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

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.

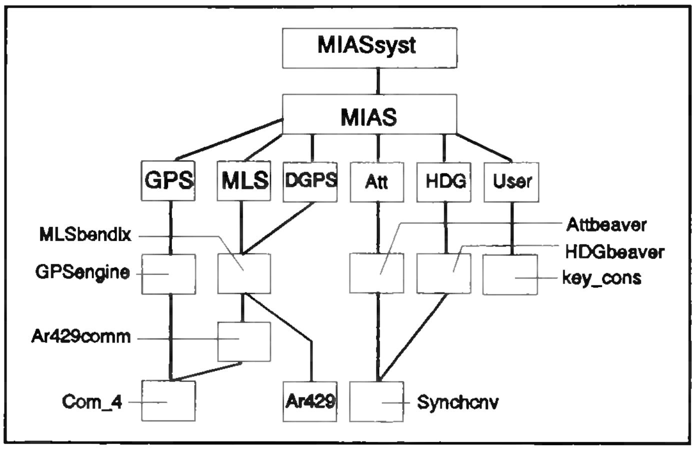

Figure 2.1 The connections between the main block of the MIAS software.

```
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[1]-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.

[1] 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.

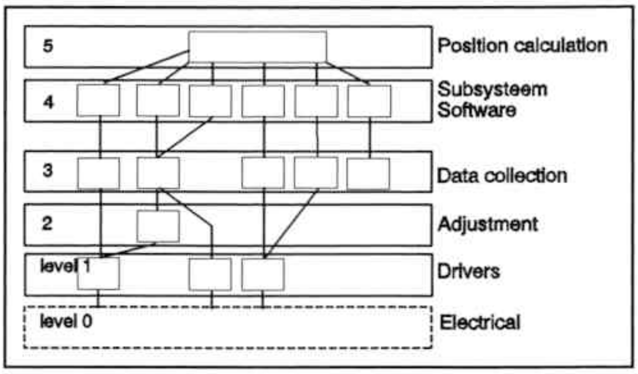

Figure 2.3 Example of an OSI approach of MIAS.

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 smoothing.

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.

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[1] should be executed from that directory. By specifying the unit and include directories[2] and selecting 'build'[3] and 'run'[4], 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 is 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

---

[1] Integrated Development Environment (IDE). The Turbo Pascal IDE is executed by typing 'turbo' at the DOS prompt.

[2] Select 'Options' at the top menu bars. Select 'Directories' from the pull down menu.

[3] Select 'Compile' from the menu bar. Select 'Build' from the pull down menu.

[4] 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 is 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~NåI=áæì;>}ó=Z)CêYaæu:+-å4âÉmëO8r‘LZ£Er
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  =  1.00000000000000E-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 =  0.00000000000000E+0000;
        alldata.pos_zerovector.y =  0.00000000000000E+0000;
        alldata.pos_zerovector.z =  0.00000000000000E+0000;
        alldata.ant_zerovector.x =  0.00000000000000E+0000;
        alldata.ant_zerovector.y =  0.00000000000000E+0000;
        alldata.ant_zerovector.z =  0.00000000000000E+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 =  1.00000000000000E+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), and 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 lattitude, 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. It is 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 send to the GPS receiver. Also a request for Ephemeris and Almanac is send.

'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 hight 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 'calcpos'

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 'execmlsreccommand'.

'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 be 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 'execcommands'.

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 'Initdgps(var dgpsdata: dgpsdatatype);'

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

'Initdgps' 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 'calcpos'.

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', 'execcommands' 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 'calcpos'.

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. It 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 if 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 and 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' ≠ '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 'Execgpsreccommand(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 DDDMM.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.
DDDMM.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,...'.

'Execgpsreccommand' 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_{HEX}. 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 okey, 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. It 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;'[1]

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_address' 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 fiels in 'MLSint'. More information can be found in [1, p 150A].

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

---

[1] 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_{HEX} 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 'Execmlsreccommand(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.

'Execmlsreccommand' 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 'Initsynchcnv(var error: boolean);'

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

'Initsynchcnv' is called by 'initatttx' 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 fro 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 is 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 okey 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 'sendmlsreccommand'.

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 datawords. 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 'miascfgname'. 'miscell' is used to open, close, etc. the configuration file.
Note: In principle it is possible to send ARINC 429 words, but his 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 'calcpos'.

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', 'calcpos', 'Getmlsdata' and 'collectMLSrec'.

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

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

'Later' is called by 'GetGPSdata', 'calcpos', '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 'execgpsreccommands' 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', 'collecport1' 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 'collecport1'.

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

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 Carrige 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 preceed 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.

```
program comtest;

uses com_4, crt;

Var
   ch   :       Char;
   x1,
   y1,
   x2,
   y2,
   x3,
   y3,
   x4,
   y4   :       Byte;

Begin
     clrscr;

     Setupcomport( 1, Ord( B4800), 8, ord( none), 1);
     Setupcomport( 2, ord( B4800), 8, ord( none), 1);
     Setupcomport( 3, Ord( B9600), 8, ord( none), 1);
     Setupcomport( 4, ord( B9600), 8, ord( none), 1);

     Emptybuffer( 1, True);
     Emptybuffer( 2, True);
     Emptybuffer( 3, True);
     Emptybuffer( 4, True);

     Installint( 1);
     Installint( 2);
     Installint( 3);
     Installint( 4);

     x1:= 1;
     y1:= 1;
     x2:= 1;
     y2:= 1;
     x3:= 1;
     y3:= 1;
     x4:= 1;
     y4:= 1;

     Repeat
           ch:= Getcharbuff( 1);
           If ch <> #00
           Then Begin
                     window( 1,1, 80, 5);
                     gotoxy( x1, y1);
                     write( ch);
                     x1:= wherex;
                     y1:= wherey;
                End;

           ch:= Getcharbuff( 2);
           If ch <> #00
           Then Begin
                     window( 1,6, 80, 10);
                     gotoxy( x2, y2);
                     write( ch);
                     x2:= wherex;
                     y2:= wherey;
                End;

           ch:= Getcharbuff( 3);
           If ch <> #00
           Then Begin
                     window( 1,11, 80, 16);
                     gotoxy( x3, y3);
                     write( ch);
                     x3:= wherex;
                     y3:= wherey;
                End;

           ch:= Getcharbuff( 4);
           If ch <> #00
           Then Begin
                     window( 1,17, 80, 24);
                     gotoxy( x4, y4);
                     write( ch);
                     x4:= wherex;
                     y4:= wherey;
                End;

     Until Keypressed;


     Removeint( 1);
     Removeint( 2);
     Removeint( 3);
     Removeint( 4);
End.
```

```
program probeersynchro;

{$N+,E+}

Uses synchcnv, crt, miasglob;

Var
   pitch,
   roll,
   heading      :       Double;
   error        :       Boolean;

begin
     ClrScr;
     Writeln( 'Press any key to end');
     initsynchcnv( error);
     Repeat
           gotoxy( 1,10);
           roll_pitch_heading( 1, pitch, error);
           roll_pitch_heading( 2, roll, error);
           roll_pitch_heading( 3, heading, error);

           writeln( 'channel 1 = ', pitch: 9: 3, ' degrees');
           writeln( 'channel 2 = ', roll: 9:3, ' degrees');
           writeln( 'channel 3 = ', heading: 9:3, ' degrees');

     Until Keypressed;
     closesynchcnv;
end.
```

```
Program read_mls;

type
    double      =       Real;
    flagtype    =       Boolean;
    mlsinttype  =       Record
                              elangle     :     double;
                              elangle_flag:     flagtype;
                              azangle     :     double;
                              azangle_flag:     flagtype;
                              elantinuse,
                              azantinuse        :byte;
                              leftclr,
                              rightclr          :boolean;
                        end;
    ar429wordtype=      Array[1..32] of byte;


   Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
                    : Word;
   {*******************************************************************}
   { This function converts single bits to a number. 'Start' indicates
     the start bit in the Arinc 429 word. 'Number' indicates the number of
     bits to be used for the number to be formed. The number is output as a
     Word. See Annex 10 p 60CC: LSB first.
                              Input : Arinc429 word
                                      startbit
                                      number of bits
                              Output: number}
   {*******************************************************************}
   Var
      value,
      mult      :       Word;
      x         :       Byte;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 0 To number - 1 Do
        Begin
             value:= value + ar429word[ start + x] * mult;
             mult:= mult * 2;
        End;
        Conv_BAS:= value;
   End;


   Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*******************************************************************}
   { This procedure converts the Arinc 429 word containing the glidepath
     information to a Pascal variable.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*******************************************************************}
   Begin
     With Mlsint Do
     Begin
        If ( Ar429word[ 27] = 1) And
           ( Ar429word[ 28] = 1) And
           ( Ar429word[ 29] = 1)
        Then Begin
                ELangle:= Conv_bas( Ar429word, 13, 14);
                ELangle:= ELangle - $4000;
                ELangle:= ELangle * 0.005;
             End;

        If ( Ar429word[ 27] = 0) And
           ( Ar429word[ 28] = 0) And
           ( Ar429word[ 29] = 0)
        Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

        ElAntInUse:= Ar429word[ 12] + 1;        { bendix: 0 = aft ant}
                                                {         1 = forward ant}
                                                { ARinc: 1,2,3 = ant no}
        ELangle_flag:= False;

     End;
   End;


   Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*******************************************************************}
   { This procedure converts the Arinc 429 word containing the azimuthangle
     information to a Pascal variable.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*******************************************************************}
   Var
      x,
      cnt       :       Byte;

   Begin
     With Mlsint Do
     Begin
        If ( Ar429word[ 27] = 1) And
           ( Ar429word[ 28] = 1) And
           ( Ar429word[ 29] = 1)
        Then Begin
                Azangle:= Conv_bas( Ar429word, 13, 14);
                Azangle:= Azangle - $4000;
                Azangle:= Azangle * 0.005;
             End;

        If ( Ar429word[ 27] = 0) And
           ( Ar429word[ 28] = 0) And
           ( Ar429word[ 29] = 0)
        Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

        AzAntInUse:= Ar429word[ 12] + 1;        { bendix: 0 = aft ant}
                                                {         1 = forward ant}
```

```
                                              { ARinc: 1,2,3 = ant no}

        cnt:= 0;                              { extra check possible :}
        For x:= 13 to 28 Do                   { |angle| < propcoverage}
            cnt:= cnt + Ar429word[ x];

        If ( cnt = 0) And ( AR429word[ 29] = 1)
        Then Leftclr:= True
        Else Leftclr:= False;

        If ( cnt = 16) And ( Ar429word[ 29] = 0)
        Then Rightclr:= True
        Else Rightclr:= False;

        If Leftclr Or Rightclr
        Then Azangle_flag:= True
        Else Begin
                  AZangle_flag:= False;

             End;
     End;
   End;

Var
   inputfile,
   azfile,
   elfile        :       Text;
   line          :       String;
   x             :       Byte;
   ar429word     :       ar429wordtype;
   code          :       Integer;
   mlsint        :       mlsinttype;


Begin
     Assign( inputfile, 'mias11.12');
     Reset( inputfile);
     Assign( elfile, 'el.dat');
     Rewrite( elfile);
     Assign( azfile, 'az.dat');
     Rewrite( azfile);

     While Not Eof( inputfile) DO
     Begin
          Readln( inputfile, line);

          If copy( line, 13, 3) = '105'
          Then Begin
                 If copy( line, 47, 2) = '11'
                 Then Begin
                    for x:= 0 to 31 Do
                    Begin
                         Val( line[ x + 17], ar429word[x+1], code);
                    End;
                    Az_Conv( mlsint, ar429word);
                    If mlsint.elangle_flag = false
                    Then writeln( azfile, copy( line, 1, 7),' ', mlsint.azangle);
                 End;
               End;

          If copy( line, 13, 3) = '106'
          Then Begin
                 If copy( line, 47, 2) = '11'
                 Then Begin
                    for x:= 0 to 31 Do
                    Begin
                         Val( line[ x + 17], ar429word[x+1], code);
                    End;
                    El_Conv( mlsint, ar429word);
                    If mlsint.azangle_flag = false
                    Then writeln( elfile, copy( line, 1, 7),' ', mlsint.elangle);
                 End;
               End;
     End;
     close( inputfile);
     Close( elfile);
     close( azfile);
End.
```

```
Program artest;
{ this program tests Bendix MLS receiver using the Max 429 PC interface card.}

Uses AR429, crt;


type
    double      =       Real;
    flagtype    =       Boolean;
    mlsinttype  =       Record
                              elangle    :       double;
                              elangle_flag:      flagtype;
                              azangle    :       double;
                              azangle_flag:      flagtype;
                              elantinuse,
                              azantinuse         :byte;
                              leftclr,
                              rightclr           :boolean;
                        end;

Const                                            { label are in octal}
     azimut_lab =       151;                     { azimuth function label}
     elevat_lab =       152;                     { elevation function label}

Var
   selecttable:arraytype;
   error                   :      Boolean;
   ar429word               :      ar429wordtype;
   noword                  :      Boolean;
   a_label                 :      Byte;
   mlsint                  :      mlsinttype;
   counter                 :      Longint;
   x                       :      Integer;
   outputfile              :      Text;


   Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
                    : Word;
   {*******************************************************************}
   { This function converts single bits to a number. 'Start' indicates
     the start bit in the Arinc 429 word. 'Number' indicates the number of
     bits to be used for the number to be formed. The number is output as a
     Word. See Annex 10 p 60CC: LSB first.
                             Input : Arinc429 word
                                     startbit
                                     number of bits
                             Output: number}
   {*******************************************************************}
   Var
      value,
      mult      :       Word;
      x         :       Byte;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 0 To number - 1 Do
        Begin
             value:= value + ar429word[ start + x] * mult;
             mult:= mult * 2;
        End;
        Conv_BAS:= value;
   End;


Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{**********************************************************************}
{ This procedure converts the Arinc 429 word containing the glidepath
  information to a Pascal variable.
                          Input : Arinc 429 word
                          Output: Mlsint}
{**********************************************************************}
Begin
  With Mlsint Do
  Begin
     If ( Ar429word[ 27] = 1) And
        ( Ar429word[ 28] = 1) And
        ( Ar429word[ 29] = 1)
     Then Begin
             ELangle:= Conv_bas( Ar429word, 13, 14);
             ELangle:= ELangle - $4000;
             ELangle:= ELangle * 0.005;
          End;

     If ( Ar429word[ 27] = 0) And
        ( Ar429word[ 28] = 0) And
        ( Ar429word[ 29] = 0)
     Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

     ElAntInUse:= Ar429word[ 12] + 1;        { bendix: 0 = aft ant}
                                             {         1 = forward ant}
                                             { ARinc: 1,2,3 = ant no}
     ELangle_flag:= False;

  End;
End;


Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{**********************************************************************}
{ This procedure converts the Arinc 429 word containing the azimuthangle
  information to a Pascal variable.
                          Input : Arinc 429 word
                          Output: Mlsint}
{**********************************************************************}
Var
  x,
  cnt       :       Byte;

Begin
```

```
    With Mlsint Do
    Begin
       If ( Ar429word[ 27] = 1) And
          ( Ar429word[ 28] = 1) And
          ( Ar429word[ 29] = 1)
       Then Begin
               Azangle:= Conv_bas( Ar429word, 13, 14);
               Azangle:= Azangle - $4000;
               Azangle:= Azangle * 0.005;
            End;

       If ( Ar429word[ 27] = 0) And
          ( Ar429word[ 28] = 0) And
          ( Ar429word[ 29] = 0)
       Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

       AzAntInUse:= Ar429word[ 12] + 1;          { bendix: 0 = aft ant}
                                                 {         1 = forward ant}
                                                 { ARinc: 1,2,3 = ant no}

       cnt:= 0;                                  { extra check possible :}
       For x:= 13 to 28 Do                       { |angle| < propcoverage}
           cnt:= cnt + Ar429word[ x];

       If ( cnt = 0) And ( AR429word[ 29] = 1)
       Then Leftclr:= True
       Else Leftclr:= False;

       If ( cnt = 16) And ( Ar429word[ 29] = 0)
       Then Rightclr:= True
       Else Rightclr:= False;

       If Leftclr Or Rightclr
       Then Azangle_flag:= True
       Else Begin
                 AZangle_flag:= False;

            End;
    End;
  End;





Begin
  Assign( outputfile, 'arinc.dat');
  Rewrite( outputfile);

  ClrScr;
  Writeln( 'Press any key to end');

  selecttable[0]:= azimut_lab;                   { program the labels}
  selecttable[1]:= elevat_lab;

  Initar429( error, selecttable, 2);

  if error
  Then Begin
               Writeln( 'Error programming ARINC 429 receiver');
               Exit;
       End;

  counter:= 0;
  Repeat
        Getar429word( ar429word, noword, a_label);
        If noword = false
        Then Begin
               For x:= 1 To 32 Do
                   Write( outputfile, ar429word[x]);
               Writeln( outputfile);

               Case a_label Of
               105: Az_conv( mlsint, ar429word);
               106: El_conv( mlsint, ar429word);
               End;

               Gotoxy( 1,15);

               If mlsint.azangle_flag
               Then Writeln( 'Azimuth   :                    flag')
               Else Writeln( 'Azimuth   : ', mlsint.azangle:10:5);

               Gotoxy( 1, 20);

               If mlsint.elangle_flag
               Then Writeln( 'Elevation :                    flag')
               Else Writeln( 'Elevation : ', mlsint.elangle:10:5);

               counter:= 10000;
             End
             Else Begin
                    If counter > 0
                    Then Dec( counter);

                    If counter = 0
                    Then Begin

                           Gotoxy( 1,15);
                           Writeln( 'No ARINC word received          ');
                           Gotoxy( 1,20);
                           Writeln( 'No ARINC word received          ');
                         End;
                  End;
        Until Keypressed;

  CLoseAr429;
  Close( outputfile);
End.
```

Appendix D Listings for MIASLOGO

```
Unit MIASglob;

{$N+,E+}

Interface

Uses Miscell;


Const
     Valid_HDGtime      :      timetype      =      (year :0;
                                                     month:0;
                                                     day  :0;
                                                     hour :0;
                                                     minute:0;
                                                     sec  :1;
                                                     sec100:0);
     Valid_Atttime      :      timetype      =      (year :0;
                                                     month:0;
                                                     day  :0;
                                                     hour :0;
                                                     minute:0;
                                                     sec  :1;
                                                     sec100:0);
     Valid_MLStime      :      timetype      =      (year :0;
                                                     month:0;
                                                     day  :0;
                                                     hour :0;
                                                     minute:0;
                                                     sec  :0;
                                                     sec100:10);
     Valid_DGPStime     :      timetype      =      (year :0;
                                                     month:0;
                                                     day  :0;
                                                     hour :0;
                                                     minute:1;
                                                     sec  :30;
                                                     sec100:0);
     Valid_GPStime      :      timetype      =      (year :0;
                                                     month:0;
                                                     day  :0;
                                                     hour :0;
                                                     minute:0;
                                                     sec  :1;
                                                     sec100:0);
     MIAScfgname        =      'MIAS.CFG';


Type
    double       =       Extended;
    flagtype     =       Boolean;    {When true, a flag is displayed}
    integritytype=       Record
                               flag: flagtype;
                         End;
    positiontype=        Record
                                WGS84lat,
                                WGS84lon,
                                WGS84alt,              { in [rad] and [m]}
                                h,                     {altitude to MSL [km]}
                                x,
                                y,
                                z          :    Double;
                                flag       :    flagtype;
                                integrity  :    integritytype;
                                EcefTrueLocalFalse:Boolean;
                         End;
    commandtype =        String;
{------------------------ GPSdatatypes ----------------------------}
    svpositiontype=      Record       {SV position}
                                   x,{ECEF}
                                   y,
                                   z : Double;
                         End;
    prntype     =        Record       {store info per satellite}
                                pr         :       Double; {pseudorange}
                                rxtime     :       Longint;
                                intcarphase:       Double; {integrated carrier}
                                flag       :       Boolean;{if True than error}
                                position   :       svpositiontype;
                         End;
    GPSdatatype =        Record
                                flag: flagtype;  { If true then error}
                                prn : Array[1..32] Of prntype;
                                deltaT :  Double;{ clockerror of user clock
                                                   relative to GPS clock}
                                present:  Byte;  { 1= present, 0=not present}
                         End;
{------------------------ MLSdatatypes ----------------------------}
    MLSpositiontype=     Record
                                x,
                                y,
                                z          :       Double;
                         End;
    ADWtype        =     Array[ 13..76] Of Byte;
    MLSdatatype =        Record
                                Azpos,             { antenna positions in}
                                Elpos,             { MLS reference system}
                                Bazpos,            { angles are in deg}
                                DMEpos     :      MLSpositiontype;
                                MLSthrespos:      positiontype;
                                Runwayhdg :       Double;
                                                   { runway heading in deg}
                                DMErange,
                                ELangle,           { corrected angles}
                                AZangle,
                                BAzangle  :       Double;
                                Leftclr,
                                Rightclr  :       Boolean;
                                ElantInUse,
                                AzantInUse:       Byte;
```

```
                        AuxB,
                        AuxC      :      ADWtype;

                        Azpos_flag,
                        Elpos_flag,
                        Bazpos_flag,
                        DMEpos_flag,
                        Runwayhdg_flag,
                        DME_flag,
                        ELangle_flag,
                        azangle_flag,
                        bazangle_flag :      flagtype;

                        AuxB_flag,
                        AuxC_flag,

                        flag      :      flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
{-----------------------------------------------------------------------}
   nablatype=      Record
                        rxtime_dgps  : Real;
                        order0,
                        order1    : Double;
                        flag      : flagtype;
                   End;

   DGPSdatatype=   Record
                        nabla     : Array [1..32] Of nablatype;
                        flag      : flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
{-----------------------------------------------------------------------}
   LORANdatatype=  Record
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   DMEdatatype =   Record
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   Altdatatype =   Record
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   DAltdatatype=   Record
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   Attdatatype =   Record
                        rollangle  :   Double;
                        pitchangle:   Double;
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   HDGdatatype =   Record
                        hdgangle  :   Double;
                        flag: flagtype;
                        present:  Byte;  { 1= present, 0=not present}
                   End;
   alldatatype =   Record
                        GPS       :   GPSdatatype;
                        MLS       :   MLSdatatype;
                        LORAN     :   LORANdatatype;
                        DME       :   DMEdatatype;
                        DGPS      :   DGPSdatatype;
                        Alt       :   Altdatatype;
                        DAlt      :   DAltdatatype;
                        Att       :   Attdatatype;
                        HDG       :   HDGdatatype;
                        pos_zerovector:positiontype;
                                          { vector from mlsant to
                                            ref pnt}
                        ant_zerovector:positiontype;
                                          { vector from mlsant to
                                            gpsant}

                   End;


Implementation

End.
```

```
Unit GPSglob;

{*************************************************************************}
{ This unit contains the types for the GPS part of MIAS.}
{*************************************************************************}

Interface

Uses MIASglob, Miscell;

Const
     Valid_Tck  :       timetype        =               (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :5;
                                                         minute:0;
                                                         sec  :0;
                                                         sec100:0);
                                                { clock is valid for 5 hours}
                                                { see STANAG }
     Valid_Tephem:      timetype        =               (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :5;
                                                         minute:0;
                                                         sec  :0;
                                                         sec100:0);
                                                { ephemeris is valid for 5}
                                                { hours, see STANAG}
     Valid_Tionos:      timetype        =               (year :0;
                                                         month:0;
                                                         day  :6;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :0;
                                                         sec100:0);
                                                { ionosphere is valid for}
                                                { 6 days}
Type
    ephemeristype=      Record
                              IODE      :       Byte;
                                { Isue of data ephemeris}
                              Crs       :       Double;
                                { Amplitude of the sin harmonic
                                  correction term to the orbit radius}
                              deltan    :       Double;
                                { Mean motion difference from computed value}
                              Mo        :       Double;
                                { Mean anomaly at reference time}
                              Cuc       :       Double;
                                { Amplitude of the cosine harmonic correction
                                  term to the argument of latitude}
                              e         :       Double;
                                { Eccentricity}
                              Cus       :       Double;
                                { Amplitude of the sine harmonic correction
                                  term to the argument of latitude}
                              Asqrt     :       Double;
                                { Square root of the semi-major axis}
                              toe       :       Double;
                                { Reference time , ephemeris}
                              Cic       :       Double;
                                { Amplitude of the cosin harmonic correction
                                  term to the angle of inclination}
                              omegao    :       Double;
                                { Longitude of ascending node of orbit plane
                                  at weekly epoch}
                              Cis       :       Double;
                                { Amplitude of the sine harmonic correction
                                  term to the angle of inclination}
                              io        :       Double;
                                { Inclination angle at reference time}
                              Crc       :       Double;
                                { Amplitude of the cosine harmonic correc-
                                  tion term to the orbit radius}
                              omega     :       Double;
                                { Argument of perigee}
                              omegadot  :       Double;
                                { rate or right ascension}
                              IDOT      :       Double;
                                { Rate of inclination angle}
                        End;
    Clocktype    =      Record
                              Tgd       :       double;
                              toc       :       double;
                              af2       :       double;
                              af1       :       double;
                              af0       :       double;
                              IODC      :       Integer;
                        End;
    prngpstype   =      Record      {store info per satellite}
                              pr        :       Double; {pseudorange}
                              health    :       Byte;   {status}
                              ephemeris :       ephemeristype;
                              clock     :       clocktype;
                              txtime    :       Double;
                              rxtime    :       Double;
                              intcarphase:      Double;
                              E         :       Double; {Elevation between
                                                         user and SV
                                                         [semicircles]}
                              A         :       Double; {Azimuth between
                                                         user and SV
                                                         [semicircles]}
                              Ek        :       Double; {EccentricAnomaly}
                              flag      :       Boolean;{if True than error}
                              position  :       svpositiontype;
                              Tck,
                              Tephem    :       timetype; { contains the
                                                            time in the
```

```
                                                          computer system
                                                          at which infor-
                                                          mation was
                                                          received.}
                    End;
   ionospheretype=  Record
                         alfa0    :      Double;
                         alfa1    :      Double;
                         alfa2    :      Double;
                         alfa3    :      Double;
                         beta0    :      Double;
                         beta1    :      Double;
                         beta2    :      Double;
                         beta3    :      Double;
                    End;
   GPSinttype  =    Record
                         flag: flagtype;    { If true then error}
                         prn : Array[1..32] Of prngpstype;
                         numofsat: Shortint;{ number of sv's being
                                              tracked}
                         Deltat     :      Double; { User clock error}
                         iono: ionospheretype;
                         Tionos: timetype;
                    End;

Implementation

Begin
End.
```

```
Unit MLSglob;
{*************************************************************************}
{ This unit contains global MLStypes and MLS variables}
{ See also the ICAO Annex 10 Part 1 page 146 and 147.}
{*************************************************************************}

Interface

Uses Miasglob, Miscell;

Const
   Valid_Bas1      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_Bas2      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :0;
                                             sec100:16);
   Valid_Bas3      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_Bas4      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_Bas5      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_Bas6      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_AuxA1     :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_AuxA2     :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_AuxA3     :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_AuxA4     :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :1;
                                             sec100:0);
   Valid_AuxB      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:2;
                                             sec  :0;
                                             sec100:0);
   Valid_AuxC      :     timetype     =     (year :0;
                                             month:0;
                                             day  :0;
                                             hour :0;
                                             minute:0;
                                             sec  :3;
                                             sec100:0);
   Az2thresdist= 1000;
   AzPropCovNegLim=40;
   AzPropCovPosLim=40;
   Cleartype=0;

   MinGP =3;
   BAZstat=0;
   DMEstat=0;
   Azstat=1;
   Elstat=1;

   AzBW=1;
   ElBW=1;
   DMEdist=0;
```

```
   AzMagOr=0;
   BazMagOr=0;
   BazPropCovNegLim=0;
   BazPropCovPosLim=0;
   BazBW=0;
   {BazStat=0;}{ is already declared}

   MLSident='AMS';

   AzOff     =0;
   Az2MLSdatdist =990;
   AzAlignRun=0;
   AzCoorSyst=0;

   ElOff=0;
   MLSdat2thres =10;
   ElHeight =0;

   DMEoff=0;
   DME2MLSdatDist=0;

   BAzOff=0;
   BAz2MLSdatDist=0;
   BAzAlignRun=0;

Type
    Basic1type  =       Record
                              Az2thresDist      :       Integer;
                                            { Azimuth to threshold distance}
                              AzPropCovNegLim,
                                            { Azimuth proportional coverage,
                                              negative limit}
                              AzPropCovPosLim,
                                            { idem, positive limit}
                              ClearType         :       Byte;
                                            { clearance signal type}
                        End;
    Basic2type  =       Record
                              MinGP     :       Real;
                                            { minimum glide path}
                              BAZstat,      { Back Azimuth status}
                              DMEstat,      { DME status}
                              Azstat,       { Azimuth status}
                              Elstat    : Byte;
                                            { Elevation status}
                        End;
    Basic3type  =       Record
                              AzBW,         { Azimuth beamwidth}
                              ElBW,         { Elevation beamwidth}
                              DMEdist    :       Real;
                                            { DME distance}
                        End;
    Basic4type  =       Record
                              AzMagOr,      { Azimuth magnetic orientation}
                              BazMagOr   :       Integer;
                                            { BackAzimuth magnetic orientation}
                        End;
    Basic5type  =       Record
                              BazPropCovNegLim,
                                            { Back Azimuth Proportional
                                              Coverage Negative limit}
                              BazPropCovPosLim:       Byte;
                                            { idem positive limit}
                              BazBW       : Real;
                                            { Back Azimuth beamwidth}
                              BazStat     : Byte;
                                            { Back Azimuth status}
                        End;
    Basic6type  =       Record
                              MLSident:         String[3];
                        End;
                                            { MLS ground equipment
                                              identification}
    AuxA1type   =       Record
                              AzOff,        { Azimuth Antenna offset}
                              Az2MLSdatdist:     Integer;
                                            { Azimuth antenna to MLS datum
                                              point distance}
                              AzAlignRun      :     Real;
                                            { Azimuth Alignment with Runway
                                              centreline}
                              AzCoorSyst      :     Byte;
                                            { Azimuth Antenna Coordinate
                                              system}
                        End;
    AuxA2type   =       Record
                              ElOff,        { Elevation antenna offset}
                              MLSdat2thres:         Integer;
                                            { MLS datumpoint 2 threshold
                                              distance}
                              ElHeight    :         Real;
                                            { Elevation Antenna Height}
                        End;
    AuxA3type   =       Record
                              DMEoff,       { DME offset}
                              DME2MLSdatDist:        Integer;
                                            { DME to MLS datum point distance}
                        End;
    AuxA4type   =       Record
                              BAzOff,       { Back azimuth antenna offset}
                              BAz2MLSdatDist:        Integer;
                                            { Back azimuth to MLS datum
                                              point distance}
                              BAzAlignRun:           Real;
                                            { Back azimuth alignment with
                                              runway centre line}
                        End;
    Discretetype=       Record
                              antenna    :       Byte;
                              test       :       Byte;
```

```
                    Azsource  :     Byte;
                    Azselwarn :     Byte;
                    Bazselwarn:     Byte;
                    GPselwarn :     Byte;
                    BAZavail  :     Byte;
                    BAZdeven  :     Byte;
                    Tuningcom :     Byte;
                    nr1antsel :     Byte;
                    changeinh :     Byte;
                    tunPrtsel :     Byte;
               End;

   MLSinttype = Record
                    Bas1      :     Basic1type;
                    Bas2      :     Basic2type;
                    Bas3      :     Basic3type;
                    Bas4      :     Basic4type;
                    Bas5      :     Basic5type;
                    Bas6      :     Basic6type;
                    AuxA1     :     AuxA1type;
                    AuxA2     :     AuxA2type;
                    AuxA3     :     AuxA3type;
                    AuxA4     :     AuxA4type;
                    AuxB,
                    AuxC      :     ADWtype;

                    Bas1_Time :     timetype;
                    Bas2_Time :     timetype;
                    Bas3_Time :     timetype;
                    Bas4_Time :     timetype;
                    Bas5_Time :     timetype;
                    Bas6_Time :     timetype;
                    AuxA1_Time:     timetype;
                    AuxA2_Time:     timetype;
                    AuxA3_Time:     timetype;
                    AuxA4_Time:     timetype;
                    AuxB_time,
                    AuxC_time :     timetype;

                    DMErange,
                    ELangle,
                    AZangle,
                    BAzangle  :     Double;
                    Discretes :     Discretestype;
                    Leftclr,
                    Rightclr  :     Boolean;
                    ElantInUse,
                    AzantInUse:     Byte;

                    Bas1_flag,
                    Bas2_flag,
                    Bas3_flag,
                    Bas4_flag,
                    Bas5_flag,
                    Bas6_flag,
                    Auxa1_flag,
                    Auxa2_flag,
                    Auxa3_flag,
                    Auxa4_flag,
                    AuxB_flag,
                    AuxC_flag,

                    ELangle_flag,
                    AZangle_flag,
                    BAzangle_flag,
                    DME_flag,
                    discretes_flag:  Boolean;

                    flag          :  Boolean;
               End;


Implementation

Begin
End.
```

```
Program MIASLogonly;

{$M 12000, 0, 650000}

Uses MIASglob, MIAS;

Var
   alldata      :       alldatatype;
   command      :       commandtype;
   position,
   filtposition,
   predposition :       positiontype;

Begin
   Init( alldata, position);
   Repeat
     SetTimerToGPSIfNotSet;
     GetUserCommands( command);
     ExecCommands( command, alldata);
     GetData( alldata);
   Until Stopcommand( command);
   CloseDown( alldata, position);
End.
```

```
Unit MIAS;

Interface

{$N+,E+}


Uses MIASglob;


Procedure Init( Var alldata: alldatatype; Var position: positiontype);
{*************************************************************************}
{Initialise the MIAS system. Open a setupfile and read some values.
                               Input :-
                               Output:status; whether or not a device present}
{*************************************************************************}


Procedure DispFlags( alldata: alldatatype; position: positiontype);
{*************************************************************************}
{Display the flags on the screen or in a file. The flags represent the valid-
 ness of a certain device. The procedure uses only the flag-fields of the
 peripheral-fields in alldata.
                               Input :alldata
                               Output:flags on screen or file}
{*************************************************************************}


Procedure GetUserCommands( Var command: commandtype);
{*************************************************************************}
{Retrieve commands for the peripherals from the keyboard.
                               Input :-
                               Output:commands string in upcase}
{*************************************************************************}


Procedure ExecCommands( command: commandtype; alldata: alldatatype);
{*************************************************************************}
{Send commands to peripherals, or execute the commands on the host.
                               Input :commands
                               Output:-}
{*************************************************************************}


Procedure GetData( Var alldata: alldatatype);
{*************************************************************************}
{Retrieve information from the peripherals.
                               Input :-
                               Output:alldata; information from peripherals}
{*************************************************************************}


Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
{*************************************************************************}
{Calculate the position from the information available.
                               Input :alldata
                               Output:position}
{*************************************************************************}


Procedure FilterPosition( position: positiontype;
                          Var filtposition: positiontype);
{*************************************************************************}
{Perform a filtering action on the positions calculated, to make sure the
 position output will be smooth.
                               Input :position
                               Output:filtered position}
{*************************************************************************}


Procedure PredictPosition( position: positiontype;
                           Var predposition: positiontype);
{*************************************************************************}
{Make a prediction of the position, to counteract the calculation and
 measurement delay.            Input :position
                               Output:predicted position}
{*************************************************************************}


Procedure SendPosition( position: positiontype);
{*************************************************************************}
{Send the position to another device from storage or displaying, or display
 the position on the screen.   Input :position
                               Output:position to the screen or a device}
{*************************************************************************}


Function Stopcommand( command: commandtype): Boolean;
{*************************************************************************}
{When a stop command is sent, this function will turn TRUE.
                               Input : commandstring
                               Output: Boolean, stop or not}
{*************************************************************************}


Procedure CloseDown( alldata: alldatatype; position: positiontype);
{*************************************************************************}
{Make sure the computer is back to 'normal'. Restore interuptvectors etc.
                               Input :-
                               Output:-}
{*************************************************************************}


Procedure SetTimerToGPSIfNotSet;
{*************************************************************************}
{ Use the GPS time to set the internal timer. If the time was already set,
  do nothing.
                               Input :-
                               Output:-}
{*************************************************************************}
```

```


Implementation

Uses GPS, DGPS, MLS, Att, HDG, PosCalc, User, Miscell, crt, dos;

Const
   Valid_tMLS          :     timetype      =      (year :0;
                                                    month:0;
                                                    day  :0;
                                                    hour :0;
                                                    minute:0;
                                                    sec  :2;
                                                    sec100:0);

Var
   stopkeypressed:     Boolean;
   DGPSflag      :     flagtype;          { Indicates DGPS mode or not}
   DGPSmode      :     Byte;
   allowed_error :     Double;
   tMLS          :     timetype;
   Old_MLS       :     MLSdatatype;
   timeset       :     Boolean;
   MIASlogname   :     String;


Procedure Init( Var alldata: alldatatype; Var position: positiontype);

Var
   setupfile     :     Text;
   error         :     Boolean;
   title,
   varname,
   line          :     String;
   value         :     String;
   code          :     Integer;

Begin
     MIASlogname:= 'c:';

     DGPSflag:= True;
     DGPSmode:= 1;

     ErrorTime( tMLS);
     With Old_MLS Do
     Begin
          flag:= True;
     End;

     TimeSet:= False;

     With position Do
     Begin
         WGS84lat:= 0;
         WGS84lon:= 0;
         WGS84alt:= 0;
         h:=0;
         x:=0;
         y:=0;
         z:=0;
         flag:= True;
         integrity.flag:= True;
     End;

     alldata.mls.MLSthresPos:= position;
     MLSantposition:= position;

     If Not FileExist( MIAScfgname)
     Then Begin
               SendUserMessage('Configfile "MIAS.CFG" not present');
               Halt( 1);                          { terminate the program }
          End;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                       { find MIAS part of
                                                    config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MIAS'));

     If Not Eof( setupfile)                       { if there is more in file}
     Then Repeat
                Readln( setupfile, line);         { get a line}
                Convert( line, varname, value);   { extract the variable name
                                                    and value}
                                                  { repeat until end of file}
                                                  { initialise comports for
                                                    communication with Engine}

                If ( varname = 'ALLOWED_ERROR')
                Then Val( value, allowed_error, code);

                If ( varname = 'POSITION.WGS84LAT')
                Then Val( value, position.wgs84lat, code);

                If ( varname = 'POSITION.WGS84LON')
                Then Val( value, position.wgs84lon, code);

                If ( varname = 'POSITION.WGS84ALT')
                Then Val( value, position.wgs84alt, code);

                If ( varname = 'DGPSMODE')
                Then Val( value, dgpsmode, code);

                If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LAT')
                Then Val( value, alldata.mls.mlsthrespos.wgs84lat, code);

                If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LON')
                Then Val( value, alldata.mls.mlsthrespos.wgs84lon, code);

                If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84ALT')
```

```
               Then Val( value, alldata.mls.mlsthrespos.wgs84alt, code);

               If ( varname = 'ALLDATA.POS_ZEROVECTOR.X')
               Then Val( value, alldata.pos_zerovector.x, code);

               If ( varname = 'ALLDATA.POS_ZEROVECTOR.Y')
               Then Val( value, alldata.pos_zerovector.y, code);

               If ( varname = 'ALLDATA.POS_ZEROVECTOR.Z')
               Then Val( value, alldata.pos_zerovector.z, code);

               If ( varname = 'ALLDATA.ANT_ZEROVECTOR.X')
               Then Val( value, alldata.ant_zerovector.x, code);

               If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Y')
               Then Val( value, alldata.ant_zerovector.y, code);

               If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Z')
               Then Val( value, alldata.ant_zerovector.z, code);

               If ( varname = 'ALLDATA.GPS.PRESENT')
               Then Val( value, alldata.gps.present, code);

               If ( varname = 'ALLDATA.DGPS.PRESENT')
               Then Val( value, alldata.dgps.present, code);

               If ( varname = 'ALLDATA.MLS.PRESENT')
               Then Val( value, alldata.mls.present, code);

               If ( varname = 'ALLDATA.ATT.PRESENT')
               Then Val( value, alldata.att.present, code);

               If ( varname = 'ALLDATA.HDG.PRESENT')
               Then Val( value, alldata.hdg.present, code);

               If ( varname = 'MIASLOGNAME')
               Then MIASlogname:= value;
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
     CloseConfig( setupfile);

     Convert_Pos_to_Ecef( position);
     Convert_Pos_to_Ecef( alldata.mls.mlsthrespos);

     MLSantposition:= position;

     InitUser( MIASlogname);

     With alldata Do
     Begin
          InitGPS( gps);
          InitMLS( mls);
          InitAtt( att);
          InitHDG( hdg);
     End;

End;


Procedure DispFlags( alldata: alldatatype; position: positiontype);

Var
   line          :       Commandtype;

Begin
     line:= '';
     With alldata Do
     Begin
          If Not GPS.flag Then line:= line + 'GPS oke '
                          Else line:= line + 'GPS err ';

          If Not DGPS.flag Then line:= line + 'DGPS oke '
                           Else line:= line + 'DGPS err ';

          If Not MLS.flag Then line:= line + 'MLS oke '
                          Else line:= line + 'MLS err ';

          If Not Att.flag Then line:= line + 'Att oke '
                          Else line:= line + 'Att err ';

          If Not HDG.flag Then line:= line + 'HDG oke '
                          Else line:= line + 'HDG err ';

          If Not position.flag Then line:= line + 'Pos oke '
                               Else line:= line + 'Pos err ';

          If ( position.integrity.flag) Then line:= line + 'int err '
                                        Else line:= line + 'int oke ';
          SendUserFlags( line);
     End;
End;


Procedure GetUserCommands( Var command: commandtype);

Begin
     GetUserMessage( command);
End;


Procedure ExecCommands( command: commandtype; alldata: alldatatype);

Var
   substr        :       String;
   varname,
   value         :       String;
   code          :       Integer;
   tempdgpsmode  :       Byte;

Begin
     substr:= Copy( command, 1, 4);
```

```

     If ( substr = 'GPS:') And ( alldata.gps.present = 1)
     Then ExecGPScommand( command);

     If ( substr = 'MLS:') And ( alldata.mls.present = 1)
     Then ExecMLScommand( command);

     If ( substr = 'ATT:') And ( alldata.att.present = 1)
     Then ExecAttcommand( command);

     If ( substr = 'HDG:') And ( alldata.hdg.present = 1)
     Then ExecHDGcommand( command);

     substr:= Copy( command, 1, 5);

     If ( substr = 'DGPS:') And ( alldata.dgps.present = 1)
     Then ExecDGPScommand( command);

     If substr = 'MIAS:'
     Then Begin
               substr:= Copy( command, 6, Length( command) - 5);
               Convert( ' '+substr, varname, value);{ add space for}
                                                    { procedure convert}
               If (varname = 'DGPSMODE')
               Then Begin
                         Val( value, tempdgpsmode, code);
                         If code <> 0
                         Then SendUserMessage( 'Error')
                         Else dgpsmode:= tempdgpsmode;
                    End;
          End;
End;


Procedure GetData( Var alldata: alldatatype);

Begin
     With alldata Do
     Begin
         If ( gps.present = 1)
         Then GetGPSdata( GPS, DGPSflag);

         If ( mls.present = 1) Or                  { call getmlsdata also}
            ( dgps.present = 1)                    { for dgps data}
         Then GetMLSdata( MLS);

         If ( dgps.present = 1)
         Then GetDGPSdata( MLS, DGPS);

         If Not gps.flag
         Then Begin
                   If ( att.present = 1)
                   Then GetAttdata( Att);

                   If ( hdg.present =1 )
                   Then GetHDGdata( HDG);
              End;
     End;
End;


Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);

Const
     MaxNumOfIt =      1;                      { 1 is for no iterations}
Var
   old_deltaT   :      Double;
   iter         :      Integer;
   t,
   result       :      timetype;


Begin
    Case DGPSmode Of
    1: DGPSflag:= False;                        { no Diff GPS}
    2: ;                                        { only DGPS in prop cov}
    3: DGPSflag:= True;                         { always DGPS if available}
    End;

    If ( alldata.mls.present = 1)
    Then Begin
              CalcMLS( alldata.MLS);

              If Not alldata.mls.flag          { if valid info, then adjust}
              Then Date_and_time( tMLS);       { receive time}

              Date_and_time( t);               { get current system time}
              AddTime( tMLS, Valid_tMLS, result);
                                               { MLS only valid for 'Valid_
                                                 tMLS' seconds}
              If alldata.MLS.flag And          { if no valid info and}
                 Not Later( t, result)         { no time out}
              Then alldata.MLS:= Old_MLS       { use old info}
              Else Old_MLS:= alldata.MLS;      { store valid info for later
                                                 use}
         End;

    If ( alldata.gps.present = 1)
    Then CalcGPS( alldata.GPS, position, DGPSflag);

    If ( alldata.dgps.present = 1)
    Then CalcDGPS( alldata);

    CalcHybridPos( alldata, allowed_error, position);
End;


Procedure FilterPosition( position: positiontype;
                          Var filtposition: positiontype);

```

```
Begin
End;


Procedure PredictPosition( position: positiontype;
                           Var predposition: positiontype);
Begin
End;


Procedure SendPosition( position: positiontype);

Var
   line        :       String;
   dum         :       String;

Begin
     If position.flag
     Then Exit;

     line:= '';
     With position DO
     Begin
          If EcefTrueLocalFalse
          Then Begin
                    Str( wgs84lat * 180 / pi:23, dum);
                    line:= line + 'lat = ' + dum + ' ';
                    Str( wgs84lon * 180 / pi:23, dum);
                    line:= line + 'lon = ' + dum + ' ';
                    Str( wgs84alt:23, dum);
                    line:= line + 'alt = ' + dum + ' ';
                    SendUserMessage( line);

                    Str( x:23, dum);
                    line:= 'x = ' + dum + ' ';
                    Str( y:23, dum);
                    line:= line + 'y = ' + dum + ' ';
                    Str( z:23, dum);
                    line:= line + 'z = ' + dum + ' ';
                    SendUserMessage( line);
               End
          Else Begin
                    Str( x:23, dum);
                    line:= line + 'a = ' + dum + ' ';
                    Str( y:23, dum);
                    line:= line + 'y = ' + dum + ' ';
                    Str( z:23, dum);
                    line:= line + 'z = ' + dum + ' ';
                    SendUserMessage( line);
               End;
     End;
End;


Function Stopcommand( command: commandtype): Boolean;

Begin
     If Copy( command, 1, 4) = 'STOP'
     Then stopcommand:= True
     Else stopcommand:= False;
End;


Procedure CloseDown( alldata: alldatatype; position: positiontype);

Var
   setupfile    :       Text;
   Value        :       String;

Begin
     OpenConfigWriteFirst( setupfile, MIAScfgname);
     Writeln( setupfile, 'MIAS');

     Str( allowed_error, value);
     Writeln( setupfile, #9'allowed_error  = ', value, ';');

     With position Do
     Begin
          Str( WGS84lat, value);
          Writeln( setupfile, #9'position.wgs84lat = ', value, ';');

          Str( WGS84lon, value);
          Writeln( setupfile, #9'position.wgs84lon = ', value, ';');

          Str( WGS84alt, value);
          Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
     End;
     Str( dgpsmode, value);
     Writeln( setupfile, #9'dgpsmode = ', value, ';');

     With alldata.mls.MLSthrespos Do
     Begin
          Str( wgs84lat, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lat = ',
                              value, ';');

          Str( wgs84lon, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lon = ',
                              value, ';');

          Str( wgs84alt, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84alt = ',
                              value, ';');
     End;

     With alldata.Pos_zerovector Do
     Begin
          Str( x, value);
          Writeln( setupfile, #9'alldata.pos_zerovector.x = ', value, ';');

```

```
          Str( y, value);
          Writeln( setupfile, #9'alldata.pos_zerovector.y = ', value, ';');

          Str( z, value);
          Writeln( setupfile, #9'alldata.pos_zerovector.z = ', value, ';');
     End;

     With alldata.Ant_zerovector Do
     Begin
          Str( x, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.x = ', value, ';');

          Str( y, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.y = ', value, ';');

          Str( z, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.z = ', value, ';');
     End;

     With alldata Do
     Begin
          Str( gps.present, value);
          Writeln( setupfile, #9'alldata.gps.present = ', value, ';');

          Str( dgps.present, value);
          Writeln( setupfile, #9'alldata.dgps.present = ', value, ';');

          Str( mls.present, value);
          Writeln( setupfile, #9'alldata.mls.present = ', value, ';');

          Str( att.present, value);
          Writeln( setupfile, #9'alldata.att.present = ', value, ';');

          Str( hdg.present, value);
          Writeln( setupfile, #9'alldata.hdg.present = ', value, ';');
     End;

     Writeln( setupfile, #9'miaslogname = ', miaslogname, ';');

     CloseConfig( setupfile);

     With alldata Do
     Begin
          If ( gps.present = 1)
          Then CloseGPS( gps, position);

          If ( dgps.present = 1)
          Then CloseDGPS( dgps);

          If ( mls.present = 1)
          Then CloseMLS( mls);

          If ( att.present = 1)
          Then CloseAtt( att);

          If ( hdg.present = 1)
          Then CloseHDG( hdg);
     End;

     CloseUser;
End;



Procedure SetTimerToGPSIfNotSet;

Var
   gpstime      :       Longint;
   valid        :       boolean;
   hour,
   minute,
   sec,
   sec100,
   year,
   month,
   day,
   dayofweek    :       Word;


Begin
     If Not TimeSet
     Then Begin
               GetGPStime( gpstime, valid);
               If valid
               Then Begin
                         day:= gpstime div 86400;
                         gpstime:= gpstime Mod 86400;

                         hour:= gpstime div 3600;
                         gpstime:= gpstime Mod 3600;

                         minute:= gpstime div 60;
                         gpstime:= gpstime Mod 60;

                         sec:= gpstime;
                         sec100:= 0;

                         SetTime( hour, minute, sec, sec100);
                         TimeSet:= True;

                         SendUserMessage(
                              'Time set to User time of GPS receiver');
                    end;
          End;
End;


Begin
End.
```

```
Unit GPS;

Interface

{$N+,E+}

Uses MIASglob, crt;

Procedure InitGps( Var GPSdata: gpsdatatype);
{****************************************************************************}
{Initialise the GPS part of the system.
                             Input :-
                             Output:error; If something went wrong, error is
                                           set to True}
{****************************************************************************}


Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
{****************************************************************************}
{Retrieve data from the GPS receiver connected to the system. Check the col-
 lected data for age. When True, the DGPS flag indicates that DGPS mode is
 active, and only ephemeris and pseudoranges should be valid.
                             Input :DGPS flag
                             Output:GPSdata; relevant data from the GPS
                                             receiver}
{****************************************************************************}


Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
                   DGPS: Boolean);
{****************************************************************************}
{ This procedure executes the nescessary calculations for GPS. If the DGPS
  flag is true, only the SV position and the Elevation and Azimuth to the SV
  are calculated (For DGPS no corrections for clock, ionosphere, troposphere
  etc are needed). If the DGPS flag is false, then all these corrections are
  needed.
                              Input : GPSdata, ephemeris etc
                                      DGPS flag
                              Output: GPSdata, SV positions}
{****************************************************************************}


Procedure ExecGPScommand( command: commandtype);
{****************************************************************************}
{ This procedure receives a command destined for the GPS part of the MIAS
  system. It passes the command on to the GPSreceiver-part of the system.
                              Input : command
                              Output:-}
{****************************************************************************}


Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
{****************************************************************************}
{Make sure the GPS part is back to 'normal'.
                             Input :-
                             Output:-}
{****************************************************************************}


Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
{****************************************************************************}
{ Get the GPStime from the 'gpsint' variable. The resolution is in seconds.
                             Input :-
                             Output:GPStime and a valid boolean}
{****************************************************************************}


Implementation

Uses GPSEngine, GPScalc, GPSglob, Miscell, user;

Var
   GPSint        :       GPSinttype;
   El_limit      :       Byte;                    { satellite elevation limit}
   HorAccFac     :       Real;                    { horizontal acceleration
                                                     factor}
   x             :       Integer;                 {counter for initialising}


Procedure InitGps( Var GPSdata: gpsdatatype);

Var
   setupfile     :       Text;
   title         :       String;
   line          :       String;
   varname       :       String;
   value         :       String;
   code          :       Integer;
   deg,
   min           :       Double;
   time          :       timetype;
   position      :       positiontype;
   x             :       Integer;                 {counter for initialising}
   error         :       Boolean;

Begin
     error:= True;
     GPSdata.flag:= error;
     GPSdata.deltaT:= 0;
     For x:= 1 To 32 Do
          With GPSdata. prn[ x] Do
          Begin
             flag:= True;
             pr  := 0;
          End;
     If ( gpsdata.present = 0)
     Then Exit;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                         { find GPS part of
```

```
                                             config file}
          Readln( setupfile, title);
    Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPS'));

    If Not Eof( setupfile)                   { if there is more in file}
    Then Repeat
              Readln( setupfile, line);      { get a line}
              Convert( line, varname, value);  { extract the variable name
                                               and value}
                                             { repeat until end of file}
              If ( varname = 'POSITION.WGS84LAT')
              Then Val( value, position.wgs84lat, code);

              If ( varname = 'POSITION.WGS84LON')
              Then Val( value, position.wgs84lon, code);

              If ( varname = 'POSITION.WGS84ALT')
              Then Val( value, position.wgs84alt, code);

              If ( varname = 'EL_LIMIT')
              Then Val( value, el_limit, code);

              If ( varname = 'HORACCFAC')
              Then Val( value, horaccfac, code);
         Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
    CloseConfig( setupfile);

    InitGPSrec( error);

    ExecGPSreccommand( 'GPS:RESET');

    Date_and_time( time);
    With time Do                             { construct a line to send}
    Begin                                    { to the gpsengine receiver}
         Str( day, value);                   { for initialisation.}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= value + ' ';                 { It contains the date and}
                                             { time, the latitude, longi-}
         Str( month, value);                 { tude and altitude of the}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value+ ' ';           { last position}

         Str( year, value);                  { don't take the 19 from }
         value:= Copy( value, 3, 2);         { 19xx, only take xx}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value + ' ';          { add zeros}

         Str( hour, value);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value;                { add zeros}

         Str( minute, value);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value + '00 ';        { add zeros}
    End;
    With position Do
    Begin
         deg:= Abs( Trunc( wgs84lat * 180 / pi));
         Str( deg :2 : 0, value);
         While value[ 1] = ' ' Do              { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value;                  { add zeros}

         min:= 60 * ( Abs( wgs84lat * 180 / pi) - deg);
         Str( min :7 : 4, value);
         While value[ 1] = ' ' Do              { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value + ' ';            { add zeros}

         If deg < 0
         Then line:= line + 'S '
         Else line:= line + 'N ';

         deg:= Abs( Trunc( wgs84lon * 180 / pi));
         Str( deg :2 :0, value);
         While value[ 1] = ' ' Do              { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 3 - Length( value)) + value;
         line:= line + value;                  { add zeros}

         min:= 60 * ( Abs( wgs84lon * 180 / pi) - deg);
         Str( min :7 :4, value);
         While value[ 1] = ' ' Do              { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value + ' ';            { add zeros}

         If deg > 0
         Then line:= line + 'E '
         Else line:= line + 'W ';

         If Abs( wgs84alt) > 99999.9           { if overflow}
         Then value:= '99999.9'                { then take maximum value}
         Else Str( wgs84alt :7 :1, value);
         While value[ 1] = ' ' Do              { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         If wgs84alt < 0
         Then value:= '-' + Copy( zeros, 1, 7 - Length( value)) +
                      Copy( value, 2, Length( value))
         Else value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value;                  { add zeros}
    End;
    Str( HorAccFac :4 :1, value);
    While value[1] = ' ' Do
         value:= Copy( value, 2, Length( value) -1);
    value:= Copy( zeros, 1, 4 - Length( value)) + value;
    line:= line + ' ' + value;
```

```
    Str( el_limit: 2, value);
    While value[1] = ' ' Do
         value:= Copy( value, 2, Length( value) -1);
    value:= Copy( zeros, 1, 2 - Length( value)) + value;
    line:= line + ' ' + value;

    ExecGPSreccommand( 'GPS:INIT '+ line);
    ExecGPSreccommand( 'GPS:SEND EPHEMERIS ETC');

    GPSdata.flag:= error;
End;


Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);


Var
   currenttime,
   result        :       timetype;
   x             :       Byte;

Begin
    CollectGPSrec( GPSint);

    Date_and_Time( currenttime);                   { check if data valid}

    GPSint.flag:= True;                            { begin with assumption}
                                                   { that info is incorrect}
    For x:= 1 To 32 Do
        GPSint.flag:= GPSint.flag And GPSint.prn[x].flag;

    For x:= 1 To 32 Do                             { update external var}
        GPSdata.prn[x].flag:= GPSint.prn[x].flag;
    GPSdata.flag:= GPSint.flag;
End;


Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
                   DGPS: Boolean);

Var
   sv_id        :       Byte;
   line,
   dum          :       String;

Begin
    If GPSint.flag                                 { GPSint not valid}
    Then Exit;                                     { GPSint is a global
                                                     variable in the GPS units}

    With GPSint Do
    For sv_id:= 1 To 32 Do
    Begin
       If Not prn[ sv_id].flag
       Then Begin
gotoxy (1,5);
write (prn[ sv_id].rxtime:10:2);
                GPSdata.prn[ sv_id]. rxtime:= Round(GPSint.prn[sv_id].rxtime);

                                                   { note: in poscalc deltaT}
                                                   { is substracted. Doing so}
                                                   { here would be obsolete}
                Clockcorrection( sv_id, GPSint);   { always correct for clock}
                RelCorrection( sv_id, GPSint);
                If ( Not DGPS) And ( Not position.flag)
                Then Begin
                        L1correction( sv_id, GPSint);
                        Convert_Pos_to_WGS( position);
                        Elev_Azim( sv_id, GPSint, position);
                        Ionosphericcorrection( sv_id, position, GPSint);
{                        Troposphericcorrection( sv_id, position, GPSint);}
                     End;

                SVposition( sv_id, GPSint);
                Calc_Pr( sv_id, GPSint);
                gpsint.deltat:= gpsdata.deltat;  { deltaT from position
                                                   calculation copied to
                                                   internal variable to
                                                   be used to correct
                                                   satellite position}
                SVpos_earthadjusted( sv_id, GPSint);

                GPSdata.prn[sv_id].position:= GPSint.prn[sv_id].position;
                GPSdata.prn[sv_id].pr:= GPSint.prn[sv_id].pr;
                GPSdata.prn[sv_id].intcarphase:=GPSint.prn[sv_id].intcarphase;
             End;
    End;
End;


Procedure ExecGPScommand( command: commandtype);

Begin
    ExecGPSreccommand( command);
End;


Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);

Var
   x     :      Byte;
   setupfile    :    Text;
   value        :    String;


Begin
    OpenConfigWrite( setupfile, MIAScfgname);
    Writeln( setupfile, 'GPS');

    With position Do
```

```
     Begin
          Str( WGS84lat, value);
          Writeln( setupfile, #9'position.wgs84lat = ', value, ';');

          Str( WGS84lon, value);
          Writeln( setupfile, #9'position.wgs84lon = ', value, ';');

          Str( WGS84alt, value);
          Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
     End;

     Str( el_limit, value);
     Writeln( setupfile, #9'el_limit', ' = ', el_limit, ';');

     Str( horaccfac, value);
     Writeln( setupfile, #9'horaccfac', ' = ', horaccfac, ';');

     CloseConfig( setupfile);

     CloseGPSrec;
End;


Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);

Var
   x    :       Byte;

Begin
     If gpsint.flag = true
     Then Begin
               valid:= False;
               Exit;
          End
     Else Begin
               x:=1;
               While ( gpsint.prn[x].flag = true) And
                     ( x < 32) Do
                     Inc( x);
               If x <= 32
               Then Begin
                         gpstime:= Round( gpsint.prn[x].rxtime);
                         valid:= True;
                    End
               Else valid:= False;
          End;
End;


Begin                                        { initialising part}
     GPSint.flag:= True;
     GPSint.numofsat := 0;
     GPSint.deltaT:= 0;
     ErrorTime( GPSint.Tionos);
     For x:= 1 To 32 Do
          With GPSint.prn[x] Do
          Begin
             flag:= True;
             Ek:=0;
             pr:= 0;
             ErrorTime( Tck);
             ErrorTime( Tephem);
          End;

     El_limit:= 0;
     HorAccFac:= 0;
End.
```

```
Unit MLS;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InitMLS( Var MLSdata: MLSdatatype);
{*****************************************************************************}
{Initialise the MLS sensor. If something went wrong, the error flag is set
 to the value True.          Input :-
                             Output:error}
{*****************************************************************************}


Procedure GetMLSdata( Var MLSdata: MLSdatatype);
{*****************************************************************************}
{Retrieve MLS data from the MLS sensor.
                             Input :-
                             Output:MLSdata}
{*****************************************************************************}



Procedure CalcMLS( Var MLSdata: MLSdatatype);
{*****************************************************************************}
{This procedure performs the necessary MLS calculations.
                              Input : MLSdata
                              Output: MLSdata.}
{*****************************************************************************}


Procedure ExecMLScommand( command: commandtype);
{*****************************************************************************}
{This procedure sends commands to the MLS receiver.
                              Input : command
                              Output: -}
{*****************************************************************************}


Procedure CloseMLS( MLSdata: MLSdatatype);

{*****************************************************************************}
{Closedown the MLS sensor.  Input :-
                            Output:-}
{*****************************************************************************}


Implementation


Uses MLSbendix, MLSglob, Miscell, crt, user;

Var
   MLSint       :       MLSinttype;


Procedure InitMLS( Var MLSdata: MLSdatatype);

Var
   setupfile    :       Text;
   title        :       String;
   line         :       String;
   varname      :       String;
   value        :       String;
   code         :       Integer;
   error        :       Boolean;

Begin
     error:= True;
     MLSdata.flag:= error;

     If ( mlsdata.present = 0)
     Then Exit;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                            { find MLS part of
                                                         config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 3) = 'MLS'));

     If Not Eof( setupfile)                            { if there is more in file}
     Then Repeat
                Readln( setupfile, line);              { get a line}
                Convert( line, varname, value);        { extract the variable name
                                                         and value}
                                                       { repeat until end of file}
                                                       { initialise comports for
                                                         communication with Engine}
          Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
     CloseConfig( setupfile);

     InitMLSrec( error);
     MLSdata.flag:= error;
End;


Procedure GetMLSdata( Var MLSdata: MLSdatatype);

Var
   result,
   currenttime    :   Timetype;

Begin
     CollectMLSrec( MLSint);

End;

```

```
Procedure CalcMLS( Var MLSdata: MLSdatatype);

Begin
    With MLSdata Do
    Begin                                      { calculate the el, az and}
                                               { baz position referenced}
                                               { to the MLS datum point}
        With Azpos Do                          { MLSint is a global var-}
        If ( Not MLSint.AuxA1_flag)            { iable in the MLS units}
        Then Begin
                x:= -1 * MLSint.AuxA1.Az2MLSdatdist;
                y:= MLSint.AuxA1.Azoff;
                z:= 0;
                Azpos_flag:= False;
            End;
        With Elpos Do
        If Not MLSint.AuxA2_flag
        Then Begin
                x:= 0;
                y:= MLSint.AuxA2.Eloff;
                z:= MLSint.AuxA2.ElHeight;
                Elpos_flag:= False;
            End;
        With Bazpos Do
        If ( Not MLSint.AuxA4_flag)
        Then Begin
                x:= MLSint.AuxA4.Baz2MLSdatdist;
                y:= MLSint.AuxA4.Bazoff;
                z:= 0;
                Bazpos_flag:= False;
            End;
        With DMEpos Do
        If ( Not MLSint.AuxA3_flag)
        Then Begin
                x:= -1 * MLSint.AuxA3.DME2MLSdatdist;
                y:= MLSint.AuxA3.DMEoff;
                z:= 0;
                DMEpos_flag:= False;
            End;
        If (Not MLSint.ELangle_flag) And (MLSint.bas2.elstat = 1)
        Then Begin
                ELangle:= MLSint.ELangle;
                ELangle_flag:= False;
            End
        Else Elangle_flag:= True;

        If (Not MLSint.AZangle_flag) And (MLSint.bas2.azstat = 1) And
           (Not MLSint.AuxA1_flag)
        Then Begin
                Azangle:= MLSint.Azangle + MLSint.AuxA1.AzAlignRun;
                Azangle_flag:= False;
            End
        Else Azangle_flag:= True;

        If (Not MLSint.BAZangle_flag) And (MLSint.bas2.bazstat = 1) And
           (MLSint.bas5.bazstat = 1) And (Not MLSint.AuxA4_flag)
        Then Begin
                Bazangle:= MLSint.Bazangle - MLSint.AuxA4.BazAlignRun;
                BAzangle_flag:= False;
            End
        Else BAzangle_flag:= True;

        If (Not MLSint.DME_flag) And (MLSint.bas2.dmestat > 0)
        Then Begin
                DMErange:= MLSint.DMErange;
                DME_flag:= False;
            End
        Else DME_flag:= True;

        flag:= ( Elangle_flag Or Elpos_flag) And
               ( Azangle_flag Or Azpos_flag) And
               ( Bazangle_flag Or Bazpos_flag) And
               ( DME_flag Or DMEpos_flag);      { set the MLS flag}

{ NOTE: this is magnetic heading}
        If ( Not MLSint.Bas4_flag) And (Not MLSint.AuxA1_flag)
        Then Begin
                Runwayhdg:= MLSint.Bas4.AzMagOr -
                            MLSint.AuxA1.AzAlignRun + 180;
                If Runwayhdg >= 360
                Then Runwayhdg:= Runwayhdg - 360;
                Runwayhdg_flag:= False;
            End
        Else Runwayhdg_flag:= True;
    End;
End;


Procedure ExecMLScommand( command: commandtype);

Begin
    ExecMLSreccommand( command);
End;


Procedure CloseMLS( MLSdata: MLSdatatype);

Begin
    CloseMLSRec;
End;


Begin
    With MLSint Do
    Begin
        Bas1_flag:= True;
        Bas2_flag:= True;
        Bas3_flag:= True;
        Bas4_flag:= True;
        Bas5_flag:= True;
```

```
          Bas6_flag:= True;
          AuxA1_flag:= True;
          AuxA2_flag:= True;
          AuxA3_flag:= True;
          AuxA4_flag:= True;
          AuxB_flag:= True;
          AuxC_flag:= True;

          ELangle_flag:= True;
          AZangle_flag:= True;
          BAzangle_flag:= True;
          DME_flag:= True;
          discretes_flag:= True;
          leftclr:= True;
          rightclr:= True;
          flag:= True;

          ErrorTime( Bas1_time);
          ErrorTime( Bas2_time);
          ErrorTime( Bas3_time);
          ErrorTime( Bas4_time);
          ErrorTime( Bas5_time);
          ErrorTime( Bas6_time);
          ErrorTime( AuxA1_time);
          ErrorTime( AuxA2_time);
          ErrorTime( AuxA3_time);
          ErrorTime( AuxA4_time);
          ErrorTime( AuxB_time);
          ErrorTime( AuxC_time);
     End;
End.
```

```
Unit Att;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InitAtt( Var Attdata: Attdatatype);
{**************************************************************************}
{Initialise the Attitude sensor. If the initialising was not succesfull, the
 error-flag will be true.        Input :-
                                 Output:error}
{**************************************************************************}


Procedure GetAttdata( Var Attdata: Attdatatype);
{**************************************************************************}
{Retrieve attitude data from the sensor.
                                 Input :-
                                 Output:Attdata}
{**************************************************************************}


Procedure ExecAttcommand( command: commandtype);
{**************************************************************************}
{ This procedure receives a command and passes it on to the att device
  driver.
                             Input : command
                             Output: -}
{**************************************************************************}


Procedure CloseAtt( Attdata: Attdatatype);
{**************************************************************************}
{CLosedown the attitude sensor.  Input :-
                                 Output:-}
{**************************************************************************}


Implementation

Uses AttBeaver;

Procedure InitAtt( Var Attdata: Attdatatype);

Var
   error        :       Boolean;

Begin
     error:= True;
     Attdata.flag:= error;

     If ( attdata.present = 0)
     Then Exit;

     InitAttTX( error);
     Attdata.flag:= error;
End;


Procedure GetAttdata( Var Attdata: Attdatatype);

Begin
     CollectAtt( attdata);
End;


Procedure ExecAttcommand( command: commandtype);
Begin
     ExecAtttxcommand( command);
End;


Procedure CloseAtt( Attdata: Attdatatype);

Begin
     CloseAttTx;
End;

Begin
End.
```

```
Unit HDG;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InitHDG( Var HDGdata: HDGdatatype);
{****************************************************************************}
{Initialise the HDG sensor. If the initialising was not succesfull, the
 error-flag will be true.        Input :-
                                 Output:error}
{****************************************************************************}


Procedure GetHDGdata( Var HDGdata: HDGdatatype);
{****************************************************************************}
{Retrieve heading data from the sensor.
                                 Input :-
                                 Output:HDGdata}
{****************************************************************************}


Procedure ExecHDGcommand( command: commandtype);
{****************************************************************************}
{ This procedure receives a command for the HDG sensor. It passes
  this command to the HDG device driver.
                              Input : command
                              Output: -}
{****************************************************************************}


Procedure CloseHDG( HDGdata: HDGdatatype);
{****************************************************************************}
{CLosedown the heading sensor.   Input :-
                                 Output:-}
{****************************************************************************}



Implementation

Uses HDGbeaver;


Procedure InitHDG( Var HDGdata: HDGdatatype);

Var
   error        :       Boolean;

Begin
     error:= True;
     HDGdata.flag:= error;

     If ( hdgdata.present = 0)
     Then Exit;

     InitHDGtx( error);
     HDGdata.flag:= error;
End;


Procedure GetHDGdata( Var HDGdata: HDGdatatype);

Begin
     CollectHdg( hdgdata);
End;


Procedure ExecHDGcommand( command: commandtype);

Begin
    ExecHDGtxcommand( command);
End;


Procedure CloseHDG( HDGdata: HDGdatatype);

Begin
     CloseHDGTX;
End;


Begin
End.
```

```
Unit DGPS;

{$N+,E+}

Interface

Uses MIASglob, crt;

Procedure InitDGPS( Var DGPSdata: DGPSdatatype);
{***************************************************************************}
{Initialise the DGPS sensor. If the initialising was not succesfull, the
 error-flag will be true.          Input :-
                                   Output:error}
{***************************************************************************}


Procedure GetDGPSdata( Var MLSdata: MLSdatatype; Var DGPSdata: DGPSdatatype);
{***************************************************************************}
{Retrieve differential GPS data from the sensor.
                                   Input :MLSdata ( ADW's )
                                   Output:DGPSdata}
{***************************************************************************}


Procedure CalcDGPS( Var alldata: alldatatype);
{***************************************************************************}
{ Calculate the corrected ranges for GPS pseudoranges
        So the record containing differential corrections, are used to
  correct. The GPS ranges are so corrected for SA etc.
                               Input : alldata, especially DGPS
                               Output: alldata, especially GPS}
{***************************************************************************}


Procedure ExecDGPScommand( command: commandtype);
{***************************************************************************}
{ This procedure receives a command for the DGPS part of MIAS. The command
  will be passed on to the DGPS device driver.
                               Input : command
                               Output: -}
{***************************************************************************}


Procedure CloseDGPS( DGPSdata: DGPSdatatype);
{***************************************************************************}
{CLosedown the differential GPS sensor.
                                   Input :-
                                   Output:-}
{***************************************************************************}



Implementation

Uses Miscell;

Procedure InitDGPS( Var DGPSdata: DGPSdatatype);

Var
   x    :       Byte;
   error:       Boolean;

Begin
     error:= True;
     With Dgpsdata Do
     Begin
          For x:= 1 To 32 DO
          Begin
               nabla[x].flag:= True;
          End;
     End;
     DGPSdata.flag:= error;
End;


Procedure GetDGPSdata( Var MLSdata: MLSdatatype; Var DGPSdata: DGPSdatatype);

Const
     resolution0 =      0.005;
     resolution1 =      0.01;

Var
   currenttime :       Timetype;
   adress,
   sv_id  :       Integer;


   Function ADW_read( ADW: ADWtype; position, bits: Integer): Longint;
   {***********************************************************************}
   { This function takes bits 13 to 18 of the ADW and translates these
     bits to an adress. This is a normal binary code. See Annex 10 p 150
                                  Input : ADW
                                  Output: adress}
   {***********************************************************************}
   Var
      value,
      mult,
      x          :       Longint;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 2 To Bits Do
           mult:= mult * 2;

        For x:= 0 To bits-1 Do
        Begin
             value:= value + ADW[ position + x] * mult;
             mult:= mult Div 2;
        End;
```

```
        ADW_read:= value;
    End;

    Function Twos_complement( ADW: ADWtype; position, bits: Integer): Longint;
    {*******************************************************************}
    { This function takes bits 13 to 18 of the ADW and translates these
      bits to an adress. This is a normal binary code. See Annex 10 p 150
                              Input : ADW
                              Output: adress}
    {*******************************************************************}
    Var
       bin_max,
       value,
       mult,
       x          :       Longint;

    Begin
        value:= 0;
        bin_max := Round ( Exp ( ( bits - 1 ) * Ln ( 2 ) ) );
        mult:= bin_max;

        For x:= 0 To bits-1 Do
        Begin
            value:= value + ADW[ position + x] * mult;
            mult:= mult Div 2;
        End;
        IF value > bin_max
        THEN BEGIN
           value := value - bin_max * 2;
        END;

        Twos_complement:= value;
    End;

Begin
End;


Procedure CalcDGPS( Var alldata: alldatatype);

Var
   x     :       Integer;
   h,
   m,
   s,
   s100 :       Word;
   use_time,
   update_time : Real;
   delta_PR    : Double;
   delayT      : Double;
   currenttime : timetype;

Begin
     Alldata.dgps.flag:= True;
     delayT     :=  5 ;   { halve window lengte }

     With alldata Do
         For x := 1 to 32 Do
             With DGPS.nabla[x] Do
             Begin
                  If NOT flag Then
                  Begin
                       update_time := alldata.GPS.prn[x].rxtime -
                                      rxtime_dgps
                                      + delayT;          { time between receiving at
                                                           ground station and mobile
                                                           receiver }
                       If Abs( update_time) > 90
                       Then dgps.nabla[x].flag:= true
                       Else Begin
                            delta_PR := order0 + update_time * order1;

                                                          { Time update for the
                                                            correction data }
                            With GPS.prn[x] Do
                                 PR := PR - delta_PR;           { correction of th
                       End;
                  End;
                  dgps.flag:= dgps.flag And dgps.nabla[x].flag;
             End;

End;



Procedure ExecDGPScommand( command: commandtype);

Begin
{     ExecDGPSreccommand( command);}
End;


Procedure CloseDGPS( DGPSdata: DGPSdatatype);

Begin
End;

Begin
End.
```

```
Unit User;

{*****************************************************************************}
{ This unit provides the necessary routines for inputting and outputting
  information from and to a user. This user may be a person, or an FMS.}
{*****************************************************************************}

Interface

Uses MiasGlob;

Procedure InitUser( MIASlogname: string);
{****************************************************************************}
{ Only here for compatibility}
{****************************************************************************}


Procedure SendUserMessage( Message: commandtype);
{****************************************************************************}
{ display a message commandtype to the user}
{                             Input : message
                              Output: message on the outputdevice}
{****************************************************************************}


Procedure GetUserMessage( Var Message: commandtype);
{****************************************************************************}
{ display a message commandtype to the user}
{                             Input : message from the inputdevice
                              Output: message }
{****************************************************************************}


Procedure SenduserFlags( Message: commandtype);
{****************************************************************************}
{ This procedure sends flags to the outputdevice}
{                             Input : message
                              Output: message on the outputdevice}
{****************************************************************************}

Procedure SaveEquipmentMessage( message: commandtype);
{****************************************************************************}
{ save the messages from equipment in a file}
{                             Input : message
                              Output: message to a file}
{****************************************************************************}


Procedure CloseUser;
{****************************************************************************}
{ Only for compatibility. Closes the outputfile 'MIAS.OUT'}
{                             Input : -
                              Output: -}
{****************************************************************************}


Implementation

Uses Key_Cons;

Procedure InitUser( MIASlogname: string);

Begin
     OpenIn_OutputDev( MIASlogname);
End;

Procedure SendUserMessage( Message: commandtype);

Begin
     SendMessage( message);
End;


Procedure GetUserMessage( Var Message: commandtype);

Begin
     GetMessage( message);
End;

Procedure SenduserFlags( Message: commandtype);

Begin
     SendFlags( message);
End;

Procedure SaveEquipmentMessage( message: commandtype);

Begin
     SaveMessage( message);
End;

Procedure CloseUser;

Begin
     CloseIn_outputDev;
End;

End.
```

```
Unit Poscalc;

{$N+,E+}

Interface

Uses MIASGLOB;

Var
  MLSantposition      :      positiontype;

Procedure CalcHybridPos( Var alldata: alldatatype; allowed_error: Double;
                         Var position: positiontype);
{*************************************************************************}
{ This procedure calculates the hybrid position of an aircraft using
  GPS and MLS information.
                              Input : alldata
                              Output: position}
{*************************************************************************}


procedure Convert_Pos_to_ECEF( Var position: positiontype );
(************************************************************************)
(* Procedure for conversion of the receiver's position from            *)
(* <lat,long,alt> coordinates to ECEF-coordinates X,Y and Z.           *)
(*     (dimension <rad,rad,m> -------> dimension <m,m,m>               *)
(*                                                                      *)
(* This procedure needs:                                               *)
(*  - position  (WGS-84 coordinates)                                   *)
(*        (latitude, logitude, altitude)      [ rad,rad, m]            *)
(*                                                                      *)
(* This procedure supplies:                                            *)
(*  - position  (ECEF - coordinates)                                   *)
(*        ( X, Y, Z)                          [ m, m, m]               *)
(*                                                                      *)
(*                            G.L. van Eendenburg, December 1988       *)
(************************************************************************)


procedure Convert_Pos_to_WGS( Var position: positiontype);
(************************************************************************)
(* Procedure for conversion of the receiver's ECEF-coordinates         *)
(*   (X,Y,Z)     to     <lat,long,alt>  -    coordinates.              *)
(*    dimension     <m,m,m>  ----->   dimension   <rad,rad,m>          *)
(* This is an iterative procedure                                      *)
(*                                                                      *)
(* This procedure needs:                                               *)
(*  - position  (ECEF - coordinates)                                   *)
(*        ( X, Y, Z)                          [ m, m, m]               *)
(*                                                                      *)
(* This procedure supplies:                                            *)
(*  - position  (WGS-84 coordinates)                                   *)
(*        (latitude, logitude, altitude)      [ rad,rad, m]            *)
(*  - procedure status                                                 *)
(*                                                                      *)
(*                            G.L. van Eendenburg, December 1988  *)
{adjusted to specific MIAS needs R.C. Meijer, August 1992}
(*************************************************************************)


Implementation

USES MATRIX, MATHX, crt, user;

Const
  flattening =  3.3528106E-3;          {flattening of the earth       }
  earthAaxis =  6.378137E+6;           {long earth axis               }
  earthEccentricitySqr = flattening * (2 - flattening);  { e^2 !! }
  pi          =       3.1415926535897932385;
  tweepi      =       2 * pi;
  c           =       2.99792458E8;              {[m/s] speed of light}

Procedure Convert_Pos_to_ECEF( Var position: positiontype );

Var
  EarthRadiusN                                                    : Double;

Begin
  EarthRadiusN := EarthAaxis / sqrt( 1 -  EarthEccentricitySqr  *
                                     sqr(sin(position.WGS84lat)) );
  position.X := (EarthRadiusN + position.WGS84alt) *
                        cos(position.WGS84lat) * cos(position.WGS84lon);
  position.Y := (EarthRadiusN + position.WGS84alt) *
                        cos(position.WGS84lat) * sin(position.WGS84lon);
  position.Z := (EarthRadiusN * ( 1-EarthEccentricitySqr ) +
                          position.WGS84alt ) * sin(position.WGS84lat);
End; (* of the procedure Convert_Pos_To_ECEF *)


Procedure Convert_Pos_to_WGS( Var position: positiontype);

Const
  smallValue = 1E-10; { small value to check if user is at a pole, }
                      { requested accuracy of NewtonRaphson etc.   }
  verySmall  = 1E-15; { small value to prevent denominator = 0     }
  maxIterAllowed = 25;    { maximum number of iterations allowed   }

Var
  numberOfRuns                                                    : byte;
  C,func1,Delta_lat                                               : Double;

Begin
  With position Do
  Begin
                                         { secures against crash on Poles }
     If (Abs( X) < smallValue) And
        (Abs( Y) < smallValue)
     Then Begin                         { user is at or close to North/South Pole }
             If ( Z > 0) Then
                WGS84lat := pi/2                              { North Pole }
```

```
          Else
             WGS84lat := -pi/2;                       { South Pole }


                                             { EXPERIMENTAL }
                      { the value of the longitude is calculated  }
                      { as zero, but with help of the course      }
                      { this value must be defined yet !!!!!!!!!! }
          WGS84lon := 0;
          WGS84alt := Abs( Z) - (EarthAaxis *
                                sqrt(1 - earthEccentricitySqr) );

       End;

                                   { calculating the longitude }
                                   { the ellipsoid is a perfect cirkel}
                                   { in the equator plane.}
    If ( Abs( X) < verySmall) then
    Begin                 { protects arctan function against zero X }
       If X >= 0 then X :=  verySmall;
       if X < 0 then X := -verySmall;
    End;
    WGS84lon := arctan( Y / X);
                     { correction of the arctan function: changes }
                     { range from -pi/2 -> pi/2  to  -pi -> pi.   }
    If ( ( X < 0) And ( Y > 0) )
    Then WGS84lon := WGS84lon + pi;

    If ( ( X < 0) And ( Y < 0) )
    Then WGS84lon := WGS84lon - pi;

               { calculation of the latitude with Newton-Raphson }
    C               := Sqrt( Sqr( X) + Sqr( Y) );
    WGS84lat        := arctan( Z / C);
    numberOfRuns    := 1;
    Delta_lat       := 10;                        { initial value }

    While ( Abs( Delta_lat) > smallValue) And
          ( numberOfRuns < maxIterAllowed) Do
    Begin
         If WGS84lat = 0.5*pi
         Then WGS84lat := 0.5*pi - smallValue;

         func1 := 1 / Sqrt( 1 - earthEccentricitySqr *
                        Sqr( sin( WGS84lat) ) );

         Delta_lat := ( Z - C * sin( WGS84lat) /
                         cos( WGS84lat) + earthEccentricitySqr *
                         EarthAaxis * func1 * sin( WGS84lat)) /
                         ( C / Sqr( cos( WGS84lat)) - EarthAaxis *
                         cos( WGS84lat) * earthEccentricitySqr *
                         func1 * (1 + earthEccentricitySqr *
                         Sqr( func1 * sin( WGS84lat)) ));

         WGS84lat := WGS84lat + Delta_lat;
         numberOfRuns     := numberOfRuns + 1;
    End;

    If numberOfRuns >= maxIterAllowed
    Then flag := True;                  { failure in Newton Raphson }

                                  { calculation of the altitude }

                                  { see "GPS"}
                                  { x = [ Na + ha] cos(lat)cos(lon)}
                                  { x / [ cos(lat)cos(lon)] = Na + ha}
                                  { x / [ cos(lat)cos(lon)] - Na = ha}
                                  { with Na = ae / sqrt[ 1 -}
                                  { sqr(e)sqr( sin(lat))]}

    WGS84alt := X /
                (cos( WGS84lat) * cos( WGS84lon))
                -
                earthAaxis /
                Sqrt( 1 - earthEccentricitySqr * Sqr( sin( WGS84lat)));

  End; { end with position}
End;  { of procedure Convert_Pos_To_WGS }


CONST

      ae = 6.378137E+0006; {metres}  {The equatorial radius}
      be = 6.356752E+0006; {metres}  {The polar distance    }

      {---------------------------------------------------------------------}
      {MaxNoOfGPSequations is the maximum number of linearised GPS-equations }
      {which is the same as the maximum number of sattelites used.          }

      MaxNoOfGPSequations = 7;

      {---------------------------------------------------------------------}
      {MaxNoOfMLSequations is the maximum number of linearised MLS-equations.}
      {The three equations are; the azimuthequation, the elevationequation   }
      {and the DME-equation.                                                 }

      MaxNoOfMLSequations = 3;
      maxiter= 25;

TYPE CHARROW  = PACKED ARRAY[1..10] OF CHAR;

     GPSROW   = ARRAY[1..MaxNoOfGPSequations] OF INTEGER;
     MLSROW   = ARRAY[1..MaxNoOfMLSequations] OF INTEGER;

     RANGE    = ARRAY[1..MaxNoOfGPSequations] OF DOUBLE;

     MLS      = ARRAY[1..MaxNoOfMLSequations] OF DOUBLE;

     MLSBLOCK = ARRAY[1..MaxNoOfMLSequations,1..3] OF DOUBLE;
```

```
    WGS84    = RECORD
                 longitude,latitude,height:DOUBLE;
                 hemilong,hemilat        :CHARROW;
               END;

    {-----------------------------------------------------------------}
    {In ECEF and LOCAL; ARRAY[1] = x-coordinate,                       }
    {                   ARRAY[2] = y-coordinate,                       }
    {                   ARRAY[3] = z-coordinate.                       }
    {                   ARRAY[4] = clockerror (dR = c*dT)              }

    ECEF     = ARRAY[1..4] OF DOUBLE;

    LOCAL    = ARRAY[1..4] OF DOUBLE;

    {-----------------------------------------------------------------}

    SATECEF   = ARRAY[1..MaxNoOfGPSequations] OF ECEF;
    SATLOCAL  = ARRAY[1..MaxNoOfGPSequations] OF LOCAL;

VAR {-----------------------------------------------------------------}
    {NumberOfGPSequations describes the total amount of GPS-equations and   }
    {NumberOfMLSequations describes the total amount of MLS-equations used in}
    {this program. TotalOfEquations is NumberOfGPSequations +               }
    {NumberOfMLSequations.                                                  }

    NumberOfGPSequations,
    NumberOfMLSequations,
    TotalOfEquations           :INTEGER;

    {-----------------------------------------------------------------}
    {First eccentricity of the earth                                   }

    e2,

    {-----------------------------------------------------------------}
    {The Nort_angle is the angle between the true north of the earth and the }
    {y-axis of the MLS-Coordinate system (Local Reference).                  }

    North_angle,

    {-----------------------------------------------------------------}
    {The rollangle, pitchangle, and the headingangle of the airplane.   }

    Rollangle,
    Pitchangle,
    Headingangle               :DOUBLE;

    {-----------------------------------------------------------------}
    {The ECEF-Coordinates of the -The estimated ECEF-position of the airborne}
    {                             GPS-antenna,                               }
    {                            -MLS-datumpoint (Datumpoint_ecef),          }
    {                            -The real (RealGPS_ecef) ECEF-position of   }
    {                             the airborne GPS-antenna,                  }
    {                            -The real (RealMIAS_ecef) ECEF-position of  }
    {                             the airborne MLS-antenna,                  }
    {                            -position of a certain defined point in the }
    {                             airplane (Fix_ecef),                       }
    {                            -variance in the estimated position of the  }
    {                             airborne GPS-antenna (Variance_ecef).      }

    Est_ecef,
    Datumpoint_ecef,
    Fix_ecef,
    Variance_ecef              :ECEF;

    {-----------------------------------------------------------------}
    {The WGS84-Coordinates of the -MLS-datumpoint (Datumpoint_wgs84),       }
    {                             -The real (RealGPS_wgs84) WGS84-position of}
    {                              the airborne GPS-antenna,                 }
    {                             -The real (RealMIAS_wgs84) WGS84-position  }
    {                              of the airborne MLS-antenna.              }

    EstWgs84,
    Datumpoint_wgs84           :wgs84;

    {-----------------------------------------------------------------}
    {The MLS-Coordinates of - the estimated position of the airborne MLS-   }
    {                         antenna,                                      }
    {                       - the azimuthantennaposition (ground),          }
    {                       - the elevationantennaposition (ground),        }
    {                       - the DME-antennaposition (ground),             }
    {                       - the real (Real_local) position of the         }
    {                         airborne MLS-antenna,                         }
    {                       - the variance in the estimated position of the }
    {                         airborne MLS-antenna in local coordinates,    }
    {                       - the final fix of a certain defined point within}
    {                         the aircraft.                                 }

    Est_local,
    Azi_local,
    Ele_local,
    DME_local,
    Variance_local,
    Fix_local,

    {-----------------------------------------------------------------}
    {The local coordinates of the Antenna_Zerovector (which is the vector   }
    {between the airborne MLS-antenna as origin and the airborne GPS-antenna }
    {when roll, pitch and heading are zero) and the GPSPosition_Zerovector  }
    {(which is the vector between the airborne GPS-antenna as origin and a  }
    {certain defined point within the airplane (for example the wheels)) and }
    {the MIASPosition_Zerovector (which is the vector between the airborne  }
    {MLS-antenna as origin and a certain defined point within the airplane). }

    Antenna_Zerovector,
    GPSPosition_Zerovector,
    MIASPosition_Zerovector    :LOCAL;

    {-----------------------------------------------------------------}
```

```
{GPS_eq describes which GPS-equations are used in the algorithm and      }
{MLS_eq describes which MLS-equations are used in the algorithm in order }
{to calculate the position with a subset of the available equations      }
{needed for integrity.                                                   }

GPS_eq                      :GPSROW;
MLS_eq                      :MLSROW;

{-----------------------------------------------------------------------}
{The measured pseudoranges (R_meas), the estimated ranges (R_est) and the}
{ranges between the airborne MLS-antenna and the satellites (R_adjust).  }

R_meas,
R_est,
R_adjust,

{-----------------------------------------------------------------------}
{The variances of the GPS-pseudoranges and the DGPS-pseudoranges.       }

Var_Rgps,
Var_Rdgps                   :RANGE;

{-----------------------------------------------------------------------}
{The measured MLS-parametres (azimuthangle, elevationangle, DME-distance)}

MLS_meas,

{-----------------------------------------------------------------------}
{The variances of the MLS-parametres.                                    }

Var_MLS                     :MLS;

{-----------------------------------------------------------------------}
{The calculated MLS-data needed for the MIASmatrix which are;            }
{   -The estimated azimuthangle,      (MLS_data[x,1])                    }
{   -The estimated elevationangle,    (MLS_data[x,1])                    }
{   -The estimated DME_distance,      (MLS_data[x,1])                    }
{   -The angle azimuthangle accent,   (MLS_data[x,2])                    }
{   -The angle elevationangle accent,(MLS_data[x,2])                     }
{   -The angle DMEangle 1,            (MLS_data[x,2])                    }
{   -The azimuthdistance,             (MLS_data[x,3])                    }
{   -The elevationdistance,           (MLS_data[x,3])                    }
{   -The angle DMEangle 2,            (MLS_data[x,3])                    }

MLS_data                    :MLSBLOCK;
mls1,
mls2,
mls3,
gps1,
gps2,
gps3,
gps4,
gps5,
gps6,
gps7                        :Byte;


{-----------------------------------------------------------------------}
{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,
```

```
                                 dgps      :       Byte;
                              End;

   Convergence_Error          :DOUBLE;
   iter: integer;


PROCEDURE Show_Matrix(TotalOfRows,
                      TotalOfCols:INTEGER;
                      Matrix     :TNMATRIX);
VAR i,j:INTEGER;

{***************************************************************************}
{-Input ; TotalOfRows (= Total of rows in Matrix)                           }
{         TotalOfCols (= Total of columns in Matrix)                        }
{         Matrix                                                            }
{-Output; -                                                                 }
{                                                                           }
{This procedure shows the values of the coefficients of Matrix on screen.   }
{***************************************************************************}

BEGIN
  FOR i:=1 TO TotalOfRows DO
  BEGIN
    FOR j:=1 TO TotalOfCols DO
      write(matrix[i,j]:16,' ');
      writeln;
    END;
    writeln;
END;{PROCEDURE Show_Matrix}



PROCEDURE Clean_Matrix(TotalOfRows,
                       TotalOfCols:INTEGER;
                   VAR Matrix     :TNMATRIX);
VAR i,j:INTEGER;

{***************************************************************************}
{-Input ; TotalOfRows (= Total of rows in Matrix)                           }
{         TotalOfCols (= Total of columns in Matrix)                        }
{-Output; Matrix                                                            }
{                                                                           }
{This procedure sets all coefficients in Matrix to zero.                    }
{***************************************************************************}
BEGIN
  FOR i:=1 TO TotalOfRows DO
  BEGIN
    FOR j:=1 TO TotalOfCols DO Matrix[i,j]:=0;
  END;
END;{PROCEDURE Clean_Matrix}



FUNCTION RowColumnMult(LineLengthM1,
                       RowM1,
                       ColM2        :INTEGER;
                       Matrix1,
                       Matrix2      :TNMATRIX):DOUBLE;
VAR i  :INTEGER;
    Sum:DOUBLE;

{***************************************************************************}
{-Input ; LineLengthM1 (= The length of a line in Matrix1 (Total of columns }
{                         in Matrix1))                                      }
{         RowM1 (= The rownumber of Matrix1)                                }
{         ColM2 (= The columnnumber of Matrix2)                             }
{         Matrix1                                                           }
{         Matrix2                                                           }
{-Output; RowColumnMult                                                     }
{                                                                           }
{This function calculates the product of a row in Matrix1 (RowM1) and a     }
{column in Matrix2 (ColM2).                                                 }
{***************************************************************************}

BEGIN
  Sum:=0;
  FOR i:=1 TO LineLengthM1 DO
    Sum:=Sum+Matrix1[RowM1,i]*Matrix2[i,ColM2];
  RowColumnMult:=Sum;
END;{FUNCTION RowColumnMult}



PROCEDURE Matrix_Mult(TotalOfRowsM1,
                      TotalOfColsM1,
                      TotalOfColsM2 :INTEGER;
                      Matrix1,
                      Matrix2       :TNMATRIX;
                  VAR SolMatrix     :TNMATRIX);
VAR i,j:INTEGER;

{***************************************************************************}
{-Input ; TotalOfRowsM1 (= Total of rows in Matrix1)                        }
{         TotalOfColsM1 (= Total of columns in Matrix1)                     }
{         TotalOfColsM2 (= Total of columns in Matrix2)                     }
{         Matrix1                                                           }
{         Matrix2                                                           }
{-Output; SolMatrix                                                         }
{                                                                           }
{This procedure calculates SolMatrix (SolutionMatrix) = Matrix1*Matrix2.    }
{***************************************************************************}

BEGIN
  FOR i:=1 TO TotalOfRowsM1 DO
  BEGIN
    FOR j:=1 TO TotalOfColsM2 DO
      SolMatrix[i,j]:=RowColumnMult(TotalOfColsM1,i,j,Matrix1,Matrix2);
  END;
END;{PROCEDURE Matrix_Mult}
```

```
PROCEDURE Transpose(TotalOfRows,
                    TotalOfCols:INTEGER;
                    Matrix     :TNMATRIX;
                VAR SolMatrix  :TNMATRIX);
VAR i,j:INTEGER;

{****************************************************************************}
{-Input ; TotalOfRows (= Total of rows in Matrix)                            }
{         TotalOfCols (= Total of columns in Matrix                          }
{         Matrix                                                             }
{-Output; SolMatrix                                                          }
{                                                                            }
{This procedure calculates SolMatrix (SolutionMatrix) = The transpose of     }
{Matrix.                                                                     }
{****************************************************************************}

BEGIN
  FOR i:=1 TO TotalOfRows DO
  BEGIN
    FOR j:=1 TO TotalOfCols DO SolMatrix[j,i]:=Matrix[i,j];
  END;
END;{PROCEDURE Transpose}



PROCEDURE WGS84_to_ECEF(POSwgs84:WGS84;
                    VAR POSecef :ECEF);
VAR Na:DOUBLE;

{****************************************************************************}
{-Input ; POSwgs84 (the position in WGS84)                                   }
{-Output; POSecef  (the position in ECEF)                                    }
{                                                                            }
{This procedure calculates a position given in WGS84-coordinates to a        }
{position given in ECEF-coordinates.                                         }
{****************************************************************************}

BEGIN
  WITH POSwgs84 DO
  BEGIN
    {------------------------------------------------------------------------}
    {Conversion from degrees into radials depending upon hemisphere.         }

    IF hemilong = 'east      ' THEN longitude:= longitude*pi/180
    ELSE {hemilong = 'west'}        longitude:=-longitude*pi/180;
    IF hemilat  ='north     '  THEN latitude:= latitude*pi/180
    ELSE {hemilat  = 'south'}       latitude:=-latitude*pi/180;

    {------------------------------------------------------------------------}
    {Take into account the flexure of the earth.                             }

    Na:=ae/sqrt(1-e2*sqr(sin(latitude)));

    {------------------------------------------------------------------------}
    {Calculate the ECEF-coordinates.                                         }

    POSecef[1]:=(Na+height)*cos(latitude)*cos(longitude);
    POSecef[2]:=(Na+height)*cos(latitude)*sin(longitude);
    POSecef[3]:=(Na*(1-e2)+height)*sin(latitude);
  END;
END;{PROCEDURE WGS84_to_ECEF}



PROCEDURE ECEF_to_WGS84(POSecef :ECEF;
                    VAR POSwgs84:WGS84);

VAR Na,difference,oldlatitude:DOUBLE;

{****************************************************************************}
{-Input ; POSecef  (the position in ECEF)                                    }
{-Output; POSwgs84 (the position in WGS84)                                   }
{                                                                            }
{This procedure calculates a position given in ECEF-coordinates to a position}
{given in WGS84-coordinates.                                                 }
{****************************************************************************}

BEGIN

  WITH POSwgs84 DO
  BEGIN
    {------------------------------------------------------------------------}
    {Calculate the longitude.                                                }

    IF POSecef[1] = 0 THEN longitude:=pi/2;
    IF POSecef[2] = 0 THEN
    BEGIN
      hemilong:='gwmeridian';
      IF POSecef[1] >= 0 THEN longitude:=0
      ELSE                    longitude:=pi;
    END
    ELSE
    BEGIN
      IF POSecef[2] > 0 THEN
      BEGIN
        hemilong:='east      ';
        IF POSecef[1] > 0 then longitude:= arctan(POSecef[2]/POSecef[1])
        ELSE                   longitude:= pi+arctan(POSecef[2]/POSecef[1]);
      END
      ELSE
      BEGIN
        hemilong:='west      ';
        IF POSecef[1] > 0 then longitude:=-arctan(POSecef[2]/POSecef[1])
        ELSE                   longitude:= pi-arctan(POSecef[2]/POSecef[1]);
      END;
    END;

```

```
    {-------------------------------------------------------------------------}
    {Calculate the latitude.                                                  }

    IF POSecef[3] = 0 THEN hemilat:='equator  '
    ELSE IF POSecef[3] > 0 THEN hemilat:='north    '
         ELSE                   hemilat:='south    ';

    IF (POSecef[1] = 0) AND (POSecef[2] = 0) THEN latitude:=pi/2
    ELSE
    BEGIN
      latitude:=arctan(POSecef[3]/sqrt(sqr(POSecef[1])+sqr(POSecef[2])));
      difference:=1;
      WHILE difference > 1E-0010*pi/180 DO {accuracy of 1E-0010 degrees}
      BEGIN
        oldlatitude:=latitude;
        latitude   :=arctan(POSecef[3]/(sqrt(sqr(POSecef[1])+sqr(POSecef[2]))-
                                  e2*cos(latitude)*
                                  ae/sqrt(1-e2*sqr(sin(latitude)))));
        difference :=abs(oldlatitude-latitude);
      END;
      latitude:=abs(latitude);
    END;

    {-------------------------------------------------------------------------}
    {Calculate the height.                                                    }

    Na    :=ae/sqrt(1-e2*sqr(sin(latitude)));
    height:=(sqrt(sqr(POSecef[1])+sqr(POSecef[2]))/cos(latitude))-Na;

    {-------------------------------------------------------------------------}
    {Conversion from radials into degrees.                                    }

    longitude:=longitude*180/pi;
    latitude :=latitude*180/pi;
  END;
END;{PROCEDURE ECEF_to_WGS84}



PROCEDURE Local_to_ECEFOrientation(ORIENTlocal:LOCAL;
                                   POSwgs84   :WGS84;
                                   Northangle :DOUBLE;
                               VAR ORIENTecef :ECEF);

VAR matrix1,matrix2,matrix3,matrix4,matrx1,matrx2,solmatrix,
    vector,solvector                                          :TNMATRIX;

{****************************************************************************}
{-Input ; ORIENTlocal (a vector described in local coordinates)             }
{         POSwgs84 (the position of the origin of the local system in WGS84- }
{                  coordinates)                                              }
{         Northangle (the angle between the true north and the y-axis of the }
{                    local system)                                           }
{-Output; ORIENTecef (a vector described in a system with the same origin as }
{                    the local system but with the axes orientated as the    }
{                    ECEF-axes)                                              }
{                                                                            }
{This procedure converts a vector described in local coordinates to a vector }
{described in a system with the same origin as the local system but with the }
{x-, y-, and z-axes orientated as the ECEF-system.                           }
{****************************************************************************}

BEGIN
  WITH POSwgs84 DO
  BEGIN
    {-------------------------------------------------------------------------}
    {Conversion from degrees into radials of the point of the origin.         }

    longitude  :=longitude *pi/180;
    latitude   :=latitude  *pi/180;
    Northangle :=Northangle*pi/180;

    {-------------------------------------------------------------------------}
    {Calculate the LONGITUDEmatrix.                                           }

    matrix1[1,1]:= cos(longitude);
    matrix1[1,2]:=-sin(longitude);
    matrix1[1,3]:= 0;
    matrix1[2,1]:= sin(longitude);
    matrix1[2,2]:= cos(longitude);
    matrix1[2,3]:= 0;
    matrix1[3,1]:= 0;
    matrix1[3,2]:= 0;
    matrix1[3,3]:= 1;

    {-------------------------------------------------------------------------}
    {Calculate the LATITUDEmatrix.                                            }

    matrix2[1,1]:= cos(latitude);
    matrix2[1,2]:= 0;
    matrix2[1,3]:=-sin(latitude);
    matrix2[2,1]:= 0;
    matrix2[2,2]:= 1;
    matrix2[2,3]:= 0;
    matrix2[3,1]:= sin(latitude);
    matrix2[3,2]:= 0;
    matrix2[3,3]:= cos(latitude);

    {-------------------------------------------------------------------------}
    {Calculate the NORTHANGLEmatrix.                                          }

    matrix3[1,1]:= 1;
    matrix3[1,2]:= 0;
    matrix3[1,3]:= 0;
    matrix3[2,1]:= 0;
    matrix3[2,2]:= cos(Northangle);
    matrix3[2,3]:=-sin(Northangle);
    matrix3[3,1]:= 0;
    matrix3[3,2]:= sin(Northangle);
    matrix3[3,3]:= cos(Northangle);
```

```

    {---------------------------------------------------------------------}
    {Rename the axes (RENAMEmatrix).                                      }

    matrix4[1,1]:=0; matrix4[1,2]:=0; matrix4[1,3]:=1;
    matrix4[2,1]:=1; matrix4[2,2]:=0; matrix4[2,3]:=0;
    matrix4[3,1]:=0; matrix4[3,2]:=1; matrix4[3,3]:=0;

    {---------------------------------------------------------------------}
    {Write the inputparametre ORIENTlocal to vector in order to use the   }
    {procedure Matrix_Mult.                                               }

    vector[1,1]:=ORIENTlocal[1];
    vector[2,1]:=ORIENTlocal[2];
    vector[3,1]:=ORIENTlocal[3];

    {---------------------------------------------------------------------}
    {LONGITUDEmatrix*LATITUDEmatrix*NORTHANGLEmatrix*RENAMEmatrix = solmatrix}

    Matrix_Mult(3,3,3,matrix3,matrix4,matrx1);
    Matrix_Mult(3,3,3,matrix2,matrx1,matrx2);
    Matrix_Mult(3,3,3,matrix1,matrx2,solmatrix);

    {---------------------------------------------------------------------}
    {solmatrix*vector = solvector.                                        }

    Matrix_Mult(3,3,1,solmatrix,vector,solvector);

    {---------------------------------------------------------------------}
    {Convert the solvector to ORIENTecef.                                 }

    ORIENTecef[1]:=solvector[1,1];
    ORIENTecef[2]:=solvector[2,1];
    ORIENTecef[3]:=solvector[3,1];
  END;
END;{PROCEDURE Local_to_ECEFOrientation}



PROCEDURE ECEF_to_LocalOrientation(ORIENTecef :ECEF;
                                   POSwgs84   :WGS84;
                                   Northangle :DOUBLE;
                               VAR ORIENTlocal:LOCAL);

VAR matrix1,matrix2,matrix3,matrix4,matrx1,matrx2,solmatrix,
    vector,solvector                                        :TNMATRIX;

{***************************************************************************}
{-Input ; ORIENTecef (a vector described in a system with the same origin as }
{                    the local system but with the axes orientated as in    }
{                    ECEF)                                                  }
{         POSwgs84 (the position of the origin of the local system in WGS84) }
{         Northangle (the angle between the true north and the y-axis of the }
{                    local system)                                          }
{-Output; ORIENTlocal (a vector described in local coordinates)             }
{                                                                           }
{This procedure converts a vector decribed by a system with a origin the same}
{as the local system but with axes orientated like the ECEF-system to a     }
{vector described by that local system.                                     }
{***************************************************************************}

BEGIN
  WITH POSwgs84 DO
  BEGIN
    {---------------------------------------------------------------------}
    {conversion from degrees into radials of the point of the origin.     }

    longitude  :=longitude *pi/180;
    latitude   :=latitude  *pi/180;
    Northangle :=Northangle*pi/180;

    {---------------------------------------------------------------------}
    {Rename the axes (RENAMEmatrix).                                      }

    matrix1[1,1]:= 0; matrix1[1,2]:= 1; matrix1[1,3]:= 0;
    matrix1[2,1]:= 0; matrix1[2,2]:= 0; matrix1[2,3]:= 1;
    matrix1[3,1]:= 1; matrix1[3,2]:= 0; matrix1[3,3]:= 0;

    {---------------------------------------------------------------------}
    {Calculate the NORTHANGLEmatrix.                                      }

    matrix2[1,1]:= 1;
    matrix2[1,2]:= 0;
    matrix2[1,3]:= 0;
    matrix2[2,1]:= 0;
    matrix2[2,2]:= cos(Northangle);
    matrix2[2,3]:= sin(Northangle);
    matrix2[3,1]:= 0;
    matrix2[3,2]:=-sin(Northangle);
    matrix2[3,3]:= cos(Northangle);

    {---------------------------------------------------------------------}
    {Calculate the LATITUDEmatrix.                                        }

    matrix3[1,1]:= cos(latitude);
    matrix3[1,2]:= 0;
    matrix3[1,3]:= sin(latitude);
    matrix3[2,1]:= 0;
    matrix3[2,2]:= 1;
    matrix3[2,3]:= 0;
    matrix3[3,1]:=-sin(latitude);
    matrix3[3,2]:= 0;
    matrix3[3,3]:= cos(latitude);

    {---------------------------------------------------------------------}
    {Calculate the LONGITUDEmatrix.                                       }

    matrix4[1,1]:= cos(longitude);
    matrix4[1,2]:= sin(longitude);
    matrix4[1,3]:= 0;
```

```
    matrix4[2,1]:=-sin(longitude);
    matrix4[2,2]:= cos(longitude);
    matrix4[2,3]:= 0;
    matrix4[3,1]:= 0;
    matrix4[3,2]:= 0;
    matrix4[3,3]:= 1;

    {--------------------------------------------------------------------}
    {Write the inputparametre ORIENTecef to vector in order to use the   }
    {procedure Matrix_Mult.                                               }

    vector[1,1]:=ORIENTecef[1];
    vector[2,1]:=ORIENTecef[2];
    vector[3,1]:=ORIENTecef[3];

    {--------------------------------------------------------------------}
    {RENAMEmatrix*NORTHANGLEmatrix*LATITUDEmatrix*LONGITUDEmatrix = solmatrix}

    Matrix_Mult(3,3,3,matrix3,matrix4,matrx1);
    Matrix_Mult(3,3,3,matrix2,matrx1,matrx2);
    Matrix_Mult(3,3,3,matrix1,matrx2,solmatrix);

    {--------------------------------------------------------------------}
    {solmatrix*vector = solvector.                                        }

    Matrix_Mult(3,3,1,solmatrix,vector,solvector);

    {--------------------------------------------------------------------}
    {Convert the solvector to ORIENTlocal.                                }

    ORIENTlocal[1]:=solvector[1,1];
    ORIENTlocal[2]:=solvector[2,1];
    ORIENTlocal[3]:=solvector[3,1];
  END;
END;{PROCEDURE ECEF_to_LocalOrientation}



PROCEDURE LocalAircraft_to_LocalMLSOrientation(AircraftLocal:LOCAL;
                                               Northangle   :DOUBLE;
                                           VAR MLSLocal     :LOCAL);
VAR matrix,vector,solvector:TNMATRIX;
{****************************************************************************}
{-Input ; AircraftLocal (The coordinates of a vector defined within the     }
{                     aircraft)                                              }
{         Northangle (the angle between the true north and the y-axis of the }
{                    MLS Local Reference system)                             }
{         MLSLocal (The coordinates of a vector defined within the aircraft  }
{                   and orientated as the MLS Local Reference system.        }
{                                                                            }
{This procedure converts a vector defined in a coordinatesystem in the      }
{aircraft to a coordinatesystem orientated as the MLS Local Reference system.}
{****************************************************************************}

BEGIN
  {--------------------------------------------------------------------------}
  {Calculate the NORTHANGLEmatrix.                                            }

  matrix[1,1]:= 1;
  matrix[1,2]:= 0;
  matrix[1,3]:= 0;
  matrix[2,1]:= 0;
  matrix[2,2]:= cos(Northangle);
  matrix[2,3]:= sin(Northangle);
  matrix[3,1]:= 0;
  matrix[3,2]:=-sin(Northangle);
  matrix[3,3]:= cos(Northangle);

  {--------------------------------------------------------------------------}
  {Write the inputparameter AircraftLocal to vector in order to use the      }
  {procedure Matrix_Mult.                                                     }

  vector[1,1]:=AircraftLocal[1];
  vector[2,1]:=AircraftLocal[2];
  vector[3,1]:=AircraftLocal[3];

  {--------------------------------------------------------------------------}
  {NORTHANGLEmatrix*vector = solvector.                                       }

  Matrix_Mult(3,3,1,matrix,vector,solvector);

  {--------------------------------------------------------------------------}
  {Convert the solvector to MLSLocal.                                         }

  MLSLocal[1]:=solvector[1,1];
  MLSLocal[2]:=solvector[2,1];
  MLSLocal[3]:=solvector[3,1];
END;{PROCEDURE LocalAircraft_to_LocalMLSOrientation}



PROCEDURE Calculate_Uvector(rollangle,pitchangle,headingangle:DOUBLE;
                                      zerovector                  :LOCAL;
                                  VAR Uvector                     :LOCAL);

VAR rollmatrix,pitchmatrix,headingmatrix,helpmatrix,solmatrix,
    vector,solvector                                          :TNMATRIX;

{****************************************************************************}
{-Input ; rollangle                                                          }
{         pitchangle                                                         }
{         headingangle                                                       }
{         zerovector                                                         }
{-Output; VectorLocal                                                        }
{                                                                            }
{This procedure calculates the Uvector. The Uvector is defined in a local   }
{system in which -the x-axis is parallel to the meridians pointing eastwards,}
{                -the y-axis is perpendicular to the x-axis pointing         }
{                 northwards,                                                }
```

```
{               -the x-axis is perpendicular to the xy-plane pointing into }
{                space.                                                    }
{The zerovector is the vector between the airborne MLS-antenna and a defined }
{point within the airplane (for example GPS-antenna or the wheels) when      }
{having a zero roll, pitch and heading.                                      }
{The Uvector decribes the zerovector during change of roll, pitch and        }
{heading.                                                                    }
{****************************************************************************}

BEGIN
  {------------------------------------------------------------------------}
  {Conversion from degrees into radials.                                   }

  rollangle   :=rollangle   *pi/180;
  pitchangle  :=pitchangle  *pi/180;
  headingangle:=headingangle*pi/180;

  {------------------------------------------------------------------------}
  {Create the ROLLmatrix.                                                  }

  rollmatrix[1,1]:= 1;
  rollmatrix[1,2]:= 0;
  rollmatrix[1,3]:= 0;
  rollmatrix[2,1]:= 0;
  rollmatrix[2,2]:= cos(rollangle);
  rollmatrix[2,3]:= sin(rollangle);
  rollmatrix[3,1]:= 0;
  rollmatrix[3,2]:=-sin(rollangle);
  rollmatrix[3,3]:= cos(rollangle);

  {------------------------------------------------------------------------}
  {Create the PITCHmatrix.                                                 }

  pitchmatrix[1,1]:= cos(pitchangle);
  pitchmatrix[1,2]:= 0;
  pitchmatrix[1,3]:= sin(pitchangle);
  pitchmatrix[2,1]:= 0;
  pitchmatrix[2,2]:= 1;
  pitchmatrix[2,3]:= 0;
  pitchmatrix[3,1]:=-sin(pitchangle);
  pitchmatrix[3,2]:= 0;
  pitchmatrix[3,3]:= cos(pitchangle);

  {------------------------------------------------------------------------}
  {Create the HEADINGmatrix.                                               }

  headingmatrix[1,1]:= cos(headingangle);
  headingmatrix[1,2]:= sin(headingangle);
  headingmatrix[1,3]:= 0;
  headingmatrix[2,1]:=-sin(headingangle);
  headingmatrix[2,2]:= cos(headingangle);
  headingmatrix[2,3]:= 0;
  headingmatrix[3,1]:= 0;
  headingmatrix[3,2]:= 0;
  headingmatrix[3,3]:= 1;

  {------------------------------------------------------------------------}
  {Write zerovector to vector in order to use the procedure Matrix_Mult.   }

  vector[1,1]:=zerovector[1];
  vector[2,1]:=zerovector[2];
  vector[3,1]:=zerovector[3];

  {------------------------------------------------------------------------}
  {ROLLmatrix*PITCHmatrix*HEADINGmatrix = solmatrix.                       }

  Matrix_Mult(3,3,3,pitchmatrix,headingmatrix,helpmatrix);
  Matrix_Mult(3,3,3,rollmatrix,helpmatrix,solmatrix);

  {------------------------------------------------------------------------}
  {solmatrix*vector = solvector.                                           }

  Matrix_Mult(3,3,1,solmatrix,vector,solvector);

  {------------------------------------------------------------------------}
  {Convert the solvector to Uvector.                                       }

  Uvector[1]:=solvector[1,1];
  Uvector[2]:=solvector[2,1];
  Uvector[3]:=solvector[3,1];
END;{PROCEDURE Calculate_Uvector}



PROCEDURE Calculate_Localcoordinates_of_Satellites(NoOfGPSequations:INTEGER;
                                                   SatEcef         :SATECEF;
                                               VAR SatLocal        :SATLOCAL);
VAR ecefvector :ECEF;
    localvector:LOCAL;
    i,j        :INTEGER;

{****************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                   }
{         SatEcef (The satellitepositions in ECEF)                           }
{-Output; SatLocal (The satellitepositions in Local Reference)               }
{                                                                            }
{This procedure transforms the satellitecoordinates in ECEF to               }
{satellitescoordinates in Local Reference.                                   }
{****************************************************************************}

BEGIN
  {------------------------------------------------------------------------}
  {Change ECEF-origin to MLS Local Reference origin.                       }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    FOR j:=1 TO 3 DO
       SatEcef[i,j]:=SatEcef[i,j]-Datumpoint_ecef[j];
  END;

```

```
  {-----------------------------------------------------------------------}
  {Calculate the Local coordinates of the satellites.                     }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    FOR j:=1 TO 3 DO ecefvector[j]:=SatEcef[i,j];
    ECEF_to_LocalOrientation(ecefvector,Datumpoint_wgs84,North_angle,
                             localvector);
    FOR j:=1 TO 3 DO SatLocal[i,j]:=localvector[j];
  END;
END;{PROCEDURE Calculate_Localcoordinates_of_Satellites}



PROCEDURE Create_GPSVarianceMatrix(NoOfGPSequations:INTEGER;
                                   GPSeq           :GPSROW;
                                   VarRgps         :RANGE;
                               VAR VMatrix         :TNMATRIX);
VAR i,j:INTEGER;

{*************************************************************************}
{-Input ; NoOfGPSequations (= Number of used GPS-equations)               }
{         GPSeq (array of used GPS-equations)                             }
{         VarRgps (the variances of the GPS-pseudoranges)                 }
{-Output; VMatrix (the variancematrix)                                    }
{                                                                         }
{This procedure creates the GPSVARIANCEmatrix.                            }
{*************************************************************************}

BEGIN
  {-----------------------------------------------------------------------}
  {Make all elements of the GPSVARIANCEmatrix zero.                       }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    FOR j:=1 TO NoOfGPSequations DO Vmatrix[i,j]:=0;
  END;

  {-----------------------------------------------------------------------}
  {Fill in the variance(s) of the GPS-pseudorange(s).                     }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Vmatrix[i,i]:=VarRgps[GPSeq[i]];
  END;

END;{PROCEDURE Create_GPSVarianceMatrix}



PROCEDURE Create_MIASVarianceMatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
                                    GPSeq                           :GPSROW;
                                    MLSeq                           :MLSROW;
                                    VarRdgps                        :RANGE;
                                    VarMLS                          :MLS;
```

```
                                VAR VMatrix                          :TNMATRIX);
VAR i,j:INTEGER;

{*************************************************************************}
{-Input ; NoOfGPSequations (= Number of used GPS-equations)               }
{         NoOfMLSequations (= Number of used MLS-equations)               }
{         GPSeq (array of used GPS-equations)                             }
{         MLSeq (array of used MLS-equations)                             }
{         VarRdgps (the variances of the DGPS-pseudoranges)               }
{         VarMLS (the variance of the azimuthangle, elevationangle and    }
{                 DME-distance)                                           }
{-Output; VMatrix (the variancematrix)                                    }
{                                                                         }
{This procedure creates the VARIANCEmatrix.                               }
{*************************************************************************}

BEGIN
  {-----------------------------------------------------------------------}
  {Make all elements of the VARIANCEmatrix zero.                          }

  FOR i:=1 TO (NoOfGPSequations+NoOfMLSequations) DO
  BEGIN
    FOR j:=1 TO (NoOfGPSequations+NoOfMLSequations) DO Vmatrix[i,j]:=0;
  END;

  {-----------------------------------------------------------------------}
  {Fill in the variance(s) of the DGPS-pseudorange(s).                    }

  FOR i:=1 TO NoOfGPSequations DO Vmatrix[i,i]:=VarRdgps[GPSeq[i]];

  {-----------------------------------------------------------------------}
  {Fill in the variance of the azimuthangle (and/or) the elevationangle   }
  {(and/or) the DME-distance.                                             }

  FOR i:=1 TO NoOfMLSequations DO
  BEGIN
    Vmatrix[NoOfGPSequations+i,NoOfGPSequations+i]:=VarMLS[MLSeq[i]];
  END;

END;{PROCEDURE Create_MIASVarianceMatrix}



PROCEDURE Create_MIASweighMatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
                                 GPSeq                           :GPSROW;
                                 MLSeq                           :MLSROW;
                                 VarRgps                         :RANGE;
                                 VarMLS                          :MLS;
                                 mlsdata                         :MLSblock;
                             VAR WMatrix                         :TNMATRIX);
VAR i,j:INTEGER;

{*************************************************************************}
{-Input ; NoOfGPSequations (= Number of used GPS-equations)               }
{         NoOfMLSequations (= Number of used MLS-equations)               }
```

```
{          GPSeq (array of used GPS-equations)                            }
{          MLSeq (array of used MLS-equations)                            }
{          VarRdgps (the variances of the DGPS-pseudoranges)              }
{          VarMLS (the variance of the azimuthangle, elevationangle and   }
{                  DME-distance)                                          }
{-Output; VMatrix (the variancematrix)                                    }
{                                                                         }
{This procedure creates the VARIANCEmatrix.                               }
{*************************************************************************}

BEGIN
  {-----------------------------------------------------------------------}
  {Make all elements of the VARIANCEmatrix zero.                          }

  FOR i:=1 TO (NoOfGPSequations+NoOfMLSequations) DO
  BEGIN
    FOR j:=1 TO (NoOfGPSequations+NoOfMLSequations) DO Wmatrix[i,j]:=0;
  END;

  {-----------------------------------------------------------------------}
  {Fill in the variance(s) of the DGPS-pseudorange(s).                    }

  FOR i:=1 TO NoOfGPSequations DO Wmatrix[i,i]:=VarRgps[GPSeq[i]];

  {-----------------------------------------------------------------------}
  {Fill in the variance of the azimuthangle (and/or) the elevationangle   }
  {(and/or) the DME-distance.                                             }

  FOR i:=1 TO NoOfMLSequations DO
  BEGIN
    If mlseq[i]=3
    Then Wmatrix[noofgpsequations+i, noofgpsequations+i]:= varmls[ mlseq[i]]
    Else Wmatrix[noofgpsequations+i, noofgpsequations+i]:= sqr( mlsdata[i,3] *
                                                           sin( varmls[ mlseq[i]]));
  END;

END;{PROCEDURE Create_MIASVarianceMatrix}




PROCEDURE New_GPSEstimation(NoOfGPSequations:INTEGER;
                            GPSeq           :GPSROW;
                            EstEcef         :ECEF;
                            SatEcef         :SATECEF;
                        VAR REst            :RANGE);
VAR i:INTEGER;

{*************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                }
{         GPSeq (array of used GPS-equations)                             }
{         EstEcef(the estimate of the airborne GPS-antennaposition in ECEF- }
{                 coordinates)                                            }
{         SatEcef (the satellitepositions in ECEF-coordinates)            }
{-Output; REst (the estimated ranges between the airborne GPS-antenna and }
{                       the satellites)                                   }
{                                                                         }
{This procedure calculates the GPS-parameters needed for the coefficients of }
{the GPS-matrix and the GPSvector.                                        }
{*************************************************************************}

BEGIN
  {-----------------------------------------------------------------------}
  {Calculate the estimated ranges between the airborne GPS-antenna and the  }
  {satellites.                                                            }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    REst[i]:=sqrt(sqr(EstEcef[1]-SatEcef[GPSeq[i],1])+
                  sqr(EstEcef[2]-SatEcef[GPSeq[i],2])+
                  sqr(EstEcef[3]-SatEcef[GPSeq[i],3]));
  END;
END;{PROCEDURE New_GPSEstimation}



PROCEDURE New_MIASEstimation(NoOfGPSequations,NoOfMLSequations  :INTEGER;
                             GPSeq                              :GPSROW;
                             MLSeq                              :MLSROW;
                             EstLocal,AziLocal,EleLocal,DMELocal:LOCAL;
                             SatLocal                           :SATLOCAL;
                         VAR REst                               :RANGE;
                         VAR MLSdata                            :MLSBLOCK);
VAR i:INTEGER;

{*************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                }
{         NoOfMLSequations (=Number of used MLS-equations)                }
{         GPSeq (array of used GPS-equations)                             }
{         MLSeq (array of used MLS-equations)                             }
{         EstLocal (the estimate of the airborne MLS-antennaposition in Local}
{                   coordinates)                                          }
{         AziLocal,EleLocal,DMELocal(the position of the groundantennas in  }
{                                   Local coordinates)                    }
{         SatLocal (the satellitepositions in Local coordinates)          }
{-Output; REst (the estimated ranges between the airborne MLS-antenna and }
{                the satellites)                                          }
{         MLSdata (the MLS-data)                                          }
{                                                                         }
{This procedure calculates the GPS- and MLS-parametres needed for the     }
{coefficients of the MIASmatrix and the MIASvector.                       }
{*************************************************************************}

var
value   :       String;

BEGIN
  {-----------------------------------------------------------------------}
  {Calculate the estimated ranges between the airborne MLS-antenna and the  }
  {satellites.                                                            }
```

```

FOR i:=1 TO NoOfGPSequations DO
BEGIN
  REst[i]:=sqrt(sqr(EstLocal[1]-SatLocal[GPSeq[i],1])+
                sqr(EstLocal[2]-SatLocal[GPSeq[i],2])+
                sqr(EstLocal[3]-SatLocal[GPSeq[i],3]));
END;

{-----------------------------------------------------------------------}
{Calculate the MLS-data.                                                }

FOR i:=1 TO NoOfMLSequations DO
BEGIN
  IF MLSeq[i]=1 THEN
  BEGIN

    {The estimated azimuthangle}

    If ( sqr(Estlocal[1] - Azilocal[1]) +
         sqr(Estlocal[3] - Azilocal[3]) = 0)
    Then If (( estlocal[2] - azilocal[2]) >= 0)
         Then MLSdata[i,1]:= pi / 2
         Else MLSdata[i,1]:= - pi / 2
    Else MLSdata[i,1]:=-arctan(( EstLocal[2]-AziLocal[2])/
                               sqrt( sqr(EstLocal[1]-AziLocal[1])+
                                     sqr(EstLocal[3]-AziLocal[3])));

    {Azimuthangle accent}

    If (( Estlocal[1] - Azilocal[1]) = 0)
    Then If (( Estlocal[3] - azilocal[3]) >= 0)
         Then MLSdata[i,2]:= pi / 2
         Else MLSdata[i,2]:= - pi / 2
    Else MLSdata[i,2]:= arctan(( EstLocal[3]-AziLocal[3])/
                              (EstLocal[1]-AziLocal[1]));

    {Distance between azimuthantenna and estimated position}

    MLSdata[i,3]:= sqrt(sqr(EstLocal[1]-AziLocal[1])+
                        sqr(EstLocal[2]-AziLocal[2])+
                        sqr(EstLocal[3]-AziLocal[3]));
  END;
  IF MLSeq[i]=2 THEN
  BEGIN
    {The estimated elevationangle}

    If ( sqr(Estlocal[1] - Elelocal[1]) +
         sqr(Estlocal[2] - Elelocal[2]) = 0)
    Then If (( estlocal[3] - elelocal[3]) >= 0)
         Then MLSdata[i,1]:= pi / 2
         Else MLSdata[i,1]:= - pi / 2
    Else MLSdata[i,1]:=-arctan(( EstLocal[3]-EleLocal[3])/
                               sqrt( sqr(EstLocal[1]-EleLocal[1])+
                                     sqr(EstLocal[2]-EleLocal[2])));

    {Elevationangle accent}

    If (( Estlocal[1] - Elelocal[1]) = 0)
    Then If (( Estlocal[2] - Elelocal[2]) >= 0)
         Then MLSdata[i,2]:= pi / 2
         Else MLSdata[i,2]:= - pi / 2
    Else MLSdata[i,2]:= arctan(( EstLocal[2]-EleLocal[2])/
                                (EstLocal[1]-EleLocal[1]));

    {Distance between the elevationantenna and the estimated position}

    MLSdata[i,3]:= sqrt(sqr(EstLocal[1]-EleLocal[1])+
                        sqr(EstLocal[2]-EleLocal[2])+
                        sqr(EstLocal[3]-EleLocal[3]));
  END;
  IF MLSeq[i]=3 THEN
  BEGIN
    {The estimated DME-distance}

    MLSdata[i,1]:= sqrt(sqr(EstLocal[1]-DMELocal[1])+
                        sqr(EstLocal[2]-DMELocal[2])+
                        sqr(EstLocal[3]-DMELocal[3]));
    {DME-angle 1}

    If (( sqr( Estlocal[1] - dmelocal[1]) +
          sqr( Estlocal[2] - dmelocal[2])) = 0)
    Then If (( Estlocal[3] - dmelocal[3]) >= 0)
         Then MLSdata[i,2]:= pi / 2
         Else MLSdata[i,2]:= - pi / 2
    Else MLSdata[i,2]:= arctan(( EstLocal[3]-DMELocal[3])/
                               sqrt( sqr(EstLocal[1]-DMELocal[1])+
                                     sqr(EstLocal[2]-DMELocal[2])));
    {DME-angle 2}

    If (( Estlocal[1] - dmelocal[1]) = 0)
    Then If (( Estlocal[2] - dmelocal[2]) >= 0)
         Then MLSdata[i,2]:= pi / 2
         Else MLSdata[i,2]:= - pi / 2
    Else MLSdata[i,3]:= arctan(( EstLocal[2]-DMELocal[2])/
                               ( EstLocal[1]-DMELocal[1]));
  END;
END;
END;{PROCEDURE New_MIASEstimation}



PROCEDURE Adjust_Measured_Pseudoranges_to_MLSantenna
                                          (NoOfGPSequations :INTEGER;
                                           GPSeq            :GPSROW;
                                           AntennaZerovector:LOCAL;
                                           EstLocal         :LOCAL;
                                           SatLocal         :SATLOCAL;
                                           REst             :RANGE;
                                           RMeas            :RANGE;
                                       VAR RAdjust          :RANGE);
```

```
VAR Ulocal1,
    Ulocal2        :LOCAL;
    Rdiff          :RANGE;
    i              :INTEGER;

{***************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                  }
{         NoOfMLSequations (=Number of used MLS-equations)                  }
{         AntennaZerovector (the vector between the airborne MLS-antenna    }
{                           (origin) and the airborne GPS-antenna)          }
{         EstLocal (the estimated position of the airborne MLS-antenna)     }
{         SatLocal (the satellitepositions in Local Reference)              }
{         REst (the distances between the estimated position and the        }
{               satellites)                                                 }
{         RMeas (the measured pseudoranges)                                 }
{-Output; RAdjust (the adjustes pseudoranges)                               }
{                                                                           }
{This procedure adjusts the measured pseudoranges (the distance between the }
{satellites to the airborne GPS-antenna) to ranges from the satellites to the}
{airborne MLS-antenna.                                                      }
{***************************************************************************}

BEGIN
  {-------------------------------------------------------------------------}
  {Calculate the Uvector described in procedure Calculate_Uvector.          }

  Ulocal1[ 1]:=0;
  Ulocal1[ 2]:=0;
  Ulocal1[ 3]:=0;



  If ( present.att = 1) and
     ( present.hdg = 1)
  Then
  Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
                    AntennaZerovector,
                    Ulocal1);

  {-------------------------------------------------------------------------}
  {Convert the local aircraftcoordinatesystem in a system orientated as the }
  {MLS-Local Reference system.                                              }

  LocalAircraft_to_LocalMLSOrientation(Ulocal1,North_angle,Ulocal2);

  {-------------------------------------------------------------------------}
  {Calculate the differences in distance between the estimated position     }
  {of the airborne GPS-antenna and the satellitepositions and between       }
  {the estimated position of the airborne MLS-antenna and the               }
  {satellitepositions                                                       }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Rdiff[i]:=sqrt(sqr(EstLocal[1]+Ulocal2[1]-SatLocal[GPSeq[i],1])+
                       sqr(EstLocal[2]+Ulocal2[2]-SatLocal[GPSeq[i],2])+
                       sqr(EstLocal[3]+Ulocal2[3]-SatLocal[GPSeq[i],3]))-REst[i];
  END;

  {-------------------------------------------------------------------------}
  {Substract these differences from the measured pseudoranges in order      }
  {to get a "measured" range between the airborne MLS-antenna and the       }
  {satellites                                                               }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    RAdjust[GPSeq[i]]:=RMeas[GPSeq[i]]-Rdiff[i];
  END;

END;{PROCEDURE Adjust_Measured_Pseudoranges_to_MLSantenna}



PROCEDURE Calculate_GPSmatrix(NoOfGPSequations:INTEGER;
                              GPSeq           :GPSROW;
                              EstEcef         :ECEF;
                              REst            :RANGE;
                              SatEcef         :SATECEF;
                          VAR Matrix          :TNMATRIX);
VAR i:INTEGER;

{***************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                  }
{         GPSeq (array of used GPS-equations)                               }
{         EstEcef (the estimated position of the airborne GPS-antenna in    }
{                  ECEF-coordinates)                                        }
{         REst (the distance between the estimated position and the         }
{               satellites)                                                 }
{         SatEcef (the satellitepositions in ECEF-coordinates)              }
{-Output; Matrix (GPSmatrix)                                                }
{                                                                           }
{This procedure creates the GPS-matrix using the parametres calculated by   }
{the procedure New_GPSEstimation.                                           }
{***************************************************************************}

BEGIN
  {-------------------------------------------------------------------------}
  {The GPS-equations.                                                       }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Matrix[i,1]:=(EstEcef[1]-SatEcef[GPSeq[i],1])/REst[i];
    Matrix[i,2]:=(EstEcef[2]-SatEcef[GPSeq[i],2])/REst[i];
    Matrix[i,3]:=(EstEcef[3]-SatEcef[GPSeq[i],3])/REst[i];
    Matrix[i,4]:=-1;
  END;
END;{PROCEDURE Calculate_GPSmatrix}
```

```
PROCEDURE Calculate_MIASmatrix(NoOfGPSequations,NoOfMLSequations:INTEGER;
                               GPSeq                            :GPSROW;
                               MLSeq                            :MLSROW;
                               EstLocal                         :LOCAL;
                               REst                             :RANGE;
                               SatLocal                         :SATLOCAL;
                               MLSdata                          :MLSBLOCK;
                           VAR Matrix                           :TNMATRIX);
VAR i:INTEGER;

{******************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                     }
{         NoOfMLSequations (=Number of used MLS-equations)                     }
{         GPSeq (array of used GPS-equations)                                  }
{         MLSeq (array of used MLS-equations)                                  }
{         EstLocal (the estimated position of the airborne MLS-antenna in      }
{                   Local Reference coordinates)                               }
{         REst (the distance between the estimated position and the            }
{               satellites)                                                    }
{         SatLocal (the satellitepositions in Local Reference coordinates)     }
{         MLSdata (estimated MLS-data+extra MLS-data)                          }
{-Output; Matrix (MIASmatrix)                                                  }
{                                                                              }
{This procedure creates the MIASmatrix using the parametres calculated by      }
{the procedure New_Estimation.                                                 }
{******************************************************************************}

BEGIN
  {----------------------------------------------------------------------------}
  {The GPS-equations.                                                          }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Matrix[i,1]:=(EstLocal[1]-SatLocal[GPSeq[i],1])/REst[i];
    Matrix[i,2]:=(EstLocal[2]-SatLocal[GPSeq[i],2])/REst[i];
    Matrix[i,3]:=(EstLocal[3]-SatLocal[GPSeq[i],3])/REst[i];
    Matrix[i,4]:=-1;
  END;

  {----------------------------------------------------------------------------}
  {The MLS-equations.                                                          }

  FOR i:=1 TO NoOfMLSequations DO
  BEGIN
    IF MLSeq[i]=1 THEN   {The azimuthequation}
    BEGIN
      If ( MLSdata[ i,3] = 0)
      Then Begin
                If ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) >= 0)
                Then if ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,1]:=1
                     Else Matrix[NoOfGPSequations+i,1]:=0
                Else if ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,1]:=-1
                     Else Matrix[NoOfGPSequations+i,1]:=0;

                If ( -cos( MLSdata[i,1]) >=0)
                Then If ( -cos( MLSdata[i,1]) =0)
                     Then Matrix[ NoOfGPSequations+i,2]:= 1
                     Else Matrix[ NoOfGPSequations+i,2]:= 0
                Else If ( -cos( MLSdata[i,1]) =0)
                     Then Matrix[ NoOfGPSequations+i,2]:= -1
                     Else Matrix[ NoOfGPSequations+i,2]:= 0;

                If ( -sin( MLSdata[i,1]) * sin( MLSdata[i,2]) >= 0)
                Then If ( -sin( MLSdata[i,1]) * sin( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,3]:= 1
                     Else Matrix[NoOfGPSequations+i,3]:= 0
                Else If ( -sin( MLSdata[i,1]) * sin( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,3]:= -1
                     Else Matrix[NoOfGPSequations+i,3]:= 0;

           End
      Else Begin
                Matrix[NoOfGPSequations+i,1]:= -sin(MLSdata[i,1])*cos(MLSdata[i,2])/
                                               MLSdata[i,3];
                Matrix[NoOfGPSequations+i,2]:= -cos(MLSdata[i,1])/MLSdata[i,3];
                Matrix[NoOfGPSequations+i,3]:= -sin(MLSdata[i,1])*sin(MLSdata[i,2])/
                                               MLSdata[i,3];
           End;

      Matrix[NoOfGPSequations+i,4]:= 0;
    END;
    IF MLSeq[i]=2 THEN   {The elevationequation}
    BEGIN
      If ( MLSdata[ i,3] = 0)
      Then Begin
                If ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) >= 0)
                Then if ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,1]:=1
                     Else Matrix[NoOfGPSequations+i,1]:=0
                Else if ( -sin( MLSdata[i,1]) * cos( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,1]:=-1
                     Else Matrix[NoOfGPSequations+i,1]:=0;

                If ( sin( MLSdata[i,1]) * sin( MLSdata[i,2]) >= 0)
                Then If ( sin( MLSdata[i,1]) * sin( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,3]:= 1
                     Else Matrix[NoOfGPSequations+i,3]:= 0
                Else If ( sin( MLSdata[i,1]) * sin( MLSdata[i,2]) = 0)
                     Then Matrix[NoOfGPSequations+i,3]:= -1
                     Else Matrix[NoOfGPSequations+i,3]:= 0;

                If ( cos( MLSdata[i,1]) >=0)
                Then If ( cos( MLSdata[i,1]) =0)
                     Then Matrix[ NoOfGPSequations+i,2]:= 1
                     Else Matrix[ NoOfGPSequations+i,2]:= 0
                Else If ( cos( MLSdata[i,1]) =0)
                     Then Matrix[ NoOfGPSequations+i,2]:= -1
                     Else Matrix[ NoOfGPSequations+i,2]:= 0;
```

```

          End
      Else Begin
             Matrix[NoOfGPSequations+i,1]:= sin(MLSdata[i,1])*sin(MLSdata[i,2])/
                                            MLSdata[i,3];
             Matrix[NoOfGPSequations+i,2]:= cos(MLSdata[i,1])/MLSdata[i,3];
             Matrix[NoOfGPSequations+i,3]:= sin(MLSdata[i,1])*sin(MLSdata[i,2])/
                                            MLSdata[i,3];
          End;
      Matrix[NoOfGPSequations+i,4]:= 0;
    END;
    IF MLSeq[i]=3 THEN   {The DME-distance-equation}
    BEGIN
      Matrix[NoOfGPSequations+i,1]:= cos(MLSdata[i,2])*cos(MLSdata[i,3]);
      Matrix[NoOfGPSequations+i,2]:= cos(MLSdata[i,2])*sin(MLSdata[i,3]);
      Matrix[NoOfGPSequations+i,3]:= sin(MLSdata[i,2]);
      Matrix[NoOfGPSequations+i,4]:= 0;
    END;
  END;
END;{PROCEDURE Calculate_MIASmatrix}



PROCEDURE Calculate_GPSvector(NoOfGPSequations:INTEGER;
                              GPSeq           :GPSROW;
                              REst,RMeas      :RANGE;
                          VAR Vector          :TNMATRIX);
VAR i:INTEGER;

{***************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                  }
{         GPSeq (array of used GPS-equations)                               }
{         REst (the distance between the estimated position of the airborne }
{               GPS-antenna and the satellites)                             }
{         RMeas (the pseudoranges)                                          }
{-Output; Vector (the GPSvector)                                            }
{                                                                           }
{This procedure creates the GPSvector using the parametres calculated by the }
{procedure New_GPSEstimation and the measured pseudoranges.                 }
{***************************************************************************}

BEGIN
  {-------------------------------------------------------------------------}
  {The GPS-parametres.                                                      }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Vector[i,1]:=RMeas[GPSeq[i]]-REst[i];
  END;
END;{PROCEDURE Calculate_GPSvector}



PROCEDURE Calculate_MIASvector(NoOfGPSequations,NoOfMLSequations:INTEGER;
                               GPSeq                            :GPSROW;
                               MLSeq                            :MLSROW;
                               REst,RAdjust                     :RANGE;
                               MLSdata                          :MLSBLOCK;
                               MLSMeas                          :MLS;
                           VAR Vector                           :TNMATRIX);
VAR i:INTEGER;

{***************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                  }
{         NoOfMLSequations (=Number of used MLS-equations)                  }
{         GPSeq (array of used GPS-equations)                               }
{         MLSeq (array of used MLS-euations)                                }
{         REst (the distance between the estimated position and the        }
{               satellites)                                                 }
{         RAdjust (the distance between the airborne MLS-antenna and the    }
{                  satellites)                                              }
{         MLSdata (estimated MLS-data)                                      }
{         MLSMeas (measured MLS-data)                                       }
{-Output; Vector (the MIASvector)                                           }
{                                                                           }
{This procedure creates the MIASvector using the parametres calculated by the}
{procedure New_Estimation and the measured GPS- and MLS-parametres.         }
{***************************************************************************}

BEGIN
  {-------------------------------------------------------------------------}
  {The GPS-parametres.                                                      }

  FOR i:=1 TO NoOfGPSequations DO
  BEGIN
    Vector[i,1]:=RAdjust[GPSeq[i]]-REst[i];
  END;

  {-------------------------------------------------------------------------}
  {The MLS-parametres.                                                      }

  FOR i:=1 TO NoOfMLSequations DO
  BEGIN
    Vector[NoOfGPSequations+i,1]:=MLSmeas[MLSeq[i]]-MLSdata[i,1];
  END;
END;{PROCEDURE Calculate_MIASvector}



PROCEDURE Calc_Weighed_LeastSqMatrix(RowTotal    :INTEGER;
                                     A,
                                     V           :TNMATRIX;
                                 VAR invAtHA_AtH:TNMATRIX;
                                     Error       :BYTE);

VAR H,              {The inverse of the variancematrix V}
    At,             {The transpose of the matrix A}
    AtH,            {The transpose of A multiplied with H}
    AtHA,           {The matrix AtH multiplied with A}
    invAtHA         {The inverse of At*HA}
```

```
            :TNMATRIX;
    i       :INTEGER;

{*****************************************************************************}
{-Input ; RowTotal (The total amount of rows)                                 }
{         A (= The MIASmatrix)                                                }
{         V (= The VARIANCEmatrix)                                            }
{-Output; invAtHA_AtH (= The weighed Least-SquareMatrix)                      }
{                                                                             }
{This procedure calculates the weighed Least-squareMatrix which is the matrix}
{inverse(At*H*A)*At*H in which At is the transpose of A and H is the inverse }
{of the VARIANCEmatrix V.                                                     }
{*****************************************************************************}

var
value                 : string;
gdop                  : Double;

BEGIN
  Transpose(RowTotal,4,A,At);                    {Solution =         At      }
  Inverse(RowTotal,V,H,Error);                   {Solution =         H       }
  Matrix_Mult(4,RowTotal,RowTotal,At,H,AtH);     {Solution =         At*H    }
  Matrix_Mult(4,RowTotal,4,AtH,A,AtHA);          {Solution =         AtH*A   }
  Inverse(4,AtHA,invAtHA,Error);                 {Solution = inverse(AtHA)   }

  Matrix_Mult(4,4,RowTotal,invAtHA,AtH,invAtHA_AtH);
                                                 {Solution =                 }
                                                 {inverse(AtHA)*AtH          }
END;{PROCEDURE Calc_Weighed_LeastSqMatrix}



PROCEDURE Add_GPSPositionvector(EstEcef              :ECEF;
                                GPSPositionZerovector:LOCAL;
                            VAR FixEcef              :ECEF);
VAR Ulocal   :LOCAL;
    EstWgs84:WGS84;
    Uecef    :ECEF;
    i        :INTEGER;

{*****************************************************************************}
{-Input ; EstEcef (the final estimated position of the airborne GPS-antenna  }
{                  in ECEF)                                                   }
{         GPSPositionZerovector (the vector between the airborne GPS-antenna }
{                               (origin) and a certain defined point within }
{                               the airplane when roll, pitch and heading   }
{                               are zero)                                    }
{         FixEcef (the position of a certain defined point within the        }
{                  airplane in Ecef)                                          }
{This procedure adds the positionvector to the calculated position of the    }
{airborne GPS-antenna.                                                        }
{*****************************************************************************}

BEGIN
  {---------------------------------------------------------------------------}
  {Calculate the Uvector out of the GPSPosition_Zerovector by using the      }
  {roll-, pitch- and headingangle.                                            }

  Ulocal[ 1]:=0;
  Ulocal[ 2]:=0;
  Ulocal[ 3]:=0;

  If ( present.att = 1) and
     ( present.hdg = 1)
  Then
  Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
                    GPSPosition_Zerovector,
                    ULocal);

  {---------------------------------------------------------------------------}
  {Convert the estimated ECEF-position to WGS84 in order to use procedure    }
  {Local_to_ECEFOrientation.                                                  }

  ECEF_to_WGS84(EstEcef,EstWgs84);

  {---------------------------------------------------------------------------}
  {The vector Ulocal converted to ECEF orientation.                           }

  Local_to_ECEFOrientation(Ulocal,EstWgs84,0,Uecef);

  {---------------------------------------------------------------------------}
  {Add Uecef to EstEcef to get the final positionfix.                         }

  FOR i:=1 TO 3 DO FixEcef[i]:=EstEcef[i]+Uecef[i];

END;{PROCEDURE Add_GPSPositionvector}



PROCEDURE Add_MIASPositionvector(EstLocal,
                                 MIASPositionZerovector:LOCAL;
                             VAR FixLocal              :LOCAL);
VAR Ulocal1,
    Ulocal2 :LOCAL;
    i       :INTEGER;

{*****************************************************************************}
{-Input ; EstLocal (the final estimated position of the airborne MLS-antenna }
{                   in MLS Local Reference)                                   }
{         MIASPositionZerovector (the vector between the airborne MLS-antenna}
{                                (origin) and a certain defined point within the}
{                                airplane when roll, pitch and heading are zero)}
{         FixLocal (the position of a certain defined point within the       }
{                   airplane in Local Reference)                              }
{This procedure adds the positionvector to the calculated position of the    }
{airborne MLS-antenna.                                                        }
{*****************************************************************************}

BEGIN
  {---------------------------------------------------------------------------}
```

```
  {Calculate the Uvector out of the Position_Zerovector by using the roll-,  }
  {pitch- and headingangle.                                                  }

  Ulocal1[ 1]:=0;
  Ulocal1[ 2]:=0;
  Ulocal1[ 3]:=0;

  If ( present.att = 1) and
     ( present.hdg = 1)
  Then
  Calculate_Uvector(Rollangle,Pitchangle,Headingangle,
                    MIASPosition_Zerovector,
                    ULocal1);

  {--------------------------------------------------------------------------}
  {Convert the Uvector to MLS Local Reference-orientation.                   }

  LocalAircraft_to_LocalMLSOrientation(Ulocal1,North_angle,Ulocal2);

  {--------------------------------------------------------------------------}
  {Add Ulocal2 to EstLocal to get the final positionfix.                     }

  FOR i:=1 TO 3 DO FixLocal[i]:=EstLocal[i]+Ulocal2[i];

END;{PROCEDURE Add_MIASPositionvector}



PROCEDURE Calculate_ECEFVariance(WLSquareMatrix,
                                 VarMatrix      :TNMATRIX;
                             VAR VarianceEcef   :ECEF);
VAR TransWLSqMatrix,
    Matrix,SolMatrix:TNMATRIX;
    i               :INTEGER;

{****************************************************************************}
{-Input ; WLSquareMatrix (The weighed Least-Squarematrix)                    }
{         VarMatrix (The VARIANCEmatrix)                                     }
{-Output; VarianceEcef (The variance in the ECEF-position)                   }
{                                                                            }
{This procedure calculates the variance in the MLS ECEF-position.            }
{The variance is given by the diagonalelements of the matrix                 }
{WLSquareMatrix*VarMatrix*TransWLSqMatrix.                                   }
{****************************************************************************}

BEGIN
  {--------------------------------------------------------------------------}
  {Calculate the transpose of the WLSquareMatrix.                            }

  Transpose(4,TotalOfEquations,WLSquareMatrix,TransWLSqMatrix);

  {--------------------------------------------------------------------------}
  {WLSquareMatrix*VarMatrix*TransWLSqMatrx.                                  }

  Matrix_Mult(TotalOfEquations,TotalOfEquations,4,VarMatrix,TransWLSqMatrix,
              Matrix);
  Matrix_Mult(4,TotalOfEquations,4,WLSquareMatrix,Matrix,SolMatrix);

  {--------------------------------------------------------------------------}
  {The variance in ECEF-position is given by the diagonalelements of the     }
  {SolMatrix.                                                                }

  FOR i:=1 TO 3 DO VarianceEcef[i]:=SolMatrix[i,i];

END;{PROCEDURE Calculate_ECEFVariance}



PROCEDURE Calculate_LOCALVariance(WLSquareMatrix,
                                  VarMatrix      :TNMATRIX;
                              VAR VarianceLocal  :LOCAL);
VAR TransWLSqMatrix,
    Matrix,SolMatrix:TNMATRIX;
    i               :INTEGER;

{****************************************************************************}
{-Input ; WLSquareMatrix (The weighed Least-Squarematrix)                    }
{         VarMatrix (The VARIANCEmatrix)                                     }
{-Output; VarianceLocal (The variance in the Local Reference position)       }
{                                                                            }
{This procedure calculates the variance in the MLS Local Reference position. }
{The variance is given by the diagonalelements of the matrix                 }
{WLSquareMatrix*VarMatrix*TransWLSqMatrix.                                   }
{****************************************************************************}

BEGIN
  {--------------------------------------------------------------------------}
  {Calculate the transpose of the WLSquareMatrix.                            }

  Transpose(4,TotalOfEquations,WLSquareMatrix,TransWLSqMatrix);

  {--------------------------------------------------------------------------}
  {WLSquareMatrix*VarMatrix*TransWLSqMatrx.                                  }

  Matrix_Mult(TotalOfEquations,TotalOfEquations,4,VarMatrix,TransWLSqMatrix,
              Matrix);
  Matrix_Mult(4,TotalOfEquations,4,WLSquareMatrix,Matrix,SolMatrix);

  {--------------------------------------------------------------------------}
  {The variance in MLS Local Reference position is given by the              }
  {diagonalelements of the SolMatrix.                                        }

  FOR i:=1 TO 3 DO VarianceLocal[i]:=SolMatrix[i,i];

END;{PROCEDURE Calculate_LOCALVariance}



PROCEDURE Calculate_Position(NoOfGPSequations,
                             gps1,gps2,gps3,gps4,gps5,gps6,gps7,
```

```
                              NoOfMLSequations,
                              mls1,mls2,mls3                    :INTEGER);
VAR i         :INTEGER;
   firsttime:BOOLEAN;


{****************************************************************************}
{-Input ; NoOfGPSequations (=Number of used GPS-equations)                   }
{         gpsx (the GPS-equations used in the algorithm)                     }
{         NoOfMLSequations (=Number of used MLS-equations)                   }
{         mlsx (the MLS-equations used in the algorithm)                     }
{                                                                            }
{This procedure calculates and prints the positionfix in ECEF, WGS84 (and    }
{Local Reference) using a (sub)set of equations defined in GPS_eq en MLS_eq. }
{****************************************************************************}


BEGIN
  {-------------------------------------------------------------------------}
  {Define the global variables NumberOfGPSequations and NumberOfMLSequations }
  {and calculate the total amount of used equations.                         }

  NumberOfGPSequations:=NoOfGPSequations;
  NumberOfMLSequations:=NoOfMLSequations;

  TotalOfEquations:=NumberOfGPSequations+NumberOfMLSequations;

  {-------------------------------------------------------------------------}
  {Fill in the GPS-equations used in the algorithn.                          }

  GPS_eq[1]:=gps1;
  GPS_eq[2]:=gps2;
  GPS_eq[3]:=gps3;
  GPS_eq[4]:=gps4;
  GPS_eq[5]:=gps5;
  GPS_eq[6]:=gps6;
  GPS_eq[7]:=gps7;

  {-------------------------------------------------------------------------}
  {Check GPSavailable and MLS available.                                     }

  GPSavailable:=(NumberOfGPSequations > 0);
  MLSavailable:=(NumberOfMLSequations > 0);

  IF ((MLSavailable) AND (GPSavailable) AND (TotalOfEquations >= 4))
  OR ((MLSavailable) AND (NumberOfMLSequations = 3)) THEN
  BEGIN
    {-----------------------------------------------------------------------}
    {Fill in the MLS-equations used in the algorithm.                        }

    MLS_eq[1]:=mls1;
    MLS_eq[2]:=mls2;
    MLS_eq[3]:=mls3;

    {-----------------------------------------------------------------------}
    {Calculate the local coordinates of the satellites.                      }

    Calculate_Localcoordinates_of_Satellites(NumberOfGPSequations,
                                             Sat_ecef,Sat_local);

    {-----------------------------------------------------------------------}
    {Create the MIASVARIANCEmatrix.                                          }

    Create_MIASVarianceMatrix(NumberOfGPSequations,NumberOfMLSequations,
                              GPS_eq,MLS_eq,
                              Var_Rdgps,Var_MLS,MIASVARIANCEmatrix);

    iter:= 0;
    REPEAT
    BEGIN
      {---------------------------------------------------------------------}
      {Make a new estimation of the ranges and angles of MIAS.               }

      New_MIASEstimation(NumberOfGPSequations,NumberOfMLSequations,
                         GPS_eq,MLS_eq,
                         Est_local,Azi_local,Ele_local,DME_local,
                         Sat_local,
                         R_est,MLS_data);

      Create_miasweighmatrix( numberofgpsequations, numberofmlsequations,
                              gps_eq, mls_eq,
                              var_rdgps, var_mls, mls_data, miasweighmatrix);
      {---------------------------------------------------------------------}
      {Convert the measured pseudoranges to "measured" ranges between the    }
      {the airborne MLS-antenna and the satellites.                          }

      Adjust_Measured_Pseudoranges_to_MLSantenna(NumberOfGPSequations,
                                                 GPS_eq,
                                                 Antenna_Zerovector,
                                                 Est_local,Sat_local,R_est,
                                                 R_meas,R_adjust);

      {---------------------------------------------------------------------}
      {Fill in the coefficients of the MIASmatrix.                           }

      Calculate_MIASmatrix(NumberOfGPSequations,NumberOfMLSequations,
                           GPS_eq,MLS_eq,
                           Est_local,R_est,Sat_local,MLS_data,MIASmatrix);

      {---------------------------------------------------------------------}
      {Fill in the coefficients of the MIASvector.                           }

      Calculate_MIASvector(NumberOfGPSequations,NumberOfMLSequations,
                           GPS_eq,MLS_eq,
                           R_est,R_adjust,MLS_data,MLS_meas,MIASvector);

      {---------------------------------------------------------------------}
      {Calculate the weighed Least-SquareMatrix of the MIASmatrix.           }

      Calc_Weighed_LeastSqMatrix(TotalOfEquations,MIASmatrix,
```

```
                                    MIASVARIANCEmatrix,WLSqMatrix,Error);

      {-------------------------------------------------------------------}
      {Multiply the weighed Least-SquareMatrix and the MIASvector.        }

      Matrix_Mult(4,TotalOfEquations,1,WLSqMatrix,MIASvector,DeltaVector);

      {-------------------------------------------------------------------}
      {Update the estimated position of the airborne MLS-antenna.         }

      FOR i:=1 TO 4 DO Est_local[i]:=Est_local[i]+DeltaVector[i,1];

      {-------------------------------------------------------------------}
      Inc( iter);
    {---------------------------------------------------------------------}
    END;
    UNTIL ((abs(DeltaVector[1,1]) <= convergence_error) AND
           (abs(DeltaVector[2,1]) <= convergence_error) AND
           (abs(DeltaVector[3,1]) <= convergence_error) AND
           (abs(DeltaVector[4,1]) <= convergence_error)) Or
           ( iter >= maxiter);
                                                  {accuracy of 1E-0004 metres in}
                                                  {x-, y- and z-direction       }

    {---------------------------------------------------------------------}
    {Calculate the variance in the LOCAL-position.                        }

    Calculate_LOCALVariance(WLSqMatrix,MIASVARIANCEmatrix,Variance_local);

    {---------------------------------------------------------------------}
    {Add MIASPositionvector to get a final positionfix.                   }

    Add_MIASPositionvector(Est_local,MIASPosition_ZeroVector,Fix_local);
  END
  ELSE
  BEGIN
    IF (GPSavailable) AND (NumberOfGPSequations >= 4) THEN
    BEGIN
      {-------------------------------------------------------------------}
      {Create the GPSVARIANCEmatrix.                                      }

      Create_GPSVarianceMatrix(NumberOfGPSequations,
                               GPS_eq,
                               Var_Rgps,GPSVARIANCEmatrix);

      iter:= 0;
      REPEAT
      BEGIN
        {-----------------------------------------------------------------}
        {Make a new estimation of the ranges towards the airborne         }
        {GPS-antenna.                                                     }

        New_GPSEstimation(NumberOfGPSequations,
                          GPS_eq,
                          Est_ecef,
                          Sat_ecef,
                          R_est);

        {-----------------------------------------------------------------}
        {Fill in the coefficients of the GPSmatrix.                       }

        Calculate_GPSmatrix(NumberOfGPSequations,
                            GPS_eq,
                            Est_ecef,R_est,Sat_ecef,GPSmatrix);

        {-----------------------------------------------------------------}
        {Fill in the coefficients of the GPSvector.                       }

        Calculate_GPSvector(NumberOfGPSequations,
                            GPS_eq,
                            R_est,R_meas,GPSvector);

        {-----------------------------------------------------------------}
        {Calculate the weighed Least-SquareMatrix of the GPSmatrix.       }

        Calc_Weighed_LeastSqMatrix(NumberOfGPSequations,GPSmatrix,
                                   GPSVARIANCEmatrix,WLSqMatrix,Error);

        {-----------------------------------------------------------------}
        {Multiply the weighed Least-SquareMatrix and the GPSvector.       }


        Matrix_Mult(4,NumberOfGPSequations,1,WLSqMatrix,GPSvector,
                    DeltaVector);

        {-----------------------------------------------------------------}
        {Update the estimated position of the airborne GPS-antenna.       }

        FOR i:=1 TO 4 DO Est_ecef[i]:=Est_ecef[i]+DeltaVector[i,1];

        {-----------------------------------------------------------------}
        Inc( iter);
        {-----------------------------------------------------------------}
      END;
      UNTIL ((abs(DeltaVector[1,1]) <= convergence_error) AND
             (abs(DeltaVector[2,1]) <= convergence_error) AND
             (abs(DeltaVector[3,1]) <= convergence_error) AND
             (abs(DeltaVector[4,1]) <= convergence_error)) Or
             ( iter >= maxiter);
                                                  {accuracy of 1E-0004 metres}
                                                  {in x-, y- and z-direction }

      {-------------------------------------------------------------------}
      {Calculate the variance in the ECEF-position .                      }

      Calculate_ECEFVariance(WLSqMatrix,GPSVARIANCEmatrix,Variance_ecef);

      {-------------------------------------------------------------------}
      {Add GPSPositionvector to get a final positionfix.                  }

```

```
      Add_GPSPositionvector(Est_ecef,GPSPosition_Zerovector,Fix_ecef);
    END
    ELSE
    BEGIN
{      writeln('Too few equations to calculate position!');}
    END;
  END;
END;{PROCEDURE Calculate_Position}



PROCEDURE Set_Measured_Values;

{*******************************************************************************}
{This procedure sets the parametres needed to solve the MIAS-position.         }
{*******************************************************************************}

BEGIN
{The variances of the GPS-pseudoranges.                                        }

  Var_Rgps[1]:= 30 ; {metres}
  Var_Rgps[2]:= 30 ;
  Var_Rgps[3]:= 30 ;
  Var_Rgps[4]:= 30 ;
  Var_Rgps[5]:= 30 ;
  Var_Rgps[6]:= 30 ;
  Var_Rgps[7]:= 30 ;

{-------------------------------------------------------------------------------}
{The variances of the DGPS-pseudoranges and the azimuthangle, elevationangle }
{and DME-distance.                                                             }

  Var_Rdgps[1]:= 5 ; {metres}
  Var_Rdgps[2]:= 5 ;
  Var_Rdgps[3]:= 5 ;
  Var_Rdgps[4]:= 5 ;
  Var_Rdgps[5]:= 5 ;
  Var_Rdgps[6]:= 5 ;
  Var_Rdgps[7]:= 5 ;

  Var_MLS[1]:= 1.7453E-0003; {0.1 degrees}
  Var_MLS[2]:= 1.7453E-0003; {0.1 degrees}
  Var_MLS[3]:= 12          ; {metres}

END;{PROCEDURE Set_Measured_Values}



PROCEDURE MRInterface(alldata:alldatatype);

VAR i:INTEGER;

{*******************************************************************************}
{-Input ; alldata (all GPS- and MLS-data written in datatypes by Marco Meijer}
{-Output; the same data written in datatypes used by René van Leeuwen.        }
{                                                                               }
{This procedure acts as a interface between the program written by Marco      }
{Meijer and the program written by René van Leeuwen.                           }
{*******************************************************************************}

BEGIN
  WITH alldata DO
  BEGIN
    WITH GPS DO
    BEGIN
      NumberOfGPSequations:=0;
      gps1:= 0;
      gps2:= 0;
      gps3:= 0;
      gps4:= 0;
      gps5:= 0;
      gps6:= 0;
      gps7:= 0;

      FOR i:=1 TO 32 DO
      BEGIN
        IF (present = 1) And (Not flag) And (Not prn[i].flag) THEN
        BEGIN
          NumberOfGPSequations:=NumberOfGPSequations+1;
          Case NumberOfGPSequations OF
          1: gps1:=1;
          2: gps2:=2;
          3: gps3:=3;
          4: gps4:=4;
          5: gps5:=5;
          6: gps6:=6;
          7: gps7:=7;
          End;

          R_meas[NumberOfGPSequations]:=prn[i].pr;

          Sat_ecef[NumberOfGPSequations,1]:=prn[i].position.x;
          Sat_ecef[NumberOfGPSequations,2]:=prn[i].position.y;
          Sat_ecef[NumberOfGPSequations,3]:=prn[i].position.z;
        END;
      END;
    END;
    WITH MLS DO
    BEGIN
      mls1:=0;
      mls2:=0;
      mls3:=0;
      numberofMLSequations:= 0;
      IF (present = 1) And (Not flag)
      THEN
      BEGIN
        IF Not AZangle_flag THEN
        BEGIN
         Azi_local[1]:=Azpos.x; Azi_local[2]:=Azpos.y; Azi_local[3]:=Azpos.z;
         MLS_meas[1]:=AZangle;
```

```
          NumberOfMLSequations:=NumberOfMLSequations+1;
          mls1:=1;
         END;
         IF Not ELangle_flag THEN
         BEGIN
          Ele_local[1]:=Elpos.x; Ele_local[2]:=Elpos.y; Ele_local[3]:=Elpos.z;
          MLS_meas[2]:=ELangle;
          NumberOfMLSequations:=NumberOfMLSequations+1;
          mls2:=2;
         END;
         IF Not DME_flag THEN
         BEGIN
          DME_local[1]:=DMEpos.x;DME_local[2]:=DMEpos.y;DME_local[3]:=DMEpos.z;
          MLS_meas[3]:=DMErange;
          NumberOfMLSequations:=NumberOfMLSequations+1;
          mls3:=3;
         END;
         North_angle:= Runwayhdg + 90;
         If North_angle >= 360
         Then North_angle:= North_angle - 360;

         If North_angle <= 0
         Then North_angle:= North_angle + 360;
       END
       Else NumberOfMLSequations:=0;
     END;
     IF ( Att.present = 1) And ( not att.flag) THEN
     BEGIN
       Rollangle:=Att.rollangle;
       Pitchangle:=Att.pitchangle;
     END;
     IF ( HDG.present = 1) And ( not hdg.flag) THEN Headingangle:=HDG.hdgangle;
     GPSPosition_Zerovector[1]:=pos_zerovector.x;
     GPSPosition_Zerovector[2]:=pos_zerovector.y;
     GPSPosition_Zerovector[3]:=pos_zerovector.z;
     Antenna_Zerovector[1]:=ant_zerovector.x;
     Antenna_Zerovector[2]:=ant_zerovector.y;
     Antenna_Zerovector[3]:=ant_zerovector.z;

     present.att:= att.present;
     present.hdg:= hdg.present;
     present.gps:= gps.present;
     present.mls:= mls.present;
     present.dgps:= dgps.present;
   END;
END;{PROCEDURE MRInterface}





Procedure CalcHybridPos( Var alldata: alldatatype; allowed_error: Double;
                         Var position: positiontype);

{MAIN PROGRAMME}
```

```
Var
value   :       String;

BEGIN
  {--------------------------------------------------------------------------}
  {Calculate the first eccentricity of the earth.                            }

  e2:=(sqr(ae)-sqr(be))/sqr(ae);

  {--------------------------------------------------------------------------}
  {Set the measured values.                                                  }

  Set_Measured_Values;

  {--------------------------------------------------------------------------}
  {The interface between the program of Marco Meijer and the program of René }
  {van Leeuwen.                                                              }

  MRInterface(alldata);

  position.flag:= True;
  If ( numberofgpsequations <> 0) Or ( numberofmlsequations <> 0)
  Then Begin
           gotoxy( 1,9);
           writeln( 'Number of GPS equations: ', numberofgpsequations,
                    ' Number of MLS equations: ', numberofmlsequations);
       End;

  If (( numberofgpsequations <4) And
     ( numberofmlsequations =0)) Or
    ( numberofgpsequations + numberofmlsequations <4) Or
    (( numberofgpsequations = 0) And
     ( numberofmlsequations <3))
  Then Exit;



  {--------------------------------------------------------------------------}
  Convergence_error:= allowed_error;

  Datumpoint_ecef[ 1]:= alldata.mls.MLSthrespos.x;
  Datumpoint_ecef[ 2]:= alldata.mls.MLSthrespos.y;
  Datumpoint_ecef[ 3]:= alldata.mls.MLSthrespos.z;

  Ecef_to_wgs84( datumpoint_ecef, datumpoint_wgs84);

  EstWGS84.longitude:= position.wgs84lon * 180 / pi;
  If estwgs84.longitude > 0
  Then Begin
           estwgs84.hemilong:= 'east      ';
           estwgs84.longitude:= abs( estwgs84.longitude);
       End
  Else estwgs84.hemilong:= 'west      ';

  EstWGS84.latitude := position.wgs84lat * 180 / pi;
```

```
  If estwgs84.latitude > 0
  Then Begin
            estwgs84.hemilat:= 'north     ';
            estwgs84.latitude:= abs( estwgs84.latitude);
       End
  Else estwgs84.hemilat:= 'south     ';

  EstWGS84.height   := position.wgs84alt;

  Wgs84_to_Ecef( Estwgs84, Est_ecef);

  Est_Local[1]:=1;
  Est_Local[2]:=1;
  Est_Local[3]:=1;
  Est_Local[4]:=0;

  Calculate_Position(NumberOfGPSequations, {total of used GPS-equations}
                     gps1,gps2,gps3,gps4,gps5,gps6,gps7, {the used GPS-equations
                     NumberOfMLSequations, {total of used MLS-equations}
                     mls1,mls2,mls3);      {the used MLS-equations     }
  If iter >= maxiter
  Then position.flag:= True
  Else position.flag:= False;

  If ( NumberOfMLSequations = 0) And ( NumberOfGPSequations >= 4)
  Then With position DO
       Begin
            x:=Est_ECEF[ 1];
            y:=Est_ECEF[ 2];
            z:=Est_ECEF[ 3];
            alldata.gps.DeltaT:= Est_Ecef[ 4];
            EcefTrueLocalFalse:= True;
            Convert_Pos_to_Wgs( position);
       End;
  If (( NumberOfMLSequations + NumberOfGPSequations >= 4) And
      ( NumberOfMLSequations > 0)) Or
     ( numberofmlsequations >= 3)
  Then With position Do
       Begin
            x:=Est_Local[ 1];
            y:=Est_Local[ 2];
            z:=Est_Local[ 3];
            alldata.gps.deltat:= Est_Local[ 4];
            EcefTrueLocalFalse:= False;
       End;

END;{MAIN PROGRAMME}

End.
```

```
Unit GPScalc;
{************************************************************************}
{ This unit provides procedures for the calculation of the SV position
  for the GPS navigation system. It is only for L1 single frequency users}
{************************************************************************}

Interface

{$N+,E+}

Uses MIASglob, GPSglob;

Procedure Clockcorrection( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure corrects the measured times for one GPS range which have an
  error because of clock. The correction used is described in Appendix 3 to
  Annex A to STANAG 4294, Draft issue L 1 August 1990.

                            Input : satellite number
                                    ephemeris, clock and SV data
                                    from Gpsint
                            Output: corrected GPStime }
{************************************************************************}


Procedure RelCorrection( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure corrects the measured times for one GPS range which have an
  error because of relativistic effects. The correction used is described in
  Appendix 3 to Annex A to STANAG 4294, Draft issue L 1 August 1990.

  To calculate Ek, a function from SVposition is used.
                            Input : satellite number
                                    ephemeris, clock and SV data
                                    from Gpsint
                            Output: corrected GPStime }
{************************************************************************}


Procedure L1correction( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure corrects the measured times for on GPS range which have an
  error because of group delay. The correction used is described in Appendix 3 to
  Annex A to STANAG 4294, Draft issue L 1 August 1990.
  Note: Do not use this for DGPS operation.
                            Input : satellite number
                                    clockparameters for on SV
                            Output: corrected GPStime in Gpsint}
{************************************************************************}


Procedure Ionosphericcorrection( sv: Byte; position: positiontype;
                                 Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure corrects the GPStime for ionospheric delay. The formulas
  used here are described in Appendix 6 to Annex A to STANAG 4294.
  USE CLOCKCORRECTION ,L1CORRECTION AND ELEV_AZIM BEFORE USING THIS PROCEDURE,
  BECAUSE THE GPSTIME, THE ELEVATION AND AZIMUTH MAY NOT BE CORRECT WHEN
  USING THIS PROCEDURE FOR THE FIRST TIME.
                            Input : satellite number
                                    position of the receiver
                                    ionospheric data from Gpsint
                            Output: GPStime corrected in Gpsint}
{************************************************************************}


Procedure Troposphericcorrection( sv: Byte; pos: positiontype;
                                  Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure corrects the transmission time for tropospheric effects.
  The range error because of troposphere is R( h, o) = f(o) * dR(h).
  The formulas used are from Appendix 6 to Annex A to STANAG 4294.
  Note: use Elev_Azim befor using Troposphericcorrection, because the ele-
        vation is used here.
                            Input : satellite number
                                    position of the receiver
                                    elevation of user to sv
                            Output: corrected transmission time}
{************************************************************************}


Procedure SVposition( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure calculates the position of a satellite. The formulas use
  can be found in Appendix 3 to Annex A to STANAG 4294.
                            Input : satellitenumber
                                    ephemerisdata
                                    GPStime
                            Output: satellite position}
{************************************************************************}


Procedure SVpos_earthadjusted( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure calculates the position of a satellite corrected for the
  rotation of the earth during the signal transmission. The formulas used
  can be found in ENAV II handout Winter 1991 from Ohio University.
  The SV position is in an earth-centered-earth-fixed coordinate system.
  This means that the satellite position at time of transmission is different
  from the SVposition at time of reception (the earth rotated). To correct
  for this, the earth rotation angle is calculated and the satellite
  position is corrected for this rotation.
                            Input : satellitenumber
                                    ephemerisdata
                                    GPStime
                            Output: satellite position}
{************************************************************************}


Procedure Elev_Azim( sv: Byte; Var Gpsint: GPSinttype;
```

```
                        position: positiontype);
{************************************************************************}
{ This procedure calculates the Azimuth and Elevation angle between
  the satellite and the user. This procedure uses the satellite and
  user position in ECEF coordinates and the userposition in WGS.
  Note: Be sure that both WGS84 and Ecef fields are valid and correspond
        to each other.
                              Input : satellitenumber
                                      satelliteposition
                                      userposition in ECEF
                                      userposition in WGS
                              Output: azimuth and elevation.}
{************************************************************************}


Procedure Convert_Pos_to_WGS( Var position: positiontype);
{************************************************************************}
{ Procedure for conversion of the receiver's ECEF-coordinates
   (X,Y,Z)    to     <lat,long,alt>  -   coordinates.
    dimension     <m,m,m>  ----->   dimension    <rad,rad,m>
  This is an iterative procedure
                              Input : position, x,y,z
                              Output: position, lat, lon, h}
{************************************************************************}


Procedure Calc_PR( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure calculates the Pseudo range from the measured transmission
  and reception time.
                              Input : satellite number
                                      Gpsint, tx and rx time
                              Output: Pseudo range}
{************************************************************************}


Procedure CalcSmoothPR( sv: Byte; Var Gpsint: GPSinttype);
{************************************************************************}
{ This procedure calculates the smoothed Pseudo range from the previous
  pseudoranges and integrated carrier phase. This procedure was written
  by Peter Vianen. It is only meant for DGPS reference stations.
                              Input : satellite number
                                      Gpsint, tx and rx time
                              Output: Pseudo range}
{************************************************************************}


Implementation

Uses Mathx;

Const
     mu =       3.986005E14;              {[m3/s2] WGS 84 value of earth's
                                           universal gravitational constant}
     c  =       2.99792458E8;             {[m/s] speed of light}
     F  =          -4.442807633E-10;          {[s/m1/2]}
     omegaedot= 7.2921151467E-5;          {[rad/s] WGS 84 value of the earth's
                                           rotation rate}
     flattening =  3.3528106E-3;          {flattening of the earth       }
     earthAaxis =  6.378137E+6;           {long earth axis               }
     earthEccentricitySqr = flattening * (2 - flattening);  { e^2 !! }
     pi          =       3.14159265358979323 85;
     tweepi      =       2 * pi;
     F_l1        =       1575420000;      { L1 frequency}

Var
   i             : Integer;
   k             : Array [1..32] Of Longint;
   previous_intcarphase,
   DR_bias       : Array [1..32] Of Double;



Procedure L1correction( sv: Byte; Var Gpsint: GPSinttype);

Begin
     With Gpsint.prn[ sv] Do            {tgps = tsv - (deltasv)L1}
          txtime:= txtime + clock.Tgd;
                                         {(deltatsv)L1 = deltatsv - Tgd}
                                         {so, tgps = tgps + Tgd}
End;            { End of procedure L1correction}


Procedure Ionosphericcorrection( sv: Byte; position: positiontype;
                                 Var Gpsint: GPSinttype);

Var
   psi           :       Double;         { Earth's central angle between user
                                           position and Earth projection of
                                           the ionospheric intersection point}
   Geomaglat     :       Double;         { Geomagnetic latitude of the earth}
                                         { projection of the ionospheric}
                                         { intersection point}
   sqrgeomaglat :        Double;         { help variable, Sqr( Geomaglat)}
   Geodetlat     :       Double;         { Geodetic lattitude of earth projec}
                                         { tion of the ionospheric }
                                         { intersection point}
   Geodetlon     :       Double;         { Geodetic longitude of earth projec}
                                         { tion of the ionospheric }
                                         { intersection point}
   PER           :       Double;         { period of model [s]}
   t             :       Double;         { Local time [s]}
   X             :       Double;         { Phase X [rad] }
   SqrX          :       Double;         { help variable, Sqr( X)}
   AMP           :       Double;         { vertical delay amplitude [s]}
   F             :       Double;         { Obliquity factor []}
   Tiono         :       Double;         { Ionospheric time correction [s]}

Begin
     With Gpsint. prn[ sv] Do           { calculate psi}
```

```
        psi:= 0.0137 / ( E + 0.11) - 0.022;

    With position Do
    Begin
        Geodetlat:= WGS84lat;
        Geodetlon:= WGS84lon;
    End;

    With Gpsint.prn[ sv] Do
    Begin
        Geodetlat:= Geodetlat + psi * Cos ( A);
        Geodetlon:= Geodetlon + psi * sin ( A) / cos ( Geodetlat);
    End;

    Geomaglat:= Geodetlat + 0.064 * ( Geodetlon - 1.617);

                                      { PER = sum from 0 to including 3 of }
    With Gpsint.iono Do              { betan* Geomaglat^n}
    Begin
        sqrgeomaglat:= Sqr( Geomaglat);
        PER:= beta0 + beta1 * Geomaglat + beta2 * sqrGeomaglat +
              beta3 * sqrgeomaglat * Geomaglat;
    End;
    If PER < 72000
    Then PER:= 72000;

    With Gpsint.prn[ sv] Do          { calculate Local time}
        t:=  4.32E4 * Geodetlon + txtime;
    If t >= 86400
    Then t:= t - 86400;
    If t < 0
    Then t:= t + 86400;

    X:= 2 * pi * ( t - 50400) / PER;   { calculate phase}

                                      { AMP = sum from 0 to including 3 of}
    With Gpsint. iono Do             { alfan * Geomaglat^n}
        AMP := alfa0 + alfa1 * Geomaglat + alfa2 * sqrgeomaglat +
               alfa3 * sqrgeomaglat * Geomaglat;
    If AMP < 0
    Then AMP := 0;

    With Gpsint. prn[ sv] Do
        F := 1 + 16 * Sqr( 0.53 - E) * ( 0.53 - E);

    sqrx:= Sqr( X);
    If Abs( X) < 1.57
    Then Tiono:= F * ( 5E-9 + AMP * (1 - SqrX/2 + Sqr( SqrX)/24))
    Else Tiono:= F * 5E-9;

    With Gpsint.prn[ sv] Do          { (deltatsv)I = deltatsv - Tiono}
        txtime:= txtime + Tiono;      { so tGPS = tGPS + Tiono}
End;      { End of procedure Ionosphericcorrection}


Procedure Troposphericcorrection( sv: Byte; pos: positiontype;
                                  Var Gpsint: GPSinttype);

Const
    Ns        =      324.8;          { Surface refractivity index at MSL}
                                     { is 324.8 with standard dev of 25.98}
                                     { If measured, it can be used}
Var
  f          :       Double;         { Range error as function of
                                       elevation angle}
  dN         :       Double;
  dR         :       Double;         { Range error as a function of
                                       altitude [m]}
  temp       :       Double;         { temporary variable}
  N1         :       Double;         { refractivity index at h=1km}

Begin
    dN        :=      -7.32 * exp( 0.005577 * Ns);

    With pos Do
    Begin
        If (h >=0) And ( h < 1)
        Then dR:= (
                      Ns * ( 1 - h) +
                      0.5 * dN * ( 1 - Sqr( h)) + 1430 + 732
                    ) * 1E-3;

        If (h > 1) And ( h <= 9)
        Then Begin
                    N1:= Ns + dN;
                    temp:= ln( N1 / 105)/8;
                    dR:= (
                           (- N1/ temp) *
                           (
                            exp( ( 1 - 9) * temp) - exp( ( 1 - h) * temp)
                           ) + 732
                          ) * 1E-3;
             End;

        If (h > 9)
        Then dR:= (
                      (-105/0.1424) *
                      (
                       exp( -0.1424 * ( 20186.8 - 9)) -
                       exp( -0.1424 * ( h - 9))
                      )
                    ) * 1E-3;
    End;

    With Gpsint. prn[ sv] Do
    Begin
        If E < 90
        Then f:= 1/
                    (
                     sin( E) + 0.00143/
```

```
                    (
                       tan( E) + 0.0455
                    )
                   )
          Else f:= 1;

          txtime:= txtime + ( f * dR / c);   { correct GPStime}
                                             { Range error/speed is time [s]}
     End;
End;                    {End of procedure Troposphericcorrection}


Function EccentricAnomaly( Mk: Double; Var Gpsint: GPSinttype;
                           sv: byte): Double;
{************************************************************************}
{ This function solves the eccentricAnomaly ( Ek) by iteration. The
  equation to be solved is : Ek = Mk + e sin( Ek). So the iteration formula
  becomes: Ekn+1 = Mk + e * sin( Ekn) which we iterate, until the difference
  between Ekn+1 and Ekn is smaller than epsilon. We start with Ekn = Mk.
                             Input : Mean Anomaly Mk
                                     Eccentricity from Gpsint
                             Output: Eccentric Anomaly Ek }
{************************************************************************}
Const
     epsilon    =       1E-9;               { accuracy for Newton Raphson}
     Maxiter    =       20;                 { maximum number of iterations}
Var
   Ekn          :       Double;             { for iteration}
   Eknplus1     :       Double;
   iter         :       Integer;                { number of iterations}

Begin
   With Gpsint.prn[ sv] Do
   With ephemeris Do
   Begin
        Eknplus1:= Mk;                          { starting condition}
        iter:= 0;
        Repeat
              Ekn:= Eknplus1;
              Eknplus1 := Mk + e * sin( Ekn);   { iterative equation}
              Inc( iter);
        Until ( Abs( Eknplus1 - Ekn) < epsilon) And (iter <= Maxiter);
                                                { stop criterion}
        If (iter > Maxiter)
        Then Gpsint.prn[ sv].flag:= True;      { not enough convergence}
   End;
   EccentricAnomaly:= Eknplus1;
End;  { End function eccentricanomaly}


Procedure SVposition( sv: Byte; Var Gpsint: GPSinttype);

Var
   Ax           :       Double;         {Semi-Major Axis}
   no           :       Double;         {computed mean motion}
   tk           :       Double;         {time from Ephemeris Reference Epoch}
   t            :       Double;         {GPStime at time of transmission
                                         GPStime corrected for transit time
                                         (rangs/speed of light)}
   n            :       Double;         {corrected mean motion}
   Mk           :       Double;         {Mean Anomaly}
   vk           :       Double;         {True Anomaly}
   vknum,                               {numerator for calculating Vk}
   vkden        :       Double;         {denumerator for calculating Vk}
   phik,                                {argument of latitude}
   twophik,                             {2 * phik}
   sintwophik,                          { sin of twophik}
   costwophik,                          { cos of twophik}
   deltauk,                             {argument of latitude correction}
   deltark,                             {radius correction}
   deltaik,                             {correction to inclination}
   uk,                                  {corrected argument of latitude}
   rk,                                  {corrected radius}
   ik,                                  {corrected inclination}
   cosik,                               {cos( ik)}
   sinik,                               {sin( ik)}
   xk1,                                 {positions in orbital plane}
   yk1,
   cosomegak,                           {cos( omegak)}
   sinomegak,                           {sin( omegak)}
   omegak       :       Double;         {corrected longitude of
                                         ascending node}


Begin                                           { begin procedure SVposition}
   With Gpsint.prn[ sv] Do                      { calculate t}
   Begin
      t  := txtime;
      With ephemeris Do
      Begin
           tk     := t - toe;
           If ( tk > 302400)
           Then tk:= tk - 604800;
           If ( tk < -302400)
           Then tk:= tk + 604800;

           Ax      := Sqr( Asqrt);
           If (Ax = 0)
           Then Exit;

           no     := Sqrt( mu/ Ax) / Ax;

           n      := no + deltan * pi;

           Mk     := Mo * pi + n * tk;

           Ek     := EccentricAnomaly( Mk, Gpsint, sv);
                                                { Compute the true anomaly}
           vknum  := Sqrt
                     (
```

```
                    1 - Sqr( e)
                   ) *
                   sin (Ek);          {calculate vk in parts}

         vkden  := cos( Ek) - e;

         If Abs( vkden) < 1E-100      { protect againt division by zero}
         Then If vknum >= 0
              Then vk:= pi * 0.5
              Else vk:= pi * 1.5
         Else vk:= arctan( vknum / vkden);

         If vknum < 0                 { 3rd or 4th quadrant}
         Then If vkden < 0
              Then vk:= vk + pi       { 3rd quadrant}
              Else vk:= vk + 2 * pi   { 4th quadrant}
         Else If vkden < 0            { 1st or 2nd quadrant}
              Then vk:= vk + pi;      { 2nd quadrant}


         phik   := vk + omega * pi;
                                               { Compute the 2nd harmonic
                                                 pertubations}
         twophik:= 2 * phik;
         sintwophik:= sin( twophik);
         costwophik:= cos( twophik);
         deltauk:= Cus * sintwophik + Cuc * costwophik;
         deltark:= Crc * costwophik + Crs * sintwophik;
         deltaik:= Cic * costwophik + Cis * sintwophik;{corrected}

         uk     := phik + deltauk;
         rk     := Ax *
                   (
                    1 - e * cos( Ek)
                   ) + deltark;
         ik     := (io + IDOT * tk) * pi + deltaik ;

         xk1    := rk * cos( uk);
         yk1    := rk * sin( uk);
                                               { correct longitude of
                                                 the ascending node}
         omegak := omegao * pi + ( omegadot * pi - omegaedot) * tk -
                   omegaedot * toe;
    End;

    With position Do                           { sv ECEF coordinates [m]}
    Begin
      sinomegak:= sin( omegak);
      cosomegak:= cos( omegak);
      cosik    := cos( ik);
      sinik    := sin( ik);
      x    := xk1 * cosomegak - yk1 * cosik * sinomegak;
      y    := xk1 * sinomegak + yk1 * cosik * cosomegak;
      z    := yk1 * sinik;
    End;
  End;
End;                                        {end of procedure SVposition}


Procedure SVpos_earthadjusted( sv: Byte; Var Gpsint: GPSinttype);

Var
   rotation     :       Double;
   travel       :       Double;
   xold,
   yold,
   cosrot,
   sinrot       :       Double;

Begin
     With GPSint.prn[sv] Do
     Begin
          travel:= pr/c - GPSint.deltat;
          rotation:= travel * omegaedot;
          cosrot:= cos( rotation);
          sinrot:= sin( rotation);
     End;

     With GPSint.prn[sv].position Do
     Begin
          xold:= x;
          yold:= y;
          x:= cosrot * xold + sinrot * yold;
          y:= -sinrot * xold + cosrot * yold;
     End;
End;

Procedure Clockcorrection( sv: Byte; Var Gpsint: GPSinttype);

Var
   tsv,                                     {SV prn code phase at message TX time
                                             [s]}
   dt,                                      {help variable}
   deltatsv                                 {SV prn code phase time offset [s]}
                :       Double;             {Relativistic correction term [s]}

Begin
     tsv:= Gpsint.prn[ sv].txtime;     {approximate t with tsv}


     With Gpsint.prn[ sv].clock Do     {tsv can be used instead of GPStime}
     Begin
        dt:= tsv - toc;
        deltatsv:= af0 + af1 * dt +
                   af2 * Sqr( dt);
     End;

     With Gpsint.prn[ sv] Do
        txtime:= tsv - deltatsv;        {delete received txtime, insert
                                         GPStime at time of transmission}
```

```
End;           { End of procedure clockcorrection}


Procedure RelCorrection( sv: Byte; Var Gpsint: GPSinttype);

Var
   t,
   tk,
   deltatr              :   Double;
   Ax,
   no,
   n,
   Mk           :       Double;

Begin
     With Gpsint.prn[ sv].ephemeris Do
        deltatr:= F * e * Asqrt;

     With Gpsint.prn[ sv] Do           { calculate Ek}
     Begin
          t  := txtime;
          With ephemeris Do
          Begin
               tk     := t - toe;
               If ( tk > 302400)
               Then tk:= tk - 604800;
               If ( tk < -302400)
               Then tk:= tk + 604800;

               Ax      := Sqr( Asqrt);
               If (Ax = 0)
               Then Exit;

               no     := Sqrt( mu/ Ax) / Ax;

               n      := no + deltan * pi;

               Mk     := Mo * pi + n * tk;
          End;
     End;

     Gpsint.prn[sv].Ek:= EccentricAnomaly( Mk, Gpsint, sv);

     With Gpsint.prn[ sv] Do
        deltatr:= deltatr * sin ( Ek);

     With Gpsint.prn[ sv] Do
        txtime:= txtime - deltatr;     {delete received txtime, insert
                                        GPStime at time of transmission}
End;


Procedure Elev_Azim( sv: Byte; Var Gpsint: GPSinttype;
                     position: positiontype);
Var
   xvec,                                     { vector coordinates from user}
   yvec,                                     { to satellite}
   zvec         :       Double;
   xrot,
   yrot         :       Double;         { transformed( rotated) coordinates}
   vec          :       Double;         { vector length}
   temp         :       Double;         { help variable}
   sintheta,
   costheta,
   sinpsi,
   cospsi       :       Double;         { helpvariables}

Begin
   With Gpsint. prn[ sv].position Do
   Begin
        xvec    := x - position.x;
        yvec    := y - position.y;
        zvec    := z - position.z;
        costheta:= cos( position.WGS84lon);
        sintheta:= sin( position.WGS84lon);
        cospsi  := cos( position.WGS84lat);
        sinpsi  := sin( position.WGS84lat);
   End;

   vec  := Sqrt( Sqr( xvec) + Sqr( yvec) + Sqr( zvec));
   temp := xvec * costheta + yvec * sintheta;
   xrot := temp * sinpsi - zvec * cospsi;
   yrot := yvec * cospsi - xvec * sinpsi;

   With Gpsint. prn[ sv] Do
   Begin
        E:= arcsin( (zvec * sinpsi + temp * cospsi)/ vec);
        A:= arctan( yrot / -xrot);

        If ( xrot < 0) And ( yrot < 0)
        Then A:= A + tweepi;

        If ( xrot < 0) And ( yrot >= 0)
        Then A:= A;

        If ( xrot >= 0) And ( yrot < 0)
        Then A:= pi - A;

        If ( xrot >= 0) And ( yrot >= 0)
        Then A:= A + pi;
   End;
End;


procedure Convert_Pos_to_WGS( Var position: positiontype);

const
  smallValue = 1E-10; { small value to check if user is at a pole, }
                      { requested accuracy of NewtonRaphson etc.   }
  verySmall  = 1E-15; { small value to prevent denominator = 0     }
```

```
  maxIterAllowed = 25;    { maximum number of iterations allowed    }

var
  numberOfRuns                                         : byte;
  C,func1,Delta_lat                                    : Double;

begin
  With position Do
  Begin
                                { secures against crash on Poles }
     If (Abs( X) < smallValue) And
        (Abs( Y) < smallValue)
     Then Begin                  { user is at or close to North/South Pole }
             If ( Z > 0) Then
                WGS84lat := pi/2                         { North Pole }
             Else
                WGS84lat := -pi/2;                       { South Pole }


                                              { EXPERIMENTAL }
                      { the value of the longitude is calculated  }
                      { as zero, but with help of the course      }
                      { this value must be defined yet !!!!!!!!!! }
             WGS84lon := 0;
             WGS84alt := Abs( Z) - (EarthAaxis *
                                    sqrt(1 - earthEccentricitySqr) );

          End;

                                      { calculating the longitude }
     If ( Abs( X) < verySmall) then
     Begin                      { protects arctan function against zero X }
        If X >= 0 then X :=  verySmall;
        if X < 0 then X := -verySmall;
     End;
     WGS84lon := arctan( Y / X);
                       { correction of the arctan function: changes }
                       { range from -pi/2 -> pi/2  to  -pi -> pi.   }
     If ( ( X < 0) And ( Y > 0) )
     Then WGS84lon := WGS84lon + pi;

     If ( ( X < 0) And ( Y < 0) )
     Then WGS84lon := WGS84lon - pi;

                  { calculation of the latitude with Newton-Raphson }
     C                 := Sqrt( Sqr( X) + Sqr( Y) );
     WGS84lat          := arctan( Z / C);
     numberOfRuns      := 1;
     Delta_lat         := 10;                       { initial value }

     While ( Abs( Delta_lat) > smallValue) And
           ( numberOfRuns < maxIterAllowed) Do
     Begin
         If WGS84lat = 0.5*pi
         Then WGS84lat := 0.5*pi - smallValue;

         func1 := 1 / Sqrt( 1 - earthEccentricitySqr *
                                Sqr( sin( WGS84lat) ) );

         Delta_lat := ( Z - C * sin( WGS84lat) /
                               cos( WGS84lat) + earthEccentricitySqr *
                               EarthAaxis * func1 * sin( WGS84lat)) /
                               ( C / Sqr( cos( WGS84lat)) - EarthAaxis *
                               cos( WGS84lat) * earthEccentricitySqr *
                               func1 * (1 + earthEccentricitySqr *
                               Sqr( func1 * sin( WGS84lat)) ));

         WGS84lat := WGS84lat + Delta_lat;
         numberOfRuns    := numberOfRuns + 1;
     End;

     If numberOfRuns >= maxIterAllowed
     Then flag := True;                 { failure in Newton Raphson }

                                        { calculation of the altitude }
     WGS84alt := X / ( cos( WGS84lat) *
                 cos( WGS84lon)) - earthAaxis /
                 Sqrt( 1 - earthEccentricitySqr *
                 Sqr( sin( WGS84lat)));
  End; { end with position}
End;  { of procedure Convert_Pos_To_WGS }


Procedure Calc_PR( sv: Byte; Var Gpsint: GPSinttype);

Begin
     With Gpsint.prn[ sv] Do
          pr:= (rxtime - txtime) * c;
End;


Procedure CalcSmoothPR ( sv: Byte; Var Gpsint: GPSinttype);

Var
   PR_measured,
   delta_phase,
   DR                      : Double;

Begin
     With Gpsint.prn[sv] Do
     Begin
          PR_measured := ( rxtime - ( txtime + ( intcarphase / F_l1 ))) * c;

          If ( k[sv] = 0 )
          Then PR := PR_measured
          Else Begin
                  delta_phase := intcarphase - previous_intcarphase [sv];
                  DR := delta_phase * c / F_l1;
                  PR := ( ( PR + DR ) * k[sv] + PR_measured ) / ( k[sv] + 1 );
                  DR_bias[sv] := ( PR - PR_measured + k[sv] * DR_bias[sv] )
```

```
                              / ( k[sv] + 1 );
                PR := PR - DR_bias[sv];
         End;

         previous_intcarphase[sv] := intcarphase;
         k[sv] := k[sv] + 1;
    End;
End;



Begin
    For i := 1 To 32 Do
    Begin
        DR_bias [i] := 0;
        k [i] := 0;
    End;

End.
```

```
Unit Mathx;

Interface

{$N+,E+}

Uses MIASglob;

Const
     pi        =       3.1415926535897932385;
     tweepi    =       2 * pi;


Function tan( arg: Double): Double;
{*************************************************************************}
{ This function provides the tangens of the argument. It is not very accurate
                             Input : argument in radians
                             Output: tangens of the argument}
{*************************************************************************}


Function arccos( x: Double): Double;
{*************************************************************************}
{ This function provides the arccosine of the argument
                             Input : argument in radians
                             Output: arccosine of the argument}
{*************************************************************************}

Function arcsin( x: Double): Double;
{*************************************************************************}
{ This function provides the arcsine of the argument
                             Input : argument in radians
                             Output: arcsine of the argument}
{*************************************************************************}


Implementation


Function tan( arg: Double): Double;

Var
   si   :      Double;                          { temporary sine}
   co   :      Double;                          { temporary cosine}

Begin
     si:= sin( arg);                            { tan x = sin x/ cos x}
     co:= cos( arg);

     If (co = 0)
     Then If si >= 0
          Then tan:= 1E38
          Else tan:= -1E38
     Else tan := si / co;
End;



Function arccos( x: Double): Double;

begin
  if (X = 0) then
    arccos := pi/2
  else
  begin

    if X < 0 then
    begin
      if sqr(X) > 1 then
        X := -1.0;
      arccos := arctan (  ( sqrt( 1 - sqr(X) ) ) / X ) + pi;
    end
    else
    begin
      if sqr(X) > 1 then
        X := 1;
      arccos := arctan (  ( sqrt( 1 - sqr(X) ) ) / X );
    end;
  end;
end;   (* of function arccos *)


Function arcsin( x: Double): Double;

begin
  If x > 1
  Then x:= 1;

  If x < -1
  Then x:= -1;

  if ( abs( X ) = 1 ) then
  begin
    if X = 1 then
      arcsin := pi/2
    else
      arcsin := -pi/2;
  end
  else
    arcsin := arctan ( X / ( sqrt( 1 - sqr(X) ) ));
end; (* of function arcsin *)


End.
```

```
unit Matrix;

{-----------------------------------------------------------------------}
{-                                                                     -}
{-     Turbo Pascal Numerical Methods Toolbox                          -}
{-     Copyright (c) 1986, 87 by Borland International, Inc.           -}
{-                                                                     -}
{-  This unit provides procedures for dealing with systems of linear   -}
{-  equations.                                                         -}
{-                                                                     -}
{-----------------------------------------------------------------------}

{$N+,E+}

interface

Uses MIASglob;

{$IFOPT N+}
type
  Float = DOuble; { 8 byte real, requires 8087 math chip }

const
  TNNearlyZero = 1E-015;
{$ELSE}
type
  Float = real;   { 6 byte real, no math chip required }

const
  TNNearlyZero = 1E-07;
{$ENDIF}

  TNArraySize = 10;               { Size of the matrix }

type
  TNvector = array[1..TNArraySize] of Float;
  TNmatrix = array[1..TNArraySize] of TNvector;

procedure Determinant(Dimen : integer;
                      Data  : TNmatrix;
                  var Det   : Float;
                  var Error : byte);

{-----------------------------------------------------------------------}
{-                                                                     -}
{-     Input: Dimen, Data                                              -}
{-     Output: Det, Error                                              -}
{-                                                                     -}
{-             Purpose : Calculate the determinant of a matrix by      -}
{-                       making it upper-triangular and then           -}
{-                       taking the product of the diagonal elements.  -}
{-                                                                     -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real;     -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector  -}
{-                                                                     -}
{-    Global Variables : Dimen : integer; Dimension of the square matrix -}
{-                       Data  : TNmatrix; Square matrix               -}
{-                       Det   : real;     Determinant of Data         -}
{-                       Error : integer;  Flags if something goes wrong -}
{-                                                                     -}
{-              Errors : 0: No errors;                                 -}
{-                       1: Dimen < 1                                  -}
{-                                                                     -}
{-----------------------------------------------------------------------}

procedure Inverse(Dimen : integer;
                  Data  : TNmatrix;
              var Inv   : TNmatrix;
              var Error : byte);

{-----------------------------------------------------------------------}
{-                                                                     -}
{-               Input: Dimen, Data                                    -}
{-              Output: Inv, Error                                     -}
{-                                                                     -}
{-             Purpose : calculate the inverse of a matrix with        -}
{-                       Gauss-Jordan elimination.                     -}
{-                                                                     -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real;     -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector  -}
{-                                                                     -}
{-    Global Variables : Dimen : integer;  Dimension of the square matrix -}
{-                       Data  : TNmatrix;  Square matrix              -}
{-                       Inv   : TNmatrix;  Inverse of Data            -}
{-                       Error : integer;   Flags if something goes wrong -}
{-                                                                     -}
{-              Errors : 0: No errors;                                 -}
{-                       1: Dimen < 1                                  -}
{-                       2: no inverse exists                          -}
{-                                                                     -}
{-----------------------------------------------------------------------}

procedure Gaussian_Elimination(Dimen        : integer;
                               Coefficients : TNmatrix;
                               Constants    : TNvector;
                           var Solution     : TNvector;
                           var Error        : byte);

{-----------------------------------------------------------------------}
{-                                                                     -}
{-               Input: Dimen, Coefficients, Constants                 -}
{-              Output: Solution, Error                                -}
{-                                                                     -}
{-             Purpose : Calculate the solution of a linear set of     -}
{-                       equations using Gaussian elimination and      -}
{-                       backwards substitution.                       -}
{-                                                                     -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real      -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector  -}
{-                                                                     -}
```

```
{-    Global Variables : Dimen : integer;         Dimension of the square   -}
{-                                                 matrix                     -}
{-                       Coefficients : TNmatrix; Square matrix               -}
{-                       Constants    : TNvector; Constants of each equation-}
{-                       Solution     : TNvector; Unique solution to the     -}
{-                                                 set of equations           -}
{-                       Error        : integer;  Flags if something goes    -}
{-                                                 wrong.                     -}
{-                                                                            -}
{-              Errors:  0: No errors;                                        -}
{-                       1: Dimen < 1                                         -}
{-                       2: no solution exists                                -}
{-                                                                            -}
{-                                                                            -}
{----------------------------------------------------------------------------}

procedure Partial_Pivoting(Dimen        : integer;
                           Coefficients : TNmatrix;
                           Constants    : TNvector;
                       var Solution     : TNvector;
                       var Error        : byte);

{----------------------------------------------------------------------------}
{-                                                                            -}
{-               Input: Dimen, Coefficients, Constants                        -}
{-              Output: Solution, Error                                       -}
{-                                                                            -}
{-             Purpose : Calculate the solution of a linear set of            -}
{-                       equations using Gaussian elimination, maximal        -}
{-                       pivoting and backwards substitution.                 -}
{-                                                                            -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real;            -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector         -}
{-                                                                            -}
{-    Global Variables : Dimen        : integer;  Dimen of the square        -}
{-                                                 matrix                     -}
{-                       Coefficients : TNmatrix; Square matrix               -}
{-                       Constants    : TNvector; Constants of each equation-}
{-                       Solution     : TNvector; Unique solution to the     -}
{-                                                 set of equations           -}
{-                       Error        : integer;  Flags if something goes    -}
{-                                                 wrong.                     -}
{-                                                                            -}
{-              Errors : 0: No errors;                                        -}
{-                       1: Dimen < 2                                         -}
{-                       2: no solution exists                                -}
{-                                                                            -}
{----------------------------------------------------------------------------}

procedure LU_Decompose(Dimen        : integer;
                       Coefficients : TNmatrix;
                   var Decomp       : TNmatrix;
                   var Permute      : TNmatrix;
                   var Error        : byte);

{----------------------------------------------------------------------------}
{-                                                                            -}
{-               Input: Dimen, Coefficients                                   -}
{-              Output: Decomp, Permute, Error                                -}
{-                                                                            -}
{-             Purpose : Decompose a square matrix into an upper              -}
{-                       triangular and lower triangular matrix such that     -}
{-                       the product of the two triangular matrices is        -}
{-                       the original matrix. This procedure also returns     -}
{-                       a permutation matrix which records the               -}
{-                       permutations resulting from partial pivoting.        -}
{-                                                                            -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real             -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector         -}
{-                                                                            -}
{-    Global Variables : Dimen        : integer;  Dimen of the coefficients -}
{-                                                 Matrix                     -}
{-                       Coefficients : TNmatrix; Coefficients matrix         -}
{-                       Decomp       : TNmatrix; Decomposition of            -}
{-                                                 Coefficients matrix        -}
{-                       Permute      : TNmatrix; Record of partial           -}
{-                                                 Pivoting                   -}
{-                       Error        : integer;  Flags if something goes    -}
{-                                                 wrong.                     -}
{-                                                                            -}
{-              Errors : 0: No errors;                                        -}
{-                       1: Dimen < 1                                         -}
{-                       2: No decomposition possible; singular matrix        -}
{-                                                                            -}
{----------------------------------------------------------------------------}

procedure LU_Solve(Dimen     : integer;
               var Decomp    : TNmatrix;
                   Constants : TNvector;
               var Permute   : TNmatrix;
               var Solution  : TNvector;
               var Error     : byte);
{----------------------------------------------------------------------------}
{-                                                                            -}
{-               Input: Dimen, Decomp, Constants, Permute                     -}
{-              Output: Solution, Error                                       -}
{-                                                                            -}
{-             Purpose : Calculate the solution of a linear set of            -}
{-                       equations using an LU decomposed matrix, a           -}
{-                       permutation matrix and backwards and forward         -}
{-                       substitution.                                        -}
{-                                                                            -}
{-  User_defined Types : TNvector = array[1..TNArraySize] of real             -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector         -}
{-                                                                            -}
{-    Global Variables : Dimen     : integer;  Dimen of the square            -}
{-                                              matrix                        -}
{-                       Decomp    : TNmatrix; Decomposition of               -}
{-                                              coefficient matrix            -}
{-                       Constants : TNvector; Constants of each equation     -}
```

```
{-                     Permute   : TNmatrix; Permutation matrix from     -}
{-                                           partial pivoting            -}
{-                     Solution  : TNvector; Unique solution to the      -}
{-                                           set of equations            -}
{-                     Error     : integer;  Flags if something goes     -}
{-                                           wrong.                      -}
{-                                                                       -}
{-              Errors : 0: No errors;                                   -}
{-                       1: Dimen < 1                                    -}
{-                                                                       -}
{-----------------------------------------------------------------------}

procedure Gauss_Seidel(Dimen        : integer;
                       Coefficients : TNmatrix;
                       Constants    : TNvector;
                       Tol          : Float;
                       MaxIter      : integer;
                   var Solution     : TNvector;
                   var Iter         : integer;
                   var Error        : byte);

{-----------------------------------------------------------------------}
{-                                                                       -}
{-               Input: Dimen, Coefficients, Constants, Tol, MaxIter     -}
{-              Output: Solution, Iter, Error                            -}
{-                                                                       -}
{-             Purpose : Calculate the solution of a linear set of       -}
{-                       equations using Gauss - Seidel iteration.       -}
{-                                                                       -}
{-  User-defined Types : TNvector = array[1..TNArraySize] of real        -}
{-                       TNmatrix = array[1..TNArraySize] of TNvector    -}
{-                                                                       -}
{-    Global Variables : Dimen : integer;        Dimen of the square     -}
{-                                               matrix                  -}
{-                       Coefficients : TNmatrix; Square matrix          -}
{-                       Constants    : TNvector; Constants of each equation-}
{-                       Tol          : real;     Tolerance in answer    -}
{-                       MaxIter      : integer;  Maximum number of      -}
{-                                                iterations allowed     -}
{-                       Solution     : TNvector; Unique solution to the -}
{-                                                set of equations       -}
{-                       Iter         : integer;  Number of iterations   -}
{-                       Error        : integer;  Flags if something goes -}
{-                                                wrong.                 -}
{-                                                                       -}
{-              Errors : 0: No errors;                                   -}
{-                       1: Iter >= MaxIter and matrix not               -}
{-                          diagonally dominant                          -}
{-                       2: Iter >= MaxIter                              -}
{-                       3: Dimen < 1                                    -}
{-                       4: Tol <= 0                                     -}
{-                       5: MaxIter < 0                                  -}
{-                       6: Zero on the diagonal of                      -}
{-                          the Coefficients matrix                      -}
{-                       7: Diverging                                    -}
{-                                                                       -}
{-                Note:  If the Gauss-Seidel iterative method is         -}
{-                       applied to an underdetermined system of equations -}
{-                       (i.e. one of the equations is a linear          -}
{-                       superposition of the others) it will converge   -}
{-                       to a (non-unique) solution. The Gauss-Seidel    -}
{-                       method is unable to distinguish between unique  -}
{-                       and non-unique solutions.                       -}
{-                       If you are concerned that  your equations       -}
{-                       may be underdetermined, solve them with         -}
{-                       Gaussian elimination (GAUSELIM.INC or           -}
{-                       PARTPIVT.INC                                    -}
{-                                                                       -}
{-----------------------------------------------------------------------}

implementation

{$I Matrix.inc}


end. { Matrix }
```

```
{----------------------------------------------------------------------------}
{-                                                                          -}
{-     Turbo Pascal Numerical Methods Toolbox                               -}
{-     Copyright (c) 1986, 87 by Borland International, Inc.                -}
{-                                                                          -}
{----------------------------------------------------------------------------}

procedure Determinant{(Dimen : integer;
                      Data   : TNmatrix;
                  var Det    : Float;
                  var Error  : byte)};

procedure Initial(Dimen : integer;
              var Data  : TNmatrix;
              var Det   : Float;
              var Error : byte);

{------------------------------------------------------}
{- This procedure tests for errors in the value of Dimen -}
{------------------------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      Det := Data[1, 1];
end; { procedure Initial }

procedure EROswitch(var Row1 : TNvector;
                    var Row2 : TNvector);

{-----------------------------------------------}
{- Elementary row operation - switching two rows -}
{-----------------------------------------------}

var
  DummyRow : TNvector;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector;
                 var ChangingRow  : TNvector);

{---------------------------------------------------------}
{- Row operation - adding a multiple of one row to another -}
{---------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier * ReferenceRow[Term];
end; { procedure EROmultAdd }

function Deter(Dimen : integer;
           var Data  : TNmatrix) : Float;

{----------------------------------------------------}
{- Input: Dimen, Data                               -}
{- Output: Deter                                    -}
{-                                                  -}
{- Function returns the determinant of the Data matrix -}
{----------------------------------------------------}

var
  PartialDeter, Multiplier : Float;
  Row, ReferenceRow : integer;
  DetEqualsZero : boolean;

procedure Pivot(Dimen         : integer;
                ReferenceRow  : integer;
            var Data          : TNmatrix;
            var PartialDeter  : Float;
            var DetEqualsZero : boolean);

{--------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, PartialDeter         -}
{- Output: Data, PartialDeter, DetEqualsZero              -}
{-                                                        -}
{- This procedure searches the ReferenceRow column of the -}
{- matrix Data for the first non-zero element below the   -}
{- diagonal. If it finds one, then the procedure switches -}
{- rows so that the non-zero element is on the diagonal.  -}
{- Switching rows changes the determinant by a factor of  -}
{- -1; this change is returned in PartialDeter.           -}
{- If it doesn't find one, the matrix is singular and the -}
{- Determinant is zero (DetEqualsZero = true is returned). -}
{--------------------------------------------------------}

var
  NewRow : integer;

begin
  DetEqualsZero := true;
  NewRow := ReferenceRow;
  while DetEqualsZero and (NewRow < Dimen) do  { Try to find a row  }
                                               { with a non-zero    }
                                               { element in this    }
                                               { column             }
  begin
    NewRow := Succ(NewRow);
```

```
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      DetEqualsZero := false;
      PartialDeter := -PartialDeter;  { Switching rows changes }
                                      { the determinant by a   }
                                      { factor of -1           }
    end;
  end;
end; { procedure Pivot }

begin  { function Deter }
  DetEqualsZero := false;
  PartialDeter := 1;
  ReferenceRow := 0;
  { Make the matrix upper triangular }
  while not(DetEqualsZero) and (ReferenceRow < Dimen - 1) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { If diagonal element is zero then switch rows }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, PartialDeter, DetEqualsZero);
    if not(DetEqualsZero) then
      for Row := ReferenceRow + 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if ABS(Data[Row, ReferenceRow]) > TNNearlyZero then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
                         Data[ReferenceRow, ReferenceRow];
          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
        end;
    { Multiply the diagonal Term into PartialDeter }
    PartialDeter := PartialDeter * Data[ReferenceRow, ReferenceRow];
  end;
  if DetEqualsZero then
    Deter := 0
  else
    Deter := PartialDeter * Data[Dimen, Dimen];
end; { function Deter }

begin { procedure Determinant }
  Initial(Dimen, Data, Det, Error);
  if Dimen > 1 then
    Det := Deter(Dimen, Data);
end; { procedure Determinant }

procedure Inverse{(Dimen : integer;
                  Data   : TNmatrix;
              var Inv    : TNmatrix;
              var Error  : byte)});


procedure Initial(Dimen : integer;
              var Data  : TNmatrix;
              var Inv   : TNmatrix;
              var Error : byte);

{-----------------------------------------------------------}
{- Input: Dimen, Data                                      -}
{- Output: Inv, Error                                      -}
{-                                                         -}
{- This procedure test for errors in the value of Dimen    -}
{-----------------------------------------------------------}

var
  Row : integer;

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    begin
      { First make the inverse-to-be the identity matrix }
      FillChar(Inv, SizeOf(Inv), 0);
      for Row := 1 to Dimen do
        Inv[Row, Row] := 1;
      if Dimen = 1 then
        if ABS(Data[1, 1]) < TNNearlyZero then
          Error := 2   { Singular matrix }
        else
          Inv[1, 1] := 1 / Data[1, 1];
    end;
end; { procedure Initial }

procedure EROdiv(Divisor : Float;
                 Dimen   : integer;
             var Row     : TNvector);

{-------------------------------------------------------}
{- Input: Divisor, Dimen, Row                          -}
{-                                                     -}
{- elementary row operation - dividing by a constant  -}
{-------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    Row[Term] := Row[Term] / Divisor;
end; { procedure EROdiv }

procedure EROswitch(var Row1 : TNvector;
                    var Row2 : TNvector);

{-----------------------------------------------}
{- Input: Row1, Row2                           -}
{- Output: Row1, Row2                          -}
```

```
{-                                              -}
{- Elementary row operation - switching two rows -}
{-----------------------------------------------}

var
  DummyRow : TNvector;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector;
                 var ChangingRow  : TNvector);

{---------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow     -}
{- Output: ChangingRow                                     -}
{-                                                         -}
{- Row operation - adding a multiple of one row to another -}
{---------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }


procedure Inver(Dimen : integer;
            var Data  : TNmatrix;
            var Inv   : TNmatrix;
            var Error : byte);

{--------------------------------------------------------}
{- Input: Dimen, Data                                    -}
{- Output: Inv, Error                                    -}
{-                                                       -}
{- This procedure computes the inverse of the matrix Data -}
{- and stores it in the matrix Inv.  If the matrix Data   -}
{- is singular, then Error = 2 is returned.               -}
{--------------------------------------------------------}

var
  Divisor, Multiplier : Float;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Data         : TNmatrix;
            var Inv          : TNmatrix;
            var Error        : byte);

{---------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, Inv                  -}
{- Output: Data, Inv, Error                               -}
{-                                                        -}
{- This procedure searches the ReferenceRow column of     -}
{- the Data matrix for the first non-zero element below   -}
{- the diagonal. If it finds one, then the procedure      -}
{- switches rows so that the non-zero element is on the   -}
{- diagonal. This same operation is applied to the Inv    -}
{- matrix. If no non-zero element exists in a column, the -}
{- matrix is singular and no inverse exists.              -}
{---------------------------------------------------------}

var
  NewRow : integer;

begin
  Error := 2;  { No inverse exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  { Try to find a       }
  { row with a non-zero }
  { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      EROswitch(Inv[NewRow], Inv[ReferenceRow]);
      Error := 0;
    end;
  end; { while }
end; { procedure Pivot }

begin { procedure Inver }
  { Make Data matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the diagonal element is zero }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, Inv, Error);
    if Error = 0 then
    begin
      Divisor := Data[ReferenceRow, ReferenceRow];
      EROdiv(Divisor, Dimen, Data[ReferenceRow]);
      EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
      for Row := 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if (Row <> ReferenceRow) and
```

```
          (ABS(Data[Row, ReferenceRow]) > TNNearlyZero) then
       begin
         Multiplier := -Data[Row, ReferenceRow] /
                        Data[ReferenceRow, ReferenceRow];
         EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
         EROmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
       end;
   end;
  end;
end; { procedure Inver }

begin { procedure Inverse }
  Initial(Dimen, Data, Inv, Error);
  if Dimen > 1 then
    Inver(Dimen, Data, Inv, Error);
end; { procedure Inverse }

procedure Gaussian_Elimination({(Dimen        : integer;
                               Coefficients : TNmatrix;
                               Constants    : TNvector;
                           var Solution     : TNvector;
                           var Error        : byte)});

procedure Initial(Dimen        : integer;
              var Coefficients : TNmatrix;
              var Constants    : TNvector;
              var Solution     : TNvector;
              var Error        : byte);

{--------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                -}
{- Output: Solution, Error                              -}
{-                                                      -}
{- This procedure test for errors in the value of Dimen. -}
{- This procedure also finds the solution for the       -}
{- trivial case Dimen = 1.                              -}
{--------------------------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      if ABS(Coefficients[1, 1]) < TNNearlyZero then
        Error := 2
      else
        Solution[1] := Constants[1] / Coefficients[1, 1];
end; { procedure Initial }

procedure EROswitch(var Row1 : TNvector;
                    var Row2 : TNvector);

{-----------------------------------------------}
{- Input: Row1, Row2                           -}
{- Output: Row1, Row2                          -}
{-                                             -}
{- elementary row operation - switching two rows -}
{-----------------------------------------------}

var
  DummyRow : TNvector;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector;
                 var ChangingRow  : TNvector);

{---------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow   -}
{- Output: ChangingRow                                   -}
{-                                                       -}
{- row operation - adding a multiple of one row to another -}
{---------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }

procedure UpperTriangular(Dimen        : integer;
                      var Coefficients : TNmatrix;
                      var Constants    : TNvector;
                      var Error        : byte);

{----------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                        -}
{- Output: Coefficients, Constants, Error                       -}
{-                                                              -}
{- This procedure makes the coefficient matrix upper triangular. -}
{- The operations which perform this are also performed on the  -}
{- Constants vector.                                            -}
{- If one of the main diagonal elements of the upper triangular -}
{- matrix is zero, then the Coefficients matrix is singular and -}
{- no solution exists (Error = 2 is returned).                  -}
{----------------------------------------------------------------}

var
  Multiplier       : Float;
  Row, ReferenceRow : integer;

```

```
procedure Pivot(Dimen          : integer;
                ReferenceRow : integer;
            var Coefficients : TNmatrix;
            var Constants    : TNvector;
            var Error        : byte);

{----------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Coefficients              -}
{- Output: Coefficients, Constants, Error                -}
{-                                                       -}
{- This procedure searches the ReferenceRow column of the -}
{- Coefficients matrix for the first non-zero element below -}
{- the diagonal. If it finds one, then the procedure switches -}
{- rows so that the non-zero element is on the diagonal.  -}
{- It also switches the corresponding elements in the     -}
{- Constants vector. If it doesn't find one, the matrix is -}
{- singular and no solution exists (Error = 2 is returned). -}
{----------------------------------------------------------}

var
  NewRow : integer;
  Dummy : Float;

begin
  Error := 2;          { No solution exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do   { Try to find a      }
                                              { row with a non-zero }
                                              { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Coefficients[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Coefficients[NewRow], Coefficients[ReferenceRow]);
      { Switch these two rows }
      Dummy := Constants[NewRow];
      Constants[NewRow] := Constants[ReferenceRow];
      Constants[ReferenceRow] := Dummy;
      Error := 0;    { Solution may exist }
    end;
  end;
end; { procedure Pivot }

begin { procedure UpperTriangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen - 1) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the main diagonal element is zero }
    if ABS(Coefficients[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Coefficients, Constants, Error);
    if Error = 0 then
      for Row := ReferenceRow + 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if ABS(Coefficients[Row, ReferenceRow]) > TNNearlyZero then
        begin
          Multiplier := -Coefficients[Row, ReferenceRow] /
                         Coefficients[ReferenceRow,ReferenceRow];
          EROmultAdd(Multiplier, Dimen,
                     Coefficients[ReferenceRow], Coefficients[Row]);
          Constants[Row] := Constants[Row] +
                            Multiplier * Constants[ReferenceRow];
        end;
  end; { while }
  if ABS(Coefficients[Dimen, Dimen]) < TNNearlyZero then
    Error := 2;    { No solution }
end; { procedure UpperTriangular }

procedure BackwardsSub(Dimen        : integer;
                   var Coefficients : TNmatrix;
                   var Constants    : TNvector;
                   var Solution     : TNvector);

{----------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                 -}
{- Output: Solution                                      -}
{-                                                       -}
{- This procedure applies backwards substitution to the upper -}
{- triangular Coefficients matrix and Constants vector. The -}
{- resulting vector is the solution to the set of equations and -}
{- is returned in the vector Solution.                   -}
{----------------------------------------------------------}

var
  Term, Row : integer;
  Sum : Float;

begin
  Term := Dimen;
  while Term >= 1 do
  begin
    Sum := 0;
    for Row := Term + 1 to Dimen do
      Sum := Sum + Coefficients[Term, Row] * Solution[Row];
    Solution[Term] := (Constants[Term] - Sum) / Coefficients[Term, Term];
    Term := Pred(Term);
  end;
end; { procedure BackwardsSub }

begin { procedure Gaussian_Elimination }
  Initial(Dimen, Coefficients, Constants, Solution, Error);
  if Dimen > 1 then
  begin
    UpperTriangular(Dimen, Coefficients, Constants, Error);
    if Error = 0 then
      BackwardsSub(Dimen, Coefficients, Constants, Solution);
  end;
end; { procedure Gaussian_Elimination }

procedure Partial_Pivoting((Dimen       : integer;
```

```
                        Coefficients : TNmatrix;
                        Constants    : TNvector;
                    var Solution     : TNvector;
                    var Error        : byte));


procedure Initial(Dimen        : integer;
              var Coefficients : TNmatrix;
              var Constants    : TNvector;
              var Solution     : TNvector;
              var Error        : byte);

{----------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                 -}
{- Output: Solution, Error                               -}
{-                                                       -}
{- This procedure test for errors in the value of Dimen. -}
{- This procedure also finds the solution for the        -}
{- trivial case Dimen = 1.                               -}
{----------------------------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1
  else
    if Dimen = 1 then
      if ABS(Coefficients[1, 1]) < TNNearlyZero then
        Error := 2
      else
        Solution[1] := Constants[1] / Coefficients[1, 1];
end; { procedure Initial }

procedure EROswitch(var Row1 : TNvector;
                    var Row2 : TNvector);

{-------------------------------------------------}
{- Input: Row1, Row2                             -}
{- Output: Row1, Row2                            -}
{-                                               -}
{- elementary row operation - switching two rows -}
{-------------------------------------------------}

var
  DummyRow : TNvector;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector;
                 var ChangingRow  : TNvector);

{--------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow  -}
{- Output: ChangingRow                                  -}
{-                                                      -}
{- Row operation - adding a multiple of one row to another -}
{--------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }

procedure UpperTriangular(Dimen        : integer;
                      var Coefficients : TNmatrix;
                      var Constants    : TNvector;
                      var Error        : byte);

{-------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                     -}
{- Output: Coefficients, Constants, Error                    -}
{-                                                           -}
{- This procedure makes the coefficient matrix upper triangular. -}
{- The operations which perform this are also performed on the   -}
{- Constants vector.                                          -}
{- If one of the main diagonal elements of the upper triangular -}
{- matrix is zero, then the Coefficients matrix is singular and -}
{- no solution exists (Error = 2 is returned).                -}
{-------------------------------------------------------------}

var
  Multiplier : Float;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Coefficients : TNmatrix;
            var Constants    : TNvector;
            var Error        : byte);

{-------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Coefficients                  -}
{- Output: Coefficients, Constants, Error                    -}
{-                                                           -}
{- This procedure searches the ReferenceRow column of the    -}
{- Coefficients matrix for the largest non-zero element below -}
{- the diagonal. If it finds one, then the procedure switches -}
{- rows so that the largest non-zero element is on the       -}
{- diagonal. It also switches the corresponding elements in  -}
{- the Constants vector. If it doesn't find a non-zero element, -}
{- the matrix is singular and no solution exists             -}
```

```
{- (Error = 2 is returned).                                    -}
{--------------------------------------------------------------}

var
  PivotRow, Row : integer;
  Dummy : Float;

begin
  { First, find the row with the largest element }
  PivotRow := ReferenceRow;
  for Row := ReferenceRow + 1 to Dimen do
    if ABS(Coefficients[Row, ReferenceRow]) >
       ABS(Coefficients[PivotRow, ReferenceRow]) then
      PivotRow := Row;
  if PivotRow <> ReferenceRow then
    { Second, switch these two rows }
    begin
      EROswitch(Coefficients[PivotRow], Coefficients[ReferenceRow]);
      Dummy := Constants[PivotRow];
      Constants[PivotRow] := Constants[ReferenceRow];
      Constants[ReferenceRow] := Dummy;
    end
  else { If the diagonal element is zero, no solution exists }
    if ABS(Coefficients[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Error := 2;     { No solution }
end; { procedure Pivot }

begin { procedure UpperTriangular }
  { Make Coefficients matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen - 1) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Find row with largest element in this column }
    { and switch this row with the ReferenceRow    }
    Pivot(Dimen, ReferenceRow, Coefficients, Constants, Error);
    if Error = 0 then
      for Row := ReferenceRow + 1 to Dimen do
        { Make the ReferenceRow element of these rows zero }
        if ABS(Coefficients[Row, ReferenceRow]) > TNNearlyZero then
        begin
          Multiplier := -Coefficients[Row, ReferenceRow] /
                         Coefficients[ReferenceRow,ReferenceRow];
          EROmultAdd(Multiplier, Dimen,
                     Coefficients[ReferenceRow], Coefficients[Row]);
          Constants[Row] := Constants[Row] +
                            Multiplier * Constants[ReferenceRow];
        end;
  end;
  if ABS(Coefficients[Dimen, Dimen]) < TNNearlyZero then
    Error := 2;    { No solution }
end; { procedure UpperTriangular }

procedure BackwardsSub(Dimen        : integer;
                   var Coefficients : TNmatrix;
                   var Constants    : TNvector;
                   var Solution     : TNvector);

{--------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                      -}
{- Output: Solution                                           -}
{-                                                            -}
{- This procedure applies backwards substitution to the upper -}
{- triangular Coefficients matrix and Constants vector. The   -}
{- resulting vector is the solution to the set of equations and -}
{- is stored in the vector Solution.                          -}
{--------------------------------------------------------------}

var
  Term, Row : integer;
  Sum : Float;

begin
  Term := Dimen;
  while Term >= 1 do
  begin
    Sum := 0;
    for Row := Term + 1 to Dimen do
      Sum := Sum + Coefficients[Term, Row] * Solution[Row];
    Solution[Term] := (Constants[Term] - Sum) / Coefficients[Term, Term];
    Term := Pred(Term);
  end;
end; { procedure BackwardsSub }

begin  { procedure Partial_Pivoting }
  Initial(Dimen, Coefficients, Constants, Solution, Error);
  if Dimen > 1 then
  begin
    UpperTriangular(Dimen, Coefficients, Constants, Error);
    if Error = 0 then
      BackwardsSub(Dimen, Coefficients, Constants, Solution);
  end;
end; { procedure Partial_Pivoting }

procedure LU_Decompose{(Dimen       : integer;
                       Coefficients : TNmatrix;
                   var Decomp       : TNmatrix;
                   var Permute      : TNmatrix;
                   var Error        : byte)};


procedure TestInput(Dimen : integer;
                var Error : byte);

{-------------------------------------}
{- Input: Dimen                      -}
{- Output: Error                     -}
{-                                   -}
{- This procedure checks to see if the -}
{- value of Dimen is greater than 1.   -}
```

```
{---------------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 1;
end; { procedure TestInput }

function RowColumnMult(Row    : integer;
                   var Lower  : TNmatrix;
                       Column : integer;
                   var Upper  : TNmatrix) : Float;

{--------------------------------------------------}
{- Input: Row, Lower, Column, Upper               -}
{- Function return: dot product of row Row of Lower -}
{-                  and column Column of Upper     -}
{--------------------------------------------------}

var
  Term : integer;
  Sum : Float;

begin
  Sum := 0;
  for Term := 1 to Row - 1 do
    Sum := Sum + Lower[Row, Term] * Upper[Term, Column];
  RowColumnMult := Sum;
end; { function RowColumnMult }


procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Coefficients : TNmatrix;
            var Lower        : TNmatrix;
            var Upper        : TNmatrix;
            var Permute      : TNmatrix;
            var Error        : byte);

{-------------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Coefficients,                 -}
{-        Lower, Upper, Permute                              -}
{- Output: Coefficients, Lower, Permute, Error               -}
{-                                                           -}
{- This procedure searches the ReferenceRow column of the    -}
{- Coefficients matrix for the element in the Row below the  -}
{- main diagonal which produces the largest value of         -}
{-                                                           -}
{-         Coefficients[Row, ReferenceRow] -                 -}
{-                                                           -}
{-          Sum K=1 to ReferenceRow - 1 of                   -}
{-                  Upper[Row, k] - Lower[k, ReferenceRow]   -}
{-                                                           -}
{- If it finds one, then the procedure switches              -}
{- rows so that this element is on the main diagonal. The    -}
{- procedure also switches the corresponding elements in the    -}
{- Permute matrix and the Lower matrix. If the largest value of -}
{- the above expression is zero, then the matrix is singular    -}
{- and no solution exists (Error = 2 is returned).              -}
{---------------------------------------------------------------}

var
  PivotRow, Row : integer;
  ColumnMax, TestMax : Float;

procedure EROswitch(var Row1 : TNvector;
                    var Row2 : TNvector);

{-----------------------------------------------}
{- Input: Row1, Row2                           -}
{- Output: Row1, Row2                          -}
{-                                             -}
{- elementary row operation - switching two rows -}
{-----------------------------------------------}

var
  DummyRow : TNvector;

begin
  DummyRow := Row1;
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

begin { procedure Pivot }
  { First, find the row with the largest TestMax }
  PivotRow := ReferenceRow;
  ColumnMax := ABS(Coefficients[ReferenceRow, ReferenceRow] -
               RowColumnMult(ReferenceRow, Lower, ReferenceRow, Upper));
  for Row := ReferenceRow + 1 to Dimen do
  begin
    TestMax := ABS(Coefficients[Row, ReferenceRow] -
               RowColumnMult(Row, Lower, ReferenceRow, Upper));

    if TestMax > ColumnMax then
    begin
      PivotRow := Row;
      ColumnMax := TestMax;
    end;
  end;

  if PivotRow <> ReferenceRow then
    { Second, switch these two rows }
    begin
      EROswitch(Coefficients[PivotRow], Coefficients[ReferenceRow]);
      EROswitch(Lower[PivotRow], Lower[ReferenceRow]);
      EROswitch(Permute[PivotRow], Permute[ReferenceRow]);
    end
  else
    { If ColumnMax is zero, no solution exists }
```

```
   if ColumnMax < TNNearlyZero then
     Error := 2;     { No solution exists }
end; { procedure Pivot }

procedure Decompose(Dimen        : integer;
                var Coefficients : TNmatrix;
                var Decomp       : TNmatrix;
                var Permute      : TNmatrix;
                var Error        : byte);

{------------------------------------------------------}
{- Input: Dimen, Coefficients                         -}
{- Output: Decomp, Permute, Error                     -}
{-                                                    -}
{- This procedure decomposes the Coefficients matrix  -}
{- into two triangular matrices, a lower and an upper -}
{- one.  The lower and upper matrices are combined    -}
{- into one matrix, Decomp.  The permutation matrix,  -}
{- Permute, records the effects of partial pivoting.  -}
{------------------------------------------------------}

var
  Upper, Lower : TNmatrix;
  Term, Index : integer;

procedure Initialize(Dimen   : integer;
                 var Lower   : TNmatrix;
                 var Upper   : TNmatrix;
                 var Permute : TNmatrix);

{-------------------------------------------------}
{- Output: Dimen, Lower, Upper, Permute          -}
{-                                               -}
{- This procedure initializes the above variables. -}
{- Lower and Upper are initialized to the zero   -}
{- matrix and Diag is initialized to the identity -}
{- matrix.                                       -}
{-------------------------------------------------}

var
  Diag : integer;

begin
  FillChar(Upper, SizeOf(Upper), 0);
  FillChar(Lower, SizeOf(Lower), 0);
  FillChar(Permute, SizeOf(Permute), 0);
  for Diag := 1 to Dimen do
    Permute[Diag, Diag] := 1;
end; { procedure Initialize }

begin { procedure Decompose }
  Initialize(Dimen, Lower, Upper, Permute);

  {  partial pivoting on row 1 }
  Pivot(Dimen, 1, Coefficients, Lower, Upper, Permute, Error);
  if Error = 0 then
  begin
    Lower[1, 1] := 1;
    Upper[1, 1] := Coefficients[1, 1];

    for Term := 1 to Dimen do
    begin
      Lower[Term, 1] := Coefficients[Term, 1] / Upper[1, 1];
      Upper[1, Term] := Coefficients[1, Term] / Lower[1, 1];
    end;
  end;

  Term := 1;
  while (Error = 0) and (Term < Dimen - 1) do
  begin
    Term := Succ(Term);

    { perform partial pivoting on row Term }
    Pivot(Dimen, Term, Coefficients, Lower, Upper, Permute, Error);

    Lower[Term, Term] := 1;
    Upper[Term, Term] := Coefficients[Term, Term] -
                         RowColumnMult(Term, Lower, Term, Upper);

    if ABS(Upper[Term, Term]) < TNNearlyZero then
      Error := 2   { no solutions }
    else
      for Index := Term + 1 to Dimen do
      begin
        Upper[Term, Index] := Coefficients[Term, Index] -
                              RowColumnMult(Term, Lower, Index, Upper);
        Lower[Index, Term] := (Coefficients[Index, Term] -
                              RowColumnMult(Index, Lower, Term, Upper)) /
                              Upper[Term, Term];
      end;
  end;

  Lower[Dimen, Dimen] := 1;
  Upper[Dimen, Dimen] := Coefficients[Dimen, Dimen] -
                         RowColumnMult(Dimen, Lower, Dimen, Upper);
  if ABS(Upper[Dimen, Dimen]) < TNNearlyZero then
    Error := 2;
  { Combine the upper and lower triangular matrices into one }
  Decomp := Upper;

  for Term := 2 to Dimen do
    for Index := 1 to Term - 1 do
      Decomp[Term, Index] := Lower[Term, Index];
end; { procedure Decompose }

begin { procedure LU_Decompose }
  TestInput(Dimen, Error);
  if Error = 0 then
    if Dimen = 1 then
      begin
```

```
      Decomp := Coefficients;
      Permute[1, 1] := 1;
    end
  else
    Decompose(Dimen, Coefficients, Decomp, Permute, Error);
end; { procedure LU_Decompose }

procedure LU_Solve({Dimen     : integer;
               var Decomp    : TNmatrix;
                   Constants : TNvector;
               var Permute   : TNmatrix;
               var Solution  : TNvector;
               var Error     : byte)};

procedure Initial(Dimen    : integer;
              var Solution : TNvector;
              var Error    : byte);

{----------------------------------------------------}
{- Input: Dimen                                     -}
{- Output: Solution, Error                          -}
{-                                                  -}
{- This procedure initializes the Solution vector.  -}
{- It also checks to see if the value of Dimen is   -}
{- greater than 1.                                  -}
{----------------------------------------------------}

begin
  Error := 0;
  FillChar(Solution, SizeOf(Solution), 0);
  if Dimen < 1 then
    Error := 1;
end; { procedure Initial }

procedure FindSolution(Dimen     : integer;
                   var Decomp    : TNmatrix;
                   var Constants : TNvector;
                   var Solution  : TNvector);

{-------------------------------------------------------------}
{- Input: Dimen, Decomp, Constants                           -}
{- Output: Solution                                          -}
{-                                                           -}
{- The Decom matrix contains a lower and upper triangular    -}
{- matrix.                                                   -}
{- This procedure performs a two step backwards substitution -}
{- to compute the solution to the system of equations.  First, -}
{- forward substitution is applied to the lower triangular   -}
{- matrix and Constants vector yielding PartialSolution.  Then -}
{- backwards substitution is applied to the Upper matrix and -}
{- the PartialSolution vector yielding Solution.             -}
{-------------------------------------------------------------}

var
  PartialSolution : TNvector;
  Term, Index : integer;
  Sum : Float;

begin { procedure FindSolution }
  { First solve the lower triangular matrix }
  PartialSolution[1] := Constants[1];
  for Term := 2 to Dimen do
  begin
    Sum := 0;
    for Index := 1 to Term - 1 do
      if Term = Index then
        Sum := Sum + PartialSolution[Index]
      else
        Sum := Sum + Decomp[Term, Index] * PartialSolution[Index];
    PartialSolution[Term] := Constants[Term] - Sum;
  end;
  { Then solve the upper triangular matrix }
  Solution[Dimen] := PartialSolution[Dimen] / Decomp[Dimen, Dimen];
  for Term := Dimen - 1 downto 1 do
  begin
    Sum := 0;
    for Index := Term + 1 to Dimen do
      Sum := Sum + Decomp[Term, Index] * Solution[Index];
    Solution[Term] := (PartialSolution[Term] - Sum)/Decomp[Term, Term];
  end;
end; { procedure FindSolution }

procedure PermuteConstants(Dimen     : integer;
                       var Permute   : TNmatrix;
                       var Constants : TNvector);

var
  Row, Column : integer;
  Entry : Float;
  TempConstants : TNvector;

begin
  for Row := 1 to Dimen do
  begin
    Entry := 0;
    for Column := 1 to Dimen do
      Entry := Entry + Permute[Row, Column] * Constants[Column];
    TempConstants[Row] := Entry;
  end;
  Constants := TempConstants;
end; { procedure PermuteConstants }

begin { procedure Solve_LU_Decompostion }
  Initial(Dimen, Solution, Error);
  if Error = 0 then
    PermuteConstants(Dimen, Permute, Constants);
  FindSolution(Dimen, Decomp, Constants, Solution);
end; { procedure LU_Solve }

procedure Gauss_Seidel({Dimen      : integer;
```

```
                         Coefficients : TNmatrix;
                         Constants    : TNvector;
                         Tol          : Float;
                         MaxIter      : integer;
                     var Solution     : TNvector;
                     var Iter         : integer;
                     var Error        : byte)};


var
  Guess : TNvector;

procedure TestInput(Dimen        : integer;
                    Tol          : Float;
                    MaxIter      : integer;
                var Coefficients : TNmatrix;
                var Constants    : TNvector;
                var Solution     : TNvector;
                var Error        : byte);

{----------------------------------}
{- Input: Dimen, Tol, MaxIter     -}
{-        Coefficients,           -}
{-        Constants               -}
{- Output: Solution, Error        -}
{-                                -}
{- test the input data for errors -}
{- The procedure also finds the   -}
{- solution for the trivial case  -}
{- Dimen = 0.                     -}
{----------------------------------}

begin
  Error := 0;
  if Dimen < 1 then
    Error := 3
  else
    if Tol <= 0 then
      Error := 4
     else
       if MaxIter < 0 then
         Error := 5;
  if (Error = 0) and (Dimen = 1) then
  begin
    if ABS(Coefficients[1, 1]) < TNNearlyZero then
      Error := 6
    else
      Solution[1] := Constants[1] / Coefficients[1, 1];
  end;
end; { procedure TestInput }

procedure TestForDiagDominance(Dimen        : integer;
                           var Coefficients : TNmatrix;
                           var Error        : byte);

{----------------------------------------------------------------}
{- Input: Dimen, Coefficients                                   -}
{- Output: Error                                                -}
{-                                                              -}
{- This procedure examines the Coefficients matrix to see if it is -}
{- diagonally dominant. If it is, then the Gauss-Seidel iterative  -}
{- method will converge to a solution of this system of equations; -}
{- if not, then convergence may not be possible with this method   -}
{- and Error = 1 (which is a warning) is returned. If one of the   -}
{- elements on the main diagonal of the Coefficients matrix is     -}
{- zero, then the matrix is singular and cannot be solved and      -}
{- Error = 6 is returned. In such a case, one of the direct        -}
{- methods for solving systems of equations (e.g. Gaussian         -}
{- elimination) should be used.                                     -}
{----------------------------------------------------------------}

var
  Row, Column : integer;
  Sum : Float;

begin
  Row := 0;
  while (Row < Dimen) and (Error < 2) do
  begin
    Row := Succ(Row);
    Sum := 0;
    for Column := 1 to Dimen do
      if Column <> Row then
        Sum := Sum + ABS(Coefficients[Row, Column]);
    if Sum > ABS(Coefficients[Row, Row]) then
      Error := 1;  { WARNING! convergence may not be }
                   { possible because matrix isn't   }
                   { diagonally dominant             }
    if ABS(Coefficients[Row, Row]) < TNNearlyZero then
      Error := 6;  { Singular matrix - can't be solved  }
                   { by the Gauss-Seidel method.        }
  end; { while }
end; { procedure TestForDiagDominance }

procedure MakeInitialGuess(Dimen        : integer;
                       var Coefficients : TNmatrix;
                       var Constants    : TNvector;
                       var Guess        : TNvector);

{----------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants                        -}
{- Output: Guess                                                -}
{-                                                              -}
{- This procedure creates an initial approximation to the solution -}
{- by dividing the Constants terms by the corresponding terms     -}
{- on the main diagonal of the Coefficients matrix.               -}
{----------------------------------------------------------------}

var
  Term : integer;
```

```
begin
  FillChar(Guess, SizeOf(Guess), 0);
  for Term := 1 to Dimen do
    if ABS(Coefficients[Term, Term]) > TNNearlyZero then
      Guess[Term] := Constants[Term] / Coefficients[Term, Term];
end; { procedure MakeInitialGuess }

procedure TestForConvergence(Dimen     : integer;
                         var OldApprox : TNvector;
                         var NewApprox : TNvector;
                             Tol       : Float;
                         var Done      : boolean;
                         var Product   : Float;
                         var Error     : byte);

{---------------------------------------------------------------------}
{- Input: Dimen, OldApprox, NewApprox, Tol, Product                  -}
{- Output: Done, Product, Error                                      -}
{-                                                                   -}
{- This procedure determines if the sequence of approximations       -}
{- has converged. For convergence to occur, the relative difference  -}
{- between each Term of OldApprox and NewApprox must be less than    -}
{- the tolerance, Tol. If so, Done = TRUE is returned.               -}
{-                                                                   -}
{- This procedure also determines if the sequence of approximations  -}
{- is diverging. Product records the total fractional change from    -}
{- the initial guess to the current iteration.  If Product is greater -}
{- than 1E20, then the sequence is assumed to have diverged. If so,  -}
{- Error = 7 is returned.                                            -}
{---------------------------------------------------------------------}

var
  Term : integer;
  PartProd : Float;

begin
  Done := true;
  PartProd := 0;
  for Term := 1 to Dimen do
  begin
    if ABS(OldApprox[Term] - NewApprox[Term]) > ABS(NewApprox[Term] * Tol) then
      Done := false;
    if (ABS(OldApprox[Term]) > TNNearlyZero) and (Error = 1) then
      { This is part of the divergence test }
      PartProd := PartProd + ABS(NewApprox[Term] / OldApprox[Term]);
  end;
  Product := Product * PartProd / Dimen;
  if Product > 1E20 then
    Error := 7   { Sequence is diverging }
end; { procedure TestForConvergence }

procedure Iterate(Dimen        : integer;
              var Coefficients : TNmatrix;
              var Constants    : TNvector;
              var Guess        : TNvector;
                  Tol          : Float;
                  MaxIter      : integer;
              var Solution     : TNvector;
              var Iter         : integer;
              var Error        : byte);

{---------------------------------------------------------------}
{- Input: Dimen, Coefficients, Constants, Guess, Tol, MaxIter  -}
{- Output: Solution, Iter, Error                               -}
{-                                                             -}
{- This procedure performs the Gauss-Seidel iteration and      -}
{- returns either an error or the approximated solution and    -}
{- the number of iterations.                                   -}
{---------------------------------------------------------------}

var
  Done : boolean;
  OldApprox, NewApprox : TNvector;
  Term, Loop : integer;
  FirstSum, SecondSum, Product : Float;

begin { procedure Iterate }
  Product := 1;
  Done := false;
  Iter := 0;
  NewApprox := Guess;
  OldApprox := Guess;
  while (Iter < MaxIter) and not(Done) and (Error <= 1) do
  begin
    Iter := Succ(Iter);
    for Term := 1 to Dimen do
    begin
      FirstSum := 0;
      SecondSum := 0;
      for Loop := 1 to Term - 1 do
        FirstSum := FirstSum + Coefficients[Term, Loop] * NewApprox[Loop];
        for Loop := Term + 1 to Dimen do
          SecondSum := SecondSum + Coefficients[Term, Loop] * OldApprox[Loop];
      NewApprox[Term] := (Constants[Term] - FirstSum - SecondSum) /
                                   Coefficients[Term, Term];
    end;
    TestForConvergence(Dimen, OldApprox, NewApprox, Tol, Done, Product, Error);
    OldApprox := NewApprox;
  end; { while }
  if (Iter < MaxIter) and (Error = 1) then
    Error := 0;  { The sequence converged, }
                 {  disregard the warning  }
  if (Iter >= MaxIter) and (Error = 1) then
    Error := 1;  { Matrix is not diagonally dominant; }
                 { convergence is probably impossible }
  if (Iter >= MaxIter) and (Error = 0) then
     Error := 2; { Convergence IS possible;   }
                 { more iterations are needed }
  Solution := NewApprox;
```

```
end; { procedure Iterate }

begin  { procedure Gauss_Seidel }
  TestInput(Dimen, Tol, MaxIter, Coefficients, Constants, Solution, Error);
  if Dimen > 1 then
  begin
    TestForDiagDominance(Dimen, Coefficients, Error);
    if Error < 2 then
    begin
      MakeInitialGuess(Dimen, Coefficients, Constants, Guess);
      Iterate(Dimen, Coefficients, Constants, Guess, Tol,
              MaxIter, Solution, Iter, Error);
    end;
  end;
end; { procedure Gauss_Seidel }
```

```
Unit GPSEngine;
{************************************************************************}
{ This unit provides procedures to be used by the GPS unit. The procedures
  are taylor made for the GPSengine from Magnavox.}
{************************************************************************}
Interface
{$N+,E+}
Uses MIASglob, GPSglob, crt;
Procedure InitGPSrec( Var error: Boolean);
{************************************************************************}
{Initialise the GPS receiver. If something went wrong, error := True.
                              Input :-
                              Output:error}
{************************************************************************}
Procedure CollectGPSrec( Var GPSint: GPSinttype);
{************************************************************************}
{get characters from buffers. Synchronise with the delimiters 'LF' and '$'.
 Check for correct header and process the information to the variable GPSint
                              Input :-
                              Output:GPSint}
{************************************************************************}
Procedure ExecGPSrecCommand( command: commandtype);
{************************************************************************}
{ Convert a GPScommand, to a commandstring for the GPS Engine and send
  the command using the IWRITECOM routine in COMM_.int. The commands
  here start with 'GPS:'.
                               Input : command
                               Output: commandstring on comport}
{************************************************************************}
Procedure CloseGPSrec;
{************************************************************************}
{Make sure the GPS receiver is back to normal. Restore interuptvectors etc
                              Input :-
                              Output:-}
{************************************************************************}
Implementation
Uses User, Miscell, com_4;{comdisc}
Const
    LF          =     #10;{ linefeed}
Var
   port0,
   port1        :       Byte;   { contains comportnr for port0 and port1
                                  of the GPSengine}
   x, y         :       Shortint; {counter}
   two2power    :       Array[ 0..55] Of Double;
   sv_id        :       Byte;   { contains the satellite identity number
                                  for ephemeris transmissions}
   Tsv_id,                      { Time when sv_id was read}
   Valid_Tsv_id :       timetype;{Time that sv_id should be valid}
   Tpr,                         { Time when first PR from a set was received}
   Valid_Tpr    :       timetype;{ time that a set is valid, for timeout}
   old_user_ms  :       Real;   { remember old time of measurement}
   svcount      :       Shortint;{ count the number of PR yet received}
   tempGPSint   :       GPSinttype;{ internal var containing all info received}
   completeinfo :       Byte;   { contains info for transfer of completeinfo}
                                { or partial info if available}
                                { 0 for partial, 1 for complete}
{--------------- start procedure initgpsrec------------------}
Procedure InitGPSrec( Var error: Boolean);
Var
   title,
   line         :       String;
   code         :       Integer;
   varname,
   value        :       String;
   setupfile    :       Text;
Begin
                                                   { set default values}
    port1       :=      1;                         { Engine port 1 is com 1}
    port0       :=      2;                         { Engine port 0 is com 2}
    error:= True;
    OpenConfigRead( setupfile, MIAScfgname);
    Repeat                                          { find GPSENGINE part of
                                                      config file}
          Readln( setupfile, title);
    Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPSENGINE'));
    If Not Eof( setupfile)                          { if there is more in file}
    Then Repeat
               Readln( setupfile, line);            { get a line}
               Convert( line, varname, value);      { extract the variable name
                                                      and value}
               If (varname = 'PORT0')
               Then Val( value, port0, code);
               If ( varname = 'PORT1')
               Then Val( value, port1, code);
```

```
              If ( varname = 'COMPLETEINFO')
              Then Val( value, completeinfo, code);
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
                                             { repeat until end of file}
     CloseConfig( setupfile);

                                             { initialise comports for
                                               communication with Engine}
     Setupcomport( port0, Ord( B4800), 8, Ord( None), 1);
     Setupcomport( port1, Ord( B9600), 8, Ord( None), 1);
                                             { empty receive and trans-
                                               mit buffers}
     Emptybuffer( port0, True);
     Emptybuffer( port1, True);
                                             { set interrupt vectors}
     Installint( port0);                     { save old interrupt}
     Installint( port1);                     { vectors}

     error:= False;
End;
{--------------- end procedure initgpsrec------------------}

{--------------- start procedure CollectGPSrec-------------------}
Procedure CollectGPSrec( Var GPSint: GPSinttype);

  {--------------- start included procedures CollectGPSrec-----------}
  Procedure Collectport0( Var GPSint: GPSinttype);
  {******************************************************************}
  { This procedure collects data from port0 of the GPS ENGINE receiver.
                            Input : data from receive buffer
                            Output: updated GPSint.}
  {******************************************************************}

  {----------------- start included procedures collectport0---------}
     Procedure StatusReport( rec: String; Var GPSint: gpsinttype);
     {****************************************************************}
     { This procedure reads the number of satellites being tracked}
     {                      Input : string from GPSengine
                            Output: number of satellites being tracked}
     {****************************************************************}

     Var
        value  :      String;
        code   :      Integer;

     Begin
          value:= Copy( rec, 12, Length( rec) -11);{ select everything}
                                                   { but the header}
          For x:= 1 To 2 Do
          Begin
              While value[1] <> ',' Do
                     value:= Copy( value, 2, Length( value) -1);
              value:= Copy( value, 2, Length( value) - 1);
          End;

          value:= Copy( value, 1, 1);
          Val( value, GPSint.numofsat, code);

          If code <> 0
          Then GPSint.numofsat := -1;
     End;
  {------------- end included procedures collectport0---------}

  Var
     rec,
     header              :      String;
     recnum,
     code                :      Integer;
     time                :      timetype;
     part                :      String;

  {-----------------------Start collectport0-----------------------}
  Begin                                        { begin collectport0}
    GPSint:= tempGPSint;

    If (charsinbuff( port0) > 80)              { wait until string long}
    Then Begin                                 { enough to be valid}
              rec:= Getcharbuff( port0);       { get on char from buffer}
              If ( rec = '$')                  { if begin of recordstring}
              Then Begin
                        Repeat                 { read chars from buffer}
                               rec:= rec + Getcharbuff( port0);
                        Until ( Pos( LF, rec) <> 0) Or
                               ( charsinbuff( port0) = 0);
                                               { until a line feed found}
                                               { or no more chars in buff}
                        If ( charsinbuff( port0) = 0)
                        Then Exit;             { if no lf found, exit}

                                               { first part of string must}
                                               { be $PMVSG for Engine}
                        header:= Copy( rec, 1, 7);
                        If ( header = '$PMVXG,')
                        Then Begin             { next part is recordnumber}
gotoxy( 1, 15);
write( rec);
SaveEquipmentMessage( 'RG: ' + rec);
                             End
                        Else;                  { what if $PMVXG not found?}
                   End
              Else While (( rec <> LF) And
                          ( Charsinbuff( port0) > 0)) Do
                                               { if not begin of record}
                                               { read buffer until lf}
                          rec:= getcharbuff( port0);
         End;
    tempGPSint:= GPSint;
  End;                                         {end collectport0}
  {-----------------------end collectport0-----------------------}
```

```
Procedure Collectport1( Var GPSint: GPSinttype);
{*******************************************************************}
{ This procedure collects data from port1 of the GPS ENGINE receiver.
                          Input : data from receive buffer
                          Output: updated GPSint.}
{*******************************************************************}

Type
    temptype        =       Array[1..24] Of Byte;

{--------------- start included procedures collectport1--------}
   Procedure Conv_ASCII_2_Val( rec: String; Var temp: temptype;
                               Var error: Boolean);
   {*******************************************************************}
   { This procedure convert a string with pairs of ascii characters
     to byte values. This string should contain 24 pairs of ascii char-
     acters with the 'values' '0'..'9' or 'A'..'F'. The first pair should
     begin at position 6 of the string. When the data is erronous, then
     the error flag is set True.
                          Input : received string
                          Output: array with 24 bytes
                                  error when flag}
   {*******************************************************************}

   Var
      x, y          :       Integer;
      value         :       Byte;
      dum           :       Char;
      code          :       Integer;

   Begin
        error:= False;                        { initialise errorflag}

        For x:= 0 To 23 Do                    { count 24 sets of }
        Begin                                 { characters}
          value:=0;                           { reset value}
          temp[ x+1] := 0;                    { reset almanac}

          For y:= 1 DownTo 0 Do               { convert two characters to}
          Begin                               { value}
             dum:= rec[6 + x * 3 + (1 - y)];  { get digit starting at }
                                              { position 6}
             If Not ( dum in ['0'..'9', 'A'..'F'])
                                              { digit is hexadecimal}
             Then Begin
                     error:= True;
                     Exit;                    { error in received record}
                  End;

             Case dum Of
             '0'..'9': Val( dum, value, code);
             'A'..'F': value:= Ord( dum) - Ord( 'A') + 10;
             End;

             value:= Round(value * two2power[ y * 4]);{ first digit is 16's}
                                              { second digit is 1's}
             temp[ x+1] := temp[ x+1] + value;
          End;
        End;                                  { complete line converted +}
   End;                                       { stored}


   Function Scale( temparray: temptype; pointer, startbit,
                   nr_of_bits: Byte): Double;
   {*******************************************************************}
   { The function takes one or more bytes from an array called tempalmanac
     and converts it into a value. Pointer indicates the first byte from
     the array to be used. Startbit indicates the number of the first bit
     to be used. The MSB has number 0, increasing to the LSB's. Nr_of_bits
     indicates the number of bits to be used.
     WARNING: THIS FUNCTION WILL ONLY WORK A MAXIMUM OF 32 BITS, BECAUSE
              OF THE ROUND FUNCTION.
                          Input : array with bytes
                                  pointer for first byte
                                  startbitnumber
                                  number of bits
                          Output: Value of the number indicated with
                                  the input variables}
   {*******************************************************************}
   Var
      temp           :       Double;
      temppointer    :       Byte;
      x              :       Byte;
      leftoverbits   :       Byte;

   Begin
      temppointer:= pointer;  { save the pointer in the array here}
      temp:= temparray[ pointer];
      Inc( pointer);          { take the pointer'th byte from the array}
      While (startbit + nr_of_bits) - ((pointer - temppointer) * 8) > 0 Do
      Begin                   { repeat this, until enough bytes are taken}
        temp:= temp * 256 + temparray[ pointer];
        Inc( pointer);        { give every byte its position ref weight}
      End;

                       { Delete MSB's that should not be used}
                       { x is number of the bits that should not be used}
                       { substract the values indicated by those bits}
      If startbit > 0
      Then For x:= 0 To startbit - 1 Do
             If ( temp >= two2power[ (pointer - temppointer) * 8
                                     - x - 1] )
             Then temp:= temp - two2power[ (pointer - temppointer) * 8
                                     - x - 1];

                       { delete LSB's that should not be used}
                       { x is number of LSB's that should not be used}
      leftoverbits:= - (( startbit + nr_of_bits) -
                     ((pointer - temppointer) * 8));
      While leftoverbits > 0 Do
```

```
                                    { divide by 2, to correct for these LSB's}
        Begin                       { as many times as there are LSB's too much}
            Dec( leftoverbits);
            temp:= temp / 2;
            temp:= Trunc( temp);
        End;
        Scale:= temp;
    End;


    Function Twoscomplement( number: Double; nr_of_bits: Byte): Double;
    {*******************************************************************}
    {A number is converted to the decimal 'number' using the natural binary
     code. This number was a twoscomplement number. This function decodes
     'number' 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.
                           Input : binary number
                                   number of bits
                           Output: two's complement of the number}
    {*******************************************************************}
    Begin
         If number > two2power[ nr_of_bits - 1]
         Then Twoscomplement:= - two2power[ nr_of_bits] + number
         Else Twoscomplement:= number;
    End;


    Procedure Rawdata( rec: String; Var GPSint: GPSinttype;
                       Var svcount: Shortint);
    {*****************************************************************}
    { This procedure converts the received string from the Engine, which
      contains rawdata, to variables in the GPSint record.
                           Input : received string
                           Output: variables in GPSint
                                   indication on end of rawdata cycle}
    {*****************************************************************}

    Const
         c     =      299792458;             { speed of light}
         l1freq=     1575420000.0;           { frequency of l1 carrier }

    Var
       user_ms,                              { user time in GPS receiver
                                               (ms)}
       chnl_ms,                              { channel time in GPS
                                               receiver (ms)}
       phi,                                  { integrated carrier phase
                                               l1 wavelengths}
       phi_frac,                             { idem, fractional. LSB=(l1
                                               wavelength)/256}
       code                                  { raw code affset, l1 wave-
                                               lengths}
                                :     Extended;
       prn                      :     Byte;          { prn-code of the satellite}
       fault                    :     Integer;
       subrec                   :     String;

       user_time,                                    { user time in GPS receiver (s)}
       channel_time                                  { channel time in GPS receiver (s)}
                                :     Extended;
    value,
    lineout        :      String;

       Begin
          subrec:= Copy( rec, 9, 2);                 { grab satellite prn}
          Val( subrec, prn, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          subrec:= Copy( rec, 12, 9);                { grab user_ms}
          Val( subrec, user_ms, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          subrec:= Copy( rec, 22, 9);                { grab channel_ms}
          Val( subrec, chnl_ms, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          subrec:= Copy( rec, 32, 10);               { grab phi}
          Val( subrec, phi, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          subrec:= Copy( rec, 43, 6);                { grab code}
          Val( subrec, code, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          subrec:= Copy( rec, 50, 4);                { grab phi_frac}
          Val( subrec, phi_frac, fault);
          If ( fault <> 0)                           { error converting ascii}
          Then Exit;                                 { to number}

          If (user_ms = old_user_ms)
          Then Begin
                  user_time:= user_ms/1000;     { calculate receive and
                                                  transmit times,
                                                  see MAGNAVOX guide}

                  With GPSint.prn[ prn] Do
                  Begin
                       intcarphase:= ( phi_frac / 256);
                       intcarphase:= intcarphase + phi;

                       txtime:= ( chnl_ms / 1000) +
                                ( intcarphase / l1freq) +
```

```
                        ( code / l1freq);
                   rxtime:= user_time;
                   flag:= False;

                End;
{ This is made a comment, to make the program work better with a disc.
  In that case, it may assume a wrong number of satellites being tracked
  giving erronous results. By not incrementing 'svcount', the errors can
  be prevented. The complete cycle of measurements is noticed as soon as
  the next cycle begins.}
                Inc( svcount);
             End
          Else Begin
                Restore_buffer( port1, LF+rec);{ put received string back
                                                 for later use}
                old_user_ms:= user_ms;
                svcount:= GPSint.numofsat;

                Date_and_time( Tpr);
             End;
      End;   { End of procedure rawdata}


      Procedure Ionoscor( rec: String; Var GPSint: GPSinttype);
      {*******************************************************************}
      { This procedure converts a received Engine string, containing Iono-
        spheric information, to variables. It takes one, two or three bytes
        from the tempionos, converts this to a value and adds an exponent.
        The information used, can be found in Appendix 3 to Annex A to
        STANAG 4249, 1 August 1990.
                             Input : received recordnumber
                                     received string
                             Output: updated almanac in GPSint}
      {*******************************************************************}

      Var
         tempionos      :       temptype;
         error          :       Boolean;

      Begin
           Conv_ASCII_2_val( rec, tempionos, error);
           If error
           Then Exit;
           With GPSint.iono Do
           Begin
                alfa0:= Twoscomplement( Scale( tempionos, 2, 0, 8), 8);
                alfa0:= alfa0 / two2power[ 30];

                alfa1:= Twoscomplement( Scale( tempionos, 3, 0, 8), 8);
                alfa1:= alfa1 / two2power[ 27];

                alfa2:= Twoscomplement( Scale( tempionos, 4, 0, 8), 8);
                alfa2:= alfa2 / two2power[ 24];

                alfa3:= Twoscomplement( Scale( tempionos, 5, 0, 8), 8);
```

```
                alfa3:= alfa3 / two2power[ 24];

                beta0:= Twoscomplement( Scale( tempionos, 6, 0, 8), 8);
                beta0:= beta0 * two2power[ 11];

                beta1:= Twoscomplement( Scale( tempionos, 7, 0, 8), 8);
                beta1:= beta1 * two2power[ 14];

                beta2:= Twoscomplement( Scale( tempionos, 8, 0, 8), 8);
                beta2:= beta2 * two2power[ 16];

                beta3:= Twoscomplement( Scale( tempionos, 9, 0, 8), 8);
                beta3:= beta3 * two2power[ 16];
           End; { End of with}
           Date_and_time( GPSint.Tionos);
      End;      { End of procedure Ionoscor}


      Procedure Get_SV_id( rec: String; Var sv_id: Byte);
      {*******************************************************************}
      { This procedure is needed, because the gpsengine sends the ephemeris
        in several records. Only the first record contains the SV-id of the
        SV from which the ephemeris originates
                             Input : received string
                             Output: satellite identity}
      {*******************************************************************}

      Var
         subrec         :       String;
         code           :       Integer;

      Begin
         subrec:= Copy( rec, 6, 3);             { sv-id, position unknown}
         Val( subrec, sv_id, code);             { convert string to value}
         If ( code <> 0)                        { on error clear sv_id}
         Then sv_id:=0;

         Date_and_time( Tsv_id);
      End;


      Procedure Clockinfo( rec: String; sv_id: Byte;
                           Var GPSint: GPSinttype);
      {*******************************************************************}
      { This procedure converts received Engine strings, containing Clock-
        information per satellite, to variables. It takes one, two or three
        bytes from the tempclock, converts this to a value and adds an
        exponent. The information used, can be found in Appendix 3 to Annex
        A to STANAG 4249, 1 August 1990.
                             Input : received string
                                     satellite identity
                             Output: updated clock info per SV in GPSint}
      {*******************************************************************}
      Var
         tempclock      :       temptype;
```

```
      error          :      Boolean;
      sumtime,
      time           :      timetype;
      dum            :      Double;

   Begin
      With GPSint Do
      Begin
         Date_and_Time( time);            { get current system time}
         Addtime( Tsv_id, Valid_Tsv_id, sumtime);
         If (sv_id = 0) Or Later( time, sumtime)
                                          { if Satellite identity
                                            number is 0, or the infor-
                                            mation is timed out, than
                                            no valid information}
         Then Begin
                 ErrorTime( Tsv_id);
                 Exit;
              End;
      End;

      Conv_ASCII_2_val( rec, tempclock, error);
      If error
      Then Exit;

      With GPSint.prn[ sv_id].clock Do
      Begin
         Tgd:= Twoscomplement( Scale( tempclock, 15, 0, 8), 8);
         Tgd:= Tgd / two2power[ 31];

         toc:= Scale( tempclock, 17, 0, 16);
         toc:= Round( toc * two2power[ 4]);
         If toc > 604784                  { check on range}
         Then Begin
                 ErrorTime( GPSint.prn[ sv_id].Tck);
                 Exit;
              End;

         af2:= Twoscomplement( Scale( tempclock, 19, 0, 8), 8);
         af2:= af2 / two2power[ 55];

         af1:= Twoscomplement( Scale( tempclock, 20, 0, 16), 16);
         af1:= af1 / two2power[ 43];

         af0:= Twoscomplement( Scale( tempclock, 22, 0, 22), 22);
         af0:= af0 / two2power[ 31];

         dum:= Scale( tempclock, 3, 6, 2) * 256;
         IODC:= Integer( Round(dum));
         dum:= Scale( tempclock, 16, 0, 8);
         IODC:= IODC + Integer( ROund(dum));
      End;     { End with}
      With GPSint.prn[ sv_id] Do
      Begin
         health:= Round( Scale( tempclock, 3, 0, 6));
                                          { set the 6 LSB's of
                                            health by health from
                                            SV record}
         If Round( Scale( tempclock, 3, 0, 1)) = 0
         Then health:= ( health And $1F)  { healthy force zero's}
         Else health:= ( health Or $E0);  { unhealthy force one's}
                                          { use MSB from SV record
                                            to set health to 'alldata
                                            bad' or 'all data ok'}

         Date_and_Time( Tck);
      End;     { End with}
   End;        { End procedure clockinfo}


Procedure Ephemeris( recnum: Integer; rec: String; sv_id: Byte;
                     Var GPSint: GPSinttype);
{*****************************************************************}
{ This procedure converts received Engine strings, containing Ephem-
  eris information, to variables. It takes one, two or three bytes
  from the tempephem, converts this to a value and adds an exponent.
  The information used, can be found in Appendix 3 to Annex A to
  STANAG 4249, 1 August 1990.
                      Input : received recordnumber
                              received string
                              satellite identity
                      Output: updated ephemeris in GPSint}
{*****************************************************************}
Var
   tempephem      :      temptype;
   error          :      Boolean;
   time,
   sumtime        :      timetype;

Begin
   Date_and_Time( time);               { get current system time}
   Addtime( Tsv_id, Valid_Tsv_id, sumtime);
   If (sv_id = 0) Or Later( time, sumtime)
                                       { if Satellite identity
                                         number is 0, or the infor-
                                         mation is timed out, than
                                         no valid information}
   Then Begin
           ErrorTime( Tsv_id);
           Exit;
        End;

   Conv_ASCII_2_val( rec, tempephem, error);
   If error
   Then Exit;

   With GPSint.prn[ sv_id].ephemeris Do
   Begin
     Case recnum Of
     202: Begin
            IODE:= tempephem[ 1];
```

```
                Crs:= Twoscomplement( Scale( tempephem, 2, 0, 16), 16);
                Crs:= Crs / two2power[ 5];

                deltan:= Twoscomplement( Scale( tempephem, 4, 0, 16), 16);
                deltan:= deltan / two2power[ 43];

                Mo:= Twoscomplement( Scale( tempephem, 6, 0, 32), 32);
                Mo:= Mo / two2power[ 31];

                Cuc:= Twoscomplement( Scale( tempephem, 10, 0, 16), 16);
                Cuc:= Cuc / two2power[ 29];

                e:= Scale( tempephem, 12, 0, 32);
                e:= e / two2power[ 33];
                If e > 0.03                     { Check on range}
                Then Begin
                        ErrorTime( GPSint.prn[ sv_id].Tephem);
                        Exit;
                     End;

                Cus:= Twoscomplement( Scale( tempephem, 16, 0, 16), 16);
                Cus:= Cus / two2power[ 29];

                Asqrt:= Scale( tempephem, 18, 0, 32);
                Asqrt:= Asqrt / two2power[ 19];

                toe:= Scale( tempephem, 22, 0, 16);
                toe:= toe * two2power[ 4];
                If toe > 604784                 { Check on range}
                Then Begin
                        ErrorTime( GPSint.prn[ sv_id].Tephem);
                        Exit;
                     End;

                Date_and_Time( GPSint.prn[ sv_id].Tephem);
              End;    { End of Case 202}
         203: Begin
                If (tempephem[22] <> IODE)   {new ephemeris being}
                Then Begin                   {uploaded. Subframe 2}
                       ErrorTime( GPSint.prn[ sv_id].Tephem);
                       Exit                  {ready, subframe 3 not}
                     End;                    {don't use this sv now}

                Cic:= Twoscomplement( Scale( tempephem, 1, 0, 16), 16);
                Cic:= Cic / two2power[ 29];

                omegao:= Twoscomplement( Scale( tempephem, 3, 0, 32), 32);
                omegao:= omegao / two2power[ 31];

                Cis:= Twoscomplement( Scale( tempephem, 7, 0, 16), 16);
                Cis:= Cis / two2power[ 29];

                io:= Twoscomplement( Scale( tempephem, 9, 0, 32), 32);
                io:= io / two2power[ 31];
```

```
                Crc:= Twoscomplement( Scale( tempephem, 13, 0, 16), 16);
                Crc:= Crc / two2power[ 5];

                omega:= Twoscomplement( Scale( tempephem, 15, 0, 32),
                                            32);
                omega:= omega / two2power[ 31];

                omegadot:= Twoscomplement( Scale( tempephem, 19, 0, 24),
                                               24);
                omegadot:= omegadot / two2power[ 43];

                IDOT:= Twoscomplement( Scale( tempephem, 23, 0, 14), 14);
                IDOT:= IDOT / two2power[ 43];

                Date_and_Time( GPSint.prn[ sv_id].Tephem);
                sv_id:= 0;                    { When last record for
                                                satellite nr: sv_id is
                                                used, then reset sv_id
                                                so, no errors are made
                                                if records 201..203 are
                                                received without record
                                                200( correct sv_id)}
              End;      { End of Case 203}
         End;           { End of case}
       End;             { End of with}
  End;                  { End of procedure Ephemeris}

{------------------ end included procedures collectport1----------}
Var
   rec,
   dumstr          :      String;
   dumch           :      Char;
   recnum,
   code            :      Integer;
   dum             :      Integer;
   x               :      Byte;
   time,
   result          :      timetype;
   part            :      String;

{--------------- start procedure collect port1-----------------}
Begin
  For x:= 1 To 32 Do
     gpsint.prn[x].flag:= true;

  If (Charsinbuff( port1) > 80)             { only start if there}
  Then Begin                                { are enough chars in buff}
          rec:= Getcharbuff( port1);        { Get first char}

          If ( rec = LF)                    { If this is a line feed}
          Then Begin                        { then we expect a new}
                  rec:='';                  { leave a LF in buffer}
                  Repeat                    { line to start}
                        dumch:= LookBuff( port1);
```

```
                                If ( dumch <> LF)
                                Then rec:= rec + Getcharbuff( port1);
                           Until ( dumch = LF) Or ( charsinbuff( port1)= 0);
                                             { continue until end of
                                               line is detected, or
                                               no more chars}
                                             { if no more chars, then
                                               error in received line}
                           If ( charsinbuff( port1) = 0)
                           Then Exit;
                           If ( rec[4] = ' ')      { for right alignment of
                                                     record number}
                           Then rec:= ' '+ rec;    { insert space if necessary}
                                                   { copy the number}
                           dumstr:= Copy( rec, 1, 4);
                                                   { convert string to value}
                           Val( dumstr, recnum, code);
                           If ( code = 0)          { if no error converting}
                           Then Begin
gotoxy( 1, 17);
write( rec);
SaveEquipmentMessage( 'RG: ' + rec);
                                Case recnum Of
                                1       : Rawdata( rec, GPSint, svcount);
                                End;               { end case}
                              End                  { end begin}
                           Else ;                   { what to do if error?}
                       End
                  Else Begin                       { if no begin of line}
                           Repeat                  { skip chars from buffer}
                                dumch:= LookBuff( port1);
                                If ( dumch <> LF)
                                Then dumch:= Getcharbuff( port1);
                           Until ( dumch = LF) Or ( charsinbuff( port1) = 0);
                                                   { a lf was found or the
                                                     buffer is empty}
                       End;
              End; { end if charsinbuff}

   End; { end collectport1}
   {---------------- end collectport1------------------------}
   {---------------- end included procedures CollectGPSrec-----------}

Var
   counter      :       Integer;
   {---------------- start procedure CollectGPSrec------------------}
Begin
     counter:= 1;                                  { now we read 2 message}
     Repeat                                        { from the engine receiver}
          Collectport0( GPSint);                   { these contain most certain}
          Collectport1( GPSint);                   { ly all the measurements}
          Dec( counter);
     Until (counter = 0);
End;
{---------------- end procedure CollectGPSrec-------------------}

{---------------- start procedure execgpsreccommand-----------}
Procedure ExecGPSrecCommand( command: commandtype);
                                                   { IN THE COMMENT, UNDER-
                                                     SCORES '_' SHOULD BE
                                                     READ AS SPACES ' '}
Var
   substr,
   sendline     :       String;
   code,
   sv           :       Integer;
   rate         :       Integer;

Begin
     sendline:= '';

     substr:= Copy( command, 5, Length( command) - 4);
                                                   { delete header: 'GPS:'}
     If substr = 'RESET'
     Then Begin
               sendline:= '$PMVXG,018,T'#13#10;  { RESTART THE ENGINE WITH
                                                   TEPID START}
{              IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
}         End;                                     { clear port0 outputlist}


     If (Copy( substr, 1, 4) = 'INIT') And
        ( Length( substr) >= 52)
     Then Begin
               sendline:= '$PMVXG,000,'+
                    Copy( substr, 6, 2) + ',' +
                    Copy( substr, 9, 2) + ',' +  { INITIALISES THE ENGINE}
                    '19' +
                    Copy( substr, 12, 2) + ',' + { RECEIVER WITH POSITION}
                    Copy( substr, 15, 6) + ',' + { ETC. FORMAT IS: 'INIT_}
                    Copy( substr, 22, 9) + ',' + { DD_MM_YY_HHMM_DDMM.MMMM}
                    Copy( substr, 32, 1) + ',' + { _N_DDDMM.MMMM_E_HHHHH.H_}
                                                 { AA.A_EL'}
                    Copy( substr, 34, 10)+ ',' + { WHERE DD_MM_YY IS THE}
                    Copy( substr, 45, 1) + ',' + { DATE IN DAY, MONTH, AND}
                    Copy( substr, 47, 7) + ',' +
                    #13#10;

               IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);


               sendline:=
                    '$PMVXG,001,,,' +            { YEAR. HHMM IS TIME, }
                    Copy( substr, 55, 4) + ',' + { DDMM.MMMM IS LATITUDE}
                    ',,,' +                      { IN DEGREES AND MINUTES,}
                    Copy( substr, 60, 2) + ',' + { N IS NORTH OR SOUTH  }
```

```
                  ','+#13#10;                    { (N/S), DDDMM.MMMM IS  }
                                                 { LONGITUDE IN DEGREES }
                                                 { AND MINUTES, E IS EAST}
                                                 { OR WEST (E/W) AND    }
                                                 { HHHHH.H IS ALTITUDE  }
                                                 { ABOVE MEAN SEA LEVEL,}
                                                 { AA.A IS HORIZONTAL}
                                                 { ACCELERATION FACTOR}
                                                 { EL IS ELEVATION LIMIT}
            IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);

            sendline:= '$PMVXG,024,-,+,+,-,-,-,-'#13#10;
            IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);

                                                { NO NAV RESULT, RAW
                                                  MEASUREMNENTS, APMANAC &
                                                  EPHEMERIS, NO CONTROL
                                                  INFO, NO TIME RECOVERY,
                                                  NO FULL DEBUG, NO PARTIAL
                                                  DEBUG}
            sendline:= '$PMVXG,023,D,G,,,,,'#13#10;
            IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
                                                { SET TIME RECOVERY ON}
            sendline:= '$PMVXG,007,,1,,,,,,,'#13#10;
            IWriteCom( port0, sendline);        { NO OUTPUT ON CTRL PORT}
SaveEquipmentMessage( 'SG: ' + sendline);
            sendline:= '$PMVXG,007,000,0,1,,1,,,,,'#13#10;
            IWriteCom( port0, sendline);        { OUTPUT MESSAGE 000}
SaveEquipmentMessage( 'SG: ' + sendline);
            sendline:= '$PMVXG,007,001,0,1,,1,,,,,'#13#10;
            IWriteCom( port0, sendline);        { OUTPUT MESSAGE 001}
SaveEquipmentMessage( 'SG: ' + sendline);
            sendline:= '$PMVXG,007,021,0,1,,1,,,,,'#13#10;
            IWriteCom( port0, sendline);        { OUTPUT MESSAGE 021}
SaveEquipmentMessage( 'SG: ' + sendline);

        End;
    If substr = 'SEND EPHEMERIS ETC'
    Then Begin
            sendline:= '$PMVXG,027,,,,2,2'#13#10;
            IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
        End;                                   { OUTPUT EPHEMERIS AND
                                                 ALMANAC NOW}

End;
{--------------- End procedure execgpsreccommand-----------}


{--------------- start procedure closegpsrec------------}
Procedure CloseGPSrec;

Var
  setupfile     :       Text;
  value         :       String;

Begin
     Removeint( port0);
     Removeint( port1);

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'GPSENGINE');

     Str( port0, value);
     Writeln( setupfile, #9'port0 = ', value, ';');

     Str( port1, value);
     Writeln( setupfile, #9'port1 = ', value, ';');

     Str( completeinfo, value);
     Writeln( setupfile, #9'completeinfo = ', value, ';');
     CloseConfig( setupfile);
End;
{--------------- end procedure closegpsrec-----------}



{--------------- start initialising -----------------}
Begin                                          { initialising part}
                                               { used to make a table}
                                               { containing the power of 2}
     two2power[ 0]:= 1;                        { fill most left position}
     For x:= 1 To 55 Do                        { calculate 2^1 to 2^55}
         two2power[ x]:= two2power[ x-1] * 2;

     sv_id:=0;                                 { set sv_id to 0, which is
                                                 invalid, so no mistake can
                                                 be made}
     ErrorTime( Tsv_id);
     ErrorTime( Valid_Tsv_id);
     Valid_Tsv_id.minute:= 5;                  { the sv_id stays valid for 5}
                                               { minutes ( see record 200)}
     ErrorTime( Tpr);
     ErrorTime( Valid_Tpr);                    { the pr are maximum valid for}
     Valid_Tpr.sec:= 1;                        { 1 second, this is the update}
                                               { rate of the GPSengine rx}
     old_user_ms:= 0;
     svcount:= 0;

     With tempGPSint Do                        { make an empty GPSint set}
     Begin
          flag:= True;
          ErrorTime( Tionos);
          For x:= 1 To 32 Do
          With prn[x] Do
          Begin
               flag:= True;
```

```
                ErrorTime( Tck);
                ErrorTime( Tephem);
            End;
            numofsat:= 0;
        End;

        completeinfo:= 1;
End.
{---------------- end initialising ------------------}
{---------------- end Unit GPSENGINE ------------------}
```

```
Unit MLSbendix;
{*************************************************************************}
{ THis unit is meant to be used with a Bendix MLS-20A receiver. This
  receiver is a quasi MLSarinc727 receiver. The mls data words are not
  passed through by this receiver. The basic datawords are simulated by
  assigning variables with static basic data values}
{*************************************************************************}

{$N+,E+}

Interface


Uses MIASglob, MLSglob, crt;


Procedure InitMLSrec( Var error: Boolean);
{*************************************************************************}
{ Initialise the Arinc 727 receiver. If something went wrong, error := True.
                              Input : -
                              Output: error}
{*************************************************************************}


Procedure CollectMLSrec( Var MLSint: MLSinttype);
{*************************************************************************}
{ Get Arinc 429 words; Skip the words that are not necessary; check the
  necessary words on parity. ADW's are checked on CRC.
                              Input : -
                              Output: Mlsint}
{*************************************************************************}


Procedure ExecMLSrecCommand( command: commandtype);
{*************************************************************************}
{ Convert a MLScommand, to a command word for the ARinc 727 MLS receiver.
  That is, a Arinc 429 word. Then send the word using the Arinc 429 tx
  channel.
                              Input : command
                              Output: Arinc 429 word on 429 tx port}
{*************************************************************************}


Procedure CloseMLSrec;
{*************************************************************************}
{ Make sure the ARINC 727 MLS receiver is back to normal. Restore interrupt-
  vectors etc.
                              Input : -
                              Output: -}
{*************************************************************************}


Implementation

Uses Ar429comm, Ar429, Miscell, User, ADW;

Type
    ADW_prestype        =       Array[1..4] Of Boolean;
    ByteArray           =       Array[1..10] Of Byte;

Const
     mlsfreq_lab=       036;                    { mlsfrequency label}
     azimut_lab =       151;                    { azimuth function label}
     elevat_lab =       152;                    { elevation function label}
     BackAz_lab =       240;                    { backazimuth function label}

Var
   ADW_A_pres,
   ADW_B_pres,
   ADW_C_pres           :       ADW_prestype;
   ADW_A,
   ADW_B,
   ADW_C                :       ADWtype;
   x                    :       Integer;
   AngleBegin           :       Boolean;
   tempMLSint           :       MLSinttype;
   Tangle,
   Valid_Tangle         :       timetype;
   completeinfo         :       Byte;
   irq                  :       Byte;
   cardaddress          :       Word;
   badwline             :       String;

{------------------------Start InitMLSrec------------------------}
Procedure InitMLSrec( Var error: Boolean);

Var
   title,
   line         :       String;
   code         :       Integer;
   varname,
   value        :       String;
   setupfile    :       Text;
   selecttable  :       arraytype;

Begin
     error:= True;
     cardaddress:= $280;
     irq:= 7;
     badwline:= '';

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                     { find MLSBENDIX part of
                                                  config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MLSBENDIX'));

     If Not Eof( setupfile)                     { if there is more in file}
     Then Repeat
                Readln( setupfile, line);       { get a line}
```

```
                Convert( line, varname, value);  { extract the variable name
                                                   and value}
                If ( varname = 'COMPLETEINFO')
                Then Val( value, completeinfo, code);

                If ( varname = 'CARDADDRESS')
                Then Val( value, cardaddress, code);

                If ( varname = 'IRQ')
                Then Val( value, irq, code);
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
                                                 { repeat until end of file}
     CloseConfig( setupfile);

     Ar429comm.InitAr429;                        { initialise comm-port for}
                                                 { VHF link}
     selecttable[ 0]:= mlsfreq_lab;              { initialise ARINC card}
     selecttable[ 1]:= azimut_lab;               { by filling in a label-}
     selecttable[ 2]:= elevat_lab;               { select table}
     selecttable[ 3]:= backaz_lab;

     Ar429.InitAr429( error, selecttable, 4);

     InitKaartadres( cardaddress);               { Initialise ADW-card}
     KiesIRQ( IRQ);
     Install_ADW_int;
     ProgTrigFunktie( 1, $0A);                   { Select BDW 1}
     ProgTrigFunktie( 2, $1F);                   { Select BDW 2}
     ProgTrigFunktie( 3, $55);
     ProgTrigFunktie( 4, $11);
     ProgTrigFunktie( 5, $1B);
     ProgTrigFunktie( 6, $58);                   { Select BDW 6}
     ProgTrigFunktie( 7, $27);                   { Select ADW A}
End;
{------------------------End InitMLSrec------------------------}


{------------------------Start CollectMLSrec------------------------}
Procedure CollectMLSrec( Var MLSint: MLSinttype);

   {--------------- start included procedures CollectMLSrec-----------}
   Function ADW_present( ADW_pres: ADW_prestype): Boolean;
   {*****************************************************************}
   { Check if all 4 arinc429 words, that form an adw, are present. A
     true is output if all 4 are present, if not, a false is output.
                              Input : Array with present flags
                              Output: Boolean.}
   {*****************************************************************}

   Begin
        ADW_present:= ADW_pres[ 1] And ADW_pres[ 2] And
                      ADW_pres[ 3] And ADW_pres[ 4];
   End;


Function Hamming_Fail( ADW: ADWtype): Boolean;
{*****************************************************************}
{ This function checks the hammingcode in the specified ADW. If the
  hammingcode was correct, then the output is true. See Annex 10 p 150B
                              Input : ADW
                              Output: Boolean}
{ Author: Maarten Uit de Haag;
  Revised: Marco Meijer}
{*****************************************************************}
Var
   x        :       Integer;
   succes   :       Boolean;
   check    :       Array[ 0..6] Of Integer;

Begin
     succes:= True;

     For x:= 0 To 4 Do
     Begin
          check[ x]:= ADW[13 + x] + ADW[14 + x] + ADW[15 + x] +
                      ADW[16 + x] + ADW[17 + x] + ADW[18 + x] +
                      ADW[20 + x] + ADW[22 + x] + ADW[24 + x] +
                      ADW[25 + x] + ADW[28 + x] + ADW[29 + x] +
                      ADW[31 + x] + ADW[32 + x] + ADW[33 + x] +
                      ADW[35 + x] + ADW[36 + x] + ADW[38 + x] +
                      ADW[41 + x] + ADW[44 + x] + ADW[45 + x] +
                      ADW[46 + x] + ADW[50 + x] + ADW[52 + x] +
                      ADW[53 + x] + ADW[54 + x] + ADW[55 + x] +
                      ADW[58 + x] + ADW[60 + x] + ADW[64 + x] +
                      ADW[65 + x];
           check[ x]:= check[ x] Mod 2;
           succes:= succes And ( check[ x] = ADW[ 70 + x]);
      End;
      check[ 5]:=     ADW[13 ] + ADW[14 ] + ADW[15 ] + ADW[16 ] +
                      ADW[17 ] + ADW[19 ] + ADW[21 ] + ADW[23 ] +
                      ADW[24 ] + ADW[27 ] + ADW[28 ] + ADW[30 ] +
                      ADW[31 ] + ADW[32 ] + ADW[34 ] + ADW[35 ] +
                      ADW[37 ] + ADW[40 ] + ADW[43 ] + ADW[44 ] +
                      ADW[45 ] + ADW[49 ] + ADW[51 ] + ADW[52 ] +
                      ADW[53 ] + ADW[54 ] + ADW[57 ] + ADW[59 ] +
                      ADW[63 ] + ADW[64 ] + ADW[69 ];
      check[ 5]:= check[ 5] Mod 2;
      succes:= succes And ( check[ 5] = ADW[ 75]);

      check[ 6]:= 0;
      For x:= 13 To 75 Do
          check[ 6]:= check[ 6] + ADW[ x];
      check[ 6]:= check[ 6] Mod 2;
      succes:= succes And ( check[ 6] = ADW[ 76]);

      Hamming_Fail:= Not Succes;
End;


Function Adress_Fail( ADW: ADWtype): Boolean;
```

```
{*******************************************************************}
{ This function checks the parity of the adress in the specified ADW.
  If the adress was correct, then the output is False. See Annex 10 p 150
                          Input : ADW
                          Output: Boolean}
{ Author: Maarten Uit de Haag;
  Revised: Marco Meijer}
{*******************************************************************}
Var
   check     :       Array[ 1..2] Of Integer;

Begin
     check[ 1]:= ADW[ 13] + ADW[ 14] + ADW[ 15] + ADW[ 16] + ADW[ 17] +
                 ADW[ 18];
     check[ 1]:= check[ 1] Mod 2;
     check[ 2]:= ADW[ 14] + ADW[ 16] + ADW[ 18];
     check[ 2]:= check[ 2] Mod 2;

     Adress_Fail:= Not ((check[ 1] = ADW[ 19]) And
                        ( check[ 2] = ADW[ 20]));
End;



Function ADW_adress( ADW: ADWtype): Byte;
{*******************************************************************}
{ This function takes bits 13 to 18 of the ADW and translates these
  bits to an adress. This is a normal binary code. See Annex 10 p 150
                          Input : ADW
                          Output: adress}
{*******************************************************************}
Var
   adress,
   mult,
   x         :       Integer;

Begin
     adress:= 0;
     mult:= 32;
     For x:= 0 To 5 Do
     Begin
          adress:= adress + ADW[ 13 + x] * mult;
          mult:= mult Div 2;
     End;
     ADW_adress:= adress;
End;


Function Conv_ADW( ADW: ADWtype; start: Byte; number: Byte): Word;
{*******************************************************************}
{ This function converts single bits to a number. 'Start' indicates
  the start bit in the ADW. First bit is bitnumber 1. 'Number' indicates
  the number of bits to be used for the number to be formed. The number
  is output as a Word. See Annex 10 p 60CC; LSB first
                          Input : ADW
```

```
                                        startbit
                                        number of bits
                                Output: number}
{*******************************************************************}
Var
   value,
   mult      :       Word;

Begin
     value:= 0;
     mult:= 1;
     For x:= 0 To number Do
     Begin
          value:= value + ADW[ start + x] * mult;
          mult:= mult * 2;
     End;
     Conv_ADW:= value;
End;



Procedure ADW_A_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                      a_label: byte; Var ADW_A_pres: ADW_prestype;
                      Var ADW_A: ADWtype);
{*******************************************************************}
{ This procedure converts the A-ADW's to Pascal variables.
  See Annex 10 p 150A etc.
                          Input : Arinc 429 word
                                  Arinc 429 label
                                  ADW_A_present array of boolean
                          Output: Mlsint
                                  A-ADW  }
{*******************************************************************}
Var
   adress   :       Byte;

Begin
     Case a_label Of
     88 :Begin   {130}                      { 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}
              For x:= 13 To 28 Do
                  ADW_A[ x]:= Ar429word[ x - 13 + 14];
              ADW_A_pres[ 1]:= True;
         End;
     89 :Begin   {131}
              If Not ADW_A_pres[1]              { second part only valid}
              Then Exit;                        { if first part presend}

              For x:= 29 To 44 Do
                  ADW_A[ x]:= Ar429word[ x - 29 + 14];
              ADW_A_pres[ 2]:= True;
```

```
            End;
        90 :Begin   {132}
                If Not ADW_A_pres[2]            { third part only valid, if}
                Then Exit;                      { second and first part}
                                                { present}
                For x:= 45 To 60 Do
                    ADW_A[ x]:= Ar429word[ x - 45 + 14];
                ADW_A_pres[ 3]:= True;
            End;
        91 :Begin   {133}
                If Not ADW_A_pres[3]            { fourth part only valid, if}
                Then Exit;                      { third, second and first }
                                                { part present}
                For x:= 61 To 76 Do
                    ADW_A[ x]:= Ar429word[ x - 61 + 14];
                ADW_A_pres[ 4]:= True;
            End;
        End;
        If (ADW_present( ADW_A_pres) And Not Hamming_Fail( ADW_A) And
            Not Adress_Fail( ADW_A))            { If the ADW is valid and
                                                  the address is correct,
                                                  begin decoding the info}
        Then Begin
                 adress:= ADW_Adress( ADW_A);
                 Case adress Of
                 1:Begin
                      Date_and_Time( Mlsint.AuxA1_time);
                      With Mlsint.auxa1 Do
                      Begin
                        If ADW_A[ 30] = 1       { MSB is sign bit}
                                                { see Annex 10 p150}
                        Then AzOff:= -1 * Conv_ADW( ADW_A, 21, 9)
                        Else AzOff:= Conv_ADW( ADW_A, 21, 9);

                        Az2MLSdatDist:= Conv_ADW( ADW_A, 31, 13);

                        If ADW_A[55] = 1
                        Then AzAlignRun:= -1 * Conv_ADW( ADW_A, 44, 11)
                        Else AzAlignRun:= Conv_ADW( ADW_A, 44, 11);
                        AzAlignRun:= AzAlignRun * 0.01;

                        AzCoorSyst:= ADW_A[ 56];
                      End;
                      Mlsint.Auxa1_flag:= False;
                   End;
                 2:Begin
                      Date_and_Time( Mlsint.AuxA2_time);
                      With Mlsint.auxa2 Do
                      Begin
                         If ADW_A[ 30] = 1
                         Then ElOff:= -1 * Conv_ADW( ADW_A, 21, 9)
                         Else ElOff:= Conv_ADW( ADW_A, 21, 9);

                         MLSdat2thres:= Conv_ADW( ADW_A, 31, 10);

                         If ADW_A[ 47] = 1
                         Then ElHeight:= -1 * Conv_ADW( ADW_A, 41, 6)
                         Else ElHeight:= Conv_ADW( ADW_A, 41, 6);
                         ElHeight:= ElHeight * 0.1;
                      End;
                      Mlsint.Auxa2_flag:= False;
                   End;
                 3:Begin
                      Date_and_Time( Mlsint.AuxA3_time);
                      With Mlsint.auxa3 Do
                      Begin
                         If ADW_A[ 30] = 1
                         Then DMEoff:= -1 * Conv_ADW( ADW_A, 21, 9)
                         Else DMEoff:= Conv_ADW( ADW_A, 21, 9);

                         If ADW_A[ 44] = 1
                         Then DME2MLSdatdist:= -1 * Conv_ADW( ADW_A, 31, 13)
                         Else DME2MLSdatdist:= Conv_ADW( ADW_A, 31, 13);
                      End;
                      Mlsint.Auxa3_flag:= False;
                   End;
                 4:Begin
                      Date_and_Time( Mlsint.AuxA4_time);
                      With Mlsint.auxa4 Do
                      Begin
                         If ADW_A[ 30] = 1
                         Then BAzoff:= -1 * Conv_ADW( ADW_A, 21, 9)
                         Else BAzoff:= Conv_ADW( ADW_A, 21, 9);

                         BAz2MLSdatdist:= Conv_ADW( ADW_A, 31, 11);

                         If ADW_A[ 53] = 1
                         Then BAzAlignRun:= -1 * Conv_ADW( ADW_A, 42, 11)
                         Else BAzAlignRun:= Conv_ADW( ADW_A, 42, 11);
                         BAzAlignRun:= BAzAlignRun * 0.01;
                      End;
                      Mlsint.Auxa3_flag:= False;
                   End;
                 End;                               { End of Case}

                 For x:= 1 To 4 Do
                     ADW_A_pres[x]:= False;
             End;                                   { End of If}
End;                                                { End of procedure}


{---------------------------------------------------------------------}
{ The following part should not necessarily be implemented, because ADW B
  and C are not yet assigned.}
{---------------------------------------------------------------------}

Procedure ADW_B_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                      a_label: byte; Var ADW_B_pres: ADW_prestype;
                      Var ADW_B: ADWtype);
{*****************************************************************}
```

```
{ This procedure converts the B-ADW's to Pascal variables.
                         Input : Arinc 429 word
                                 Arinc 429 label
                                 ADW_B_present array of boolean
                         Output: Mlsint
                                 B-ADW  }
{*******************************************************************}
Var
   adress              :       Byte;
Begin
     Case a_label Of
     92 :Begin    {134}
              For x:= 13 To 28 Do
                  ADW_B[ x]:= Ar429word[ x - 13 + 14];
              ADW_B_pres[ 1]:= True;
         End;
     93 :Begin    {135}
              If Not ADW_B_pres[1]
              Then Exit;

              For x:= 29 To 44 Do
                  ADW_B[ x]:= Ar429word[ x - 29 + 14];
              ADW_B_pres[ 2]:= True;
         End;
     94 :Begin    {136}
              If Not ADW_B_pres[2]
              Then Exit;

              For x:= 45 To 60 Do
                  ADW_B[ x]:= Ar429word[ x - 45 + 14];
              ADW_B_pres[ 3]:= True;
         End;
     95 :Begin    {137}
              If Not ADW_B_pres[3]
              Then Exit;

              For x:= 61 To 76 Do
                  ADW_B[ x]:= Ar429word[ x - 61 + 14];
              ADW_B_pres[ 4]:= True;
         End;
     End;
     If (ADW_present( ADW_B_pres) And Not Hamming_Fail( ADW_B) And
         Not Adress_Fail( ADW_B))
     Then Begin
               adress:= ADW_Adress( ADW_B);
               Date_and_Time( MLSint.AuxB_time);
               MLSint.AuxB:= ADW_B;

               For x:= 1 To 4 Do
                   ADW_B_pres[x]:= False;
          End;
End;


Procedure ADW_C_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                            a_label: byte; Var ADW_C_pres: ADW_prestype;
                            Var ADW_C: ADWtype);
{*******************************************************************}
{ This procedure converts the C-ADW's to Pascal variables.
                         Input : Arinc 429 word
                                 Arinc 429 label
                                 ADW_C_present array of boolean
                         Output: Mlsint
                                 C-ADW  }
{*******************************************************************}
Var
   adress              :       Byte;

Begin
     Case a_label Of
     96 :Begin    {140}
              For x:= 13 To 28 Do
                  ADW_C[ x]:= Ar429word[ x - 13 + 14];
              ADW_C_pres[ 1]:= True;
         End;
     97 :Begin    {141}
              If Not ADW_C_pres[1]
              Then Exit;

              For x:= 29 To 44 Do
                  ADW_C[ x]:= Ar429word[ x - 29 + 14];
              ADW_C_pres[ 2]:= True;
         End;
     98 :Begin    {142}
              If Not ADW_C_pres[2]
              Then Exit;

              For x:= 45 To 60 Do
                  ADW_C[ x]:= Ar429word[ x - 45 + 14];
              ADW_C_pres[ 3]:= True;
         End;
     99 :Begin    {143}
              If Not ADW_C_pres[3]
              Then Exit;

              For x:= 61 To 76 Do
                  ADW_C[ x]:= Ar429word[ x - 61 + 14];
              ADW_C_pres[ 4]:= True;
         End;
     End;
     If (ADW_present( ADW_C_pres) And Not Hamming_Fail( ADW_C) And
         Not Adress_Fail( ADW_C))            { If the ADW is valid and
                                               the address is correct,
                                               begin decoding the info}
     Then Begin
               adress:= ADW_Adress( ADW_C);
               Date_and_Time( MLSint.AuxC_time);
               MLSint.AuxC:= ADW_C;

               For x:= 1 To 4 Do
```

```
                  ADW_C_pres[x]:= False;
         End;
   End;
{----end of part---------------------------------------------------------}

   Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
                    : Word;
   {*********************************************************************}
   { This function converts single bits to a number. 'Start' indicates
     the start bit in the Arinc 429 word. 'Number' indicates the number of
     bits to be used for the number to be formed. The number is output as a
     Word. See Annex 10 p 60CC: LSB first.
                              Input : Arinc429 word
                                      startbit
                                      number of bits
                              Output: number}
   {*********************************************************************}
   Var
      value,
      mult      :       Word;
      x         :       Byte;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 0 To number - 1 Do
        Begin
             value:= value + ar429word[ start + x] * mult;
             mult:= mult * 2;
        End;
        Conv_BAS:= value;
   End;



   Procedure Bas_1_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*********************************************************************}
   { This procedure converts the Basic dataword nr 1 to Pascal variables.
     See Annex 10 p 146.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*********************************************************************}
   Begin
        Date_and_Time( Mlsint.Bas1_time);

        With Mlsint.Bas1 Do
        Begin
             Az2Thresdist:= Conv_bas( Ar429word, 12, 6) * 100;
             AzPropCovNegLim:= Conv_bas( Ar429word, 19, 5) * 2;
             AzPropCovPosLim:= Conv_bas( Ar429word, 24, 5) * 2;
             Cleartype:= Ar429word[ 29];
        End;
        Mlsint. Bas1_flag:= False;
   End;


   Procedure Bas_2_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*********************************************************************}
   { This procedure converts the Basic dataword nr 2 to Pascal variables.
     See Annex 10 p 146.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*********************************************************************}
   Begin
        Date_and_Time( Mlsint.Bas2_time);

        With Mlsint.Bas2 Do
        Begin
             MinGP:= Conv_bas( Ar429word, 13, 7) * 0.1;
             BAZstat:= Ar429word[ 20];
             DMEstat:= Conv_bas( ar429word, 21, 2);
             Azstat:= Ar429word[ 23];
             Elstat:= Ar429word[ 24];
        End;
        Mlsint. Bas2_flag:= False;
   End;


   Procedure Bas_3_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*********************************************************************}
   { This procedure converts the Basic dataword nr 3 to Pascal variables.
     See Annex 10 p 147.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*********************************************************************}
   Begin
        Date_and_Time( Mlsint.Bas3_time);

        With Mlsint.Bas3 Do
        Begin
             AzBW:= Conv_bas( Ar429word, 13, 3) * 0.5;
             ElBW:= Conv_bas( Ar429word, 16, 3) * 0.5;
             DMEdist:= Conv_bas( Ar429word, 19, 9) * 12.5;
        End;
        Mlsint. Bas3_flag:= False;
   End;


   Procedure Bas_4_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
   {*********************************************************************}
   { This procedure converts the Basic dataword nr 4 to Pascal variables.
     See Annex 10 p 147.
                              Input : Arinc 429 word
                              Output: Mlsint}
   {*********************************************************************}
   Begin
        Date_and_Time( Mlsint.Bas4_time);

        With Mlsint.Bas4 Do
        Begin
```

```
          AzMagOr:= Conv_bas( Ar429word, 13, 9);
          BazMagor:= Conv_bas( Ar429word, 22, 9);
     End;
     Mlsint. Bas4_flag:= False;
End;


Procedure Bas_5_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Basic dataword nr 5 to Pascal variables.
  See Annex 10 p 147.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
     Date_and_Time( Mlsint.Bas5_time);

     With Mlsint.Bas5 Do
     Begin
          BAZPropCovNegLim:= Conv_bas( Ar429word, 13, 5) * 2;
          BazPropCovPosLim:= Conv_bas( Ar429word, 18, 5) * 2;
          BazBW:= Conv_bas( Ar429word, 23, 3) * 0.5;
          Bazstat:= Ar429word[ 26];
     End;
     Mlsint. Bas5_flag:= False;
End;


Procedure Bas_6_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Basic dataword nr 6 to Pascal variables.
  See Annex 10 p 147.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
     Date_and_Time( Mlsint.Bas6_time);

     With Mlsint.Bas6 Do
     Begin
          MLSident[1]:= Char( Conv_bas( ar429word, 13, 6));
          MLSident[2]:= Char( Conv_bas( ar429word, 19, 6));
          MLSident[3]:= Char( Conv_bas( ar429word, 25, 6));
     End;
     Mlsint. Bas6_flag:= False;
End;


Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Arinc 429 word containing the glidepath
  information to a Pascal variable.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
  With Mlsint Do
  Begin
     If ( Ar429word[ 27] = 1) And
        ( Ar429word[ 28] = 1) And
        ( Ar429word[ 29] = 1)
     Then Begin
             ELangle:= Conv_bas( Ar429word, 13, 14);
             ELangle:= ELangle - $4000;
             ELangle:= ELangle * 0.005;
          End;

     If ( Ar429word[ 27] = 0) And
        ( Ar429word[ 28] = 0) And
        ( Ar429word[ 29] = 0)
     Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

     ElAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
                                              {         1 = forward ant}
                                              { ARinc: 1,2,3 = ant no}
     ELangle_flag:= False;

  End;
End;


Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Arinc 429 word containing the azimuthangle
  information to a Pascal variable.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Var
   x,
   cnt      :      Byte;

Begin
  With Mlsint Do
  Begin
     If ( Ar429word[ 27] = 1) And
        ( Ar429word[ 28] = 1) And
        ( Ar429word[ 29] = 1)
     Then Begin
             Azangle:= Conv_bas( Ar429word, 13, 14);
             Azangle:= Azangle - $4000;
             Azangle:= Azangle * 0.005;
          End;

     If ( Ar429word[ 27] = 0) And
        ( Ar429word[ 28] = 0) And
        ( Ar429word[ 29] = 0)
     Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

     AzAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
```

```
                                                   {         1 = forward ant}
                                                   { ARinc: 1,2,3 = ant no}

          cnt:= 0;                                 { extra check possible :}
          For x:= 13 to 28 Do                      { |angle| < propcoverage}
              cnt:= cnt + Ar429word[ x];

          If ( cnt = 0) And ( AR429word[ 29] = 1)
          Then Leftclr:= True
          Else Leftclr:= False;

          If ( cnt = 16) And ( Ar429word[ 29] = 0)
          Then Rightclr:= True
          Else Rightclr:= False;

          If Leftclr Or Rightclr
          Then Azangle_flag:= True
          Else Begin
                    AZangle_flag:= False;

               End;
       End;
     End;


     Procedure BAZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
     {*********************************************************************}
     { This procedure converts the Arinc 429 word containing the backazimuth
       angle information to a Pascal variable.
                                Input : Arinc 429 word
                                Output: Mlsint}
     {*********************************************************************}
     Var
        cnt,
        x   :     Byte;

     Begin
       With Mlsint Do
       Begin
          If ( Ar429word[ 27] = 1) And
             ( Ar429word[ 28] = 1) And
             ( Ar429word[ 29] = 1)
          Then Begin
                  BAzangle:= Conv_bas( Ar429word, 13, 14);
                  BAzangle:= BAzangle - $4000;
                  BAzangle:= BAzangle * 0.005;
               End;

          If ( Ar429word[ 27] = 0) And
             ( Ar429word[ 28] = 0) And
             ( Ar429word[ 29] = 0)
          Then BAzangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

          AzAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
                                                   {         1 = forward ant}
                                                   { ARinc: 1,2,3 = ant no}
          cnt:= 0;
          For x:= 13 to 28 Do
              cnt:= cnt + Ar429word[ x];

          If ( cnt = 0) And ( AR429word[ 29] = 1)
          Then Leftclr:= True
          Else Leftclr:= False;

          If ( cnt = 16) And ( Ar429word[ 29] = 0)
          Then Rightclr:= True
          Else Rightclr:= False;

          If Leftclr Or Rightclr
          Then BAzangle_flag:= True
          Else Begin
                    BAZangle_flag:= False;

               End;
       End;
     End;


     Procedure Discretes_conv( Var Mlsint: MLSinttype;
                                   Ar429word: Ar429wordtype);
     {*********************************************************************}
     { This procedure converts the Arinc 429 word containing the MLS discretes
       to Pascal variables.
                                Input : Arinc 429 word
                                Output: Mlsint}
     {*********************************************************************}
     Begin
          With Mlsint.discretes Do
          Begin
               antenna:= Conv_bas( Ar429word, 11, 2);
               test             := Ar429word[ 13];
               Azsource         := Ar429word[ 14];
               Azselwarn        := Ar429word[ 15];
               Bazselwarn       := Ar429word[ 16];
               GPselwarn        := Ar429word[ 17];
               BAzavail         := Ar429word[ 18];
               BAzdeven         := Ar429word[ 19];
               tuningcom        := Ar429word[ 20];
               nr1antsel        := Ar429word[ 21];
               changeinh        := Ar429word[ 22];
               tunprtsel        := Ar429word[ 23];
          End;
          Mlsint. discretes_flag:=False;
     End;

     Function Hex2String( x: byte): string;
     {**********************************************************************}
     { This function converts a byte to its hexadecimal representation in a
       string. Written by Rob Luxen.
                                Input : Byte;
```

```
                          Output: A string;
  {**************************************************************************}
  Const
      hex: Array[ 0..15] Of Char = '0123456789ABCDEF';

  Var
     i :      Byte;

  Begin
     Hex2String[ 0]:= Chr(2);
     For i:= 0 to 1 Do
     Begin
         hex2String[2-I]:= Hex[ x and $000F];
         x:= x shr 4;
     End;
  End;
  {------------------------end included procedures CollectMLSrec-----------}

Var
  line,
  part          :  String;
  Ar429word     :  Ar429wordtype;
  a_label       :  Byte;
  adress        :  Byte;
  NoWord        :  Boolean;
  time,
  result        :  timetype;
  d,d1          :  Byte;

{------------------------Start procedure CollectMLSrec-------------------}
Begin      { begin  procedure CollectMLSrec}
    MLSint:= tempMLSint;

    Ar429comm.GetAr429word( Ar429word, NoWord, a_label);
                                                  { the label is in dec}
Gotoxy( 1, 13);
If Not ( Noword)
Then Begin
 Str( a_label: 3, part);
 line:= part + ' ';
 For x:= 1 To 32 Do
 Begin
    Str( ar429word[x]:1, part);
    line:= line + part;
 End;
 Write( line, ' ');

 SaveEquipmentMessage( 'RM: ' + line);
End;

          Ar429.GetAr429word( Ar429word, NoWord, a_label);
                                                  { the label is in dec}
If Not ( Noword) Then Begin
Str( a_label: 3, part);
line:= part + ' ';
For x:= 1 To 32 Do
Begin
    Str( ar429word[x]:1, part);
    line:= line + part;
End;
Writeln( line);

SaveEquipmentMessage( 'RM: ' + line);
End;
    With Fifo Do                                         { Get BDW and ADW from}
    Begin                                                { FIFO buffer}
        If Not FifoEmpty                                 { A new word starts with}
        Then Begin                                       { $AA $55. This is not}
                d:= GetFifo;                             { transmitted}
                If d = $AA
                Then Begin
                        If Not FifoEmpty
                        Then Begin
                                d1:= GetFifo;
                                If d1 = $55
                                Then Begin
SaveEquipmentMessage( 'RM: ' + badwline);
                                        badwline:= '';
                                     End
                                Else Begin
                                        badwline:= badwline +
                                                   Hex2String( d) +
                                                   ' ';
                                        badwline:= badwline +
                                                   Hex2String( d1) +
                                                   ' ';
                                     End;
                             End;
                     End
                Else Begin
                        badwline:= badwline +
                                   Hex2String( d) +
                                   ' ';
                     End;
             End;
    End;
    tempMLSint:= MLSint;

End;
{------------------------End procedure CollectMLSrec-------------------}




{------------------------Start procedure Execmlsreccommand------------}
Procedure ExecMLSrecCommand( command: commandtype);

Const
    Prate     =      1000;

```

```
Var
   error          :       Boolean;
   datain         :       Longint;
   oct_lab        :       Byte;
   part,
   line           :       String;
   ar429word      :       ar429wordtype;

Begin
     { room for decoding the command. The result should be a valid
       arinc429 word of 32 bits. This should be coded in a 32 bit longint}

     Ar429.SendAr429word( dataIn, oct_lab, Prate, error);

line:= '';
For x:= 1 To 32 Do
Begin
     Str( ar429word[x]:1, part);
     line:= line + part;
End;
SaveEquipmentMessage( 'SM: ' + line);
End;
{-------------------------End procedure Execmlsreccommand-------------}


{-------------------------Start procedure Closemlsrec-------------}
Procedure CloseMLSrec;

Var
   setupfile      :       Text;
   value          :       String;

Begin
     Ar429comm.CloseAR429;
     Ar429.CloseAr429;
     Remove_ADW_int;

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'MLSBENDIX');

     Str( completeinfo, value);
     Writeln( setupfile, #9'completeinfo = ', value, ';');

     Str( cardaddress, value);
     Writeln( setupfile, #9'cardaddress = ', value, ';');

     Str( irq, value);
     Writeln( setupfile, #9'irq = ', value, ';');

     CloseConfig( setupfile);
End;
{-------------------------End procedure Closemlsrec-------------}


{-------------------------Start initialising----------------------}
Begin
     For x:= 1 To 4 Do
     Begin
          ADW_A_pres[x]:= False;
          ADW_B_pres[x]:= False;
          ADW_C_pres[x]:= False;
     End;
     For x:= 13 To 76 Do
     Begin
          ADW_A[ x]:= 0;
          ADW_B[ x]:= 0;
          ADW_C[ x]:= 0;
     End;

     With TempMLSint Do
     Begin
          Bas1_flag      := True;
          Bas2_flag      := true;
          Bas3_flag      := true;
          Bas4_flag      := true;
          Bas5_flag      := true;
          Bas6_flag      := true;
          auxa1_flag     := true;
          auxa2_flag     := true;
          auxa3_flag     := true;
          auxa4_flag     := true;
          ELangle_flag   := true;
          AZangle_flag   := true;
          BAzangle_flag  := true;
          DME_flag       := true;
          discretes_flag:= true;
          leftclr        := true;
          rightclr       := true;
          flag           := true;

          ErrorTime( Bas1_time);
          ErrorTime( Bas2_time);
          ErrorTime( Bas3_time);
          ErrorTime( Bas4_time);
          ErrorTime( Bas5_time);
          ErrorTime( Bas6_time);
          ErrorTime( AuxA1_time);
          ErrorTime( AuxA2_time);
          ErrorTime( AuxA3_time);
          ErrorTime( AuxA4_time);
          ErrorTime( AuxB_time);
          ErrorTime( AuxC_time);
     End;

     ErrorTime( Tangle);
     ErrorTime( Valid_Tangle);                         { angles are no longer}
     Valid_Tangle.sec100:= 8;                          { valid than 80 msec}

     anglebegin:= False;
     completeinfo:= 1;
```

```
End.
{------------------------End initialising---------------------}
{------------------------End Unit MLSBENDIX---------------------}
```

```
Unit AttBeaver;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InitAttTX( Var error: Boolean);
{*****************************************************************************}
{ This procedure initialises the ATTitude transmitter}
{                            Input : -
                             Output: -}
{*****************************************************************************}

Procedure CollectAtt( Var attdata: attdatatype);
{*****************************************************************************}
{ This procedure collects the attitude angles}
{                            Input : -
                             Output: attitude angles in radians}
{*****************************************************************************}

Procedure ExecAttTxcommand( command: commandtype);
{*****************************************************************************}
{ This procedure passes a command to the Attitude transmitter.
  NotE: This procedure is empty}
{                            Input : -
                             Output: -}
{*****************************************************************************}

Procedure CloseAttTx;
{*****************************************************************************}
{ This procedure restores the ATTitude transmitter to its original state}
{                            Input : -
                             Output: -}
{*****************************************************************************}


Implementation


Uses Miscell, Synchcnv, crt, user;


Const
  pi        =       3.1415926535897932385;


Var
   PitchOffset ,
   RollOffset  :        Double;

{------------------------start procedure initatttx ------------------}
Procedure InitAttTX( Var error: Boolean);

Var
   setupfile    :       Text;
   title,
   varname,
   line         :       String;
   value        :       String;
   code         :       Integer;

Begin
     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                       { find attbeaver part of
                                                    config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'ATTBEAVER'));

     If Not Eof( setupfile)                       { if there is more in file}
     Then Repeat
                Readln( setupfile, line);         { get a line}
                Convert( line, varname, value);   { extract the variable name
                                                    and value}

                If ( varname = 'PITCHOFFSET')
                Then Val( value, pitchoffset, code);

                If ( varname = 'ROLLOFFSET')
                Then Val( value, rolloffset, code);
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
     CloseConfig( setupfile);

     InitSynchcnv( error);
End;
{------------------------end procedure initatttx ------------------}


{------------------------Start procedure collectatt ------------------}
Procedure CollectAtt( Var attdata: attdatatype);

Var
   error        :       Boolean;
   part,
   line         :       String;
   time         :       Timetype;

Begin
     attdata.flag:= True;

     Roll_Pitch_heading( 1, attdata.rollangle, error);
     attdata.flag:= error;
     Roll_Pitch_heading( 2, attdata.pitchangle, error);
     attdata.flag:= attdata.flag Or error;

If Not ( error) Then Begin
Gotoxy( 1, 11);
Str( attdata.rollangle: 6: 2, part);
Write( 'rollangle = ', part);
```

```
line:= part + ' ';

Str( attdata.pitchangle: 6: 2, part);
Write( ' pitchangle = ', part);
line:= line + part + ' ';

SaveEquipmentMessage( 'RA: ' + line);
End;

     attdata.rollangle:= ( attdata.rollangle - rolloffset) * pi / 180;
     attdata.pitchangle:= ( attdata.pitchangle - pitchoffset) * pi / 180;
End;
{-------------------------end procedure collectatt ------------------}


{-------------------------start procedure execatttxcommand-------------}
Procedure ExecAttTxcommand( command: commandtype);

Begin
End;
{-------------------------end procedure execatttxcommand-------------}


{-------------------------start procedure closeatttx-------------}
Procedure CloseAttTx;

Var
   setupfile    :       Text;
   Value        :       String;

Begin
     CloseSynchcnv;

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'ATTBEAVER');

     Str( Pitchoffset, value);
     Writeln( setupfile, #9'pitchoffset = ', value, ';');

     Str( Rolloffset, value);
     Writeln( setupfile, #9'rolloffset = ', value, ';');

     CloseConfig( setupfile);
End;
{-------------------------End procedure closeatttx-------------}

{-------------------------Start initialising-------------------}
Begin
     PitchOffset:= 0;
     RollOffset:= 0;
End.
{-------------------------End initialising-------------------}
{-------------------------End Unit ATTBEAVER-------------------}
```

```
Unit hdgBeaver;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InithdgTX( Var error: Boolean);
{*****************************************************************************}
{ This procedure initialises the hdgitude transmitter}
{*****************************************************************************}

Procedure Collecthdg( Var hdgdata: hdgdatatype);
{*****************************************************************************}
{ This procedure collects the hdgitude data}
{*****************************************************************************}

Procedure ExechdgTxcommand( command: commandtype);
{*****************************************************************************}
{ This procedure passes a command to the hdgitude transmitter.
  NotE: This procedure is empty}
{*****************************************************************************}

Procedure ClosehdgTx;
{*****************************************************************************}
{ This procedure restores the hdgitude transmitter to its original state}
{*****************************************************************************}


Implementation


Uses Miscell, Synchcnv, crt, user;

Const
  pi         =       3.1415926535897932385;

Var
   HdgOffset  :        Double;

{--------------- start procedure inithdgtx --------------------}
Procedure InithdgTX( Var error: Boolean);

Var
   setupfile    :       Text;
   title,
   varname,
   line         :       String;
   value        :       String;
   code         :       Integer;

Begin
     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                           { find headingbeaver part of
                                                        config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'HDGBEAVER'));

     If Not Eof( setupfile)                           { if there is more in file}
     Then Repeat
                Readln( setupfile, line);             { get a line}
                Convert( line, varname, value);       { extract the variable name
                                                        and value}

                If ( varname = 'HDGOFFSET')
                Then Val( value, hdgoffset, code);
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
     CloseConfig( setupfile);

     InitSynchcnv( error);
End;
{--------------- end procedure inithdgtx --------------------}


{--------------- Start procedure collecthdg --------------------}
Procedure Collecthdg( Var hdgdata: hdgdatatype);

Var
   error        :       Boolean;
   part,
   line         :       String;
   time         :       Timetype;

Begin
     hdgdata.flag:= True;

     Roll_pitch_heading( 3, hdgdata.hdgangle, error);

     hdgdata.flag:= error;

If Not ( error) Then Begin
Gotoxy( 40, 11);
Str( hdgdata.hdgangle: 6: 2, part);
Write( 'hdgangle = ', part);
line:= part + ' ';

SaveEquipmentMessage( 'RH: ' + line);
End;
     hdgdata.hdgangle:= ( hdgdata.hdgangle - hdgoffset) * pi / 180;
End;
{--------------- End procedure collecthdg-------------------------}


{--------------- start procedure exechdgtxcommand--------------------}
Procedure ExechdgTxcommand( command: commandtype);

Begin
End;
{--------------- End procedure exechdgtxcommand--------------------}
```

```


{---------------- Start procedure Closehdgtx---------------------}
Procedure ClosehdgTx;

Var
   setupfile    :       Text;
   Value        :       String;

Begin
     CloseSynchcnv;

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'HDGBEAVER');

     Str( Hdgoffset, value);
     Writeln( setupfile, #9'hdgoffset = ', value, ';');

     CloseConfig( setupfile);

End;
{---------------- End procedure Closehdgtx---------------------}


{---------------- start initialising---------------------------}
Begin
     hdgoffset:= 0;
End.
{---------------- End initialising-----------------------------}
{---------------- End unit HDGBEAVE----------------------------}
```

```
Unit Key_Cons;

Interface

Uses MIASglob;

Procedure OpenIn_OutputDev( MIASlogname: String);
{**************************************************************************}
{ opens a file for output.}
{                            Input : -
                             Output: -}
{**************************************************************************}

Procedure GetMessage( Var command: commandtype);
{**************************************************************************}
{ This procedure retreives a message from the keyboard. The procedure will
  return a string as soon as the 'ENTER' key is pressed.
                             Input : keystrokes
                             Output: string with keystrokes}
{**************************************************************************}

Procedure SendMessage( command: commandtype);
{**************************************************************************}
{ This procedure sends a message to the display connected to the computer.
  This is the standard display.
                             Input : string with message
                             Output: string with message on screen.}
{**************************************************************************}

Procedure SendFlags( command: commandtype);
{**************************************************************************}
{ This procedure sends a message ( flags) to a prescribed position on
  the screen.
                             Input : flags
                             Output: flags on the screen}
{**************************************************************************}


Procedure SaveMessage( message: commandtype);
{**************************************************************************}
{ this procedure saves a message to disc.
                             Input : message
                             Output: message on disc}
{**************************************************************************}


Procedure CloseIn_OutputDev;
{**************************************************************************}
{ closes the file for output.}
{                            Input : -
                             Output: -}
{**************************************************************************}

Implementation

Uses Dos, Crt, Miscell;

Const
     Cr          =       #13;

Var
   inputmessage :        commandtype;
   outputfile   :        Text;
   outname      :        String;
   Disknr       :        Byte;
   Free         :        Longint;
   blockused    :        Longint;


Procedure SendMessageToDisplay( command: commandtype);

Begin
     Gotoxy( 1, 21);
     Write( command);
     If ( command[ length( command)] = #10) Or    { LF}
        ( command[ length( command)] = #13)       { CR}
     Then Gotoxy( Length( command), 21)
     Else Gotoxy( Length( command) + 1, 21);
     Write( '                                       ');
End;


Procedure OpenIn_OutputDev( MIASlogname: String);

Begin                                             { init for Key_cons.int}
    ClrScr;
    inputmessage:= '';

    If ( MIASlogname[2] = ':') And ( Length( MIASlogname) >=2)
    Then Case MIASlogname[1] Of
         'a', 'A': Disknr:= 1;
         'b', 'B': Disknr:= 2;
         'c', 'C': Disknr:= 3;
         End
    Else Disknr:= 0;

    SendMessageToDisplay( 'Insert empty disc '+
                          'and press Enter when ready'#10#13+
                          'Disc should contain "MIAS.CFG"');
    Readln;

    outname:= MIASlogname + 'mias.dat';
    Assign( outputfile, outname);                 { defined in MIASglob}

    If FileExist( outname)
    Then Append( outputfile)
    Else Rewrite( outputfile);

    Free:= DiskFree( disknr);
    Blockused:= 0;
```

```
End;


Procedure GetMessage( Var command: commandtype);

Var
   ch   :       Char;

Begin
     If Keypressed
     Then Begin
               gotoxy( 1, 23);
               ch:= Upcase( readkey);

               If ch = #8                       { backspace}
               Then inputmessage:= Copy( inputmessage, 1,
                                   Length( inputmessage) -1)
               Else inputmessage:= inputmessage + ch;
                                                { Enter}
               If inputmessage[ Length( inputmessage)] = CR
               Then Begin
                         command:= Copy( inputmessage, 1,
                                   Length( inputmessage) -1);
                         inputmessage:= '';
                         SendMessage( command);
                    End
               Else command:= '';
               Write( inputmessage);
               Write( '                                              ');
          End
     Else command:= '';
End;


Procedure SendMessage( command: commandtype);

Begin
     SendMessageToDisplay( command);
     SaveMessage( command);
End;


Procedure SendFlags( command: commandtype);

Begin
     Gotoxy( 1, 19);
     Write( command);
End;

Procedure SaveMessage( message: commandtype);

Var
   hour,
   minute,
   sec,
   sec100       :       Word;
   part,
   line         :       String;
   seconds      :       Longint;

Begin
     GetTime( hour, minute, sec, sec100);
     seconds:= hour * 60;
     seconds:= seconds + minute;
     seconds:= seconds * 60;
     seconds:= seconds + sec;
     seconds:= seconds * 100;
     seconds:= seconds + sec100;

     str( seconds: 7, part);
     line:= part + ' ';

     While ( message[ length( message)] = #10) Or
           ( message[ length( message)] = #13) Do
           message:= Copy( message, 1, Length( message) -1);

     Writeln( outputfile, line + message);

     Blockused:= Blockused + Length( line+message) + 2;
     If Blockused >= 10000
     Then Begin
               Flush( outputfile);
               Close( outputfile);
               Append( outputfile);
               Blockused:= 0;
          End;

     Free:= Free - Length( line+message) - 2;
     If (Free <= 512) And ( Free >= 0)
     Then Begin
               Flush( outputfile);
               Close( outputfile);
               SendMessageToDisplay( 'Disc full. Insert empty disc '+
                                     'and press Enter when ready');
               Readln;
               Free:= DiskFree( disknr);
               Assign( outputfile, outname);
               Rewrite( outputfile);
          End;
End;


Procedure CloseIn_OutputDev;

Begin
     Close( outputfile);                         { cannot be checked effect}
                                                 { ively with 'IOresult'}
End;

Begin
```

```
End.
```

```
Unit Synchcnv;
{******************************************************************************}
{ This unit performs interfacing with the synchroconverter card.
  The procedure 'ROLL_PITCH_HEADING' was written by Dennis Willemsen.

  The rest of this unit was written by Marco Meijer}
{******************************************************************************}

{$N+,E+}

Interface

Uses MIASglob;


Procedure InitSynchcnv( Var error: Boolean);
{******************************************************************************}
{ This procedure resets the synchroconverter card.}
{                           Input : -
                            Output: -}
{******************************************************************************}


PROCEDURE roll_pitch_heading (number:INTEGER;VAR angle:Double;
                              VAR error:BOOLEAN);
{******************************************************************************}
{ This procedure instructs the synchroconverter card to output the angle
  specified by the number: nummber. The angle is output in the variable
  hoek. The angle is output in degrees.
                            Input : angle number
                            output: angle}
{******************************************************************************}


Procedure CloseSynchcnv;
{******************************************************************************}
{ This procedure restores the synchroconverter card to its original state.
                            Input : -
                            Output: -}
{******************************************************************************}


Implementation


{*******************************************************************************}
{* This function replaces the bits, which means that bit 8 will switch        *}
{* place with bit 1, bit 7 will switch place with bit 2, etc. This is         *}
{* done by first determining the value of the bit and then multiply it        *}
{* with the value of the place it should be on. The new value of the byte *}
{* is obtained by adding all the separate value's.                            *}
{*******************************************************************************}


FUNCTION shift (old:BYTE):BYTE;

VAR  new, conversion     :    BYTE;
     power, p, weight    :    INTEGER;

BEGIN
   weight := 1;
   power := 128;
   new := 0;
   shift := 0;
   conversion := 0;
   FOR p := 1 to 8 DO
   BEGIN
      new :=((old AND weight) * power) DIV weight;
      conversion := conversion + new;
      power := power DIV 2;
      weight := weight * 2;
   END;
   shift := conversion;
END;


Procedure InitSynchcnv( Var error: Boolean);

Var
   result       :       Boolean;
   x            :       Integer;
   hoek         :       Double;

Begin
     result:= True;
     For x:= 1 To 3 Do
     Begin
          roll_pitch_heading( x, hoek, error);
          result:= result And error;
     End;
     error:= result;
End;


{*******************************************************************************}
{* This procedure gives the value's of one of three synchro's. The dif- *}
{* ferent synchro's are selected by addressing the procedure with the   *}
{* numbers 1,2 or 3. It will reset the synchro's and it will read each  *}
{* byte 2 times, so that error tests can be performed. The bytes that   *}
{* are read first have to be shifted and this is done by the function   *}
{* shift. The hardware card is addressed by the addresses $300-$30F. The*}
{* stepsize is obtained by deviding 360 by (2^16 - 1).                  *}
{*******************************************************************************}

PROCEDURE roll_pitch_heading (number:INTEGER;VAR angle:Double;
                              VAR error:BOOLEAN);

CONST
  roll1    = $300;
  roll2    = $308;
```

```
  pitch1   = $304;
  pitch2   = $30C;
  heading1 = $302;
  heading2 = $30A;
  reset    = $30F;
  stepsize = 5.493080245e-3;

VAR
  b                          :   INTEGER;
  amount_of_steps            :   WORD;
  lsb1, msb1, msb, lsb, rsb,
  lsbc, msbc                 :   BYTE;

BEGIN
  angle := 0;
  amount_of_steps := 0;
  rsb := PORT[reset];
  CASE number OF
  1: BEGIN
       rsb  := PORT[reset];
       msb1 := PORT[roll1];
       lsb1 := PORT[roll2];
       msbc := PORT[roll1];
       lsbc := PORT[roll2]
     END;
  2: BEGIN
       rsb  := PORT[reset];
       msb1 := PORT[pitch1];
       lsb1 := PORT[pitch2];
       msbc := PORT[pitch1];
       lsbc := PORT[pitch2]
     END;
  3: BEGIN
       rsb  := PORT[reset];
       msb1 := PORT[heading1];
       lsb1 := PORT[heading2];
       msbc := PORT[heading1];
       lsbc := PORT[heading2]
     END;
  END;

       lsb := shift(lsb1);
       msb := shift(msb1);
       amount_of_steps := (lsb + msb * 256);
       angle := amount_of_steps * stepsize;


  {*************************************************************************}
  {* This is where the error-testing starts. First the values of the     *}
  {* bytes are tested on their correctness. Of the lsb1 only the 7 least *}
  {* significant bits are compared, because the 8th bit alters al the    *}
  {* time, this is due to the accuracy of the chip. Also the amount of   *}
  {* steps is checked as wel as the angle.                               *}
  {*************************************************************************}

  lsbc := lsbc AND 127;
  lsb1 := lsb1 AND 127;
  IF (lsb = lsbc) AND (msb = msbc)
     THEN error := FALSE
     ELSE error := TRUE;

  IF ((amount_of_steps < 0) OR (amount_of_steps > 65535))
     AND (error = FALSE)
  THEN error := TRUE
  ELSE error := FALSE;

  IF ((angle < 0) OR (angle >= 360)) AND (error = FALSE)
  THEN error := TRUE
  ELSE error := FALSE;

END;



Procedure CloseSynchcnv;

Begin
End;

End.
```

```
Unit AR429;


Interface

Type
    AR429wordtype       =       Array[1..32] Of Byte;
    arraytype           =       Array[ 0..255] Of Byte;

Procedure InitAR429( Var error: Boolean; selecttable: arraytype; num: Integer);
{***********************************************************************}
{ This procedure installs the ARINC 429 handler. It catches the transmissions
  from the external ARINC 429 channel. The label numbers to be caught should
  be programmed in selecttable in octal. The number of labels should be
  passed with num. Error indicates, that something went wrong during
  initialising of the receiver.
                              Input : selecttable
                                      number of labels in table
                              Output: error}
{***********************************************************************}


Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
                        Var a_label: Byte);
{***********************************************************************}
{ This procedure delivers one ARINC 429 word. If no word is available,
  the NoWord-flag will be set True. The label will be output also. If
  no word is available, the word is set to all zero's and a_label = 0.
                              Input : -
                              Output: Arinc 429 word
                                      NoWord flag
                                      adress label}
{***********************************************************************}


Procedure SendAr429word( dataIn: Longint; oct_lab: Byte; Prate: Word;
                         Var error: Boolean);
{***********************************************************************}
{ This procedure sends a command to an external ARINC 429 channel.
                              Input : ARINC 429 word
                              Output: ARINC 429 word on 429 channel}
{***********************************************************************}


Procedure CloseAR429;
{***********************************************************************}
{ This procedure makes sure, the computer and peripherals are in their
  starting state.
                              Input : -
                              Output: -}
{***********************************************************************}


Implementation

Uses Lib429P;

Const
    Pspeed      =       S100;                   { 100 Kbits per sec}
    Pgap        =       G4_00;                  { gap time of 4 bittimes}
                                                { NOTE: LABELS ARE OCTAL}
    ssm         =       0;                      { sign status matrix}
    sdi         =       0;                      { source destination indicator}
    max429buf   =       100;
    parity      =       Podd;
    on          =       1;
    off         =       0;


Var
  nextfreeslot :        Byte;
  arinc429buffer:       Array[0..Max429buf-1] Of Longint;
  inp           :       Integer;
  outp          :       integer;
  arTxChannel   :       Byte;
  ptr1,
  ptr2          :       Byte;

Procedure InitAR429( Var error: Boolean; selecttable: arraytype; num: Integer);

Var
  x             :       Byte;
  Rspeed,
  Rgap          :       Integer;
  readtable     :       arraytype;

Begin
  error:= False;

  arSetBase($D000, 0, 0, 0);                    { set card base address}
  arReset(0);                                   { Turn OFF all channels,
                                                  all blocks and all
                                                  receiver modes}
                                                { INITIALISE TRANSMITTER}
  arTxSetSpeed( 0, Pspeed, Pgap);               { set transmitter speed}
  arTxReadSpeed( 0, Rspeed, Rgap);

  If ( Pspeed <> Rspeed) Or ( Pgap <> Rgap)     { verify transmitter speed}
  Then Begin
          error:= True;
          Exit;
     End;

  arTxSetParity( 0, Pnone);                     { set transmitter channel}
                                                { parity to none, to allow}
                                                { other data to overide}
  nextfreeslot:= 0;
  arTxChannel := off;

                                                { INITIALISE RECEIVER}
```

```
    arRxStop( 0);                               { stop receiver}
    arRxSpeedHigh( 0);                          { set receiver on high speed}

    For x:= 0 To num -1 Do
        selecttable[ x]:= oct( selecttable[ x]);  { convert labels to decimal}

    arRxSelectSet( 0, @Selecttable, num);       { select the labels to be read}

    arRxSelectRead( 0, @ReadTable);             { verify the labels to be read}

    For x:= 0 To num -1 Do                      { check the read labels}
         If readtable[ selecttable[ x]] = 0     { with respect to programmed}
         Then Begin                             { labels}
                  error:= True;
                  Exit;
              End;

    arRxConfigWrite( 0, circular, shared, 42, nosync, 0, 0, 0);
                                                { configure the receiver}
    arRxStatTrig( 0);                           { start the receiver}
    ptr1:= arRxGetPtr( 0);

    outp:=0;
    inp:= 0;
End;


Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
                        Var a_label: Byte);

Type
    stat_buffer =       Record
                              data:     Longint;
                              tagtime:  word;
                        End;

Var
   x           : Integer;
   arincw      : Longint;
   Realarincw  : Real;
   temp        : Real;
   time        : word;


Begin
     ptr2:= arRxGetPtr( 0);
     While (ptr1 <> ptr2) Do
     Begin
          ptr1:= arRxBuffer( 0, ptr1, arinc429buffer[ inp], time);
          inp:= ( inp + 1 ) Mod max429buf;
     End;

     If outp <> Inp                             { if word in buffer}
     Then Begin
                arincw:= arinc429buffer[ outp];   { get word}
                outp:= (outp + 1 ) Mod max429buf; { increment outpointer}

                a_label:= arLabel( arincw);       { get label}


                realarincw:= arincw;
                temp:= 1;                         { decode longint to bits}
                For x:= 1 to 32 do
                    temp:= temp * 2;

                If arincw < 0
                Then realarincw:= realarincw + temp;{ make positive}

                temp:= temp / 2;
                For x:= 32 DownTo 1 Do            { 32 bits}
                Begin
                    ar429word[ x]:= Trunc( realarincw / temp);
                    If ar429word[x] > 0           { test on specific bit}
                    Then realarincw:= Round( realarincw - temp);
                    temp:= Trunc (temp / 2);      { select next bit}
                End;
                noword:= False;
           End
      Else noword:= True;

End;


Procedure SendAr429word( dataIn: Longint; oct_lab: Byte; Prate: Word;
                         Var error: Boolean);

Var
   Rstatus     :      Byte;
   Rrate       :      Word;
   Rarincw     :      longint;
   Rlab        :      integer;
   Rdata       :      longint;
   Rssm,
   Rsdi,
   Rpar        :      Integer;
   Parincw     :      Longint;
   Pstatus     :      Byte;
   done        :      Boolean;
   x           :      Integer;

Begin
   arCompose( Parincw, oct( oct_lab), datain, ssm, sdi, parity);

   If ( parity = Podd)                          { program the status mask}
   Then Pstatus:= $00                           { for parity}
   Else Pstatus:= $01;

   Pstatus:= Pstatus Or $80;                    { program status mask for}
                                                { 'slot on'}
   error:= False;
```

```
   done:= False;                              { ready inserting word?}
   For x:= 0 To nextfreeslot -1 Do            { check all used slots for}
   Begin
        arTxReadSlot( 0, x, Rstatus, Rrate, Rarincw);
        arDecompose( Rarincw, Rlab, Rdata, Rssm, Rsdi, Rpar);

        If ( oct( oct_lab) = Rlab)            { labels already in use}
        Then Begin
                  arTxSetdata( 0, x, Parincw);   { update these labels}
                  arTxSetRate( 0, x, Prate);
                  arTxReadSlot( 0, x, Rstatus, Rrate, Rarincw);

                  done:= True;
                  If ( (Rstatus And Pstatus) <> Pstatus) Or
                                              { verify transmitter}
                     ( Rrate <> Prate) Or       { contents}
                     ( Rarincw <> Parincw)
                  Then Begin
                            error:= True;
                            Exit;
                       End;
             End;
   End;

   If done = False
   Then Begin
           arTxSetdata( 0, nextfreeslot, Parincw);{ put data in next free slot}
                                              { on the channel}
           arTxSetRate( 0, nextfreeslot, Prate); { set the transmit rate for}
                                              { slot 0}
           arTxSlotOn( 0, nextfreeslot);        { enable slot 0}

           arTxReadSlot( 0, nextfreeslot, Rstatus, Rrate, Rarincw);

           If ( (Rstatus And Pstatus) <> Pstatus) Or
              ( Rrate <> Prate) Or              { verify transmitter}
              ( Rarincw <> Parincw)             { contents}
           Then Begin
                     error:= True;
                     Exit;
                End;

           If ( arTxChannel = off)
           Then Begin
                     arTxChannelOn( 0);          { turn channel on}
                     arTxChannel:= On;
                End;

           nextfreeslot:= (nextfreeslot + 1) Mod 63;
                                              { if 63 slots active, over-}
                                              { write slot 0}
        End;
End;


Procedure CloseAR429;

Begin
     arReset( 0);                             { turn off channel 0}
                                              { all slots and all receiver}
                                              { modes}
End;


Begin
End.
```

```
Unit AR429comm;
{***************************************************************************}
{ This unit contains routines for decoding and interfacing with an Arinc
  429 datastream. The datastream is input using a comport}
{***************************************************************************}

Interface

Uses AR429;{ This statement is placed here instead of:"
                 Type AR429wordtype = Array[1..32] Of Byte;"
                 This statement caused compile time errors due to
                 duplicate type declarations with the same name.
                 The first declaration is done in AR429comm.
                 If AR429comm is not used, replaced the statement}


Procedure InitAR429;
{***************************************************************************}
{ This procedure installs the ARINC 429 handler. It catches the transmissions
  from the external ARINC 429 channel.
                             Input : -
                             Output: -}
{***************************************************************************}


Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
                        Var a_label: Byte);
{***************************************************************************}
{ This procedure delivers one ARINC 429 word. If no word is available,
  the NoWord-flag will be set True. The label will be output also. If
  no word is available, the word is set to all zero's and a_label = 0.
                             Input : -
                             Output: Arinc 429 word
                                     NoWord flag
                                     adress label}
{***************************************************************************}


Procedure SendAr429word( word: AR429wordtype);
{***************************************************************************}
{ This procedure sends a command to an external ARINC 429 channel.
                             Input : ARINC 429 word
                             Output: ARINC 429 word on 429 channel}
{***************************************************************************}


Procedure CloseAr429;
{***************************************************************************}
{ This procedure makes sure, the computer and peripherals are in their
  starting state.
                             Input : -
                             Output: -}
{***************************************************************************}


Implementation

Uses MIASglob, Miscell, com_4; {comdisc}

Var
   mlsport             :       Byte;


Procedure InitAR429;

Var
   setupfile    :       Text;
   title,
   line         :       String;
   varname,
   value        :       String;
   code         :       Integer;

Begin
     MLSport    :=     3;                          { MLSport is com 3}

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                        { find AR429COMM part of
                                                     config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'AR429COMM'));

     If Not Eof( setupfile)                        { if there is more in file}
     Then Repeat
                Readln( setupfile, line);          { get a line}
                Convert( line, varname, value);    { extract the variable name
                                                     and value}
                If ( varname = 'MLSPORT')
                Then Val( value, mlsport, code);
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
                                                   { repeat until end of file}
     CloseConfig( setupfile);
                                                   { initialise comports for
                                                     communication with Engine}
     Setupcomport( mlsport, Ord( B2400), 8, Ord( None), 1);
                                                   { empty receive and trans-
                                                     mit buffers}
     Emptybuffer( mlsport, True);
                                                   { set interrupt vectors}
     Installint( mlsport);                         { save old interrupt}
End;


Procedure GetAr429word( Var ar429word: AR429wordtype; Var NoWord: Boolean;
                        Var a_label: Byte);

Var
   x,
   y    :       Byte;
   dum  :       Byte;
```

```
  bitsum,
  weight      :       Byte;

Begin
     If Charsinbuff( mlsport) < 4                { if not enough bytes in the}
     Then Begin                                  { buffer than deliver a clean}
               For x:= 1 To 32 Do                { AR429 word and exit}
                   Ar429word[ x]:= 0;
               NoWord:= True;
               a_label:= 0;
               Exit;
          End;

     For x:= 1 To 4 Do
     Begin
          dum:= Byte( GetCharbuff( mlsport));    { take a byte from the buffer}
          For y:= 1 To 8 Do
              Ar429word[ (x-1) * 8 + (9 - y)]:= (dum Shr ( y -1)) And $01;
                                                  { take the bits from a byte
                                                    from the buffer}
     End;

     bitsum:= 0;                                 { check the parity of the}
     For x:= 1 To 31 Do                          { arinc word. It is odd }
         bitsum:= bitsum + AR429word[x];         { parity. Bit 32 is the }
     bitsum:= bitsum Mod 2;                      { paritybit.}

     If bitsum = Ar429word[ 32]
     Then Begin                                  { if bad parity, clear the}
               For x:= 1 To 32 Do                { AR429 word and exit}
                   Ar429word[ x]:= 0;
               a_label:=0;
               NoWord:= True;
               Exit;
          End
     Else Begin
               NoWord:= False;                   { if the parity was ok,}
                                                 { the NoWord becomes False}
               a_label:= 0;                        { determine the label}
               weight:= 128;                     { for the weights of the bits}
               For x:= 1 To 8 Do
               Begin
                   a_label:= a_label + Ar429word[x] * weight;
                   weight:= weight Div 2;
               End;
          End;
End;


Procedure SendAr429word;

Begin
End;


Procedure CloseAr429;

Var
   setupfile    :       Text;
   Value        :       String;

Begin
     Removeint( mlsport);

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'AR429COMM');

     Str( mlsport, value);
     Writeln( setupfile, #9'mlsport  = ', value, ';');

     CloseConfig( setupfile);
End;


End.
```

```
{$A+,B-,D+,E+,F-,G+,I+,L+,N+,O-,R+,S+,V+,X-}
{$M 16384,0,655360}
Unit ADW;

(****************************) interface (**********************************)

uses
  Intrupt;

const
  BufferLengte =  1024;
  TO_Tijd      = 50000; {timeout tijd}

type
  FifoOBJ = object
    Kop, Staart         : word;
    Buffer              : array [0..BufferLengte] of byte;
    function  FIFOfull  : boolean;
    function  FIFOempty : boolean;
    function  GetFIFO   : byte;
    procedure ResetFIFO;
    procedure PutFIFO (Data : byte);
  end;

var
  FIFO    : FifoOBJ;
  TimeOut : boolean;

procedure InitKaartAdres  (Adres   : word);
procedure ProgTrigFunktie (TrigNum : byte; Funktie : byte);
procedure ResetTrigger    (TrigNum : byte);
procedure ResetTriggers;
procedure IRQ_Aan;
procedure IRQ_Uit;
procedure ResetIntLatch;
procedure Install_ADW_Int;
procedure Remove_ADW_Int;
procedure KiesIRQ (IRQ : byte);

function  TrigFunktie (TrigNum : byte) : byte;

(**************************) implementation (*******************************)

uses
  Crt, DOS;

const
  MaxFunkties  = 9;
  IRQ_Bit      = $80; {aan/uit hardware-interrupt, alleen funktie 1}
  FE_Bit       = $80; {aan/uit funktie, voor funkties 2 t/m 9}
  KaartAdres : word = $280;
(*
Adres van een triggerfunktie is gelijk aan het kaartadres plus het
trigger-funktienummer min 1.
*)
  oFunktie_FF  =   9; {offset t.o.v. kaartadres}
  oData_FF     =  10; {offset t.o.v. kaartadres}
  oReset       =  11; {offset t.o.v. kaartadres}
(*
Bitmasker voor uitlezen Funktie_FlipFlop.
*)
  mFunktie     = $1F; {funktiebits}
  mFunkPar     = $60; {pariteitbits van funktie}
  mKlok        = $40; {klok van schuifregister}
  mPreamble    = $80; {indikatie aanwezigheid gewenste preamble}


var
  Trigger : array [1..MaxFunkties] of byte;
  IRQlijn : byte;

procedure ResetIntLatch;

begin
  Port [KaartAdres+oReset] := 0;
end;


procedure KiesIRQ (IRQ : byte);

begin
  if (IRQ >= 3) and (IRQ <= 7)
  then IRQlijn := IRQ;
end;


{======================= INTERRUPT PROCEDURE =============================}

procedure ADW_Handler; interrupt;
(*
Er is een preamble gedetekteerd. Schakel de IRQ-lijn af om eventuele
triggering op data te voorkomen. Lees vervolgens de ontvangen funktiebits
uit de Data_FlipFlop en bepaal hieruit hoeveel bits nog zullen volgen.
*)
const
  L_BasicDataWord = 32;
  L_AuxDataWord   = 76;

var
  Funktie    : word;
  Data       : word;
  KlokTeller : word;
  Kloknivo   : word;
  Bitteller  : word;
  Klaar      : boolean;
  TO_Teller  : longint;
  MaxBits    : word;

begin
  DisableInterrupt;
```

```
  Funktie  := Port [KaartAdres+oFunktie_FF];
  KlokNivo := (Funktie and mKlok);                    {onthoud kloknivo}
  Data     := Port [KaartAdres+oData_FF];

  case (Funktie and mFunktie) of

    $0A, $1E, $05, $11, $1B, $18 : MaxBits := L_BasicDataWord;
                                   { basic data woorden 1 t/m 6 }
    $07, $15, $0F                : MaxBits := L_AuxDataWord;
                                   { auxiliary data woorden A t/m C }
    else MaxBits := 0
  end; {case}

  TimeOut   := false;
  TO_Teller := TO_Tijd;

  if MaxBits > 0
  then begin
    with Fifo do
    begin
      if not FifoFull
      then PutFifo (Funktie and mFunktie);     {funktie identifikatie > fifo}

      if not FifoFull
      then PutFifo (Data);                              {volgend byte > fifo}

      repeat                                   {herhalen tot kloknivo laag is}
        repeat                            {herhalen tot nivo-verandering van klok}
          Funktie := Port [KaartAdres+oFunktie_FF];
(*
          if TO_Teller > 0 then Dec (TO_Teller);
          TimeOut := TO_Teller = 0;
*)
        until (KlokNivo <> (Funktie and mKlok)) or TimeOut;
        KlokNivo := (Funktie and mKlok);                {verwerk verandering}
      until (KlokNivo = 0) or TimeOut;
(*
Om de volgende 8 bits in de Data_Flipflop te krijgen moeten 8 klokpulsen
geteld worden. Daarna kunnen ze uitgelezen worden en naar de fifo gestuurd
worden.
*)
      KlokTeller := 21;                    {aantal reeds ontvangen klokpulsen}
      Klaar      := false;

      while not (Klaar or TimeOut) do
      begin
        BitTeller := 1;
        repeat                         {herhalen tot volledig byte is ingeschoven}
          repeat                                 {herhalen tot kloknivo laag is}
            repeat                        {herhalen tot nivo-verandering van klok}
              Funktie  := Port [KaartAdres+oFunktie_FF];
(*
              if TO_Teller > 0 then Dec (TO_Teller);
              TimeOut := TO_Teller = 0;
*)
              until (KlokNivo <> (Funktie and mKlok)){ or TimeOut};
              KlokNivo := (Funktie and mKlok);          {verwerk verandering}

            until (KlokNivo = 0) {or TimeOut};
            Inc (BitTeller);                        {er is een bit ingeschoven}
            Inc (KlokTeller);
            Klaar := KlokTeller = MaxBits;          {test op einde bitstroom}

          until (BitTeller mod 8 = 0) or        {volledig byte ingeschoven of}
                Klaar {or TimeOut};             {maximum aantal bits ontvangen}

          if not FifoFull                              {schrijf byte naar fifo}
          then PutFifo (Port [KaartAdres+oData_FF]);
        end;

        if not FifoFull then PutFifo ($AA);
        if not FifoFull then PutFifo ($55);
      end; {with Fifo do}
    end

    else begin                   {onjuiste triggering, wacht tot oorzaak weg is}
      repeat
        Funktie := Port [KaartAdres+oFunktie_FF];

        if TO_Teller > 0 then Dec (TO_Teller);
        TimeOut := TO_Teller = 0;

      until (Funktie and mPreamble = 0) or TimeOut;
    end;

    ResetIntLatch;                                      {reset interrupt latch}
    EnableInterrupt;                               {interrupts weer toegestaan}
    ResetInterrupt;                             {reset hardware interrupt chip}
end;


{============================ FIFO PERIKELEN ============================}

procedure FifoOBJ.ResetFIFO;

begin
  DisableInterrupt;
  Kop    := 0;
  Staart := 0;
  EnableInterrupt;
end;


function FifoOBJ.FIFOempty : Boolean;

begin
  DisableInterrupt;
  FIFOempty := Kop = Staart;
  EnableInterrupt;
end;
```

```


function FifoOBJ.GetFIFO : byte;

begin
  DisableInterrupt;
  GetFIFO := Buffer [Staart];
  Inc (Staart);
  if Staart = BufferLengte then Staart := 0;
  EnableInterrupt;
end;


procedure FifoOBJ.PutFIFO (Data : byte);

begin
  Buffer [Kop] := Data;
  Inc (Kop);
  if Kop = BufferLengte then Kop := 0;
end;


function FifoOBJ.FIFOfull : Boolean;

begin
  if Kop = BufferLengte - 1
  then FIFOfull := 0 = Staart
  else FIFOfull := (Kop + 1) = Staart;
end;

{======================== KAART INSTELLEN ================================}

procedure InitKaartAdres (Adres : word);

begin
  KaartAdres := Adres and $03F0; {kaart beslaat 16 adressen}
end;


procedure ProgTrigFunktie (TrigNum : byte; Funktie : byte);

begin
  if TrigNum <= MaxFunkties
  then begin
    Funktie := Funktie and (mFunktie or mFunkPar);

    if TrigNum = 1
    then Trigger [TrigNum] := Trigger [TrigNum] and not
                              (mFunktie or mFunkPar) or
                               Funktie  {IRQ-bit ontzien}

    else Trigger [TrigNum] := Funktie or FE_Bit;

    Port [KaartAdres+TrigNum-1] := Trigger [TrigNum];
  end;
end;


function TrigFunktie (TrigNum : byte) : byte;

begin
  TrigFunktie := Trigger [TrigNum];
end;


procedure ResetTrigger (TrigNum : byte);

begin
  if TrigNum <= MaxFunkties
  then begin
    if TrigNum = 1
    then Trigger [TrigNum] := Trigger [TrigNum] or $7F      {IRQ-bit ontzien}
    else Trigger [TrigNum] := $7F;
    Port [KaartAdres+TrigNum-1] := Trigger [TrigNum];
  end;
end;


procedure ResetTriggers;

var
  n : byte;

begin
  for n := 1 to MaxFunkties do ResetTrigger (n);
end;


procedure IRQ_Aan;

begin
  Trigger [1]        := Trigger [1] or IRQ_bit;
  Port [KaartAdres] := Trigger [1];
end;


procedure IRQ_Uit;

begin
  Trigger [1]        := Trigger [1] and not IRQ_bit;
  Port [KaartAdres] := Trigger [1];
end;

{==========================================================================}

procedure Install_ADW_Int;

begin
  if IRQlijn <> 0
  then begin
```

```
    SetInterrupt (IRQlijn, ON, Addr (ADW_Handler));
    ResetTriggers;
    IRQ_Aan;
    ResetIntLatch;                                      {reset interrupt latch}
  end;
end;


procedure Remove_ADW_Int;

begin
  SetInterrupt (IRQlijn, OFF, Addr (ADW_Handler));
  IRQ_Uit;
  ResetIntLatch;                                        {reset interrupt latch}
end;


{============================== INITIALISATIE ==============================}

begin
  IRQlijn := 0;
  Fifo.ResetFifo;
  IRQ_Uit;
end.
```

```
{======================================================================}
{                                                                      }
{                   unit:  INTERRUPT HANDLING                          }
{                          ==================                          }
{                                                                      }
{  File: INTRUPT.PAS                                                   }
{                                                                      }
{  This unit contains functions to handle interrupts                   }
{                                                                      }
{                                                     feb 88           }
{                                                                      }
{======================================================================}

{$D-,R-,S-,V-}
UNIT Intrupt;

INTERFACE
{**********************************************************************}

USES Dos;

CONST
  On  = true;
  Off = false;

PROCEDURE EnableInterrupt;  Inline ($FB);
PROCEDURE DisableInterrupt; Inline ($FA);
PROCEDURE ResetInterrupt;   Inline ($B0/$20/$E6/$A0/$E6/$20);
PROCEDURE SetInterrupt      (IRQ: byte; ON: boolean; VEKTOR: pointer);
PROCEDURE ExeOldInterrupt   (IRQ: byte);

IMPLEMENTATION
{**********************************************************************}

const
  IntNrs = 15;
  Int : array [0..IntNrs] of byte = ($08,$09,$0A,$0B,$0C,$0D,$0E,$0F,
                                     $70,$71,$72,$73,$74,$75,$76,$78);
var
  IRQPTR : array [0..IntNrs] of pointer;
  n      : byte;

{----------------------------------------------------------------------}
{ PROCEDURE EnableInterrupt;  Inline ($FB);                            }
{----------------------------------------------------------------------}
{ in:  -                                                               }
{ out: -                                                               }
{ rem: This is the machine instruction STI Set Interrupt Flag          }
{----------------------------------------------------------------------}

{----------------------------------------------------------------------}
{ PROCEDURE DisableInterrupt; Inline ($FA);                            }
{----------------------------------------------------------------------}
{ in:  -                                                               }
{ out: -                                                               }
{ rem: This is the machine instruction CLI Clear Interrupt Flag        }
{----------------------------------------------------------------------}

{----------------------------------------------------------------------}
{ PROCEDURE ResetInterrupt; Inline ($B0/$20/$E6/$A0/$E6/$20);          }
{----------------------------------------------------------------------}
{ in:  -                                                               }
{ out: -                                                               }
{ rem: This is a macro for the following machine instructions:         }
{              MOV  AL,20h                                             }
{              OUT  0A0h,AL    clear slave  interrupt unit 8259        }
{              OUT  020h,AL    clear master interrupt unit 8259        }
{         The master interrupt handles hardware interrupts IRQ0..IRQ7  }
{         The slave  interrupt handles hardware interrupts IRQ8..IRQ15 }
{         The slave chip is NOT present into IBM-XT                    }
{         ResetInterrupt must be the last instruction in a hardware    }
{         interrupt serving procedure.                                 }
{----------------------------------------------------------------------}

PROCEDURE SetInterrupt (IRQ: byte; ON: boolean; VEKTOR: pointer);

{----------------------------------------------------------------------}
{ in:  IRQ - the number 0..15 of the interrupt request line            }
{      ON  - TRUE:  enable  hardware interrupt IRQ                     }
{            FALSE: disable hardware interrupt IRQ                     }
{ out: -                                                               }
{ rem: This call must be used to enable user interrupts at initalizing }
{      time, and must be used as well to disable user interrupts       }
{      before ending the program. The following hardware is present:   }
{                                                                      }
{   Line    Interrupt    PC - XT              AT - PS/2(50,60)         }
{   ---------------------------------------------------------------    }
{   IRQ0    Int $08      Timer                Timer                    }
{   IRQ1        $09      Keyboard             Keyboard                 }
{   IRQ2        $0A      Vert. retrace EGA    cascade interrupts 8..15 }
{   IRQ3        $0B      COM2                 COM2                     }
{   IRQ4        $0C      COM1                 COM1                     }
{   IRQ5        $0D      Fixed Disk adapter   Printer 2                }
{   IRQ6        $0E      Diskette adapter     Diskette adapter         }
{   IRQ7        $0F      Printer  adapter     Printer 1                }
{                                                                      }
{   IRQ8        $70        --                 Real time clock chip     }
{   IRQ9        $71        --                 Reserved, NetWork        }
```

```
{   IRQ10       $72         --                  COM3                        }
{   IRQ11       $73         --                  COM4                        }
{   IRQ12       $74         --                  Reserved (PS/2: Mouse)      }
{   IRQ13       $75         --                  80x87 co processor          }
{   IRQ14       $76         --                  Fixed Disk adapter          }
{   IRQ15       $78         --                  Reserved                    }
{-----------------------------------------------------------------------------}

VAR
  NewInt : word;
  Mask : record
    case boolean of
      TRUE : (All              : word);
      FALSE: (LowByte,HighByte: byte);
    end;

BEGIN
  DisableInterrupt;                        { no interrupts while changing  }
  NewInt := 1 SHL IRQ;                     { set bit according to value    }
  Mask.HighByte := Port[$A1];              { all enabled slave  interrupts }
  Mask.LowByte  := Port[$21];              { all enabled master interrupts }

  if ON and                                { IF NEW INTERRUPT ENABLED and  }
    (IRQPTR [IRQ] = nil)                   { not enabled before            }
  then begin
    NewInt   := NewInt XOR $FF;            { swap all bits                 }
    Mask.All := Mask.All AND NewInt;       { clear the new bit in Mask     }
    GetIntVec (Int [IRQ], IRQPTR [IRQ]);   { onthoud huidige pointer       }
    SetIntVec (Int [IRQ], Vektor);         { pointer naar programma        }
  end
  else begin                               { IF INTERRUPT DISABLED         }
    if IRQPTR [IRQ] <> nil                 { and interrupt was saved       }
    then begin
      SetIntVec (Int [IRQ], IRQPTR [IRQ]);{ herstel originele pointer     }
      IRQPTR [IRQ] := nil;                 { geen interrupt meer bewaard   }
      Mask.All := Mask.All OR NewInt;      { set the new bit in Mask       }
    end;
  end;

  Port[$A1] := Mask.HighByte;              { set new slave mask            }
  Port[$21] := Mask.LowByte;               { set new master mask           }
  EnableInterrupt;                         { interrupts back on            }
END; { SetInterrupt }


{$F+}
PROCEDURE ExeOldInterrupt (IRQ: byte);

var
  Ptr : pointer;

begin
  if IRQPTR [IRQ] <> nil
  then begin
    Ptr := IRQPTR [IRQ];
    inline ($9C);           { PushF }
    inline ($FF/$1E/Ptr);   { Call Far [oude interrupt] }
  end;
end;
{$F-}


BEGIN
  for n := 0 to IntNrs do IRQPTR [n] := nil; { maak alle pointers nil }
END.
```

```
Unit miscell;

Interface

Const
     zeros       =       '000000000000000000000000000000';

Type
   timetype     =       Record
                          year,
                          month,
                          day,
                          hour,
                          minute,
                          sec,
                          sec100    :      Word;
                        End;

Procedure Convert( line: String; Var varname, value: String);
{*************************************************************************}
{ This procedure converts a line from the MIAS.CFG file. It outputs
  the variable name and the value in upcase. A line should begin with one
  or more spaces, then the variable name, followed by '=', terminated by
  the value in ascii and a semicolon.
                              Input : line from CFG file
                              Output: variablename
                                      value}
{*************************************************************************}


Procedure OpenConfigRead( Var setupfile: Text; filename: string);
{*************************************************************************}
{ This procedure opens the config-file for reading.
                              Input : -
                              Output: file of Text}
{*************************************************************************}


Procedure CloseConfig( Var setupfile: Text);
{*************************************************************************}
{ This procedure closes the config-file.
                              Input : file of Text
                              Output: -}
{*************************************************************************}


Procedure OpenConfigWrite( Var setupfile: Text; filename: String);
{*************************************************************************}
{ This procedure appends the config-file for writing.
                              Input : filename
                              Output: file of Text}
{*************************************************************************}


Procedure OpenConfigWriteFirst( Var setupfile: Text; filename: String);
{*************************************************************************}
{ This procedure opens the config-file for writing.
                              Input : filename
                              Output: file of Text}
{*************************************************************************}


Procedure Date_and_Time( Var time: timetype);
{*************************************************************************}
{ Fills the input variable with the current system date and time.
                              Input : -
                              Output: system date and time}
{*************************************************************************}


Procedure ErrorTime( Var time: timetype);
{*************************************************************************}
{ This procedure sets the time variable to 0,0,0,0,0 to indicate an error
  in the data it represents.
                              Input : -
                              Output: 0,0,0,0,0 in time}
{*************************************************************************}


Procedure Addtime( t1, t2: timetype; Var sum: timetype);
{*************************************************************************}
{ This procedure adds two times. The result is corrected for leap-years.
                              Input : t1, t2; times to be added
                              Output: sum; the addition result}
{*************************************************************************}


Function Later( t1, t2: timetype): Boolean;
{*************************************************************************}
{ This function becomes True if t1 is later than or equal to t2. Leap years
  are accounted for.
                              Input : t1, t2, times to be compared
                              Output: Boolean.}
{*************************************************************************}


Function FileExist( filename: String): Boolean;
{*************************************************************************}
{ This function checks the current directory for a file specified bij
  filename. If the file is present, true is output.
                              Input : filename
                              Output: true/false}
{*************************************************************************}


Implementation

Uses Dos;

```

```
Procedure Convert( line: String; Var varname, value: String);

Begin
     varname:= '';
     value:= '';

     If (line[1] <> ' ') And ( line[1] <> #9){ exit when no leading space}
     Then Begin                              { or tab}
               Exit;
          End;

     While (( line[1] = ' ') Or ( line[1] = #9)) And
          ( Length( line) > 1) Do            { skip leading spaces}
          line:= Copy( line, 2, Length( line) -1);
     If ( Length( line) = 0)                 { line too short}
     Then Exit;

     While ( line[ 1] <> ' ') And ( line[ 1] <> '=') And
          ( Length( line) > 1) Do            { get varname}
     Begin
          varname:= varname + Upcase( line[1]);
          line:= Copy( line, 2, Length( line) -1);
     End;
     If ( Length( line) = 0)                 { line too short}
     Then Exit;

     While ( line[ 1] <> '=') And ( Length( line) > 1) Do
                                             { search for '='}
          line:= Copy( line, 2, Length( line) -1);
     If ( Length( line) < 2)
     Then Exit;

     line:= Copy( line, 2, Length( line) -1);{ skip '='}
     While (( line[1] = ' ') Or ( line[1] = #9)) And
          ( Length( line) <> 0) Do           { skip leading spaces}
          line:= Copy( line, 2, Length( line) -1);
     If ( Length( line) = 0)
     Then Exit;

     While ( line[ 1] <> ';') And ( Length( line) > 1) Do
                                             { get value}
     Begin
          value:= value + line[1];
          line:= Copy( line, 2, Length( line) -1);
     End;
End;


Procedure OpenConfigRead( Var setupfile: Text; filename: string);

Begin
          If FileExist( filename)
          Then Begin
                    Assign( setupfile, filename);
                    Reset( setupfile);
               End
          Else Begin
                    Assign( setupfile, filename);
                    Rewrite( setupfile);
                    Writeln( setupfile);         { put some dummy load in file}
                    Close( setupfile);
                    Reset( setupfile);
               End;

End;


Procedure CloseConfig( Var setupfile: Text);

Begin
     If (TextRec( setupfile). mode = fmInput) Or
        (TextRec( setupfile). mode = fmOutput) Or
        (TextRec( setupfile). mode = fmInOut)
     Then Close( setupfile);
End;


Procedure OpenConfigWrite( Var setupfile: Text; filename: String);

Begin
     If FileExist( filename)
     Then Begin
               Assign( setupfile, filename);
               Append( setupfile);
          End
     Else Begin
               Assign( setupfile, filename);
               Rewrite( setupfile);
          End;
End;


Procedure OpenConfigWriteFirst( Var setupfile: Text; filename: String);

Begin
        Assign( setupfile, filename);
        Rewrite( setupfile);
End;


Procedure Date_and_Time( Var time: timetype);

Var
   dum  :      Word;
Begin
     With time Do
     Begin
          Getdate( year, month, day, dum);
          Gettime( hour, minute, sec, sec100);
     End;
```

```
End;


Procedure ErrorTime( Var time: timetype);

Begin
     With time Do
     Begin
          year:= 0;
          month:= 0;
          day:= 0;
          hour:= 0;
          minute:= 0;
          sec:= 0;
          sec100:= 0;
     End;
End;


Procedure Addtime( t1, t2: timetype; Var sum: timetype);

Begin
     sum.sec100:= t1.sec100 + t2.sec100;
     If sum.sec100 > 99 Then Begin
                              sum.sec100:= 99;
                              sum.sec:= 1;
                            End
                       Else sum.sec:= 0;

     sum.sec:= sum.sec + t1.sec + t2.sec;
     If sum.sec > 59 Then Begin
                              sum.sec:= 59;
                              sum.minute:= 1;
                         End
                    Else sum.minute:= 0;

     sum.minute:= sum.minute + t1.minute + t2.minute;
     If sum.minute > 59 Then Begin
                              sum.minute:= 59;
                              sum.hour:=1;
                            End
                       Else sum.hour:= 0;

     sum.hour:= sum.hour + t1.hour + t2.hour;
     If sum.hour > 23 Then Begin
                              sum.hour:= 23;
                              sum.day:=1;
                          End
                     Else sum.day:= 0;

     sum.day:= sum.day + t1.day + t2.day;
     sum.month:= 0;
     Case t1.month Of
     1,3,5,7,8,10,12: If sum.day > 31
                      Then Begin
                                sum.day:= 31;
                                sum.month:= 1;
                           End;
     4,6,9,11:        If sum.day > 30
                      Then Begin
                                sum.day:= 30;
                                sum.month:=1;
                           End;
     2:               If ( t1.year Mod 4 = 0) And ( sum.day > 29)
                      Then Begin
                                sum.day:= 29;
                                sum.month:=1;
                           End
                      Else If sum.day > 28
                           Then Begin
                                     sum.day:= 28;
                                     sum.month:= 1;
                                End;
     End;

     sum.month:= sum.month + t1.month + t2.month;
     If sum.month > 12
     Then Begin
               sum.month:= 12;
               sum.year:=1;
          End
     Else sum.year:= 0;

     sum.year:= sum.year + t1.year + t2.year;
End;


Function Later( t1, t2: timetype): Boolean;

Begin
  If t1.year > t2.year
  Then later:= True
  Else If t1.year = t2.year

    Then If t1.month > t2.month
         Then later:= True
         Else If t1.month = t2.month

           Then If t1.day > t2.day
                Then later:= True
                Else If t1.day = t2.day

                  Then If t1.hour > t2.hour
                       Then later:= True
                       Else If t1.hour = t2.hour

                         Then If t1.minute > t2.minute
                              Then later:= True
                              Else If t1.minute = t2.minute

```

```
                                Then If t1.sec > t2.sec
                                     Then later:= True
                                     Else If t1.sec = t2.sec

                                       Then If t1.sec100 >= t2.sec100
                                            Then later:= True
                                            Else later:= False

                                       Else later:= False
                                Else later:= False
                         Else later:= False
                  Else later:= False
           Else later:= False
    Else later:= False;
End;


Function FileExist( filename: String): Boolean;

Var
   infile      :      File Of Byte;

Begin
   {$I-}
   Assign( infile, filename);
   Reset( infile);

   If IOResult <> 0
   Then FileExist:= False
   Else Begin                                    { on error, file is not open}
             FileExist:= True;                   { so don't close. If no error}
             Close( infile);                     { file is open so do close}
        End;
   {$I+}
End;

End.
```

```
UNIT Com_4;

{ The original program is called:}
{        Comm_TP4.PAS Ver. 1.50 - RS-232 Support for IBM Compatibles       }
{                          (c) Copyright, 1989                              }
{                           Kevin R. Bulgrien                               }
{                             October, 1989                                 }
{                                                                           }
{ See the accompanying file COMM_TP4.DOC for specific information regarding  }
{ the distribution policies and usage information for this source code file. }
{                                                                           }
{ Written by:  Kevin R. Bulgrien           Version 1.50 completed 11/13/89    }
{                                                                           }
{ Contact at:  LeTourneau University       LeTourneau University BBS          }
{              Microcomputer Services      2400/1200/300 Baud                 }
{              P.O. Box 7001               (214) 237-2742                     }
{              Longview, TX  75607                                           }
{                                                                           }
{ This program works with Turbo Pascal 4.0 and 5.x.  See Comm_TP4.DOC for the }
{ instructions.  Comm_TP3, by the same author, works under Turbo Pascal 3.0,  }
{ and Comm_TC2 works with Turbo C.  Upcoming is a Turbo Assembler Comm_TA1.   }
{                                                                           }
{ This software directly accesses the 8250 UART as well as the 8259 interrupt }
{ controller hardware.  Though they are IBM's standard, it is possible that   }
{ some manufacturers could use different hardware to perform these functions. }
{
{ This unit was adapted and expanded by R.C. Meijer}

{ Functions Getcharbuff, lookbuff, restore_buff and charsinbuff added by      }
{ Comments added in Interface section }

{ NOTE: THE INTERRUPT NUMBER FOR COM3 IS CHANGED TO IRQ5 AND}
{       THE INTERRUPT NUMBER FOR COM4 IS CHANGED TO IRQ7 !!!!!!}
{       IT IS NOW POSSIBLE TO RUN COM1, COM2, COM3 AND COM4}
{       AT THE SAME TIME}
{       ON THE I/O CARD, THE INTERRUPT FOR LPT1 AND LPT2 SHOULD BE DISABLED}
{       A SPECIAL COMPORT CARD SHOULD BE USED PROVIDING THE APPROPRIATE{
{       INTERRUPTS. }
{                                                                           }
{$S-} { Interrupt handlers should not be compiled with stack checking enabled }
      {-Else the system may crash! }
{$DEFINE ErrorChecking}                          {-Enable/Disable Error Check }
{$DEFINE NoMessageCode}                          {-Enable/Disable Error Msgs  }
{$DEFINE FWriteCOM}                              {-WriteCOM is a Func or Proc }

INTERFACE

USES DOS, CRT;

CONST
  MaxPorts = 4;                                  {-Max # of COM ports to use  }
  MaxInSize = 1000;                              {-Maximum input buffer size  }
  MaxOutSize = 255;                              {-Maximum output buffer size }
                                                 {-These constants are used   }
                                                 { to make the code readable. }
                                                 { They are the offsets from  }
                                                 { the base address of the    }
                                                 { 8250 UART. If, for example,}
                                                 { you need the value of the  }
                                                 { #3 Line Status Register,   }
                                                 { you could get it using:    }
                                                 {  PORT [COMPort [3] + LSR]  }
  THR = 0;                                       { Transmit Holding Register  }
  RHR = 0;                                       { Receive Holding Register   }
  DLL = 0;                                       { Divisor Latch Register LSB }
  IER = 1;                                       { Interrupt Enable Register  }
  DLM = 1;                                       { Divisor Latch Register MSB }
  IIR = 2;                                       { Interrupt ID Register      }
  LCR = 3;                                       { Line Control Register      }
  MCR = 4;                                       { Modem Control Register     }
  LSR = 5;                                       { Line Status Register       }
  MSR = 6;                                       { Modem Status Register      }

  { The following declarations are crucial to the operation of this program. }
  { I would advise you not to change the information unless you are sure you }
  { know what you are doing.  See the .DOC file for further information. For }
  { standard MaxPorts settings of 1 - 4, move the comment bracket as needed. }
  { Improper setting of the IRQNmbr array may cause a system crash!          }

  COMNmbr  : ARRAY [1..MaxPorts] OF BYTE    = (     1,     2,     3,     4 );
                                                 {-Define the COM port number }
  COMPort  : ARRAY [1..MaxPorts] OF WORD    = ( $03F8, $02F8, $03E8, $02E8 );
                                                 {-Base addresses 8250 Regs   }
  IRQNmbr  : ARRAY [1..MaxPorts] OF BYTE    = (     4,     3,     5,     7 );
                                                 {-IRQ numbers of the ports   }
  ChainInt : ARRAY [1..MaxPorts] OF BOOLEAN = ( FALSE, FALSE, FALSE, FALSE );
                                                 {-When port interrupt done,  }
                                                 { jump to OldIntVector [x]?  }



Type
  BaudType = (B110,B150,B300,B600,B1200,B2400,B4800,B9600,B19200,B38400);
                                                 {-Baud rates supported       }
  ParityType = (None, Odd, Null, Even, MarkOff, Mark, SpaceOff, Space);
                                                 {-Parity types supported     }
  ProcNameType = STRING [20];                    {-Used by error checking code}

VAR
  Framing, Overrun, Parity, Break : ARRAY [1..MaxPorts] OF WORD;
                                                 {-Port error counters        }
  OutBuffer : ARRAY [1..MaxPorts, 0..MaxOutSize] OF BYTE;
                                                 {-Port output buffers        }
  InBuffer : ARRAY [1..MaxPorts, 0..MaxInSize] OF BYTE;
                                                 {-Port input buffers         }
  CTS, DSR, RI, CD : ARRAY [1..MaxPorts] OF BOOLEAN;
                                                 {-Port input line status     }
  OutHead, OutTail : ARRAY [1..MaxPorts] OF WORD;{-Output buffer pointers     }
  InHead, InTail : ARRAY [1..MaxPorts] OF WORD;  {-Input buffer pointers      }
  IntInstalled : ARRAY [1..MaxPorts] OF BOOLEAN; {-TRUE if interrupt in place }
```

```
  DTR_RTS : ARRAY [1..MaxPorts] OF BOOLEAN;     {-Allowed to alter DTR/RTS?  }
  ErrorCode, ErrorPort : BYTE;                  {-Error type code & the port }
  {$IFNDEF NoMessageCode}                       { which had the error.       }
    ErrMsgX, ErrMsgY : BYTE;                    {-Error message coordinates  }
    ShowMessages : BOOLEAN;                     {-FALSE disables the error   }
  {$ENDIF}                                      { messages/TRUE enables them.}
  cnter:Byte;
PROCEDURE DisableInts; INLINE ($FA);            {-Disable hardware interrupts}
{****************************************************************************}
{ This procedure disables all interrupts. It is used, so that some
  operations, which would give wrong result when interrupted, can work
  properly.
                               Input : -
                               Output: -}
{****************************************************************************}
PROCEDURE EnableInts; INLINE ($FB);             {-Enable hardware interrupts }
{****************************************************************************}
{ This procedure enables all interrupts. It is used to undo disableints.
                               Input : -
                               Output: -}
{****************************************************************************}
PROCEDURE Set_DTR_RTS (Com : BYTE; Status : BOOLEAN);
{****************************************************************************}
{ This procedure changes the setting of DTR & RTS if the global variable
  DTR_RTS is set to TRUE.  The port and the desired state are passed as
  parameters.  TRUE for on, FALSE for off.  Since hardware handshaking
  requirements vary according to the hardware being used, you may have
  to rewrite Set_DTR_RTS to accommodate the hardware.  Bit 0 controls
  DTR and bit 1 controls RTS.  Writing a 0 to the bit will turn the line
  on.
                               Input : comportnumber
                                       status flag
                               Output: -}
{****************************************************************************}
PROCEDURE SetupCOMPort (Com, Baud, DataBits, Parity, StopBits : BYTE);
{****************************************************************************}
{ This procedure is used to program the 8250 UART chip. You can use the
  UART for serial communications and as a mouse interface.
                               Input : comportnumber
                                       baudrate
                                       number of databits
                                       parity
                                       number of stopbits
                               Output: -}
{****************************************************************************}
PROCEDURE InstallInt (Com : BYTE);
{****************************************************************************}
{ This procedure is used to install the interrupts used for the serial
  communications. For comport 1, the IRQ nr 4 is used and interruptvector
  $0C. For comport 2, the IRQ nr 3 is used and interruptvector $0B.
                               Input : comportnumber
                               Output: -}
{****************************************************************************}
PROCEDURE RemoveInt (Com : BYTE);
{****************************************************************************}
{ This procedure is used to uninstall the interrupts. The original inter-
  rupts are installed again, to insure further operation.
                               Input : comportnumber
                               Output: -}
{****************************************************************************}
PROCEDURE EmptyBuffer (Buffer : BYTE; TrueInFalseOut : BOOLEAN);
{****************************************************************************}
{ This procedure is used to empty a buffer used for communications port.
                               Input : buffer
                                       flag to indicate input/output
                                       buffer
                               Output: -}
{****************************************************************************}
{$IFDEF FWriteCOM}
FUNCTION WriteCOM (Com : BYTE; Data : STRING) : BOOLEAN;
{$ELSE}
PROCEDURE WriteCOM (Com : BYTE; Data : STRING);
{$ENDIF}
{****************************************************************************}
{ Depending on the definition of the environment variable FWRITECOM, the
  procedure or function is used to transmit a string of data on a certain
  comport.
                               Input : comportnumber
                                       datastring
                               Output: (-/ flag indicating success)}
{****************************************************************************}
PROCEDURE IWriteCOM (Com : BYTE; Data : STRING);
{****************************************************************************}
{ This procedure writes a string to the comport specified. It uses inter-
  rupts to do its job.
                               Input : comportnumber
                                       datastring
                               Output: -}
{****************************************************************************}
FUNCTION  ReadCOM (Com : BYTE) : CHAR;
{****************************************************************************}
```

```
{ This function reads a character from a comport. It is used for polling.
                                  Input : comportnumber
                                  Output: character}
{*************************************************************************}


FUNCTION  TimedReadCOM (Com : BYTE; VAR Data : CHAR) : BOOLEAN;
{*************************************************************************}
{ This function reads a character from a comport. It wait some time to
  catch a character. If the character was not caught, the it return a false.
                                  Input : comportnumber
                                  Output: character from comport
                                          flag indicating timed out}
{*************************************************************************}


Function Getcharbuff( comport: Byte): Char;
{*************************************************************************}
{ This function fetches one character from the buffer that belongs to
  a comport. It return #00 if there was no character.
                                  Input : comportnumber
                                  Output: character}
{*************************************************************************}


Function Lookbuff( comport: Byte): Char;
{*************************************************************************}
{ This function copies one character from the buffer that belongs to a com-
  port. It is not erased in the buffer. It returns #00 if there was no
  character.
                                 Input : comportnumber
                                 Output: character}
{*************************************************************************}


Function Charsinbuff( comport: Byte): Integer;
{*************************************************************************}
{ This function returns the number of characters in the buffer for one
  comport.
                                 Input : comportnumber
                                 Output: number of characters in buffer}
{*************************************************************************}


Procedure Restore_buffer( comport: Byte; line: String);
{*************************************************************************}
{ This procedure restores a line into the read buffer.
                              Input : comportnumber
                                      line to restore
                              Output: restored buffer}
{*************************************************************************}

IMPLEMENTATION


VAR
  OldIntVector : ARRAY [1..MaxPorts] OF POINTER; {-Original COMx int. vectors }
  IRQMask, Loop : BYTE;
  ExitSave : POINTER;                            {-Saves original ExitProc    }
  restorebuff   :       Array[1..Maxports] Of String;
  restoreflag   :       Array[1..Maxports] Of Boolean;

{ This procedure sets the global error identification variables ErrorCode and
  ErrorPort.  It also prints an appropriate error message if the $DEFINE
  NoMessageCode directive was not present at compile time and if ShowMessages
  is TRUE at run-time.  The error messages are displayed at the current
  cursor position when ErrMsgX and/or ErrMsgY = 0 (Default).  If the calling
  program needs the messages at a specific location, it must set ErrMsgX and
  ErrMsgY to the desired screen coordinates.  Error messages consist of a
  beep, the procedure or function in which the error took place, the port
  number, and a description of the problem encountered.  At all times,
  ErrorCode and ErrorPort can also be used to find error conditions.
  ErrorCode is 0 for no errors, or a value from 1-5 describing the error.
  ErrorPort is the handler number which had the problem.  These values are
  valid for the last serial operation requested.  }

PROCEDURE MakeError (Code, Port : BYTE; ProcName : ProcNameType);
BEGIN
  ErrorCode := Code;                             {-Set the global error type  }
  ErrorPort := Port;                             { and port variables.        }
  {$IFNDEF NoMessageCode}
    IF ShowMessages
      THEN BEGIN
             IF (ErrMsgX > 0) THEN GOTOXY (ErrMsgX, WHEREY);
                                                 {-Print error messages. A 0  }
             IF (ErrMsgY > 0) THEN GOTOXY (WHEREX, ErrMsgY);
                                                 { coordinate uses current    }
             WRITE (ProcName, ' ERROR: ', #7);   { cursor position.           }
             CASE Code OF
               1 : WRITELN ('Invalid port # ', Port);
                                                 { 1 <= Good Port <= MaxPorts }
               2 : WRITELN ('Port # ', Port, 'already installed');
                                                 {-Use InstallInt once/port   }
               3 : WRITELN ('Port # ', Port, 'not installed yet');
                                                 {-RemoveInt w/o InstallInt   }
               4 : WRITELN ('Timeout writing port # ', Port);
                                                 {-WriteCOM error             }
               5 : WRITELN ('Timeout reading port # ', Port);
                                                 {-TimedReadCOM error         }
             END;
           END;
  {$ENDIF}
END;

{ This function is used to make sure that the requested ports are valid and
  if the interrupt handlers are properly installed or uninstalled.  It calls
  MakeError to set the global error variables ErrorCode and ErrorPort, and to
  print error messages.  Status should be set to 0 if the port handler should
  not be installed yet, and 1 if it is supposed to be installed already.  Use
  a status of -1 if the installation status is not critical.  ProcName is
```

```
  used to pass the procedure or function name to MakeError so it can be used
  in an error message. A TRUE is returned if everything checks out okay,
  otherwise a FALSE is returned.}

FUNCTION ValidPort (Port:BYTE; Status:INTEGER; ProcName:ProcNameType):BOOLEAN;
BEGIN
  ErrorCode := 0;                                 {-Default of no errors found }
  IF (Port < 1) OR (Port > MaxPorts)              {-Check requested port # for }
    THEN MakeError (1, Port, ProcName)            { validity.                  }
    ELSE IF (Status >= 0) AND (ORD (IntInstalled [Port]) <> Status)
                                                  {-Check port installation    }
           THEN MakeError (2+Status, Port, ProcName);
                                                  { state with needed status.  }
  ValidPort := (ErrorCode = 0);                   {-Returns TRUE if no errors  }
END;

{ This procedure changes the setting of DTR & RTS if the global variable
  DTR_RTS is set to TRUE.  The port and the desired state are passed as
  parameters.  TRUE for on, FALSE for off.  Since hardware handshaking
  requirements vary according to the hardware being used, you may have to
  rewrite Set_DTR_RTS to accommodate the hardware.  Bit 0 controls DTR and
  bit 1 controls RTS.  Writing a 0 to the bit will turn the line on.}

PROCEDURE Set_DTR_RTS (Com : BYTE; Status : BOOLEAN);
BEGIN
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, -1, 'Set_DTR_RTS') THEN EXIT;
                                                  {-Optional error trapping    }
  {$ENDIF}
  IF DTR_RTS [Com]                                {-Provided to allow modem    }
    THEN IF Status                                { programs to prevent the    }
           THEN PORT [COMPort [Com]+MCR] := PORT [COMPort [Com]+MCR] AND $FC
                                                  { modem from being hung up   }
           ELSE PORT [COMPort [Com]+MCR] := PORT [COMPort [Com]+MCR] OR $03;
                                                  { during port setups due to  }
END;                                              { dropping DTR and/or RTS.   }

{ This procedure sets up a selected serial port to use specified parameters.
  Com specifies the port to set up.  The Baud parameter must be in the range
  0 to 9, and is not range checked, but its maximum valid value is determined
  by the number of entries in BaudTable.  BaudType documents the baud rates
  supported by BaudTable.  ParityType is provided to document the parity
  settings allowed.  Use ORD() to get the correct value to pass: ORD(B110)
  returns the BYTE that selects 110 baud and ORD(None) gives the value that
  selects no parity.  1.5 stop bits are used when StopBits = 2 AND DataBits
  = 5, otherwise StopBits will set the indicated number of stop bits in the
  range 1 to 2.  DataBits may be set with 5 to 8 for the number of data bits
  to use.  Mark parity means that the parity bit is always set to 0. Space
  parity means that the parity bit is always set to 1.  MarkOff, SpaceOff,
  NONE, & NULL all disable parity but are here for completeness. }

PROCEDURE SetupCOMPort (Com, Baud, DataBits, Parity, StopBits : BYTE);
CONST BaudTable : ARRAY [0..9] OF WORD = ($0417, $0300, $0180, $00C0, $0060,
                                          $0030, $0018, $000C, $0006, $0003);
                                                  { Set baud rate of 8250 when }
                                                  { written to DLL & DLM.      }
VAR
  Temporary : BYTE;
BEGIN
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, -1, 'SetupCOMPort') THEN EXIT;
                                                  {-Optional error trapping    }
  {$ENDIF}
  Set_DTR_RTS (Com, FALSE);
  PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] OR $80;
                                                  {-Set DLL & DLM active       }
  PORT [COMPort [Com] + DLL] := LO (BaudTable [Baud]);
                                                  {-Set the baud rate with the }
  PORT [COMPort [Com] + DLM] := HI (BaudTable [Baud]);
                                                  { predefined divisor values. }
  Temporary := (DataBits - 5) AND $03 OR (((StopBits - 1) SHL 2) AND $04);
                                                  {-Set data bits, stop bits,  }
  PORT [COMPort [Com] + LCR] := Temporary OR ((Parity SHL 3) AND $38);
                                                  { and parity protocol.       }
  Set_DTR_RTS (Com, TRUE);
END;

{ These procedures handle all interrupts from the 8250 communications chip.
  All interrupt types are at least minimally supported.  Enough code is
  present to help you know how to modify the code for your specific
  requirements.  Incoming data is ignored if the buffer is full, otherwise
  it is placed into the InBuffer circular queue. Data to be transmitted by
  interrupt is taken from the OutBuffer queue.  The transmitter is the only
  interrupt which has to be manually invoked.  Do so by placing the first
  character of each trans- mission into the THR.  It automatically shuts off
  when all the data in the buffer has been sent. The other interrupt types
  will take care of themselves once enabled. Modem/Port input lines may be
  monitored by this handler.  BOOLEAN arrays CTS, DSR, RI, and CD always
  show current status of these lines IF the interrupt handler is active and
  the Modem Status Change interrupt has been enabled.  A TRUE indicates the
  signal is  active.  CD and RI are very helpful for modem related programs.
  Line Status errors are counted.  It is up to you add any corrective action.
  All ports with interrupts pending on the IRQ level which invoked this
  handler are processed regardless of which port generated the actual
  interrupt. This is simple to implement, yet it preserves the interrupt
  priority handling between ports. }

{$F+}                                             {-Interrupt handlers MUST be }
PROCEDURE IntHandler1; INTERRUPT;                 { FAR calls.                 }

Const
     com       =        1;

Var
   imr,
   temp  :       Byte;

BEGIN
  IMR := PORT [$21];                              {-Backup IMR for later use   }
  PORT [$21] := IMR OR IRQMask;                   {-Disable Comm_TP4 interrupts}
```

```
EnableInts;                               {-Allow other interrupts      }

    PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;

                                          {-Set THR, RHR & IER active  }
    CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type   }
     0 : BEGIN
          Temp := PORT [COMPort [Com] + MSR]; {     MODEM STATUS CHANGES     }
          CD  [Com] := $80 AND Temp <> 0;   {        Carrier Detect        }
          CTS [Com] := $10 AND Temp <> 0;   {        Clear To Send         }
          DSR [Com] := $20 AND Temp <> 0;   {        Data Set Ready        }
          RI  [Com] := $40 AND Temp <> 0;   {        Ring Indicator        }
         END;
     2 : BEGIN
          IF (OutHead [Com] = OutTail [Com])  {  TRANSMIT REGISTER EMPTY   }
           THEN
            PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
                                          {  If no more data to send,  }
           ELSE                           {  shut off the transmitter. }
            BEGIN                         {  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;
         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                           { Otherwise, the character   }
           ELSE                           { 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-}


{$F+}                                     {-Interrupt handlers MUST be }
PROCEDURE IntHandler2; INTERRUPT;         { FAR calls.                 }

Const
    com        =      2;

Var
   imr,
   temp  :       Byte;

BEGIN
  IMR := PORT [$21];                      {-Backup IMR for later use   }
  PORT [$21] := IMR OR IRQMask;           {-Disable Comm_TP4 interrupts}
  EnableInts;                             {-Allow other interrupts     }

    PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;

                                          {-Set THR, RHR & IER active  }
    CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type   }
     0 : BEGIN
          Temp := PORT [COMPort [Com] + MSR]; {     MODEM STATUS CHANGES     }
          CD  [Com] := $80 AND Temp <> 0;   {        Carrier Detect        }
          CTS [Com] := $10 AND Temp <> 0;   {        Clear To Send         }
          DSR [Com] := $20 AND Temp <> 0;   {        Data Set Ready        }
          RI  [Com] := $40 AND Temp <> 0;   {        Ring Indicator        }
         END;
     2 : BEGIN
          IF (OutHead [Com] = OutTail [Com])  {  TRANSMIT REGISTER EMPTY   }
           THEN
            PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
                                          {  If no more data to send,  }
           ELSE                           {  shut off the transmitter. }
            BEGIN                         {  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;
         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                           { Otherwise, the character   }
```

```
          ELSE                                 { 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-}


{$F+}                                          {-Interrupt handlers MUST be }
PROCEDURE IntHandler3; INTERRUPT;              { FAR calls.                 }

Const
     com        =       3;

Var
   imr,
   temp  :       Byte;

BEGIN
  IMR := PORT [$21];                           {-Backup IMR for later use   }
  PORT [$21] := IMR OR IRQMask;                {-Disable Comm_TP4 interrupts}
  EnableInts;                                  {-Allow other interrupts     }

    PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;

                                               {-Set THR, RHR & IER active  }
    CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type    }
     0 : BEGIN
          Temp := PORT [COMPort [Com] + MSR];  {    MODEM STATUS CHANGES    }
          CD  [Com] := $80 AND Temp <> 0;      {       Carrier Detect       }
          CTS [Com] := $10 AND Temp <> 0;      {       Clear To Send        }
          DSR [Com] := $20 AND Temp <> 0;      {       Data Set Ready       }
          RI  [Com] := $40 AND Temp <> 0;      {       Ring Indicator       }
        END;
     2 : BEGIN
          IF (OutHead [Com] = OutTail [Com])   {  TRANSMIT REGISTER EMPTY   }
           THEN
            PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
                                               {  If no more data to send,  }
          ELSE                                 {  shut off the transmitter. }
           BEGIN                               {  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;
        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                                 { Otherwise, the character   }
          ELSE                                 { 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-}


{$F+}                                          {-Interrupt handlers MUST be }
PROCEDURE IntHandler4; INTERRUPT;              { FAR calls.                 }

Const
     com        =       4;

Var
   imr,
   temp  :       Byte;

BEGIN
```

```
IMR := PORT [$21];                              {-Backup IMR for later use   }
PORT [$21] := IMR OR IRQMask;                   {-Disable Comm_TP4 interrupts}
EnableInts;                                     {-Allow other interrupts     }

    PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;

                                                {-Set THR, RHR & IER active  }
    CASE PORT [COMPort [Com] + IIR] AND $06 OF {-Identify interrupt type    }
     0 : BEGIN
          Temp := PORT [COMPort [Com] + MSR];  {    MODEM STATUS CHANGES    }
          CD  [Com] := $80 AND Temp <> 0;      {       Carrier Detect       }
          CTS [Com] := $10 AND Temp <> 0;      {       Clear To Send        }
          DSR [Com] := $20 AND Temp <> 0;      {       Data Set Ready       }
          RI  [Com] := $40 AND Temp <> 0;      {       Ring Indicator       }
         END;
     2 : BEGIN
          IF (OutHead [Com] = OutTail [Com])   {  TRANSMIT REGISTER EMPTY   }
           THEN
            PORT [COMPort [Com] + IER] := PORT [COMPort [Com] + IER] AND $FD
                                               {  If no more data to send,  }
           ELSE                                {  shut off the transmitter. }
            BEGIN                              {  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;
         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                                { Otherwise, the character   }
           ELSE                                { 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-}

{ This procedure installs and enables the specified serial port interrupt.  All
  port input line monitoring variables are set to match the actual line
  states.  The old serial port interrupt vector is saved so it can be
  reinstalled when we remove our serial port interrupt.  DTR and RTS are
  forced to the ready state. The 8250 interrupts are enabled by ORing the
  MCR with $08.  To enable all four 8250 interrupt types, write $0F to the
  IER.  (Receive Buffer Full, Line Status, & Modem Status interrupts are
  enabled by writing $0D to the IER).  ORing $EF with PORT [$21] enables
  IRQ4 (COM1 or COM3), while $F7 enables IRQ3 (COM2 or COM4). Hardware
  interrupts should be disabled during the installation process since the
  8259 ports are being set. Error checking is always in place since a crash
  could occur if it is used when a the handler for the same port has already
  been installed. }

PROCEDURE InstallInt (Com : BYTE);
VAR
  Temporary : BYTE;
BEGIN
  IF ValidPort (Com, 0, 'InstallInt')             {-Error checking important!  }
   THEN
    BEGIN
     DisableInts;                                 {-Accessing 8259 hardware    }
     Temporary := PORT [COMPort [Com]+MSR];
     CD  [Com] := ($80 AND Temporary <> 0);       {-Carrier Detect status      }
     CTS [Com] := ($10 AND Temporary <> 0);       {-Clear to Send status       }
     DSR [Com] := ($20 AND Temporary <> 0);       {-Data Set Ready status      }
     RI  [Com] := ($40 AND Temporary <> 0);       {-Ring Indicator status      }
     Temporary := PORT [COMPort [Com] + LSR]; {-Reset interrupts that were }
     Temporary := PORT [COMPort [Com] + RHR]; { waiting to be processed.   }
     Temporary := 1 SHL IRQNmbr [Com];            {-If other port using same   }
     IF (IRQMask AND Temporary) = 0               { IRQ then nothing must be   }
      THEN BEGIN                                  { done to 8259 or vectors.   }
             IRQMask := IRQMask OR Temporary;{-Update interrupt record    }
             GETINTVEC ($08 + IRQNmbr [Com], OldIntVector [Com]);
                                                  {-Save old interrupt vector  }
             Case Com Of
             1: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler1);
             2: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler2);
             3: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler3);
             4: SETINTVEC ($08 + IRQNmbr [Com], @IntHandler4);
             End;
                                                  {-Install Comm_TP4 vector    }
             PORT [$21] := PORT [$21] AND NOT Temporary;
                                                  {-Enable 8259 IRQ handling   }
           END;
     PORT [COMPort [Com] + MCR] := PORT [COMPort [Com] + MCR] OR $08;
                                                  {-Enable 8250 interrupt line }
     PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
```

```
                                                {-Set THR/RHR/IER active     }
         PORT [COMPort [Com] + IER] := $01;     {-Enable 8250 interrupts     }
         IntInstalled [Com] := TRUE;            {-The interrupt is installed }
         Set_DTR_RTS (Com, TRUE);               {-DTR/RTS on so the other    }
         EnableInts;                            { device knows we are ready  }
       END;                                     { to receive data.           }
END;
                                                                              {
{ This procedure removes the specified serial port interrupt and reinstalls
  the original interrupt vectors. DTR & RTS are set OFF and 8250 interrupt
  line is disabled by ANDing the MCR with $F7.  All 8250 interrupt types are
  disabled by writing $00 to the IER.  ORing $10 with PORT [$21] disables
  IRQ4 (COM1 or COM3), while $08 disables IRQ3 (COM2 or COM4).  Hardware
  interrupts must be disabled since the 8259 ports are being set.  Some error
  checking is always in place since attempting RemoveInt on a handler which
  has not been installed can cause the computer to eventually crash.}

PROCEDURE RemoveInt (Com : BYTE);
VAR
  Temporary : BYTE;
BEGIN
  IF ValidPort (Com, 1, 'RemoveInt')            {-Error checking important!  }
    THEN
      BEGIN
        DisableInts;                            {-Accessing 8259 hardware    }
        Set_DTR_RTS (Com, FALSE);               {-DTR/RTS off                }
        IntInstalled [Com] := FALSE;            {-Uninstalling interrupt     }
        PORT [COMPort [Com] + MCR] := PORT [COMPort [Com] + MCR] AND $F7;
                                                {-Disable 8250 interrupt line}
        PORT [COMPort [Com] + LCR] := PORT [COMPort [Com] + LCR] AND $7F;
                                                {-Set THR, THE & IER active  }
        PORT [COMPort [Com] + IER] := $00;      {-Disable 8250 interrupts    }
        Temporary := 1 SHL IRQNmbr [Com];       {-Get bit for IRQ mask       }
        IF (IRQMask AND Temporary) <> 0         {-If no other Comm_TP4       }
          THEN BEGIN                            { interrupt is on this IRQ:  }
                 PORT [$21] := PORT [$21] OR Temporary;
                                                { Disable 8259 IRQ handling, }
                 SETINTVEC ($08 + IRQNmbr [Com], OldIntVector [Com]);
                                                { Replace original vector,   }
                 IRQMask := IRQMask AND NOT Temporary;
                                                { and update IRQ mask.       }
               END;
        EnableInts;                             {-Done with 8259 setup       }
      END;
END;

{ This procedure is provided for situations where you want to be sure that
  any of the serial port buffers are empty.  This could be used for aborting
  an interrupt driven transmission or for clearing unwanted data from the
  input buffers.  The first parameter is the buffer number and the second
  parameter is TRUE for the input buffer or FALSE for the output buffer.
  A buffer number of 0 causes all buffers to be emptied. }

PROCEDURE EmptyBuffer (Buffer : BYTE; TrueInFalseOut : BOOLEAN);
VAR
  LoopVar : BYTE;
BEGIN
  FOR LoopVar := 1 to MaxPorts DO                       {-A buffer number of 0 will  }
    IF (Buffer = 0) OR (LoopVar = Buffer)               { empty all buffers. Invalid }
      THEN BEGIN                                        { buffer numbers do nothing. }
             DisableInts;                               {-Disable buffer activity    }
             IF TrueInFalseOut                          { before changing pointers.  }
               THEN InHead [LoopVar] := InTail [LoopVar]
                                                        {-Clear an input buffer      }
               ELSE OutHead [LoopVar] := OutTail [LoopVar];
                                                        {-Clear an output buffer     }
             EnableInts;                                {-Reenable buffer activity   }
           END;
END;

{ This function/procedure writes character or string data to the serial port.
  It does this by reading and writing to the 8250 communications chip.  This
  is an example that may be modified to suit your purposes. As is, it pauses
  the program while it sends the data.  If it cannot send a character after
  65535 tries, it aborts the sending process.  Use the conditional
  compilation directive $DEFINE FWriteCOM in order to make WriteCOM a
  function which returns a BOOLEAN TRUE if the transmission did not timeout.
  The statement: (PORT [COMPort [Com]+LSR] AND $20) <> $20 indicates when
  the THR is ready for a new character to send.  CTS and DSR are not checked,
  but if you want to check for them (PORT [COMPort [Com]+MSR] AND $30) must
  equal $30.}

{$IFDEF FWriteCOM}                                      {-A $DEFINE/$UNDEF FWriteCOM }
  FUNCTION WriteCOM (Com : BYTE; Data : STRING) : BOOLEAN;
{$ELSE}                                                 { directive determines if    }
  PROCEDURE WriteCOM (Com : BYTE; Data : STRING);{ a procedure.                      }
{$ENDIF}
                                                        { WriteCOM is a function or  }

VAR
 LoopVar,                                               {-Pointer to output char     }
 TimeLoop : WORD;                                       {-Timeout counter variable   }
 TimeOut, Ready : BOOLEAN;                              {-TRUE if port timed out     }
BEGIN
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, 1, 'WriteCOM') THEN EXIT;
                                                        {-Optional error trapping    }
  {$ENDIF}
  LoopVar := 0;
  TimeOut := FALSE;
  WHILE (LoopVar < LENGTH (Data)) AND NOT TimeOut DO
                                                        {-Send the data one char at  }
    BEGIN                                               { a time unless the port was }
      INC (LoopVar);                                    { timed out.                 }
      TimeLoop := 0;
      REPEAT
        Ready := (PORT [COMPort [Com]+LSR] AND $20) <> 0;
        INC (TimeLoop);
      UNTIL Ready OR (TimeLoop = 65535);
      IF Ready
```

```
      THEN BEGIN
             PORT [COMPort [Com]+LCR] := PORT [COMPort [Com]+LCR] AND $7F;
                                                  {-Allow THR,RHR & IER access }
             PORT [COMPort [Com]+THR] := ORD (Data [LoopVar]);
                                                  {-Put the data to send in    }
           END                                    { the THR.                   }
      ELSE BEGIN
             TimeOut := TRUE;                     {-WriteCOM aborts if the THR }
             {$IFDEF ErrorChecking}               { takes too long to become   }
               MakeError (4, Com, 'WriteCOM'); { empty & optionally creates }
             {$ENDIF}                             { an error condition.        }
           END;
  END;
  {$IFDEF FWriteCOM}                              {-With a compiler directive  }
    WriteCOM := NOT TimeOut;                      { of $DEFINE FWriteCOM, then }
  {$ENDIF}                                        { WriteCOM is a function and }
END;                                              { returns TRUE if no timeout }

{ This procedure is an example of how to write an interrupt driven send
  routine.  The main idea is that you add data to the output buffer, then
  you get things going by manually placing one byte into the transmitter
  holding register.  After doing this, the rest of the buffer will be sent
  automatically.  Strenuous testing of this procedure under very high data
  rates has not been done, and it might be possible to rewrite it to provide
  better throughput.  Data is the information to send to the port in either
  CHAR or STRING form.  It is impractical to use this procedure for sending
  single characters since it calls WriteCOM at least once. It is best suited
  for high volume data transfers.  Interrupts should be off during buffer
  operations.}

PROCEDURE IWriteCOM (Com : BYTE; Data : STRING);
VAR
  BuffFull : BOOLEAN;                             {-TRUE if output buffer full }
  StartChr : CHAR;                                {-Temporary Buffer           }
  Loop : WORD;                                    {-Points to current Data item}
BEGIN
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, 1, 'IWriteCOM') THEN EXIT;
                                                  {-Optional error trapping    }
  {$ENDIF}
  Loop := 1;
  PORT [COMPort [Com]+LCR] := PORT [COMPort [Com]+LCR] AND $7F;
                                                  {-Enable access to 8250 IER  }
  WHILE (Loop <= LENGTH (Data)) DO                {-Load the output buffer one }
    BEGIN                                         { byte at a time.            }
      DisableInts;                                {-During buffer operations   }
      BuffFull := (OutTail [Com] + 1) MOD (MaxOutSize+1) = OutHead[Com];
                                                  {-Is the buffer full?        }
      IF NOT BuffFull                             {-If not, add a character to }
        THEN BEGIN                                { buffer and update pointers }
               OutBuffer [Com, OutTail [Com]] := ORD (Data [Loop]);
               OutTail [Com] := (OutTail [Com] + 1) MOD (MaxOutSize + 1);
                                                  { NOTE: Interrupts should be }
               INC (Loop);                        { enabled within the Loop so }
             END;                                 { the interrupt can empty    }
      EnableInts;                                 {-the buffer as we fill it.  }
      IF BuffFull OR (Loop > LENGTH (Data))       {-Check the interrupt status }
        THEN                                      { if the buffer gets full or }
          BEGIN                                   { after data is all loaded.  }
            IF (PORT [COMPort [Com] + IER] AND $02 <> 2)
                                                  {-If the transmit interrupt  }
              THEN                                { is not on, start it up.    }
                BEGIN
                  DisableInts;
                  StartChr := CHR (OutBuffer [Com, OutHead [Com]]);
                                                  {-Get the first character &  }
                  OutHead [Com] := (OutHead [Com] + 1) MOD (MaxOutSize + 1);
                                                  { take it out of the buffer. }
                  PORT [COMPort [Com]+IER] := PORT [COMPort [Com]+IER] OR $02;
                                                  {-Turn transmit interrupt on }
                  EnableInts;
                  {$IFDEF FWriteCOM}              {-Kickstart the transmitter  }
                    IF WriteCOM (Com, StartChr) THEN {};
                                                  { interrupt by sending one   }
                  {$ELSE}                         { character.  WriteCOM is    }
                    WriteCOM (Com, StartChr);     { used for simplicity.       }
                  {$ENDIF}
                END;
          END;
    END;
END;


{ This function is an example of how to get a character from the serial port.
  As is, if the buffer is empty, it waits until a character arrives, so this
  will not work for the TTY emulation. The interrupts are always disabled
  when the buffer pointers are checked or modified.  Beware!  Do not
  completely disable interrupts in the wait loop or else you never will get
  a character if there is not one there already. }

FUNCTION ReadCOM (Com : BYTE) : CHAR;
VAR
  CharReady : BOOLEAN;                            {-TRUE if there is data in   }
BEGIN                                             { the input buffer           }
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, 1, 'ReadCOM') THEN EXIT;
                                                  {-Optional error trapping    }
  {$ENDIF}
  CharReady := FALSE;
  REPEAT                                          {-Wait for data to arrive    }
    DisableInts;
    CharReady := InTail [Com] <> InHead [Com];    {-Check to see if buffer is  }
    EnableInts;                                   { empty                      }
  UNTIL CharReady;
  DisableInts;
  ReadCOM := CHR(InBuffer [Com, InHead [Com]]);   {-Read a character of data   }
  InHead [Com] := (InHead [Com] + 1) MOD (MaxInSize + 1);
                                                  {-Update the buffer pointer  }
  EnableInts;
END;
```

```

{ This function is an example of how to get a character from the serial port.
  Unlike ReadCOM, this routine  returns if no data appears in the buffer for
  a short period of time. This makes it useful for applications which process
  data only if it is available.  The interrupts are always disabled when the
  buffer pointers are checked or modified.  Beware!  Do not completely
  disable interrupts in the wait loop or else you never will get a character
  if there is not one there already.  Returns TRUE if valid data has been
  returned. You may optimize the time out period to a shorter one for your
  applications.  65535 is the maximum wait time.}


FUNCTION TimedReadCOM (Com : BYTE; VAR Data : CHAR) : BOOLEAN;
VAR
  CharReady : BOOLEAN;                          {-TRUE if there is data in   }
  TimeOut : WORD;                               { the input buffer           }
BEGIN
  {$IFDEF ErrorChecking}
    IF NOT ValidPort (Com, 1, 'TimedReadCOM') THEN EXIT;
                                                {-Optional error trapping    }
  {$ENDIF}
  TimeOut := 0;
  REPEAT                                        {-Wait for data to arrive    }
    DisableInts;
    CharReady := InTail [Com] <> InHead [Com];  {-Is the buffer empty or not }
    EnableInts;
    INC (TimeOut);                              {-Increment the timer        }
  UNTIL CharReady OR (TimeOut = 65535);         {-Set the maximum time to    }
  IF CharReady                                  { wait for data here.  Lower }
    THEN BEGIN                                  { number for shorter timeout.}
           DisableInts;
           Data := CHR (InBuffer [Com, InHead [Com]]);
                                                {-If data became available,  }
           InHead [Com] := (InHead [Com] + 1) MOD (MaxInSize + 1);
                                                { read a character, and set  }
           EnableInts;                          { the buffer pointers.       }
         END
    ELSE MakeError (5, Com, 'TimedReadCOM');    {-Record the timeout error   }
  TimedReadCOM := CharReady;
END;

{$F+}                                           {-VERY IMPORTANT!  When the  }
PROCEDURE RemoveIntOnExit;                      { program quits normally or  }
BEGIN                                           { abnormally, the interrupt  }
  FOR Loop := 1 TO MaxPorts DO                  { handler is automatically   }
    IF IntInstalled [Loop]                      { uninstalled when Turbo     }
      THEN RemoveInt (Loop);                    { invokes this procedure.    }
  ExitProc := ExitSave;                         {-Return control to the      }
END;                                            { original exit procedure    }
{$F-}

Function Getcharbuff( comport: Byte): Char;
{*************************************************************************}
{ Function to retrieve one character from the inputbuffer. #00 is returned}
{ whenever, there was no character in the inputbuffer}
{                            Input : comportnumber}
{                            Output: character}
{*************************************************************************}

Var
   dataready    :       Boolean;
   tempcharbuff :       Char;

Begin
     If restoreflag[comport]
     Then Begin
                  tempCharbuff:= restorebuff[comport,1];
                  If (Length( restorebuff[comport]) > 1)
                  Then restorebuff[ comport]:= Copy( restorebuff[comport], 2,
                                                  Length( restorebuff[ comport])-1)
                  Else restoreflag[comport]:= False;
             End
     Else Begin
                  Disableints;
                  dataready:= ( Intail[ comport] <> Inhead[ comport]);
                  Enableints;
                  If dataready
                  Then Begin
                               Disableints;
                               tempcharbuff:= Char( inbuffer[ comport,
                                                   Inhead[ comport]]);
                               Inhead[ comport]:= ( Inhead[ comport] + 1) MOD
                                                  ( Maxinsize+1);
                               Enableints;
                       End
                  Else tempcharbuff:=#00;
             End;
     GetCharbuff:= tempcharbuff;
End;


Function Lookbuff( comport: Byte): Char;
{*************************************************************************}
{ Function to see the next character in the inputbuffer. The character is  }
{ not retrieved. #00 is returned whenever, there was no character in the   }
{ inputbuffer.             Input : comportnumber}
{                          Output: character}
{*************************************************************************}

Var
   dataready    :       Boolean;

Begin
     If restoreflag[comport]
     Then Lookbuff:= restorebuff[comport,1]
     Else Begin
                  Disableints;
                  dataready:= ( Intail[ comport] <> Inhead[ comport]);
                  Enableints;
                  If dataready
```

```
                Then Begin
                        Disableints;
                        Lookbuff:= Char( inbuffer[ comport, Inhead[ comport]]);
                        Enableints;
                     End
                Else Lookbuff:=#00;
            End;
End;


Function Charsinbuff( comport: Byte): Integer;
{****************************************************************************}
{ Function which returns the number of characters in the inputbuffer for a}
{ specific comport.       Input : comportnumber}
{                         Output: number of characters in the buffer}
{****************************************************************************}

Var
   temp          : Integer;
Begin
    Disableints;
    If ( intail[ comport] >= inhead[ comport])
    Then temp:= intail[ comport] - inhead[ comport]
    Else temp:= intail[ comport] + ( maxinsize - inhead[ comport]);
    If Restoreflag[comport]
    Then temp:= temp + Length( restorebuff[ comport]);

    charsinbuff:= temp;
    Enableints;
End;


Procedure Restore_buffer( comport: Byte; line: String);

Var
   x    : Word;

Begin
    Disableints;
    restorebuff[comport]:= line;
    restoreflag[comport]:= True;
    Enableints;
End;

{ The following code is executed when any program which uses this unit first
  Changeable defaults are starts up.  It performs all necessary
  initializations. marked with a '*'          }

BEGIN
  ExitSave := ExitProc;                        {-VERY IMPORTANT!  This lets }
  ExitProc := @RemoveIntOnExit;                { the program halt safely.   }
  FOR Loop := 1 TO MaxPorts DO
    BEGIN
      InHead [Loop] := 0;                      {-Default all buffers set to }
      InTail [Loop] := 0;                      { empty on startup.          }
      OutHead [Loop] := 0;
      OutTail [Loop] := 0;
      DTR_RTS [Loop] := FALSE;                          {*Default DTR/RTS setting on }
      IntInstalled [Loop] := FALSE;                     {-Default interrupts not on  }
      CD  [Loop] := ($80 AND PORT [COMPort [Loop]+MSR] <> 0);
                                                        {-Carrier Detect status set  }
      CTS [Loop] := ($10 AND PORT [COMPort [Loop]+MSR] <> 0);
                                                        {-Clear To Send status set   }
      DSR [Loop] := ($20 AND PORT [COMPort [Loop]+MSR] <> 0);
                                                        {-Data Set Ready status set  }
      RI  [Loop] := ($40 AND PORT [COMPort [Loop]+MSR] <> 0);
                                                        {-Ring Indicator status set  }
      Framing [Loop] := 0;                              {-Reset framing error count  }
      Overrun [Loop] := 0;                              {-Reset overrun error count  }
      Parity [Loop] := 0;                               {-Reset parity error count   }
      Break [Loop] := 0;                                {-Reset break interrupt count}
    END;
  ErrorCode := 0;                                       {-Default of no port errors  }
  ErrorPort := 0;
  IRQMask := 0;                                         {-No interrupts installed    }
  {$IFNDEF NoMessageCode}
    ShowMessages := TRUE;                               {*Default error messages on  }
    ErrMsgX := 0;                                       {*Default of error messages  }
    ErrMsgY := 0;                                       {*placed at cursor position. }
  {$ENDIF}
    For cnter:= 1 To Maxports Do
        restoreflag[cnter]:= False;
END.
```

Appendix E Listings for MIASNAV

This appendix lists only the files of 'MIASNAV' that are different from 'MIASLOGO'.

```
Program MIASSystem;

{$M 40000, 0, 650000}

Uses MIASglob, MIAS;

Var
   alldata      :       alldatatype;
   command      :       commandtype;
   position,
   filtposition,
   predposition :       positiontype;

Begin
   Init( alldata, position);
   Repeat
     SetTimerToGPSIfNotSet;
     GetUserCommands( command);
     ExecCommands( command, alldata);
     GetData( alldata);
     DispFlags( alldata, position);
     CalcPos( alldata, position);
     FilterPosition( position, filtposition);
     PredictPosition( position, predposition);
     SendPosition( position);
   Until Stopcommand( command);
   CloseDown( alldata, position);
End.
```

```
Unit MLSglob;
{***************************************************************************}
{ This unit contains global MLStypes and MLS variables}
{ See also the ICAO Annex 10 Part 1 page 146 and 147.}
{***************************************************************************}

Interface

Uses Miasglob, Miscell;

Const
   Valid_Bas1          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_Bas2          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :0;
                                                         sec100:16);
   Valid_Bas3          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_Bas4          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_Bas5          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_Bas6          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_AuxA1         :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_AuxA2         :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_AuxA3         :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_AuxA4         :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :1;
                                                         sec100:0);
   Valid_AuxB          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:2;
                                                         sec  :0;
                                                         sec100:0);
   Valid_AuxC          :       timetype       =        (year :0;
                                                         month:0;
                                                         day  :0;
                                                         hour :0;
                                                         minute:0;
                                                         sec  :3;
                                                         sec100:0);
   Az2thresdist= 3700;
   AzPropCovNegLim=54;
   AzPropCovPosLim=54;
   Cleartype=1;

   MinGP =3;
   BAZstat=0;
   DMEstat=1;
   Azstat=1;
   Elstat=1;

   AzBW=2;
   ElBW=2;
   DMEdist=3387.5;

```

```
   AzMagOr=186;
   BazMagOr=0;

   BazPropCovNegLim=0;
   BazPropCovPosLim=0;
   BazBW=0;
   {BazStat=0;}{ is already declared}

   MLSident='MSC';

   AzOff     =0;
   Az2MLSdatdist =3394;
   AzAlignRun=0;
   AzCoorSyst=0;

   ElOff=-80;
   MLSdat2thres =275;
   ElHeight =1.2;
   {MLS datum point elevation = -3}
   {Runway threshold height = 0}

   DMEoff=56;
   DME2MLSdatDist=3385;
   {DME antenna height = 5}
   {Runway stopend distance = 3122}

   BAzOff=0;
   BAz2MLSdatDist=0;
   BAzAlignRun=0;

Type
    Basic1type  =       Record
                              Az2thresDist      :       Integer;
                                          { Azimuth to threshold distance}
                              AzPropCovNegLim,
                                          { Azimuth proportional coverage,
                                            negative limit}
                              AzPropCovPosLim,
                                          { idem, positive limit}
                              ClearType         :       Byte;
                                          { clearance signal type}
                        End;
    Basic2type  =       Record
                              MinGP      :       Real;
                                          { minimum glide path}
                              BAZstat,    { Back Azimuth status}
                              DMEstat,    { DME status}
                              Azstat,     { Azimuth status}
                              Elstat     : Byte;
                                          { Elevation status}
                        End;
    Basic3type  =       Record
                              AzBW,       { Azimuth beamwidth}
                              ElBW,       { Elevation beamwidth}
                              DMEdist    :       Real;
                                          { DME distance}
                        End;
    Basic4type  =       Record
                              AzMagOr,    { Azimuth magnetic orientation}
                              BazMagOr   :      Integer;
                                          { BackAzimuth magnetic orientation}
                        End;
    Basic5type  =       Record
                              BazPropCovNegLim,
                                          { Back Azimuth Proportional
                                            Coverage Negative limit}
                              BazPropCovPosLim:      Byte;
                                          { idem positive limit}
                              BazBW      : Real;
                                          { Back Azimuth beamwidth}
                              BazStat    : Byte;
                                          { Back Azimuth status}
                        End;
    Basic6type  =       Record
                              MLSident:         String[3];
                        End;
                                          { MLS ground equipment
                                            identification}
    AuxA1type   =       Record
                              AzOff,      { Azimuth Antenna offset}
                              Az2MLSdatdist:     Integer;
                                          { Azimuth antenna to MLS datum
                                            point distance}
                              AzAlignRun     :     Real;
                                          { Azimuth Alignment with Runway
                                            centreline}
                              AzCoorSyst     :     Byte;
                                          { Azimuth Antenna Coordinate
                                            system}
                        End;
    AuxA2type   =       Record
                              ElOff,      { Elevation antenna offset}
                              MLSdat2thres:        Integer;
                                          { MLS datumpoint 2 threshold
                                            distance}
                              ElHeight   :         Real;
                                          { Elevation Antenna Height}
                        End;
    AuxA3type   =       Record
                              DMEoff,     { DME offset}
                              DME2MLSdatDist:        Integer;
                                          { DME to MLS datum point distance}
                        End;
    AuxA4type   =       Record
                              BAzOff,     { Back azimuth antenna offset}
                              BAz2MLSdatDist:        Integer;
                                          { Back azimuth to MLS datum
                                            point distance}
                              BAzAlignRun:           Real;
                                          { Back azimuth alignment with
```

```
                                      runway centre line}
                          End;
    Discretestype=        Record
                                antenna    :      Byte;
                                test       :      Byte;
                                Azsource   :      Byte;
                                Azselwarn  :      Byte;
                                Bazselwarn:       Byte;
                                GPselwarn  :      Byte;
                                BAZavail   :      Byte;
                                BAZdeven   :      Byte;
                                Tuningcom  :      Byte;
                                nr1antsel  :      Byte;
                                changeinh  :      Byte;
                                tunPrtsel  :      Byte;
                          End;

    MLSinttype =          Record
                                Bas1       :      Basic1type;
                                Bas2       :      Basic2type;
                                Bas3       :      Basic3type;
                                Bas4       :      Basic4type;
                                Bas5       :      Basic5type;
                                Bas6       :      Basic6type;
                                AuxA1      :      AuxA1type;
                                AuxA2      :      AuxA2type;
                                AuxA3      :      AuxA3type;
                                AuxA4      :      AuxA4type;
                                AuxB,
                                AuxC       :      ADWtype;

                                Bas1_Time :       timetype;
                                Bas2_Time :       timetype;
                                Bas3_Time :       timetype;
                                Bas4_Time :       timetype;
                                Bas5_Time :       timetype;
                                Bas6_Time :       timetype;
                                AuxA1_Time:       timetype;
                                AuxA2_Time:       timetype;
                                AuxA3_Time:       timetype;
                                AuxA4_Time:       timetype;
                                AuxB_time,
                                AuxC_time :       timetype;

                                DMErange,
                                ELangle,
                                AZangle,
                                BAzangle   :      Double;
                                Discretes :       Discretestype;
                                Leftclr,
                                Rightclr   :      Boolean;
                                ElantInUse,
                                AzantInUse:       Byte;

                                Bas1_flag,
                                Bas2_flag,
                                Bas3_flag,
                                Bas4_flag,
                                Bas5_flag,
                                Bas6_flag,
                                Auxa1_flag,
                                Auxa2_flag,
                                Auxa3_flag,
                                Auxa4_flag,
                                AuxB_flag,
                                AuxC_flag,

                                ELangle_flag,
                                AZangle_flag,
                                BAzangle_flag,
                                DME_flag,
                                discretes_flag:   Boolean;

                                flag          :   Boolean;
                          End;


Implementation

Begin
End.
```

```
Unit MIAS;

Interface

{$N+,E+}


Uses MIASglob;


Procedure Init( Var alldata: alldatatype; Var position: positiontype);
{***************************************************************************}
{Initialise the MIAS system. Open a setupfile and read some values.
                              Input :-
                              Output:status; whether or not a device present}
{***************************************************************************}


Procedure DispFlags( alldata: alldatatype; position: positiontype);
{***************************************************************************}
{Display the flags on the screen or in a file. The flags represent the valid-
 ness of a certain device. The procedure uses only the flag-fields of the
 peripheral-fields in alldata.
                              Input :alldata
                              Output:flags on screen or file}
{***************************************************************************}


Procedure GetUserCommands( Var command: commandtype);
{***************************************************************************}
{Retrieve commands for the peripherals from the keyboard.
                              Input :-
                              Output:commands string in upcase}
{***************************************************************************}


Procedure ExecCommands( command: commandtype; alldata: alldatatype);
{***************************************************************************}
{Send commands to peripherals, or execute the commands on the host.
                              Input :commands
                              Output:-}
{***************************************************************************}


Procedure GetData( Var alldata: alldatatype);
{***************************************************************************}
{Retrieve information from the peripherals.
                              Input :-
                              Output:alldata; information from peripherals}
{***************************************************************************}


Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);
{***************************************************************************}
{Calculate the position from the information available.
                              Input :alldata
                              Output:position}
{***************************************************************************}


Procedure FilterPosition( position: positiontype;
                          Var filtposition: positiontype);
{***************************************************************************}
{Perform a filtering action on the positions calculated, to make sure the
 position output will be smooth.
                              Input :position
                              Output:filtered position}
{***************************************************************************}


Procedure PredictPosition( position: positiontype;
                           Var predposition: positiontype);
{***************************************************************************}
{Make a prediction of the position, to counteract the calculation and
 measurement delay.           Input :position
                              Output:predicted position}
{***************************************************************************}


Procedure SendPosition( position: positiontype);
{***************************************************************************}
{Send the position to another device from storage or displaying, or display
 the position on the screen.  Input :position
                              Output:position to the screen or a device}
{***************************************************************************}


Function Stopcommand( command: commandtype): Boolean;
{***************************************************************************}
{When a stop command is sent, this function will turn TRUE.
                              Input : commandstring
                              Output: Boolean, stop or not}
{***************************************************************************}


Procedure CloseDown( alldata: alldatatype; position: positiontype);
{***************************************************************************}
{Make sure the computer is back to 'normal'. Restore interuptvectors etc.
                              Input :-
                              Output:-}
{***************************************************************************}


Procedure SetTimerToGPSIfNotSet;
{***************************************************************************}
{ Use the GPS time to set the internal timer. If the time was already set,
  do nothing.
                              Input :-
                              Output:-}
{***************************************************************************}
```

```


Implementation

Uses GPS, DGPS, MLS, Att, HDG, PosCalc, User, Miscell, crt, dos;

Const
   Valid_tMLS          :       timetype       =       (year :0;
                                                        month:0;
                                                        day  :0;
                                                        hour :0;
                                                        minute:0;
                                                        sec  :2;
                                                        sec100:0);

Var
   stopkeypressed:      Boolean;
   DGPSflag      :      flagtype;         { Indicates DGPS mode or not}
   DGPSmode      :      Byte;
   allowed_error :      Double;
   tMLS          :      timetype;
   Old_MLS       :      MLSdatatype;
   timeset       :      Boolean;
   MIASlogname   :      String;


Procedure Init( Var alldata: alldatatype; Var position: positiontype);

Var
   setupfile    :       Text;
   error        :       Boolean;
   title,
   varname,
   line         :       String;
   value        :       String;
   code         :       Integer;

Begin
     MIASlogname:= 'c:';

     DGPSflag:= True;
     DGPSmode:= 1;

     ErrorTime( tMLS);
     With Old_MLS Do
     Begin
          flag:= True;
     End;

     TimeSet:= False;

     With position Do
     Begin
         WGS84lat:= 0;
         WGS84lon:= 0;
         WGS84alt:= 0;
         h:=0;
         x:=0;
         y:=0;
         z:=0;
         flag:= True;
         integrity.flag:= True;
     End;

     alldata.mls.MLSthresPos:= position;
     MLSantposition:= position;

     If Not FileExist( MIAScfgname)
     Then Begin
                SendUserMessage('Configfile "MIAS.CFG" not present');
                Halt( 1);                           { terminate the program }
          End;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                         { find MIAS part of
                                                      config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MIAS'));

     If Not Eof( setupfile)                         { if there is more in file}
     Then Repeat
                 Readln( setupfile, line);          { get a line}
                 Convert( line, varname, value);    { extract the variable name
                                                      and value}
                                                    { repeat until end of file}
                                                    { initialise comports for
                                                      communication with Engine}

                 If ( varname = 'ALLOWED_ERROR')
                 Then Val( value, allowed_error, code);

                 If ( varname = 'POSITION.WGS84LAT')
                 Then Val( value, position.wgs84lat, code);

                 If ( varname = 'POSITION.WGS84LON')
                 Then Val( value, position.wgs84lon, code);

                 If ( varname = 'POSITION.WGS84ALT')
                 Then Val( value, position.wgs84alt, code);

                 If ( varname = 'DGPSMODE')
                 Then Val( value, dgpsmode, code);

                 If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LAT')
                 Then Val( value, alldata.mls.mlsthrespos.wgs84lat, code);

                 If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84LON')
                 Then Val( value, alldata.mls.mlsthrespos.wgs84lon, code);

                 If ( varname = 'ALLDATA.MLS.MLSTHRESPOS.WGS84ALT')
```

```
                Then Val( value, alldata.mls.mlsthrespos.wgs84alt, code);

                If ( varname = 'ALLDATA.POS_ZEROVECTOR.X')
                Then Val( value, alldata.pos_zerovector.x, code);

                If ( varname = 'ALLDATA.POS_ZEROVECTOR.Y')
                Then Val( value, alldata.pos_zerovector.y, code);

                If ( varname = 'ALLDATA.POS_ZEROVECTOR.Z')
                Then Val( value, alldata.pos_zerovector.z, code);

                If ( varname = 'ALLDATA.ANT_ZEROVECTOR.X')
                Then Val( value, alldata.ant_zerovector.x, code);

                If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Y')
                Then Val( value, alldata.ant_zerovector.y, code);

                If ( varname = 'ALLDATA.ANT_ZEROVECTOR.Z')
                Then Val( value, alldata.ant_zerovector.z, code);

                If ( varname = 'ALLDATA.GPS.PRESENT')
                Then Val( value, alldata.gps.present, code);

                If ( varname = 'ALLDATA.DGPS.PRESENT')
                Then Val( value, alldata.dgps.present, code);

                If ( varname = 'ALLDATA.MLS.PRESENT')
                Then Val( value, alldata.mls.present, code);

                If ( varname = 'ALLDATA.ATT.PRESENT')
                Then Val( value, alldata.att.present, code);

                If ( varname = 'ALLDATA.HDG.PRESENT')
                Then Val( value, alldata.hdg.present, code);

                If ( varname = 'MIASLOGNAME')
                Then MIASlogname:= value;
          Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
     CloseConfig( setupfile);

     Convert_Pos_to_Ecef( position);
     Convert_Pos_to_Ecef( alldata.mls.mlsthrespos);

     MLSantposition:= position;

     InitUser( MIASlogname);

     With alldata Do
     Begin
          InitGPS( gps);
          InitDGPS( dgps);
          InitMLS( mls);
          InitAtt( att);
          InitHDG( hdg);
     End;

     DispFlags( alldata, position);
End;


Procedure DispFlags( alldata: alldatatype; position: positiontype);

Var
   line        :       Commandtype;

Begin
     line:= '';
     With alldata Do
     Begin
          If Not GPS.flag Then line:= line + 'GPS oke '
                          Else line:= line + 'GPS err ';

          If Not DGPS.flag Then line:= line + 'DGPS oke '
                           Else line:= line + 'DGPS err ';

          If Not MLS.flag Then line:= line + 'MLS oke '
                          Else line:= line + 'MLS err ';

          If Not Att.flag Then line:= line + 'Att oke '
                          Else line:= line + 'Att err ';

          If Not HDG.flag Then line:= line + 'HDG oke '
                          Else line:= line + 'HDG err ';

          If Not position.flag Then line:= line + 'Pos oke '
                               Else line:= line + 'Pos err ';

          If ( position.integrity.flag) Then line:= line + 'int err '
                                        Else line:= line + 'int oke ';
          SendUserFlags( line);
     End;
End;


Procedure GetUserCommands( Var command: commandtype);

Begin
     GetUserMessage( command);
End;


Procedure ExecCommands( command: commandtype; alldata: alldatatype);

Var
   substr      :       String;
   varname,
   value       :       String;
   code        :       Integer;
   tempdgpsmode :      Byte;

```

```
Begin
     substr:= Copy( command, 1, 4);

     If ( substr = 'GPS:') And ( alldata.gps.present = 1)
     Then ExecGPScommand( command);

     If ( substr = 'MLS:') And ( alldata.mls.present = 1)
     Then ExecMLScommand( command);

     If ( substr = 'ATT:') And ( alldata.att.present = 1)
     Then ExecAttcommand( command);

     If ( substr = 'HDG:') And ( alldata.hdg.present = 1)
     Then ExecHDGcommand( command);

     substr:= Copy( command, 1, 5);

     If ( substr = 'DGPS:') And ( alldata.dgps.present = 1)
     Then ExecDGPScommand( command);

     If substr = 'MIAS:'
     Then Begin
               substr:= Copy( command, 6, Length( command) - 5);
               Convert( ' '+substr, varname, value);{ add space for}
                                                     { procedure convert}
               If (varname = 'DGPSMODE')
               Then Begin
                         Val( value, tempdgpsmode, code);
                         If code <> 0
                         Then SendUserMessage( 'Error')
                         Else dgpsmode:= tempdgpsmode;
                    End;
          End;
End;


Procedure GetData( Var alldata: alldatatype);

Begin
     With alldata Do
     Begin
         If ( gps.present = 1)
         Then GetGPSdata( GPS, DGPSflag);

         If ( mls.present = 1) Or                { call getmlsdata also}
            ( dgps.present = 1)                  { for dgps data}
         Then GetMLSdata( MLS);

         If ( dgps.present = 1)
         Then GetDGPSdata( MLS, DGPS);

         If Not gps.flag
         Then Begin
                   If ( att.present = 1)
                   Then GetAttdata( Att);

                   If ( hdg.present =1 )
                   Then GetHDGdata( HDG);
              End;
     End;
End;


Procedure CalcPos( Var alldata: alldatatype; Var position: positiontype);

Const
     MaxNumOfIt =        1;                      { 1 is for no iterations}
Var
   old_deltaT   :        Double;
   iter         :        Integer;
   t,
   result       :        timetype;


Begin
    Case DGPSmode Of
    1: DGPSflag:= False;                         { no Diff GPS}
    2: ;                                         { only DGPS in prop cov}
    3: DGPSflag:= True;                          { always DGPS if available}
    End;

    If ( alldata.mls.present = 1)
    Then Begin
              CalcMLS( alldata.MLS);

              If Not alldata.mls.flag            { if valid info, then adjust}
              Then Date_and_time( tMLS);         { receive time}

              Date_and_time( t);                 { get current system time}
              AddTime( tMLS, Valid_tMLS, result);
                                                 { MLS only valid for 'Valid_
                                                   tMLS' seconds}
              If alldata.MLS.flag And            { if no valid info and}
                 Not Later( t, result)           { no time out}
              Then alldata.MLS:= Old_MLS         { use old info}
              Else Old_MLS:= alldata.MLS;        { store valid info for later
                                                   use}
         End;

    If ( alldata.gps.present = 1)
    Then CalcGPS( alldata.GPS, position, DGPSflag);

    If ( alldata.dgps.present = 1)
    Then CalcDGPS( alldata);

    CalcHybridPos( alldata, allowed_error, position);
End;


Procedure FilterPosition( position: positiontype;
```

```
                              Var filtposition: positiontype);

Begin
End;


Procedure PredictPosition( position: positiontype;
                           Var predposition: positiontype);
Begin
End;


Procedure SendPosition( position: positiontype);

Var
   line          :       String;
   dum           :       String;

Begin
     If position.flag
     Then Exit;

     line:= '';
     With position DO
     Begin
          If EcefTrueLocalFalse
          Then Begin
                    Str( wgs84lat * 180 / pi:23, dum);
                    line:= line + 'lat = ' + dum + ' ';
                    Str( wgs84lon * 180 / pi:23, dum);
                    line:= line + 'lon = ' + dum + ' ';
                    Str( wgs84alt:23, dum);
                    line:= line + 'alt = ' + dum + ' ';
                    SendUserMessage( line);

                    Str( x:23, dum);
                    line:= 'x = ' + dum + ' ';
                    Str( y:23, dum);
                    line:= line + 'y = ' + dum + ' ';
                    Str( z:23, dum);
                    line:= line + 'z = ' + dum + ' ';
                    SendUserMessage( line);
               End
          Else Begin
                    Str( x:23, dum);
                    line:= line + 'a = ' + dum + ' ';
                    Str( y:23, dum);
                    line:= line + 'y = ' + dum + ' ';
                    Str( z:23, dum);
                    line:= line + 'z = ' + dum + ' ';
                    SendUserMessage( line);
               End;
     End;
End;


Function Stopcommand( command: commandtype): Boolean;

Begin
     If Copy( command, 1, 4) = 'STOP'
     Then stopcommand:= True
     Else stopcommand:= False;
End;


Procedure CloseDown( alldata: alldatatype; position: positiontype);

Var
   setupfile     :       Text;
   Value         :       String;

Begin
     OpenConfigWriteFirst( setupfile, MIAScfgname);
     Writeln( setupfile, 'MIAS');

     Str( allowed_error, value);
     Writeln( setupfile, #9'allowed_error  = ', value, ';');

     With position Do
     Begin
          Str( WGS84lat, value);
          Writeln( setupfile, #9'position.wgs84lat = ', value, ';');

          Str( WGS84lon, value);
          Writeln( setupfile, #9'position.wgs84lon = ', value, ';');

          Str( WGS84alt, value);
          Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
     End;
     Str( dgpsmode, value);
     Writeln( setupfile, #9'dgpsmode = ', value, ';');

     With alldata.mls.MLSthrespos Do
     Begin
          Str( wgs84lat, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lat = ',
                              value, ';');

          Str( wgs84lon, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84lon = ',
                              value, ';');

          Str( wgs84alt, value);
          Writeln( setupfile, #9'alldata.mls.mlsthrespos.wgs84alt = ',
                              value, ';');
     End;

     With alldata.Pos_zerovector Do
     Begin
          Str( x, value);
```

```
          Writeln( setupfile, #9'alldata.pos_zerovector.x = ', value, ';');

          Str( y, value);
          Writeln( setupfile, #9'alldata.pos_zerovector.y = ', value, ';');

          Str( z, value);
          Writeln( setupfile, #9'alldata.pos_zerovector.z = ', value, ';');
     End;

     With alldata.Ant_zerovector Do
     Begin
          Str( x, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.x = ', value, ';');

          Str( y, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.y = ', value, ';');

          Str( z, value);
          Writeln( setupfile, #9'alldata.ant_zerovector.z = ', value, ';');
     End;

     With alldata Do
     Begin
          Str( gps.present, value);
          Writeln( setupfile, #9'alldata.gps.present = ', value, ';');

          Str( dgps.present, value);
          Writeln( setupfile, #9'alldata.dgps.present = ', value, ';');

          Str( mls.present, value);
          Writeln( setupfile, #9'alldata.mls.present = ', value, ';');

          Str( att.present, value);
          Writeln( setupfile, #9'alldata.att.present = ', value, ';');

          Str( hdg.present, value);
          Writeln( setupfile, #9'alldata.hdg.present = ', value, ';');
     End;

     Writeln( setupfile, #9'miaslogname = ', miaslogname, ';');

     CloseConfig( setupfile);

     With alldata Do
     Begin
          If ( gps.present = 1)
          Then CloseGPS( gps, position);

          If ( dgps.present = 1)
          Then CloseDGPS( dgps);

          If ( mls.present = 1)
          Then CloseMLS( mls);

          If ( att.present = 1)
          Then CloseAtt( att);

          If ( hdg.present = 1)
          Then CloseHDG( hdg);
     End;

     CloseUser;
End;


Procedure SetTimerToGPSIfNotSet;

Var
   gpstime       :       Longint;
   valid         :       boolean;
   hour,
   minute,
   sec,
   sec100,
   year,
   month,
   day,
   dayofweek     :       Word;


Begin
     If Not TimeSet
     Then Begin
               GetGPStime( gpstime, valid);
               If valid
               Then Begin
                         day:= gpstime Div 86400;
                         gpstime:= gpstime Mod 86400;

                         hour:= gpstime div 3600;
                         gpstime:= gpstime Mod 3600;

                         minute:= gpstime div 60;
                         gpstime:= gpstime Mod 60;

                         sec:= gpstime;
                         sec100:= 0;

                         SetTime( hour, minute, sec, sec100);
                         TimeSet:= True;

                         SendUserMessage(
                              'Time set to User time of GPS receiver');
                    end;
          End;
End;


Begin
```

```
End.
```

```
Unit GPS;

Interface

{$N+,E+}

Uses MIASglob, crt;

Procedure InitGps( Var GPSdata: gpsdatatype);
{*****************************************************************************}
{Initialise the GPS part of the system.
                              Input :-
                              Output:error; If something went wrong, error is
                                            set to True}
{*****************************************************************************}


Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);
{*****************************************************************************}
{Retrieve data from the GPS receiver connected to the system. Check the col-
 lected data for age. When True, the DGPS flag indicates that DGPS mode is
 active, and only ephemeris and pseudoranges should be valid.
                              Input :DGPS flag
                              Output:GPSdata; relevant data from the GPS
                                              receiver}
{*****************************************************************************}


Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
                   DGPS: Boolean);
{*****************************************************************************}
{ This procedure executes the nescessary calculations for GPS. If the DGPS
  flag is true, only the SV position and the Elevation and Azimuth to the SV
  are calculated (For DGPS no corrections for clock, ionosphere, troposphere
  etc are needed). If the DGPS flag is false, then all these corrections are
  needed.
                               Input : GPSdata, ephemeris etc
                                       DGPS flag
                               Output: GPSdata, SV positions}
{*****************************************************************************}


Procedure ExecGPScommand( command: commandtype);
{*****************************************************************************}
{ This procedure receives a command destined for the GPS part of the MIAS
  system. It passes the command on to the GPSreceiver-part of the system.
                               Input : command
                               Output:-}
{*****************************************************************************}


Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);
{*****************************************************************************}
{Make sure the GPS part is back to 'normal'.
                              Input :-
                              Output:-}
{*****************************************************************************}


Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);
{*****************************************************************************}
{ Get the GPStime from the 'gpsint' variable. The resolution is in seconds.
                              Input :-
                              Output:GPStime and a valid boolean}
{*****************************************************************************}


Implementation

Uses GPSEngine, GPScalc, GPSglob, Miscell, user;

Var
   GPSint       :      GPSinttype;
   El_limit     :      Byte;                   { satellite elevation limit}
   HorAccFac    :      Real;                   { horizontal acceleration
                                                 factor}
   x            :      Integer;                {counter for initialising}


Procedure InitGps( Var GPSdata: gpsdatatype);

Var
   setupfile    :      Text;
   title        :      String;
   line         :      String;
   varname      :      String;
   value        :      String;
   code         :      Integer;
   deg,
   min          :      Double;
   time         :      timetype;
   position     :      positiontype;
   x            :      Integer;                {counter for initialising}
   error        :      Boolean;

Begin
     error:= True;
     GPSdata.flag:= error;
     GPSdata.deltaT:= 0;
     For x:= 1 To 32 Do
          With GPSdata. prn[ x] Do
          Begin
             flag:= True;
             pr  := 0;
          End;
     If ( gpsdata.present = 0)
     Then Exit;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                    { find GPS part of
```

```
                                                 config file}
          Readln( setupfile, title);
    Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPS'));

    If Not Eof( setupfile)                      { if there is more in file}
    Then Repeat
               Readln( setupfile, line);        { get a line}
               Convert( line, varname, value);  { extract the variable name
                                                  and value}
                                                { repeat until end of file}
               If ( varname = 'POSITION.WGS84LAT')
               Then Val( value, position.wgs84lat, code);

               If ( varname = 'POSITION.WGS84LON')
               Then Val( value, position.wgs84lon, code);

               If ( varname = 'POSITION.WGS84ALT')
               Then Val( value, position.wgs84alt, code);

               If ( varname = 'EL_LIMIT')
               Then Val( value, el_limit, code);

               If ( varname = 'HORACCFAC')
               Then Val( value, horaccfac, code);
         Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
    CloseConfig( setupfile);

    InitGPSrec( error);

    ExecGPSreccommand( 'GPS:RESET');

    Date_and_time( time);
    With time Do                                { construct a line to send}
    Begin                                       { to the gpsengine receiver}
         Str( day, value);                      { for initialisation.}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= value + ' ';                    { It contains the date and}
                                                { time, the latitude, longi-}
         Str( month, value);                    { tude and altitude of the}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value+ ' ';              { last position}

         Str( year, value);                     { don't take the 19 from }
         value:= Copy( value, 3, 2);            { 19xx, only take xx}
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value + ' ';             { add zeros}

         Str( hour, value);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value;                   { add zeros}

         Str( minute, value);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value + '00 ';           { add zeros}
    End;
    With position Do
    Begin
         deg:= Abs( Trunc( wgs84lat * 180 / pi));
         Str( deg :2 : 0, value);
         While value[ 1] = ' ' Do               { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 2 - Length( value)) + value;
         line:= line + value;                   { add zeros}

         min:= 60 * ( Abs( wgs84lat * 180 / pi) - deg);
         Str( min :7 : 4, value);
         While value[ 1] = ' ' Do               { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value + ' ';             { add zeros}

         If deg < 0
         Then line:= line + 'S '
         Else line:= line + 'N ';

         deg:= Abs( Trunc( wgs84lon * 180 / pi));
         Str( deg :2 :0, value);
         While value[ 1] = ' ' Do               { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 3 - Length( value)) + value;
         line:= line + value;                   { add zeros}

         min:= 60 * ( Abs( wgs84lon * 180 / pi) - deg);
         Str( min :7 :4, value);
         While value[ 1] = ' ' Do               { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value + ' ';             { add zeros}

         If deg > 0
         Then line:= line + 'E '
         Else line:= line + 'W ';

         If Abs( wgs84alt) > 99999.9            { if overflow}
         Then value:= '99999.9'                 { then take maximum value}
         Else Str( wgs84alt :7 :1, value);
         While value[ 1] = ' ' Do               { delete leading spaces}
               value:= Copy( value, 2, Length( value) -1);
         If wgs84alt < 0
         Then value:= '-' + Copy( zeros, 1, 7 - Length( value)) +
                      Copy( value, 2, Length( value))
         Else value:= Copy( zeros, 1, 7 - Length( value)) + value;
         line:= line + value;                   { add zeros}
    End;
    Str( HorAccFac :4 :1, value);
    While value[1] = ' ' Do
          value:= Copy( value, 2, Length( value) -1);
    value:= Copy( zeros, 1, 4 - Length( value)) + value;
    line:= line + ' ' + value;
```

```
     Str( el_limit: 2, value);
     While value[1] = ' ' Do
          value:= Copy( value, 2, Length( value) -1);
     value:= Copy( zeros, 1, 2 - Length( value)) + value;
     line:= line + ' ' + value;

     ExecGPSreccommand( 'GPS:INIT '+ line);
     ExecGPSreccommand( 'GPS:SEND EPHEMERIS ETC');

     GPSdata.flag:= error;
End;


Procedure GetGPSdata( Var GPSdata: GPSdatatype; DGPS: Boolean);


Var
   currenttime,
   result        :       timetype;
   x             :       Byte;

Begin
     CollectGPSrec( GPSint);

     Date_and_Time( currenttime);                 { check if data valid}
     For x:= 1 To 32 Do                           { check for every sv}
     Begin
       With GPSint.prn[x] Do
       Begin
          If Not flag                             { The flag is True if no}
          Then Begin                              { PR was received. if no}
                                                  { PR is received, exit}
                  AddTime( Tck, Valid_Tck, result);  { also for DGPS mode}
                  flag:= Later( currenttime, result);
                                                  { because health is trans-}
                                                  { mitted in same subframe}
                  If Not flag
                  Then Begin
                          AddTime( Tephem, Valid_Tephem, result);
                          flag:= Later( currenttime, result);
                          If Not flag
                          Then flag:= ( (health And $80) <> 0);
                       End;                       { check health here also}
               End;
       End;      { end of with}
     End;        { End of for}

     GPSint.flag:= True;                          { begin with assumption}
                                                  { that info is incorrect}
     For x:= 1 To 32 Do
         GPSint.flag:= GPSint.flag And GPSint.prn[x].flag;

     If Not DGPS                                  { If not dgps, these should}
     Then With GPSint Do
          Begin                                   { be checked also}
              AddTime( Tionos, Valid_Tionos, result);
              flag:= flag Or Later( currenttime, result);
                                                  { if ionosphere is timed out}
                                                  { then flag should be True}
          End;

     For x:= 1 To 32 Do                           { update external var}
        GPSdata.prn[x].flag:= GPSint.prn[x].flag;
     GPSdata.flag:= GPSint.flag;
End;


Procedure CalcGPS( Var GPSdata: GPSdatatype; position: positiontype;
                   DGPS: Boolean);

Var
   sv_id        :       Byte;
   line,
   dum          :       String;

Begin
     If GPSint.flag                               { GPSint not valid}
     Then Exit;                                   { GPSint is a global
                                                    variable in the GPS units}

     With GPSint Do
     For sv_id:= 1 To 32 Do
     Begin
        If Not prn[ sv_id].flag
        Then Begin
gotoxy (1,5);
write (prn[ sv_id].rxtime:10:2);
                GPSdata.prn[ sv_id]. rxtime:= Round(GPSint.prn[sv_id].rxtime);

                                                  { note: in poscalc deltaT}
                                                  { is substracted. Doing so}
                                                  { here would be obsolete}
                Clockcorrection( sv_id, GPSint); { always correct for clock}
                RelCorrection( sv_id, GPSint);
                If ( Not DGPS) And ( Not position.flag)
                Then Begin
                        L1correction( sv_id, GPSint);
                        Convert_Pos_to_WGS( position);
                        Elev_Azim( sv_id, GPSint, position);
                        Ionosphericcorrection( sv_id, position, GPSint);
{                        Troposphericcorrection( sv_id, position, GPSint);}
                     End;

                SVposition( sv_id, GPSint);
                Calc_Pr( sv_id, GPSint);
                gpsint.deltat:= gpsdata.deltat;  { deltaT from position
                                                    calculation copied to
                                                    internal variable to
                                                    be used to correct
                                                    satellite position}
```

```
                SVpos_earthadjusted( sv_id, GPSint);

                GPSdata.prn[sv_id].position:= GPSint.prn[sv_id].position;
                GPSdata.prn[sv_id].pr:= GPSint.prn[sv_id].pr;
                GPSdata.prn[sv_id].intcarphase:=GPSint.prn[sv_id].intcarphase;
             End;
    End;
End;


Procedure ExecGPScommand( command: commandtype);

Begin
    ExecGPSreccommand( command);
End;


Procedure CloseGps( GPSdata: GPSdatatype; position: positiontype);

Var
  x    :      Byte;
  setupfile    :    Text;
  value        :    String;


Begin
    OpenConfigWrite( setupfile, MIAScfgname);
    Writeln( setupfile, 'GPS');

    With position Do
    Begin
        Str( WGS84lat, value);
        Writeln( setupfile, #9'position.wgs84lat = ', value, ';');

        Str( WGS84lon, value);
        Writeln( setupfile, #9'position.wgs84lon = ', value, ';');

        Str( WGS84alt, value);
        Writeln( setupfile, #9'position.wgs84alt = ', value, ';');
    End;

    Str( el_limit, value);
    Writeln( setupfile, #9'el_limit', ' = ', el_limit, ';');

    Str( horaccfac, value);
    Writeln( setupfile, #9'horaccfac', ' = ', horaccfac, ';');

    CloseConfig( setupfile);

    CloseGPSrec;
End;


Procedure GetGPStime( Var gpstime: Longint; Var valid: boolean);

Var
  x    :      Byte;

Begin
    If gpsint.flag = true
    Then Begin
             valid:= False;
             Exit;
         End
    Else Begin
             x:=1;
             While ( gpsint.prn[x].flag = true) And
                   ( x < 32) Do
                   Inc( x);
             If x <= 32
             Then Begin
                      gpstime:= Round( gpsint.prn[x].rxtime);
                      valid:= True;
                  End
             Else valid:= False;
         End;
End;


Begin                                                    { initialising part}
    GPSint.flag:= True;
    GPSint.numofsat := 0;
    ErrorTime( GPSint.Tionos);
    For x:= 1 To 32 Do
        With GPSint.prn[x] Do
        Begin
           flag:= True;
           Ek:=0;
           pr:= 0;
           ErrorTime( Tck);
           ErrorTime( Tephem);
        End;

    El_limit:= 0;
    HorAccFac:= 0;
End.
```

```
Unit MLS;

{$N+,E+}

Interface

Uses MIASglob;

Procedure InitMLS( Var MLSdata: MLSdatatype);
{****************************************************************************}
{Initialise the MLS sensor. If something went wrong, the error flag is set
 to the value True.          Input :-
                             Output:error}
{****************************************************************************}


Procedure GetMLSdata( Var MLSdata: MLSdatatype);
{****************************************************************************}
{Retrieve MLS data from the MLS sensor.
                             Input :-
                             Output:MLSdata}
{****************************************************************************}



Procedure CalcMLS( Var MLSdata: MLSdatatype);
{****************************************************************************}
{This procedure performs the necessary MLS calculations.
                              Input : MLSdata
                              Output: MLSdata.}
{****************************************************************************}


Procedure ExecMLScommand( command: commandtype);
{****************************************************************************}
{This procedure sends commands to the MLS receiver.
                              Input : command
                              Output: -}
{****************************************************************************}


Procedure CloseMLS( MLSdata: MLSdatatype);

{****************************************************************************}
{Closedown the MLS sensor.  Input :-
                            Output:-}
{****************************************************************************}


Implementation


Uses MLSbendix, MLSglob, Miscell, crt, user;

Var
   MLSint       :       MLSinttype;


Procedure InitMLS( Var MLSdata: MLSdatatype);

Var
   setupfile    :       Text;
   title        :       String;
   line         :       String;
   varname      :       String;
   value        :       String;
   code         :       Integer;
   error        :       Boolean;

Begin
     error:= True;
     MLSdata.flag:= error;

     If ( mlsdata.present = 0)
     Then Exit;

     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                          { find MLS part of
                                                       config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 3) = 'MLS'));

     If Not Eof( setupfile)                          { if there is more in file}
     Then Repeat
                Readln( setupfile, line);            { get a line}
                Convert( line, varname, value);      { extract the variable name
                                                       and value}
                                                     { repeat until end of file}
                                                     { initialise comports for
                                                       communication with Engine}
          Until ( Eof( setupfile) Or ( Pos( ' ', line) = 0));
     CloseConfig( setupfile);

     InitMLSrec( error);
     MLSdata.flag:= error;
End;


Procedure GetMLSdata( Var MLSdata: MLSdatatype);

Var
   result,
   currenttime    :   Timetype;

Begin
     CollectMLSrec( MLSint);

     Date_and_Time( currenttime);                    { check if data valid}

     With MLSint Do
```

```
    Begin
        AddTime( Bas1_time, Valid_Bas1, result);
        Bas1_flag:= Later( currenttime, result);

        AddTime( Bas2_time, Valid_Bas2, result);
        Bas2_flag:= Later( currenttime, result);

        AddTime( Bas3_time, Valid_Bas3, result);
        Bas3_flag:= Later( currenttime, result);

        AddTime( Bas4_time, Valid_Bas4, result);
        Bas4_flag:= Later( currenttime, result);

        AddTime( Bas5_time, Valid_Bas5, result);
        Bas5_flag:= Later( currenttime, result);

        AddTime( Bas6_time, Valid_Bas6, result);
        Bas6_flag:= Later( currenttime, result);

        AddTime( AuxA1_time, Valid_AuxA1, result);
        AuxA1_flag:= Later( currenttime, result);

        AddTime( AuxA2_time, Valid_AuxA2, result);
        AuxA2_flag:= Later( currenttime, result);

        AddTime( AuxA3_time, Valid_AuxA3, result);
        AuxA3_flag:= Later( currenttime, result);

        AddTime( AuxA4_time, Valid_AuxA4, result);
        AuxA4_flag:= Later( currenttime, result);

        AddTime( AuxB_time, Valid_AuxB, result);
        AuxB_flag:= Later( currenttime, result);

        AddTime( AuxC_time, Valid_AuxC, result);
        AuxC_flag:= Later( currenttime, result);

        flag:= (Elangle_flag And Azangle_flag) Or
               Bas1_flag Or Bas2_flag Or Bas3_flag Or Bas4_flag Or
               Bas6_flag Or AuxA1_flag Or AuxA2_flag Or AuxA3_flag;
        flag:= flag And ( BazAngle_flag Or Bas4_flag Or Bas5_flag Or
                          Bas6_flag Or AuxA3_flag Or AuxA4_flag);
                                         { According to Annex 10
                                           p 61, $3.11.5.4, When
                                           BAZ is provided, AuxA4
                                           And Bas5 should be trans-
                                           mitted in the Approach
                                           region also. This is not
                                           implemented here}


        MLSdata.Elangle:= Elangle;
        MLSdata.Azangle:= Azangle;
        MLSdata.BAZangle:= Bazangle;
        MLSdata.DMErange:= DMErange;
        MLSdata.leftclr:= leftclr;
        MLSdata.rightclr:= rightclr;
        MLSdata.ElantInUse:= ElantInUse;
        MLSdata.AzantInUse:= AzantInUse;
        MLSdata.AuxB:= AuxB;
        MLSdata.AuxC:= AuxC;
        MLSdata.ELangle_flag:= Elangle_flag;
        MLSdata.AZangle_flag:= AZangle_flag;
        MLSdata.BAZangle_flag:= BAZangle_flag;
        MLSdata.DME_flag:= DME_flag;
        MLSdata.AuxB_flag:= AuxB_flag;
        MLSdata.AuxC_flag:= AuxC_flag;


        MLSdata.flag:= flag;
    End;
End;


Procedure CalcMLS( Var MLSdata: MLSdatatype);

Begin
    With MLSdata Do
    Begin                                       { calculate the el, az and}
                                                { baz position referenced}
                                                { to the MLS datum point}
        With Azpos Do                           { MLSint is a global var-}
        If ( Not MLSint.AuxA1_flag)             { iable in the MLS units}
        Then Begin
                x:= -1 * MLSint.AuxA1.Az2MLSdatdist;
                y:= MLSint.AuxA1.Azoff;
                z:= 0;
                Azpos_flag:= False;
            End;
        With Elpos Do
        If Not MLSint.AuxA2_flag
        Then Begin
                x:= 0;
                y:= MLSint.AuxA2.Eloff;
                z:= MLSint.AuxA2.ElHeight;
                Elpos_flag:= False;
            End;
        With Bazpos Do
        If ( Not MLSint.AuxA4_flag)
        Then Begin
                x:= MLSint.AuxA4.Baz2MLSdatdist;
                y:= MLSint.AuxA4.Bazoff;
                z:= 0;
                Bazpos_flag:= False;
            End;
        With DMEpos Do
        If ( Not MLSint.AuxA3_flag)
        Then Begin
                x:= -1 * MLSint.AuxA3.DME2MLSdatdist;
                y:= MLSint.AuxA3.DMEoff;
```

```
                    z:= 0;
                    DMEpos_flag:= False;
               End;
          If (Not MLSint.ELangle_flag) And (MLSint.bas2.elstat = 1)
          Then Begin
                    ELangle:= MLSint.ELangle;
                    ELangle_flag:= False;
               End
          Else Elangle_flag:= True;

          If (Not MLSint.AZangle_flag) And (MLSint.bas2.azstat = 1) And
             (Not MLSint.AuxA1_flag)
          Then Begin
                    Azangle:= MLSint.Azangle + MLSint.AuxA1.AzAlignRun;
                    Azangle_flag:= False;
               End
          Else Azangle_flag:= True;

          If (Not MLSint.BAZangle_flag) And (MLSint.bas2.bazstat = 1) And
             (MLSint.bas5.bazstat = 1) And (Not MLSint.AuxA4_flag)
          Then Begin
                    Bazangle:= MLSint.Bazangle - MLSint.AuxA4.BazAlignRun;
                    BAzangle_flag:= False;
               End
          Else BAzangle_flag:= True;

          If (Not MLSint.DME_flag) And (MLSint.bas2.dmestat > 0)
          Then Begin
                    DMErange:= MLSint.DMErange;
                    DME_flag:= False;
               End
          Else DME_flag:= True;

          flag:= ( Elangle_flag Or Elpos_flag) And
                 ( Azangle_flag Or Azpos_flag) And
                 ( Bazangle_flag Or Bazpos_flag) And
                 ( DME_flag Or DMEpos_flag);     { set the MLS flag}

{ NOTE: this is magnetic heading}
          If ( Not MLSint.Bas4_flag) And (Not MLSint.AuxA1_flag)
          Then Begin
                    Runwayhdg:= MLSint.Bas4.AzMagOr -
                                MLSint.AuxA1.AzAlignRun + 180;
                    If Runwayhdg >= 360
                    Then Runwayhdg:= Runwayhdg - 360;
                    Runwayhdg_flag:= False;
               End
          Else Runwayhdg_flag:= True;
     End;
End;


Procedure ExecMLScommand( command: commandtype);

Begin
     ExecMLSreccommand( command);
End;


Procedure CloseMLS( MLSdata: MLSdatatype);

Begin
     CloseMLSRec;
End;


Begin
     With MLSint Do
     Begin
          Bas1_flag:= True;
          Bas2_flag:= True;
          Bas3_flag:= True;
          Bas4_flag:= True;
          Bas5_flag:= True;
          Bas6_flag:= True;
          AuxA1_flag:= True;
          AuxA2_flag:= True;
          AuxA3_flag:= True;
          AuxA4_flag:= True;
          AuxB_flag:= True;
          AuxC_flag:= True;

          ELangle_flag:= True;
          AZangle_flag:= True;
          BAzangle_flag:= True;
          DME_flag:= True;
          discretes_flag:= True;
          leftclr:= True;
          rightclr:= True;
          flag:= True;

          ErrorTime( Bas1_time);
          ErrorTime( Bas2_time);
          ErrorTime( Bas3_time);
          ErrorTime( Bas4_time);
          ErrorTime( Bas5_time);
          ErrorTime( Bas6_time);
          ErrorTime( AuxA1_time);
          ErrorTime( AuxA2_time);
          ErrorTime( AuxA3_time);
          ErrorTime( AuxA4_time);
          ErrorTime( AuxB_time);
          ErrorTime( AuxC_time);
     End;
End.
```

```
Unit GPSEngine;
{****************************************************************************}
{ This unit provides procedures to be used by the GPS unit. The procedures
  are taylor made for the GPSengine from Magnavox.}
{****************************************************************************}


Interface

{$N+,E+}

Uses MIASglob, GPSglob, crt;


Procedure InitGPSrec( Var error: Boolean);
{****************************************************************************}
{Initialise the GPS receiver. If something went wrong, error := True.
                               Input :-
                               Output:error}
{****************************************************************************}


Procedure CollectGPSrec( Var GPSint: GPSinttype);
{****************************************************************************}
{get characters from buffers. Synchronise with the delimiters 'LF' and '$'.
 Check for correct header and process the information to the variable GPSint
                               Input :-
                               Output:GPSint}
{****************************************************************************}


Procedure ExecGPSrecCommand( command: commandtype);
{****************************************************************************}
{ Convert a GPScommand, to a commandstring for the GPS Engine and send
  the command using the IWRITECOM routine in COMM_.int. The commands
  here start with 'GPS:'.
                                Input : command
                                Output: commandstring on comport}
{****************************************************************************}


Procedure CloseGPSrec;
{****************************************************************************}
{Make sure the GPS receiver is back to normal. Restore interuptvectors etc
                               Input :-
                               Output:-}
{****************************************************************************}


Implementation


Uses User, Miscell, com_4;{comdisc}


Const
    LF          =     #10;{ linefeed}

Var
   port0,
   port1        :       Byte;   { contains comportnr for port0 and port1
                                  of the GPSengine}
   x, y         :       Shortint; {counter}
   two2power    :       Array[ 0..55] Of Double;
   sv_id        :       Byte;   { contains the satellite identity number
                                  for ephemeris transmissions}
   Tsv_id,                      { Time when sv_id was read}
   Valid_Tsv_id :       timetype;{Time that sv_id should be valid}
   Tpr,                         { Time when first PR from a set was received}
   Valid_Tpr    :       timetype;{ time that a set is valid, for timeout}
   old_user_ms  :       Real;   { remember old time of measurement}
   svcount      :       Shortint;{ count the number of PR yet received}
   tempGPSint   :       GPSinttype;{ internal var containing all info received}
   completeinfo :       Byte;   { contains info for transfer of completeinfo}
                                { or partial info if available}
                                { 0 for partial, 1 for complete}

{---------------- start procedure initgpsrec-------------------}
Procedure InitGPSrec( Var error: Boolean);

Var
   title,
   line         :       String;
   code         :       Integer;
   varname,
   value        :       String;
   setupfile    :       Text;

Begin
                                                  { set default values}
     port1      :=     1;                         { Engine port 1 is com 1}
     port0      :=     2;                         { Engine port 0 is com 2}
     error:= True;
     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                          { find GPSENGINE part of
                                                       config file}
           Readln( setupfile, title);
     Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'GPSENGINE'));

     If Not Eof( setupfile)                          { if there is more in file}
     Then Repeat
                Readln( setupfile, line);            { get a line}
                Convert( line, varname, value);      { extract the variable name
                                                       and value}
                If (varname = 'PORT0')
                Then Val( value, port0, code);

                If ( varname = 'PORT1')
                Then Val( value, port1, code);

```

```
                If ( varname = 'COMPLETEINFO')
                Then Val( value, completeinfo, code);
            Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
                                                    { repeat until end of file}
     CloseConfig( setupfile);

                                                    { initialise comports for
                                                      communication with Engine}
     Setupcomport( port0, Ord( B4800), 8, Ord( None), 1);
     Setupcomport( port1, Ord( B4800), 8, Ord( None), 1);
                                                    { empty receive and trans-
                                                      mit buffers}
     Emptybuffer( port0, True);
     Emptybuffer( port1, True);
                                                    { set interrupt vectors}
     Installint( port0);                            { save old interrupt}
     Installint( port1);                            { vectors}

     error:= False;
End;
{---------------- end procedure initgpsrec------------------}

{---------------- start procedure CollectGPSrec-------------------}
Procedure CollectGPSrec( Var GPSint: GPSinttype);

   {--------------- start included procedures CollectGPSrec----------}
   Procedure Collectport0( Var GPSint: GPSinttype);
   {*******************************************************************}
   { This procedure collects data from port0 of the GPS ENGINE receiver.
                           Input : data from receive buffer
                           Output: updated GPSint.}
   {*******************************************************************}

   {----------------- start included procedures collectport0---------}
      Procedure StatusReport( rec: String; Var GPSint: gpsinttype);
      {*******************************************************************}
      { This procedure reads the number of satellites being tracked}
      {                        Input : string from GPSengine
                               Output: number of satellites being tracked}
      {*******************************************************************}

      Var
         value  :       String;
         code   :       Integer;

      Begin
           value:= Copy( rec, 12, Length( rec) -11);{ select everything}
                                                    { but the header}
           For x:= 1 To 2 Do
           Begin
               While value[1] <> ',' Do
                     value:= Copy( value, 2, Length( value) -1);
               value:= Copy( value, 2, Length( value) - 1);
           End;

           value:= Copy( value, 1, 1);
           Val( value, GPSint.numofsat, code);

           If code <> 0
           Then GPSint.numofsat := -1;
      End;
   {------------- end included procedures collectport0---------}

   Var
      rec,
      header              :       String;
      recnum,
      code                :       Integer;
      time                :       timetype;
      part                :       String;

   {------------------------Start collectport0------------------------}
   Begin                                             { begin collectport0}
     GPSint:= tempGPSint;

     If (charsinbuff( port0) > 80)                   { wait until string long}
     Then Begin                                      { enough to be valid}
               rec:= Getcharbuff( port0);            { get on char from buffer}
               If ( rec = '$')                       { if begin of recordstring}
               Then Begin
                         Repeat                      { read chars from buffer}
                               rec:= rec + Getcharbuff( port0);
                         Until ( Pos( LF, rec) <> 0) Or
                               ( charsinbuff( port0) = 0);
                                                     { until a line feed found}
                                                     { or no more chars in buff}
                         If ( charsinbuff( port0) = 0)
                         Then Exit;                  { if no lf found, exit}

                                                     { first part of string must}
                                                     { be $PMVSG for Engine}
                         header:= Copy( rec, 1, 7);
                         If ( header = '$PMVXG,')
                         Then Begin                  { next part is recordnumber}
gotoxy( 1, 15);
write( rec);
SaveEquipmentMessage( 'RG: ' + rec);
                                   header:= Copy( rec, 8,3);
                                                     { convert string to value}
                                   Val( header, recnum, code);
                                                     { If no error converting}
                                   If (code = 0)
                                   Then Case recnum Of
                                        0: StatusReport( rec, GPSint);
                                        End
                                   Else;             { what if error converting?}
                              End
                         Else;                       { what if $PMVXG not found?}
                    End
               Else While (( rec <> LF) And
```

```
                          ( Charsinbuff( port0) > 0)) Do
                                                  { if not begin of record}
                                                  { read buffer until lf}
                           rec:= getcharbuff( port0);
        End;
    tempGPSint:= GPSint;
  End;                                            {end collectport0}
  {-------------------------end collectport0------------------------}

  Procedure Collectport1( Var GPSint: GPSinttype);
  (*********************************************************************)
  { This procedure collects data from port1 of the GPS ENGINE receiver.
                            Input : data from receive buffer
                            Output: updated GPSint.}
  (*********************************************************************)

  Type
      temptype         =      Array[1..24] Of Byte;

  {--------------- start included procedures collectport1--------}
     Procedure Conv_ASCII_2_Val( rec: String; Var temp: temptype;
                                    Var error: Boolean);
     (*****************************************************************)
     { This procedure convert a string with pairs of ascii characters
       to byte values. This string should contain 24 pairs of ascii char-
       acters with the 'values' '0'..'9' or 'A'..'F'. The first pair should
       begin at position 6 of the string. When the data is erronous, then
       the error flag is set True.
                            Input : received string
                            Output: array with 24 bytes
                                    error when flag}
     (*****************************************************************)

     Var
        x, y           :       Integer;
        value          :       Byte;
        dum            :       Char;
        code           :       Integer;

     Begin
          error:= False;                         { initialise errorflag}

          For x:= 0 To 23 Do                     { count 24 sets of }
          Begin                                  { characters}
            value:=0;                            { reset value}
            temp[ x+1] := 0;                     { reset almanac}

            For y:= 1 DownTo 0 Do                { convert two characters to}
            Begin                                { value}
               dum:= rec[6 + x * 3 + (1 - y)];  { get digit starting at }
                                                 { position 6}
               If Not ( dum in ['0'..'9', 'A'..'F'])
                                                 { digit is hexadecimal}
               Then Begin
                         error:= True;
                         Exit;                  { error in received record}
                    End;

               Case dum Of
               '0'..'9': Val( dum, value, code);
               'A'..'F': value:= Ord( dum) - Ord( 'A') + 10;
               End;

               value:= Round(value * two2power[ y * 4]);{ first digit is 16's}
                                                 { second digit is 1's}
               temp[ x+1] := temp[ x+1] + value;
            End;
          End;                                   { complete line converted +}
     End;                                        { stored}


     Function Scale( temparray: temptype; pointer, startbit,
                     nr_of_bits: Byte): Double;
     (*****************************************************************)
     { The function takes one or more bytes from an array called tempalmanac
       and converts it into a value. Pointer indicates the first byte from
       the array to be used. Startbit indicates the number of the first bit
       to be used. The MSB has number 0, increasing to the LSB's. Nr_of_bits
       indicates the number of bits to be used.
       WARNING: THIS FUNCTION WILL ONLY WORK A MAXIMUM OF 32 BITS, BECAUSE
                OF THE ROUND FUNCTION.
                            Input : array with bytes
                                    pointer for first byte
                                    startbitnumber
                                    number of bits
                            Output: Value of the number indicated with
                                    the input variables}
     (*****************************************************************)
     Var
          temp             :          Double;
          temppointer      :          Byte;
          x                :          Byte;
          leftoverbits     :          Byte;

     Begin
          temppointer:= pointer;  { save the pointer in the array here}
          temp:= temparray[ pointer];
          Inc( pointer);           { take the pointer'th byte from the array}
          While (startbit + nr_of_bits) - ((pointer - temppointer) * 8) > 0 Do
          Begin                    { repeat this, until enough bytes are taken}
            temp:= temp * 256 + temparray[ pointer];
            Inc( pointer);         { give every byte its position ref weight}
          End;

                            { Delete MSB's that should not be used}
                            { x is number of the bits that should not be used}
                            { substract the values indicated by those bits}
          If startbit > 0
          Then For x:= 0 To startbit - 1 Do
                 If ( temp >= two2power[ (pointer - temppointer) * 8
```

```
                                    - x - 1] )
                Then temp:= temp - two2power[ (pointer - temppointer) * 8
                                          - x - 1];

                              { delete LSB's that should not be used}
                              { x is number of LSB's that should not be used}
          leftoverbits:= - (( startbit + nr_of_bits) -
                           ((pointer - temppointer) * 8));
          While leftoverbits > 0 Do
                                   { divide by 2, to correct for these LSB's}
          Begin                    { as many times as there are LSB's too much}
              Dec( leftoverbits);
              temp:= temp / 2;
              temp:= Trunc( temp);
          End;
          Scale:= temp;
     End;


     Function Twoscomplement( number: Double; nr_of_bits: Byte): Double;
     {*********************************************************************}
     {A number is converted to the decimal 'number' using the natural binary
      code. This number was a twoscomplement number. This function decodes
      'number' 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.
                              Input : binary number
                                      number of bits
                              Output: two's complement of the number}
     {*********************************************************************}
     Begin
          If number > two2power[ nr_of_bits - 1]
          Then Twoscomplement:= - two2power[ nr_of_bits] + number
          Else Twoscomplement:= number;
     End;


     Procedure Rawdata( rec: String; Var GPSint: GPSinttype;
                        Var svcount: Shortint);
     {*****************************************************************}
     { This procedure converts the received string from the Engine, which
       contains rawdata, to variables in the GPSint record.
                              Input : received string
                              Output: variables in GPSint
                                      indication on end of rawdata cycle}
     {*****************************************************************}

     Const
          c     =       299792458;              { speed of light}
          l1freq=      1575420000.0;            { frequency of l1 carrier }

     Var
        user_ms,                                { user time in GPS receiver
                                                  (ms)}
        chnl_ms,                                { channel time in GPS
                                                  receiver (ms)}
        phi,                                    { integrated carrier phase
                                                  l1 wavelengths}
        phi_frac,                               { idem, fractional. LSB=(l1
                                                  wavelength)/256}
        code                                    { raw code affset, l1 wave-
                                                  lengths}
                                :     Extended;
        prn                     :     Byte;     { prn-code of the satellite}
        fault                   :     Integer;
        subrec                  :     String;

        user_time,                              { user time in GPS receiver (s)}
        channel_time                            { channel time in GPS receiver (s)}
                                :     Extended;
     value,
     lineout       :       String;

       Begin
          subrec:= Copy( rec, 9, 2);            { grab satellite prn}
          Val( subrec, prn, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          subrec:= Copy( rec, 12, 9);           { grab user_ms}
          Val( subrec, user_ms, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          subrec:= Copy( rec, 22, 9);           { grab channel_ms}
          Val( subrec, chnl_ms, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          subrec:= Copy( rec, 32, 10);          { grab phi}
          Val( subrec, phi, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          subrec:= Copy( rec, 43, 6);           { grab code}
          Val( subrec, code, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          subrec:= Copy( rec, 50, 4);           { grab phi_frac}
          Val( subrec, phi_frac, fault);
          If ( fault <> 0)                      { error converting ascii}
          Then Exit;                            { to number}

          If (user_ms = old_user_ms)
          Then Begin
                    user_time:= user_ms/1000;   { calculate receive and
                                                  transmit times,
```

```
                                                        see MAGNAVOX guide}

                With GPSint.prn[ prn] Do
                Begin
                     intcarphase:= ( phi_frac / 256);
                     intcarphase:= intcarphase + phi;

                     txtime:= ( chnl_ms / 1000) +
                              ( intcarphase / l1freq) +
                              ( code / l1freq);
                     rxtime:= user_time;
                     flag:= False;

                End;
{ This is made a comment, to make the program work better with a disc.
  In that case, it may assume a wrong number of satellites being tracked
  giving erronous results. By not incrementing 'svcount', the errors can
  be prevented. The complete cycle of measurements is noticed as soon as
  the next cycle begins.}
                Inc( svcount);
           End
      Else Begin
                Restore_buffer( port1, LF+rec);{ put received string back
                                                  for later use}
                old_user_ms:= user_ms;
                svcount:= GPSint.numofsat;

                Date_and_time( Tpr);
           End;
   End;  { End of procedure rawdata}


   Procedure Ionoscor( rec: String; Var GPSint: GPSinttype);
   {*****************************************************************}
   { This procedure converts a received Engine string, containing Iono-
     spheric information, to variables. It takes one, two or three bytes
     from the tempionos, converts this to a value and adds an exponent.
     The information used, can be found in Appendix 3 to Annex A to
     STANAG 4249, 1 August 1990.
                              Input : received recordnumber
                                      received string
                              Output: updated almanac in GPSint}
   {*****************************************************************}

   Var
      tempionos      :       temptype;
      error          :       Boolean;

   Begin
        Conv_ASCII_2_val( rec, tempionos, error);
        If error
        Then Exit;
        With GPSint.iono Do
        Begin
             alfa0:= Twoscomplement( Scale( tempionos, 2, 0, 8), 8);
             alfa0:= alfa0 / two2power[ 30];

             alfa1:= Twoscomplement( Scale( tempionos, 3, 0, 8), 8);
             alfa1:= alfa1 / two2power[ 27];

             alfa2:= Twoscomplement( Scale( tempionos, 4, 0, 8), 8);
             alfa2:= alfa2 / two2power[ 24];

             alfa3:= Twoscomplement( Scale( tempionos, 5, 0, 8), 8);
             alfa3:= alfa3 / two2power[ 24];

             beta0:= Twoscomplement( Scale( tempionos, 6, 0, 8), 8);
             beta0:= beta0 * two2power[ 11];

             beta1:= Twoscomplement( Scale( tempionos, 7, 0, 8), 8);
             beta1:= beta1 * two2power[ 14];

             beta2:= Twoscomplement( Scale( tempionos, 8, 0, 8), 8);
             beta2:= beta2 * two2power[ 16];

             beta3:= Twoscomplement( Scale( tempionos, 9, 0, 8), 8);
             beta3:= beta3 * two2power[ 16];
        End; { End of with}
        Date_and_time( GPSint.Tionos);
   End;      { End of procedure Ionoscor}


   Procedure Get_SV_id( rec: String; Var sv_id: Byte);
   {*****************************************************************}
   { This procedure is needed, because the gpsengine sends the ephemeris
     in several records. Only the first record contains the SV-id of the
     SV from which the ephemeris originates
                         Input : received string
                         Output: satellite identity}
   {*****************************************************************}

   Var
      subrec         :       String;
      code           :       Integer;

   Begin
      subrec:= Copy( rec, 6, 3);              { sv-id, position unknown}
      Val( subrec, sv_id, code);              { convert string to value}
      If ( code <> 0)                         { on error clear sv_id}
      Then sv_id:=0;

      Date_and_time( Tsv_id);
   End;


   Procedure Clockinfo( rec: String; sv_id: Byte;
                        Var GPSint: GPSinttype);
   {*****************************************************************}
   { This procedure converts received Engine strings, containing Clock-
     information per satellite, to variables. It takes one, two or three
```

```
    bytes from the tempclock, converts this to a value and adds an
    exponent. The information used, can be found in Appendix 3 to Annex
    A to STANAG 4249, 1 August 1990.
                        Input : received string
                                satellite identity
                        Output: updated clock info per SV in GPSint}
  {******************************************************************}
  Var
     tempclock      :      temptype;
     error          :      Boolean;
     sumtime,
     time           :      timetype;
     dum            :      Double;

  Begin
       With GPSint Do
       Begin
            Date_and_Time( time);           { get current system time}
            Addtime( Tsv_id, Valid_Tsv_id, sumtime);
            If (sv_id = 0) Or Later( time, sumtime)
                                            { if Satellite identity
                                              number is 0, or the infor-
                                              mation is timed out, than
                                              no valid information}
            Then Begin
                        ErrorTime( Tsv_id);
                        Exit;
                  End;
       End;

       Conv_ASCII_2_val( rec, tempclock, error);
       If error
       Then Exit;

       With GPSint.prn[ sv_id].clock Do
       Begin
            Tgd:= Twoscomplement( Scale( tempclock, 15, 0, 8), 8);
            Tgd:= Tgd / two2power[ 31];

            toc:= Scale( tempclock, 17, 0, 16);
            toc:= Round( toc * two2power[ 4]);
            If toc > 604784                 { check on range}
            Then Begin
                      ErrorTime( GPSint.prn[ sv_id].Tck);
                      Exit;
                  End;

            af2:= Twoscomplement( Scale( tempclock, 19, 0, 8), 8);
            af2:= af2 / two2power[ 55];

            af1:= Twoscomplement( Scale( tempclock, 20, 0, 16), 16);
            af1:= af1 / two2power[ 43];

            af0:= Twoscomplement( Scale( tempclock, 22, 0, 22), 22);
            af0:= af0 / two2power[ 31];

            dum:= Scale( tempclock, 3, 6, 2) * 256;
            IODC:= Integer( Round(dum));
            dum:= Scale( tempclock, 16, 0, 8);
            IODC:= IODC + Integer( ROund(dum));
       End;      { End with}
       With GPSint.prn[ sv_id] Do
       Begin
            health:= Round( Scale( tempclock, 3, 0, 6));
                                            { set the 6 LSB's of
                                              health by health from
                                              SV record}
            If Round( Scale( tempclock, 3, 0, 1)) = 0
            Then health:= ( health And $1F)  { healthy force zero's}
            Else health:= ( health Or $E0);  { unhealthy force one's}
                                            { use MSB from SV record
                                              to set health to 'alldata
                                              bad' or 'all data ok'}
            Date_and_Time( Tck);
       End;      { End with}
  End;           { End procedure clockinfo}


  Procedure Ephemeris( recnum: Integer; rec: String; sv_id: Byte;
                       Var GPSint: GPSinttype);
  {******************************************************************}
  { This procedure converts received Engine strings, containing Ephem-
    eris information, to variables. It takes one, two or three bytes
    from the tempephem, converts this to a value and adds an exponent.
    The information used, can be found in Appendix 3 to Annex A to
    STANAG 4249, 1 August 1990.
                        Input : received recordnumber
                                received string
                                satellite identity
                        Output: updated ephemeris in GPSint}
  {******************************************************************}
  Var
     tempephem      :      temptype;
     error          :      Boolean;
     time,
     sumtime        :      timetype;

  Begin
       Date_and_Time( time);                { get current system time}
       Addtime( Tsv_id, Valid_Tsv_id, sumtime);
       If (sv_id = 0) Or Later( time, sumtime)
                                            { if Satellite identity
                                              number is 0, or the infor-
                                              mation is timed out, than
                                              no valid information}
       Then Begin
                   ErrorTime( Tsv_id);
                   Exit;
             End;
```

```
        Conv_ASCII_2_val( rec, tempephem, error);
        If error
        Then Exit;

        With GPSint.prn[ sv_id].ephemeris Do
        Begin
          Case recnum Of
          202: Begin
                 IODE:= tempephem[ 1];

                 Crs:= Twoscomplement( Scale( tempephem, 2, 0, 16), 16);
                 Crs:= Crs / two2power[ 5];

                 deltan:= Twoscomplement( Scale( tempephem, 4, 0, 16), 16);
                 deltan:= deltan / two2power[ 43];

                 Mo:= Twoscomplement( Scale( tempephem, 6, 0, 32), 32);
                 Mo:= Mo / two2power[ 31];

                 Cuc:= Twoscomplement( Scale( tempephem, 10, 0, 16), 16);
                 Cuc:= Cuc / two2power[ 29];

                 e:= Scale( tempephem, 12, 0, 32);
                 e:= e / two2power[ 33];
                 If e > 0.03                  { Check on range}
                 Then Begin
                        ErrorTime( GPSint.prn[ sv_id].Tephem);
                        Exit;
                      End;

                 Cus:= Twoscomplement( Scale( tempephem, 16, 0, 16), 16);
                 Cus:= Cus / two2power[ 29];

                 Asqrt:= Scale( tempephem, 18, 0, 32);
                 Asqrt:= Asqrt / two2power[ 19];

                 toe:= Scale( tempephem, 22, 0, 16);
                 toe:= toe * two2power[ 4];
                 If toe > 604784              { Check on range}
                 Then Begin
                        ErrorTime( GPSint.prn[ sv_id].Tephem);
                        Exit;
                      End;

                 Date_and_Time( GPSint.prn[ sv_id].Tephem);
               End;   { End of Case 202}
          203: Begin
                 If (tempephem[22] <> IODE)   {new ephemeris being}
                 Then Begin                   {uploaded. Subframe 2}
                      ErrorTime( GPSint.prn[ sv_id].Tephem);
                      Exit                    {ready, subframe 3 not}
                      End;                    {don't use this sv now}

                 Cic:= Twoscomplement( Scale( tempephem, 1, 0, 16), 16);
                 Cic:= Cic / two2power[ 29];

                 omegao:= Twoscomplement( Scale( tempephem, 3, 0, 32), 32);
                 omegao:= omegao / two2power[ 31];

                 Cis:= Twoscomplement( Scale( tempephem, 7, 0, 16), 16);
                 Cis:= Cis / two2power[ 29];

                 io:= Twoscomplement( Scale( tempephem, 9, 0, 32), 32);
                 io:= io / two2power[ 31];

                 Crc:= Twoscomplement( Scale( tempephem, 13, 0, 16), 16);
                 Crc:= Crc / two2power[ 5];

                 omega:= Twoscomplement( Scale( tempephem, 15, 0, 32),
                                                32);
                 omega:= omega / two2power[ 31];

                 omegadot:= Twoscomplement( Scale( tempephem, 19, 0, 24),
                                                24);
                 omegadot:= omegadot / two2power[ 43];

                 IDOT:= Twoscomplement( Scale( tempephem, 23, 0, 14), 14);
                 IDOT:= IDOT / two2power[ 43];

                 Date_and_Time( GPSint.prn[ sv_id].Tephem);
                 sv_id:= 0;                   { When last record for
                                                satellite nr: sv_id is
                                                used, then reset sv_id
                                                so, no errors are made
                                                if records 201..203 are
                                                received without record
                                                200( correct sv_id)}
               End;    { End of Case 203}
          End;         { End of case}
        End;           { End of with}
  End;                 { End of procedure Ephemeris}

{----------------- end included procedures collectport1----------}
Var
  rec,
  dumstr          :     String;
  dumch           :     Char;
  recnum,
  code            :     Integer;
  dum             :     Integer;
  x               :     Byte;
  time,
  result          :     timetype;
  part            :     String;

{--------------- start procedure collect port1-----------------}
Begin
  GPSint:= tempGPSint;                        { update GPSint with temp}
                                              { orarily stored GPSint}
  If (Charsinbuff( port1) > 80)               { only start if there}
```

```
    Then Begin                                  { are enough chars in buff}
            rec:= Getcharbuff( port1);          { Get first char}

            If ( rec = LF)                      { If this is a line feed}
            Then Begin                          { then we expect a new}
                    rec:='';                    { leave a LF in buffer}
                    Repeat                      { line to start}
                         dumch:= LookBuff( port1);
                         If ( dumch <> LF)
                         Then rec:= rec + Getcharbuff( port1);
                    Until ( dumch = LF) Or ( charsinbuff( port1)= 0);
                                                { continue until end of
                                                  line is detected, or
                                                  no more chars}
                                                { if no more chars, then
                                                  error in received line}
                    If ( charsinbuff( port1) = 0)
                    Then Exit;
                    If ( rec[4] = ' ')          { for right alignment of
                                                  record number}
                    Then rec:= ' '+ rec;        { insert space if necessary}

                                                { copy the number}
                    dumstr:= Copy( rec, 1, 4);
                                                { convert string to value}
                    Val( dumstr, recnum, code);
                    If ( code = 0)              { if no error converting}
                    Then Begin
gotoxy( 1, 17);
write( rec);
SaveEquipmentMessage( 'RG: ' + rec);
                          Case recnum Of
                          1         : Rawdata( rec, GPSint, svcount);
                          135       : Ionoscor( rec, GPSint);
                          200       : Get_SV_id( rec, sv_id);
                          201       : Clockinfo( rec, sv_id, GPSint);
                          202..203  : Ephemeris( recnum, rec, sv_id,
                                                   GPSint);
                          End;                  { end case}
                        End                     { end begin}
                    Else ;                       { what to do if error?}
                End
            Else Begin                          { if no begin of line}
                    Repeat                      { skip chars from buffer}
                          dumch:= LookBuff( port1);
                          If ( dumch <> LF)
                          Then dumch:= Getcharbuff( port1);
                    Until ( dumch = LF) Or ( charsinbuff( port1) = 0);
                                                { a lf was found or the
                                                  buffer is empty}
                 End;
        End; { end if charsinbuff}
        tempGPSint:= GPSint;

        Date_and_Time( time);
        Addtime( Tpr, Valid_Tpr, result);
        If Later( time, result) Or ( svcount = GPSint.numofsat)
        Then Begin
               For x:= 1 To 32 Do
                  tempGPSint.prn[x].flag:= True;{ reset flags if all PR's}
                                                { are present}
               svcount:= 0;
             End
        Else If (completeinfo = 1)              { If only complete info}
             Then For x:= 1 To 32 Do            { should be transfered}
                     GPSint.prn[x].flag:= True;{ keep flags True}
   End; { end collectport1}
   {---------------- end collectport1-------------------------}
   {---------------- end included procedures CollectGPSrec-----------}

Var
   counter      :       Integer;
   {---------------- start procedure CollectGPSrec------------------}
Begin
    counter:= 1;                                { now we read 2 message}
    Repeat                                      { from the engine receiver}
          Collectport0( GPSint);                { these contain most certain}
          Collectport1( GPSint);                { ly all the measurements}
          Dec( counter);
    Until (counter = 0);
End;
{---------------- end procedure CollectGPSrec------------------}


{---------------- start procedure execgpsreccommand-----------}
Procedure ExecGPSrecCommand( command: commandtype);
                                                { IN THE COMMENT, UNDER-
                                                  SCORES '_' SHOULD BE
                                                  READ AS SPACES ' '}
Var
   substr,
   sendline     :       String;
   code,
   sv           :       Integer;
   rate         :       Integer;

Begin
    sendline:= '';

    substr:= Copy( command, 5, Length( command) - 4);
                                                { delete header: 'GPS:'}
    If substr = 'RESET'
    Then Begin
            sendline:= '$PMVXG,018,T'#13#10;    { RESTART THE ENGINE WITH
                                                  TEPID START}
{           IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
}          End;                                 { clear port0 outputlist}
```

```
    If (Copy( substr, 1, 4) = 'INIT') And
       ( Length( substr) >= 52)
    Then Begin
              sendline:= '$PMVXG,000,'+
                    Copy( substr, 6, 2) + ',' +
                    Copy( substr, 9, 2) + ',' +  { INITIALISES THE ENGINE}
                    '19' +
                    Copy( substr, 12, 2) + ',' + { RECEIVER WITH POSITION}
                    Copy( substr, 15, 6) + ',' + { ETC. FORMAT IS: 'INIT_}
                    Copy( substr, 22, 9) + ',' + { DD_MM_YY_HHMM_DDMM.MMMM}
                    Copy( substr, 32, 1) + ',' + { _N_DDDMM.MMMM_E_HHHHH.H_}
                                                 { AA.A_EL'}
                    Copy( substr, 34, 10)+ ',' + { WHERE DD_MM_YY IS THE}
                    Copy( substr, 45, 1) + ',' + { DATE IN DAY, MONTH, AND}
                    Copy( substr, 47, 7) + ',' +
                    #13#10;

              IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);


              sendline:=
                   '$PMVXG,001,,,' +             { YEAR. HHMM IS TIME, }
                    Copy( substr, 55, 4) + ',' + { DDMM.MMMM IS LATITUDE}
                    ',,,' +                      { IN DEGREES AND MINUTES,}
                    Copy( substr, 60, 2) + ',' + { N IS NORTH OR SOUTH  }
                    ','+#13#10;                  { (N/S), DDDMM.MMMM IS  }
                                                 { LONGITUDE IN DEGREES }
                                                 { AND MINUTES, E IS EAST}
                                                 { OR WEST (E/W) AND    }
                                                 { HHHHH.H IS ALTITUDE  }
                                                 { ABOVE MEAN SEA LEVEL,}
                                                 { AA.A IS HORIZONTAL}
                                                 { ACCELERATION FACTOR}
                                                 { EL IS ELEVATION LIMIT}
              IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);

              sendline:= '$PMVXG,024,-,+,+,-,-,-,-'#13#10;
              IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);

                                               { NO NAV RESULT, RAW
                                                 MEASUREMNENTS, APMANAC &
                                                 EPHEMERIS, NO CONTROL
                                                 INFO, NO TIME RECOVERY,
                                                 NO FULL DEBUG, NO PARTIAL
                                                 DEBUG}
              sendline:= '$PMVXG,023,D,G,,,,,'#13#10;
              IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
                                               { SET TIME RECOVERY ON}
              sendline:= '$PMVXG,007,,1,,,,,,,'#13#10;
              IWriteCom( port0, sendline);     { NO OUTPUT ON CTRL PORT}
SaveEquipmentMessage( 'SG: ' + sendline);
                   sendline:= '$PMVXG,007,000,0,1,,1,,,,,'#13#10;
                   IWriteCom( port0, sendline);      { OUTPUT MESSAGE 000}
SaveEquipmentMessage( 'SG: ' + sendline);
                   sendline:= '$PMVXG,007,001,0,1,,1,,,,,'#13#10;
                   IWriteCom( port0, sendline);      { OUTPUT MESSAGE 001}
SaveEquipmentMessage( 'SG: ' + sendline);
                   sendline:= '$PMVXG,007,021,0,1,,1,,,,,'#13#10;
                   IWriteCom( port0, sendline);      { OUTPUT MESSAGE 021}
SaveEquipmentMessage( 'SG: ' + sendline);

              End;
    If substr = 'SEND EPHEMERIS ETC'
    Then Begin
                sendline:= '$PMVXG,027,,,,2,2'#13#10;
                IWriteCom( port0, sendline);
SaveEquipmentMessage( 'SG: ' + sendline);
              End;                                  { OUTPUT EPHEMERIS AND
                                                      ALMANAC NOW}

End;
{--------------- End procedure execgpsreccommand-----------}


{--------------- start procedure closegpsrec-----------}
Procedure CloseGPSrec;

Var
   setupfile    :       Text;
   value        :       String;

Begin
     Removeint( port0);
     Removeint( port1);

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'GPSENGINE');

     Str( port0, value);
     Writeln( setupfile, #9'port0 = ', value, ';');

     Str( port1, value);
     Writeln( setupfile, #9'port1 = ', value, ';');

     Str( completeinfo, value);
     Writeln( setupfile, #9'completeinfo = ', value, ';');
     CloseConfig( setupfile);
End;
{--------------- end procedure closegpsrec-----------}



{--------------- start initialising ----------------}
Begin                                           { initialising part}
                                                { used to make a table}
                                                { containing the power of 2}
```

```
     two2power[ 0]:= 1;                      { fill most left position}
     For x:= 1 To 55 Do                      { calculate 2^1 to 2^55}
         two2power[ x]:= two2power[ x-1] * 2;

     sv_id:=0;                               { set sv_id to 0, which is
                                               invalid, so no mistake can
                                               be made}
     ErrorTime( Tsv_id);
     ErrorTime( Valid_Tsv_id);
     Valid_Tsv_id.minute:= 5;                { the sv_id stays valid for 5}
                                             { minutes ( see record 200)}
     ErrorTime( Tpr);
     ErrorTime( Valid_Tpr);                  { the pr are maximum valid for}
     Valid_Tpr.sec:= 1;                      { 1 second, this is the update}
                                             { rate of the GPSengine rx}
     old_user_ms:= 0;
     svcount:= 0;

     With tempGPSint Do                     { make an empty GPSint set}
     Begin
          flag:= True;
          ErrorTime( Tionos);
          For x:= 1 To 32 Do
          With prn[x] Do
          Begin
               flag:= True;
               ErrorTime( Tck);
               ErrorTime( Tephem);
          End;
          numofsat:= 0;
     End;

     completeinfo:= 1;
End.
{---------------- end initialising ------------------}
{---------------- end Unit GPSENGINE ------------------}
```

```
Unit MLSbendix;
{***************************************************************************}
{ THis unit is meant to be used with a Bendix MLS-20A receiver. This
  receiver is a quasi MLSarinc727 receiver. The mls data words are not
  passed through by this receiver. The basic datawords are simulated by
  assigning variables with static basic data values}
{***************************************************************************}

{$N+,E+}

Interface


Uses MIASglob, MLSglob, crt;


Procedure InitMLSrec( Var error: Boolean);
{***************************************************************************}
{ Initialise the Arinc 727 receiver. If something went wrong, error := True.
                              Input : -
                              Output: error}
{***************************************************************************}


Procedure CollectMLSrec( Var MLSint: MLSinttype);
{***************************************************************************}
{ Get Arinc 429 words; Skip the words that are not necessary; check the
  necessary words on parity. ADW's are checked on CRC.
                              Input : -
                              Output: Mlsint}
{***************************************************************************}


Procedure ExecMLSrecCommand( command: commandtype);
{***************************************************************************}
{ Convert a MLScommand, to a command word for the ARinc 727 MLS receiver.
  That is, a Arinc 429 word. Then send the word using the Arinc 429 tx
  channel.
                              Input : command
                              Output: Arinc 429 word on 429 tx port}
{***************************************************************************}


Procedure CloseMLSrec;
{***************************************************************************}
{ Make sure the ARINC 727 MLS receiver is back to normal. Restore interrupt-
  vectors etc.
                              Input : -
                              Output: -}
{***************************************************************************}


Implementation

Uses Ar429comm, Ar429, Miscell, User, ADW;

Type
    ADW_prestype        =       Array[1..4] Of Boolean;
    ByteArray           =       Array[1..10] Of Byte;

Const
     mlsfreq_lab=       036;                    { mlsfrequency label}
     azimut_lab =       151;                    { azimuth function label}
     elevat_lab =       152;                    { elevation function label}
     BackAz_lab =       240;                    { backazimuth function label}

Var
   ADW_A_pres,
   ADW_B_pres,
   ADW_C_pres           :       ADW_prestype;
   ADW_A,
   ADW_B,
   ADW_C                :       ADWtype;
   x                    :       Integer;
   AngleBegin           :       Boolean;
   tempMLSint           :       MLSinttype;
   Tangle,
   Valid_Tangle         :       timetype;
   completeinfo         :       Byte;
   irq                  :       Byte;
   cardaddress          :       Word;
   badwline             :       String;
   bytes                :       ByteArray;
   bytecount            :       Byte;
   wordready            :       Boolean;

{-------------------------Start InitMLSrec------------------------}
Procedure InitMLSrec( Var error: Boolean);

Var
   title,
   line         :       String;
   code         :       Integer;
   varname,
   value        :       String;
   setupfile    :       Text;
   selecttable  :       arraytype;

Begin
     error:= True;
     cardaddress:= $280;
     irq:= 7;
     badwline:= '';
     wordready:= False;
     bytecount:= 1;


     OpenConfigRead( setupfile, MIAScfgname);
     Repeat                                     { find MLSBENDIX part of
                                                  config file}
```

```
            Readln( setupfile, title);
      Until (EOF( setupfile) OR ( Copy( title, 1, 9) = 'MLSBENDIX'));

      If Not Eof( setupfile)                      { if there is more in file}
      Then Repeat
                 Readln( setupfile, line);        { get a line}
                 Convert( line, varname, value);  { extract the variable name
                                                    and value}
                 If ( varname = 'COMPLETEINFO')
                 Then Val( value, completeinfo, code);

                 If ( varname = 'CARDADDRESS')
                 Then Val( value, cardaddress, code);

                 If ( varname = 'IRQ')
                 Then Val( value, irq, code);
           Until ( Eof( setupfile) Or ( (line[1] <> #9) And (line[1] <> ' ')));
                                                  { repeat until end of file}
      CloseConfig( setupfile);

      Ar429comm.InitAr429;                        { Initialise comm-port}
                                                  { for VHF datalink}
      selecttable[ 0]:= mlsfreq_lab;              { Initialise ARINC card}
      selecttable[ 1]:= azimut_lab;               { by selecting the labels}
      selecttable[ 2]:= elevat_lab;               { for triggering}
      selecttable[ 3]:= backaz_lab;

      Ar429.InitAr429( error, selecttable, 4);

      wordready:= False;                          { Initialise ADW-card}
      bytecount:= 1;

      InitKaartadres( cardaddress);
      KiesIRQ( IRQ);
      Install_ADW_int;
      ProgTrigFunktie( 1, $0A);                   { Select BDW 1}
      ProgTrigFunktie( 2, $1F);                   { Select BDW 2}
      ProgTrigFunktie( 3, $55);
      ProgTrigFunktie( 4, $11);
      ProgTrigFunktie( 5, $1B);
      ProgTrigFunktie( 6, $58);                   { Select BDW 6}
      ProgTrigFunktie( 7, $27);                   { Select ADW A}
End;
{------------------------End InitMLSrec------------------------}


{------------------------Start CollectMLSrec------------------------}
Procedure CollectMLSrec( Var MLSint: MLSinttype);

   {--------------- start included procedures CollectMLSrec-----------}
   Function ADW_present( ADW_pres: ADW_prestype): Boolean;
   {******************************************************************}
   { Check if all 4 arinc429 words, that form an adw, are present. A
     true is output if all 4 are present, if not, a false is output.
                           Input : Array with present flags
                           Output: Boolean.}
   {******************************************************************}

   Begin
        ADW_present:= ADW_pres[ 1] And ADW_pres[ 2] And
                      ADW_pres[ 3] And ADW_pres[ 4];
   End;


   Function Hamming_Fail( ADW: ADWtype): Boolean;
   {******************************************************************}
   { This function checks the hammingcode in the specified ADW. If the
     hammingcode was correct, then the output is true. See Annex 10 p 150B
                           Input : ADW
                           Output: Boolean}
   { Author: Maarten Uit de Haag;
     Revised: Marco Meijer}
   {******************************************************************}
   Var
      x        :       Integer;
      succes   :       Boolean;
      check    :       Array[ 0..6] Of Integer;

   Begin
        succes:= True;

        For x:= 0 To 4 Do
        Begin
             check[ x]:= ADW[13 + x] + ADW[14 + x] + ADW[15 + x] +
                         ADW[16 + x] + ADW[17 + x] + ADW[18 + x] +
                         ADW[20 + x] + ADW[22 + x] + ADW[24 + x] +
                         ADW[25 + x] + ADW[28 + x] + ADW[29 + x] +
                         ADW[31 + x] + ADW[32 + x] + ADW[33 + x] +
                         ADW[35 + x] + ADW[36 + x] + ADW[38 + x] +
                         ADW[41 + x] + ADW[44 + x] + ADW[45 + x] +
                         ADW[46 + x] + ADW[50 + x] + ADW[52 + x] +
                         ADW[53 + x] + ADW[54 + x] + ADW[55 + x] +
                         ADW[58 + x] + ADW[60 + x] + ADW[64 + x] +
                         ADW[65 + x];
              check[ x]:= check[ x] Mod 2;
              succes:= succes And ( check[ x] = ADW[ 70 + x]);
         End;
         check[ 5]:=     ADW[13 ] + ADW[14 ] + ADW[15 ] + ADW[16 ] +
                         ADW[17 ] + ADW[19 ] + ADW[21 ] + ADW[23 ] +
                         ADW[24 ] + ADW[27 ] + ADW[28 ] + ADW[30 ] +
                         ADW[31 ] + ADW[32 ] + ADW[34 ] + ADW[35 ] +
                         ADW[37 ] + ADW[40 ] + ADW[43 ] + ADW[44 ] +
                         ADW[45 ] + ADW[49 ] + ADW[51 ] + ADW[52 ] +
                         ADW[53 ] + ADW[54 ] + ADW[57 ] + ADW[59 ] +
                         ADW[63 ] + ADW[64 ] + ADW[69 ];
         check[ 5]:= check[ 5] Mod 2;
         succes:= succes And ( check[ 5] = ADW[ 75]);

         check[ 6]:= 0;
         For x:= 13 To 75 Do
```

```
             check[ 6]:= check[ 6] + ADW[ x];
         check[ 6]:= check[ 6] Mod 2;
         succes:= succes And ( check[ 6] = ADW[ 76]);

         Hamming_Fail:= Not Succes;
   End;


   Function Adress_Fail( ADW: ADWtype): Boolean;
   {*****************************************************************}
   { This function checks the parity of the adress in the specified ADW.
     If the adress was correct, then the output is False. See Annex 10 p 150
                               Input : ADW
                               Output: Boolean}
   { Author: Maarten Uit de Haag;
     Revised: Marco Meijer}
   {*****************************************************************}
   Var
      check     :       Array[ 1..2] Of Integer;

   Begin
        check[ 1]:= ADW[ 13] + ADW[ 14] + ADW[ 15] + ADW[ 16] + ADW[ 17] +
                    ADW[ 18];
        check[ 1]:= check[ 1] Mod 2;
        check[ 2]:= ADW[ 14] + ADW[ 16] + ADW[ 18];
        check[ 2]:= check[ 2] Mod 2;

        Adress_Fail:= Not ((check[ 1] = ADW[ 19]) And
                           ( check[ 2] = ADW[ 20]));
   End;



   Function ADW_adress( ADW: ADWtype): Byte;
   {*****************************************************************}
   { This function takes bits 13 to 18 of the ADW and translates these
     bits to an adress. This is a normal binary code. See Annex 10 p 150
                               Input : ADW
                               Output: adress}
   {*****************************************************************}
   Var
      adress,
      mult,
      x         :       Integer;

   Begin
        adress:= 0;
        mult:= 32;
        For x:= 0 To 5 Do
        Begin
             adress:= adress + ADW[ 13 + x] * mult;
             mult:= mult Div 2;
        End;
        ADW_adress:= adress;
   End;


   Function Conv_ADW( ADW: ADWtype; start: Byte; number: Byte): Word;
   {*****************************************************************}
   { This function converts single bits to a number. 'Start' indicates
     the start bit in the ADW. First bit is bitnumber 1. 'Number' indicates
     the number of bits to be used for the number to be formed. The number
     is output as a Word. See Annex 10 p 60CC; LSB first
                               Input : ADW
                                       startbit
                                       number of bits
                               Output: number}
   {*****************************************************************}
   Var
      value,
      mult      :       Word;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 0 To number - 1 Do
        Begin
             value:= value + ADW[ start + x] * mult;
             mult:= mult * 2;
        End;
        Conv_ADW:= value;
   End;

   Procedure ADW_A_conv( Var MLSint: mlsinttype; ADW_A: ADWtype);

   Var
      adress    :       Byte;

   Begin
                adress:= ADW_Adress( ADW_A);
                Case adress Of
                1:Begin
                     Date_and_Time( Mlsint.AuxA1_time);
                     With Mlsint.auxa1 Do
                     Begin
                       If ADW_A[ 30] = 1        { MSB is sign bit}
                                                { see Annex 10 p150}
                       Then AzOff:= -1 * Conv_ADW( ADW_A, 21, 9)
                       Else AzOff:= Conv_ADW( ADW_A, 21, 9);

                       Az2MLSdatDist:= Conv_ADW( ADW_A, 31, 13);

                       If ADW_A[55] = 1
                       Then AzAlignRun:= -1 * Conv_ADW( ADW_A, 44, 11)
                       Else AzAlignRun:= Conv_ADW( ADW_A, 44, 11);
                       AzAlignRun:= AzAlignRun * 0.01;

                       AzCoorSyst:= ADW_A[ 56];
                     End;
                     Mlsint.Auxa1_flag:= False;
```

```
                End;
              2:Begin
                   Date_and_Time( Mlsint.AuxA2_time);
                   With Mlsint.auxa2 Do
                   Begin
                      If ADW_A[ 30] = 1
                      Then ElOff:= -1 * Conv_ADW( ADW_A, 21, 9)
                      Else ElOff:= Conv_ADW( ADW_A, 21, 9);

                      MLSdat2thres:= Conv_ADW( ADW_A, 31, 10);

                      If ADW_A[ 47] = 1
                      Then ElHeight:= -1 * Conv_ADW( ADW_A, 41, 6)
                      Else ElHeight:= Conv_ADW( ADW_A, 41, 6);
                      ElHeight:= ElHeight * 0.1;
                   End;
                   Mlsint.Auxa2_flag:= False;
                End;
              3:Begin
                   Date_and_Time( Mlsint.AuxA3_time);
                   With Mlsint.auxa3 Do
                   Begin
                      If ADW_A[ 30] = 1
                      Then DMEoff:= -1 * Conv_ADW( ADW_A, 21, 9)
                      Else DMEoff:= Conv_ADW( ADW_A, 21, 9);

                      If ADW_A[ 44] = 1
                      Then DME2MLSdatdist:= -1 * Conv_ADW( ADW_A, 31, 13)
                      Else DME2MLSdatdist:= Conv_ADW( ADW_A, 31, 13);
                   End;
                   Mlsint.Auxa3_flag:= False;
                End;
              4:Begin
                   Date_and_Time( Mlsint.AuxA4_time);
                   With Mlsint.auxa4 Do
                   Begin
                      If ADW_A[ 30] = 1
                      Then BAzoff:= -1 * Conv_ADW( ADW_A, 21, 9)
                      Else BAzoff:= Conv_ADW( ADW_A, 21, 9);

                      BAz2MLSdatdist:= Conv_ADW( ADW_A, 31, 11);

                      If ADW_A[ 53] = 1
                      Then BAzAlignRun:= -1 * Conv_ADW( ADW_A, 42, 11)
                      Else BAzAlignRun:= Conv_ADW( ADW_A, 42, 11);
                      BAzAlignRun:= BAzAlignRun * 0.01;
                   End;
                   Mlsint.Auxa3_flag:= False;
                End;
              End;                              { End of Case}
   End;                                         { End of Procedure}


   Procedure ADW_A_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                      a_label: byte; Var ADW_A_pres: ADW_prestype;
                      Var ADW_A: ADWtype);
{********************************************************************}
{ This procedure converts the A-ADW's to Pascal variables.
  See Annex 10 p 150A etc.
                                 Input : Arinc 429 word
                                         Arinc 429 label
                                         ADW_A_present array of boolean
                                 Output: Mlsint
                                         A-ADW  }
{********************************************************************}
Var
   adress   :       Byte;

Begin
     Case a_label Of
     88 :Begin   {130}                         { 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}
              For x:= 13 To 28 Do
                  ADW_A[ x]:= Ar429word[ x - 13 + 14];
              ADW_A_pres[ 1]:= True;
         End;
     89 :Begin   {131}
              If Not ADW_A_pres[1]              { second part only valid}
              Then Exit;                        { if first part presend}

              For x:= 29 To 44 Do
                  ADW_A[ x]:= Ar429word[ x - 29 + 14];
              ADW_A_pres[ 2]:= True;
         End;
     90 :Begin   {132}
              If Not ADW_A_pres[2]              { third part only valid, if}
              Then Exit;                        { second and first part}
                                                { present}
              For x:= 45 To 60 Do
                  ADW_A[ x]:= Ar429word[ x - 45 + 14];
              ADW_A_pres[ 3]:= True;
         End;
     91 :Begin   {133}
              If Not ADW_A_pres[3]              { fourth part only valid, if}
              Then Exit;                        { third, second and first }
                                                { part present}
              For x:= 61 To 76 Do
                  ADW_A[ x]:= Ar429word[ x - 61 + 14];
              ADW_A_pres[ 4]:= True;
         End;
     End;
     If (ADW_present( ADW_A_pres) And Not Hamming_Fail( ADW_A) And
         Not Adress_Fail( ADW_A))               { If the ADW is valid and
                                                  the address is correct,
                                                  begin decoding the info}
     Then Begin
```

```
                  ADW_A_conv( MLSint, ADW_A);
                  For x:= 1 To 4 Do
                      ADW_A_pres[x]:= False;
             End;                                { End of If}
   End;                                          { End of procedure}


{------------------------------------------------------------------------}
{ The following part should not necessarily be implemented, because ADW B
  and C are not yet assigned.}
{------------------------------------------------------------------------}

   Procedure ADW_B_conv( Var MLSint: mlsinttype; ADW_B: ADWtype);

   Begin
                  Date_and_Time( MLSint.AuxB_time);
                  MLSint.AuxB:= ADW_B;
   End;

   Procedure ADW_B_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                         a_label: byte; Var ADW_B_pres: ADW_prestype;
                         Var ADW_B: ADWtype);
   {*******************************************************************}
   { This procedure converts the B-ADW's to Pascal variables.
                             Input : Arinc 429 word
                                     Arinc 429 label
                                     ADW_B_present array of boolean
                             Output: Mlsint
                                     B-ADW  }
   {*******************************************************************}
   Var
      adress            :       Byte;
   Begin
        Case a_label Of
        92 :Begin    {134}
                 For x:= 13 To 28 Do
                     ADW_B[ x]:= Ar429word[ x - 13 + 14];
                 ADW_B_pres[ 1]:= True;
             End;
        93 :Begin    {135}
                 If Not ADW_B_pres[1]
                 Then Exit;

                 For x:= 29 To 44 Do
                     ADW_B[ x]:= Ar429word[ x - 29 + 14];
                 ADW_B_pres[ 2]:= True;
             End;
        94 :Begin    {136}
                 If Not ADW_B_pres[2]
                 Then Exit;

                 For x:= 45 To 60 Do
                     ADW_B[ x]:= Ar429word[ x - 45 + 14];
                 ADW_B_pres[ 3]:= True;
             End;
        95 :Begin    {137}
                 If Not ADW_B_pres[3]
                 Then Exit;

                 For x:= 61 To 76 Do
                     ADW_B[ x]:= Ar429word[ x - 61 + 14];
                 ADW_B_pres[ 4]:= True;
            End;
        End;
        If (ADW_present( ADW_B_pres) And Not Hamming_Fail( ADW_B) And
            Not Adress_Fail( ADW_B))
        Then Begin
                  ADW_B_conv( MLSint, ADW_B);
                  For x:= 1 To 4 Do
                      ADW_B_pres[x]:= False;
             End;
   End;


   Procedure ADW_C_conv( Var MLSint: mlsinttype; ADW_C: ADWtype);

   Begin
                  Date_and_Time( MLSint.AuxC_time);
                  MLSint.AuxC:= ADW_C;
   End;


   Procedure ADW_C_coll( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype;
                         a_label: byte; Var ADW_C_pres: ADW_prestype;
                         Var ADW_C: ADWtype);
   {*******************************************************************}
   { This procedure converts the C-ADW's to Pascal variables.
                             Input : Arinc 429 word
                                     Arinc 429 label
                                     ADW_C_present array of boolean
                             Output: Mlsint
                                     C-ADW  }
   {*******************************************************************}
   Var
      adress            :       Byte;

   Begin
        Case a_label Of
        96 :Begin    {140}
                 For x:= 13 To 28 Do
                     ADW_C[ x]:= Ar429word[ x - 13 + 14];
                 ADW_C_pres[ 1]:= True;
            End;
        97 :Begin    {141}
                 If Not ADW_C_pres[1]
                 Then Exit;

                 For x:= 29 To 44 Do
                     ADW_C[ x]:= Ar429word[ x - 29 + 14];
                 ADW_C_pres[ 2]:= True;
```

```
            End;
        98 :Begin    {142}
                 If Not ADW_C_pres[2]
                 Then Exit;

                 For x:= 45 To 60 Do
                     ADW_C[ x]:= Ar429word[ x - 45 + 14];
                 ADW_C_pres[ 3]:= True;
            End;
        99 :Begin    {143}
                 If Not ADW_C_pres[3]
                 Then Exit;

                 For x:= 61 To 76 Do
                     ADW_Cl x]:= Ar429word[ x - 61 + 14];
                 ADW_C_pres[ 4]:= True;
            End;
        End;
        If (ADW_present( ADW_C_pres) And Not Hamming_Fail( ADW_C) And
            Not Adress_Fail( ADW_C))             { If the ADW is valid and
                                                   the address is correct,
                                                   begin decoding the info}
        Then Begin
                  For x:= 1 To 4 Do
                      ADW_C_pres[x]:= False;
             End;
   End;
{----end of part-----------------------------------------------------}

   Function Conv_BAS( ar429word: ar429wordtype; start: Byte; number: Byte)
                      : Word;
   {*****************************************************************}
   { This function converts single bits to a number. 'Start' indicates
     the start bit in the Arinc 429 word. 'Number' indicates the number of
     bits to be used for the number to be formed. The number is output as a
     Word. See Annex 10 p 60CC: LSB first.
                           Input : Arinc429 word
                                   startbit
                                   number of bits
                           Output: number}
   {*****************************************************************}
   Var
      value,
      mult      :       Word;
      x         :       Byte;

   Begin
        value:= 0;
        mult:= 1;
        For x:= 0 To number - 1 Do
        Begin
             value:= value + ar429word[ start + x] * mult;
             mult:= mult * 2;
        End;
        Conv_BAS:= value;
   End;


Procedure Bas_1_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*****************************************************************}
{ This procedure converts the Basic dataword nr 1 to Pascal variables.
  See Annex 10 p 146.
                           Input : Arinc 429 word
                           Output: Mlsint}
{*****************************************************************}
Begin
     Date_and_Time( Mlsint.Bas1_time);

     With Mlsint.Bas1 Do
     Begin
          Az2Thresdist:= Conv_bas( Ar429word, 12, 6) * 100;
          AzPropCovNegLim:= Conv_bas( Ar429word, 18, 5) * 2;
          AzPropCovPosLim:= Conv_bas( Ar429word, 23, 5) * 2;
          Cleartype:= Ar429word[ 29];
     End;
     Mlsint. Bas1_flag:= False;
End;


Procedure Bas_2_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*****************************************************************}
{ This procedure converts the Basic dataword nr 2 to Pascal variables.
  See Annex 10 p 146.
                           Input : Arinc 429 word
                           Output: Mlsint}
{*****************************************************************}
Begin
     Date_and_Time( Mlsint.Bas2_time);

     With Mlsint.Bas2 Do
     Begin
          MinGP:= Conv_bas( Ar429word, 12, 7) * 0.1;
          BAZstat:= Ar429word[ 19];
          DMEstat:= Conv_bas( ar429word, 20, 2);
          Azstat:= Ar429word[ 22];
          Elstat:= Ar429word[ 23];
     End;
     Mlsint. Bas2_flag:= False;
End;


Procedure Bas_3_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*****************************************************************}
{ This procedure converts the Basic dataword nr 3 to Pascal variables.
  See Annex 10 p 147.
                           Input : Arinc 429 word
                           Output: Mlsint}
{*****************************************************************}
Begin
```

```
     Date_and_Time( Mlsint.Bas3_time);

     With Mlsint.Bas3 Do
     Begin
          AzBW:= Conv_bas( Ar429word, 12, 3) * 0.5;
          ElBW:= Conv_bas( Ar429word, 15, 3) * 0.5;
          DMEdist:= Conv_bas( Ar429word, 18, 9) * 12.5;
     End;
     Mlsint. Bas3_flag:= False;
End;


Procedure Bas_4_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Basic dataword nr 4 to Pascal variables.
  See Annex 10 p 147.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
     Date_and_Time( Mlsint.Bas4_time);

     With Mlsint.Bas4 Do
     Begin
          AzMagOr:= Conv_bas( Ar429word, 12, 9);
          BazMagor:= Conv_bas( Ar429word, 21, 9);
     End;
     Mlsint. Bas4_flag:= False;
End;


Procedure Bas_5_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Basic dataword nr 5 to Pascal variables.
  See Annex 10 p 147.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
     Date_and_Time( Mlsint.Bas5_time);

     With Mlsint.Bas5 Do
     Begin
          BAZPropCovNegLim:= Conv_bas( Ar429word, 12, 5) * 2;
          BazPropCovPosLim:= Conv_bas( Ar429word, 17, 5) * 2;
          BazBW:= Conv_bas( Ar429word, 22, 3) * 0.5;
          Bazstat:= Ar429word[ 25];
     End;
     Mlsint. Bas5_flag:= False;
End;


Procedure Bas_6_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Basic dataword nr 6 to Pascal variables.
  See Annex 10 p 147.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
     Date_and_Time( Mlsint.Bas6_time);

     With Mlsint.Bas6 Do
     Begin
          MLSident[1]:= Char( Conv_bas( ar429word, 12, 6));
          MLSident[2]:= Char( Conv_bas( ar429word, 18, 6));
          MLSident[3]:= Char( Conv_bas( ar429word, 24, 6));
     End;
     Mlsint. Bas6_flag:= False;
End;


Procedure EL_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Arinc 429 word containing the glidepath
  information to a Pascal variable.
                          Input : Arinc 429 word
                          Output: Mlsint}
{*******************************************************************}
Begin
  With Mlsint Do
  Begin
     If ( Ar429word[ 27] = 1) And
        ( Ar429word[ 28] = 1) And
        ( Ar429word[ 29] = 1)
     Then Begin
             ELangle:= Conv_bas( Ar429word, 13, 14);
             ELangle:= ELangle - $4000;
             ELangle:= ELangle * 0.005;
          End;

     If ( Ar429word[ 27] = 0) And
        ( Ar429word[ 28] = 0) And
        ( Ar429word[ 29] = 0)
     Then ELangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

     ElAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
                                              {         1 = forward ant}
                                              { ARinc: 1,2,3 = ant no}
     ELangle_flag:= False;

     If (Not anglebegin) Or (Not Elangle_flag)
     Then Date_and_time( Tangle);
  End;
End;


Procedure AZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
{*******************************************************************}
{ This procedure converts the Arinc 429 word containing the azimuthangle
```

```
    information to a Pascal variable.
                            Input : Arinc 429 word
                            Output: Mlsint}
  {*******************************************************************}
  Var
     x,
     cnt       :       Byte;

  Begin
    With Mlsint Do
    Begin
       If ( Ar429word[ 27] = 1) And
          ( Ar429word[ 28] = 1) And
          ( Ar429word[ 29] = 1)
       Then Begin
               Azangle:= Conv_bas( Ar429word, 13, 14);
               Azangle:= Azangle - $4000;
               Azangle:= Azangle * 0.005;
            End;

       If ( Ar429word[ 27] = 0) And
          ( Ar429word[ 28] = 0) And
          ( Ar429word[ 29] = 0)
       Then Azangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

       AzAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
                                                {         1 = forward ant}
                                                { ARinc: 1,2,3 = ant no}

       cnt:= 0;                                 { extra check possible :}
       For x:= 13 to 28 Do                      { |angle| < propcoverage}
           cnt:= cnt + Ar429word[ x];

       If ( cnt = 0) And ( AR429word[ 29] = 1)
       Then Leftclr:= True
       Else Leftclr:= False;

       If ( cnt = 16) And ( Ar429word[ 29] = 0)
       Then Rightclr:= True
       Else Rightclr:= False;

       If Leftclr Or Rightclr
       Then Azangle_flag:= True
       Else Begin
               AZangle_flag:= False;

               If (Not anglebegin) Or (Not Azangle_flag)
               Then Date_and_time( Tangle);
            End;
    End;
  End;


  Procedure BAZ_conv( Var Mlsint: MLSinttype; Ar429word: Ar429wordtype);
  {*******************************************************************
  { This procedure converts the Arinc 429 word containing the backazimuth
    angle information to a Pascal variable.
                            Input : Arinc 429 word
                            Output: Mlsint}
  {*******************************************************************}
  Var
     cnt,
     x   :     Byte;

  Begin
    With Mlsint Do
    Begin
       If ( Ar429word[ 27] = 1) And
          ( Ar429word[ 28] = 1) And
          ( Ar429word[ 29] = 1)
       Then Begin
               BAzangle:= Conv_bas( Ar429word, 13, 14);
               BAzangle:= BAzangle - $4000;
               BAzangle:= BAzangle * 0.005;
            End;

       If ( Ar429word[ 27] = 0) And
          ( Ar429word[ 28] = 0) And
          ( Ar429word[ 29] = 0)
       Then BAzangle:= Conv_bas( Ar429word, 13, 14) * 0.005;

       AzAntInUse:= Ar429word[ 12] + 1;         { bendix: 0 = aft ant}
                                                {         1 = forward ant}
                                                { ARinc: 1,2,3 = ant no}
       cnt:= 0;
       For x:= 13 to 28 Do
           cnt:= cnt + Ar429word[ x];

       If ( cnt = 0) And ( AR429word[ 29] = 1)
       Then Leftclr:= True
       Else Leftclr:= False;

       If ( cnt = 16) And ( Ar429word[ 29] = 0)
       Then Rightclr:= True
       Else Rightclr:= False;

       If Leftclr Or Rightclr
       Then BAzangle_flag:= True
       Else Begin
               BAZangle_flag:= False;

               If (Not anglebegin) Or (Not BAzangle_flag)
               Then Date_and_time( Tangle);
            End;
    End;
  End;


  Procedure Discretes_conv( Var Mlsint: MLSinttype;
                                Ar429word: Ar429wordtype);
```

```
    {*******************************************************************}
    { This procedure converts the Arinc 429 word containing the MLS discretes
      to Pascal variables.
                              Input : Arinc 429 word
                              Output: Mlsint}
    {*******************************************************************}
    Begin
         With Mlsint.discretes Do
         Begin
              antenna:= Conv_bas( Ar429word, 11, 2);
              test                 := Ar429word[ 13];
              Azsource             := Ar429word[ 14];
              Azselwarn            := Ar429word[ 15];
              Bazselwarn           := Ar429word[ 16];
              GPselwarn            := Ar429word[ 17];
              BAzavail             := Ar429word[ 18];
              BAzdeven             := Ar429word[ 19];
              tuningcom            := Ar429word[ 20];
              nr1antsel            := Ar429word[ 21];
              changeinh            := Ar429word[ 22];
              tunprtsel            := Ar429word[ 23];
         End;
         Mlsint. discretes_flag:=False;
    End;

    Function Hex2String( x: byte): string;
    {*********************************************************************}
    { This function converts a byte to its hexadecimal representation in a
      string. Written by Rob Luxen.
                              Input : Byte;
                              Output: A string;
    {*********************************************************************}
    Const
         hex: Array[ 0..15] Of Char = '0123456789ABCDEF';

    Var
       i :       Byte;

    Begin
       Hex2String[ 0]:= Chr(2);
       For i:= 0 to 1 Do
       Begin
            hex2String[2-I]:= Hex[ x and $000F];
            x:= x shr 4;
       End;
    End;
    {-------------------------end included procedures CollectMLSrec------------}

Var
    line,
    part          :    String;
    Ar429word     :    Ar429wordtype;
    a_label       :    Byte;
    adress        :    Byte;
    NoWord        :    Boolean;
    time,
    result        :    timetype;
    d,d1,i,j      :    Byte;
    ADword        :    ADWtype;

{-------------------------Start procedure CollectMLSrec--------------------}
Begin       { begin  procedure CollectMLSrec}
     MLSint:= tempMLSint;

     Ar429comm.GetAr429word( Ar429word, NoWord, a_label);
                                                       { the label is in dec}
Gotoxy( 1, 13);
If Not ( Noword)
Then Begin
 Str( a_label: 3, part);
 line:= part + ' ';
 For x:= 1 To 32 Do
 Begin
     Str( ar429word[x]:1, part);
     line:= line + part;
 End;
 Write( line, ' ');

 SaveEquipmentMessage( 'RM: ' + line);
End;
           Case a_label Of
           88..91: {130..133} ADW_A_coll( MLSint, Ar429word, a_label,
                                          ADW_A_pres, ADW_A);
           92..95: {134..137} ADW_B_coll( MLSint, Ar429word, a_label,
                                          ADW_B_pres, ADW_B);
           96..99: {140..143} ADW_C_coll( MLSint, Ar429word, a_label,
                                          ADW_C_pres, ADW_C);
           110   : {156}      Bas_1_conv( MLSint, Ar429word);
           111   : {157}      Bas_2_conv( MLSint, Ar429word);
           112   : {160}      Bas_3_conv( MLSint, Ar429word);
           113   : {161}      Bas_4_conv( MLSint, Ar429word);
           114   : {162}      Bas_5_conv( MLSint, Ar429word);
           115   : {163}      Bas_6_conv( MLSint, Ar429word);
           106   : {152}      EL_conv( MLSint, Ar429word);
           105   : {151}      Az_conv( MLSint, Ar429word);
           126   : {176}      Baz_conv( MLSint, Ar429word);
           184   : {270}      Discretes_conv( MLSint, Ar429word);
           End;

           Ar429.GetAr429word( Ar429word, NoWord, a_label);
                                                       { the label is in dec}
If Not ( Noword) Then Begin
Str( a_label: 3, part);
line:= part + ' ';
For x:= 1 To 32 Do
Begin
     Str( ar429word[x]:1, part);
     line:= line + part;
End;
Writeln( line);
```

```

SaveEquipmentMessage( 'RM: ' + line);
End;

          Case a_label Of
          88..91: {130..133} ADW_A_coll( MLSint, Ar429word, a_label,
                                         ADW_A_pres, ADW_A);
          92..95: {134..137} ADW_B_coll( MLSint, Ar429word, a_label,
                                         ADW_B_pres, ADW_B);
          96..99: {140..143} ADW_C_coll( MLSint, Ar429word, a_label,
                                         ADW_C_pres, ADW_C);
          110   : {156}      Bas_1_conv( MLSint, Ar429word);
          111   : {157}      Bas_2_conv( MLSint, Ar429word);
          112   : {160}      Bas_3_conv( MLSint, Ar429word);
          113   : {161}      Bas_4_conv( MLSint, Ar429word);
          114   : {162}      Bas_5_conv( MLSint, Ar429word);
          115   : {163}      Bas_6_conv( MLSint, Ar429word);
          106   : {152}      EL_conv( MLSint, Ar429word);
          105   : {151}      Az_conv( MLSint, Ar429word);
          126   : {176}      Baz_conv( MLSint, Ar429word);
          184   : {270}      Discretes_conv( MLSint, Ar429word);
          End;

     tempMLSint:= MLSint;

     With Fifo Do
     Begin
          If Not FifoEmpty
          Then Begin
                    d:= GetFifo;
                    If d = $AA
                    Then Begin
                              If Not FifoEmpty
                              Then Begin
                                        d1:= GetFifo;
                                        If d1 = $55
                                        Then Begin
                                                  wordready:= True;
SaveEquipmentMessage( 'RM: ' + badwline);
                                                  badwline:= '';
                                             End
                                        Else Begin
                                                  bytes[ bytecount]:= d;
                                                  Inc( bytecount);
                                                  bytes[ bytecount]:= d1;
                                                  Inc( bytecount);
                                                  wordready:= False;
                                                  badwline:= badwline +
                                                             Hex2String( d) +
                                                             ' ';
                                                  badwline:= badwline +
                                                             Hex2String( d1) +
                                                             ' ';
                                             End;
                                   End;
```

```
                         End
                    Else Begin
                              bytes[ bytecount]:= d;
                              Inc( bytecount);
                              wordready:= False;
                              badwline:= badwline +
                                         Hex2String( d) +
                                         ' ';
                         End;
               End;
     End;
     If wordready
     Then Begin
               Case Bytecount Of
               5: Begin
                    For i:= 2 To 3 Do            { take 2 bytes and convert}
                       For j:= 0 to 7 Do         { to bits}
                         AR429word[ (i-2)*8 + j + 12]:= (bytes[i] Shr j) And $01;
                    For j:= 4 To 5 Do            { from last byte only bit}
                        AR429word[ 29]:= ( bytes[4] Shr j) And $01;
                                                 { 5 and 6 are used. LSb=1}
                                                 { because they are not}
                                                 { shifted further in}

                    Case Bytes[1] Of
                    $0A : Bas_1_conv( MLSint, AR429word);
                    $1E : Bas_2_conv( MLSint, AR429word);
                    $05 : Bas_3_conv( MLSint, AR429word);
                    $11 : Bas_4_conv( MLSint, AR429word);
                    $1B : Bas_5_conv( MLSint, AR429word);
                    $18 : Bas_6_conv( MLSint, AR429word);
                    End;
                    wordready:= False;
                    bytecount:= 1;
                  End;
               10:Begin
                    For i:= 2 To 9 Do            { take bits of 7 bytes}
                      For j:= 0 To 7 Do
                        ADWord[13 + (i-2)*8 + j]:= (bytes[ i] Shr j) And $01;
                    Case bytes[1] Of
                    $07: If ((Not Hamming_Fail( ADWord)) And
                             (Not Adress_Fail( ADWord)))
                         Then ADW_A_conv( MLSint, ADWord);
                    $15: If ((Not Hamming_Fail( ADWord)) And
                             (Not Adress_Fail( ADWord)))
                         Then ADW_B_conv( MLSint, ADWord);
                    $0F: If ((Not Hamming_Fail( ADWord)) And
                             (Not Adress_Fail( ADWord)))
                         Then ADW_C_conv( MLSint, ADWord);
                    End;
                    wordready:= False;
                    bytecount:= 1;
                  End;                           { end case 10}
               End;                              { end case}
          End;                                   { end if wordready}
```

```

    With MLSint Do                               { This part is to simulate}
    Begin                                        { basic datawords.}
                                                 { In the bendix receiver,}
                                                 { they are not available}
         bas1.Az2thresdist    := Az2thresdist;
         bas1.AzPropCovNegLim := AzPropCovNegLim;
         bas1.AzPropCovPosLim := AzPropCovPosLim;
         bas1.Cleartype       := Cleartype;

         Date_and_time( bas1_time);
         bas2.MinGP           := MinGP;
         bas2.BAZstat         := Bazstat;
         bas2.DMEstat         := DMEstat;
         bas2.Azstat          := Azstat;
         bas2.Elstat          := Elstat;

         Date_and_time( bas2_time);

         bas3.AzBW            := AzBW;
         bas3.ElBW            := ElBW;
         bas3.DMEdist         := DMEdist;

         Date_and_time( bas3_time);

         bas4.AzMagOr         := AzMagOr;
         bas4.BazMagOr        := BazMagOr;

         Date_and_time( bas4_time);

         bas5.BazPropCovNegLim:= BazPropCovNegLim;
         bas5.BazPropCovPosLim:= BazPropCovPosLim;
         bas5.BazBW           := BazBW;
         bas5.BazStat         := BazStat;

         Date_and_time( bas5_time);

         bas6.MLSident        := MLSident;

         Date_and_time( bas6_time);

         auxa1.AzOff           := AzOff;
         auxa1.Az2MLSdatdist   := Az2MLSdatdist;
         auxa1.AzAlignRun      := AzAlignRun;
         auxa1.AzCoorSyst      := AzCoorSyst;

         Date_and_time( AuxA1_time);

         auxa2.ElOff           := ElOff;
         auxa2.MLSdat2thres    := MLSdat2thres;
         auxa2.ElHeight        := ElHeight;

         Date_and_time( AuxA2_time);

         auxa3.DMEoff          := DMEoff;
         auxa3.DME2MLSdatDist  := DME2MLSdatDist;

         Date_and_time( AuxA3_time);

         auxa4.BAzOff          := BazOff;
         auxa4.BAz2MLSdatDist  := Baz2MLSdatdist;
         auxa4.BAzAlignRun     := BazAlignRun;

         Date_and_time( AuxA4_time);
    End;

    Date_and_Time( time);

    With MLSint Do
    Begin
         Addtime( bas2_time, Valid_bas2, result);
         If Not Later( time, result)             { determine if all info}
         Then Begin                              { for a cycle is present}
                   If (( bas2.Azstat = 1) And ( bas2.Elstat = 1) And
                       ( Azangle_flag XOR Elangle_flag))
                   Then anglebegin:= True
                   Else anglebegin:= False;
              End;
    End;

    Addtime( Tangle, Valid_Tangle, result);
    If Later( time, result) Or ( anglebegin = False)
    Then With tempMLSint Do                      { when the info is passed}
         Begin                                   { through, make flags in}
              AZangle_flag:= True;               { temporarily variable}
              Elangle_flag:= True;               { True, to prevent sending}
              Bazangle_flag:= True;              { info twice}
              DME_flag:= True;
              RightClr:= True;
              LeftClr:= True;
              Discretes_Flag:= True;
         End
    Else If ( completeinfo = 1)
         Then With MLSint Do
              Begin
                  AZangle_flag:= True;
                  Elangle_flag:= True;
                  Bazangle_flag:= True;
                  DME_flag:= True;
                  RightClr:= True;
                  LeftClr:= True;
                  Discretes_Flag:= True;
              End;
End;
{------------------------End procedure CollectMLSrec-------------------}




{------------------------Start procedure Execmlsreccommand------------}
```

```
Procedure ExecMLSrecCommand( command: commandtype);

Const
     Prate      =       1000;

Var
  error        :        Boolean;
  datain       :        Longint;
  oct_lab      :        Byte;
  part,
  line         :        String;
  ar429word    :        ar429wordtype;

Begin
     { room for decoding the command. The result should be a valid
       arinc429 word of 32 bits. This should be coded in a 32 bit longint}

     Ar429.SendAr429word( dataIn, oct_lab, Prate, error);

line:= '';
For x:= 1 To 32 Do
Begin
     Str( ar429word[x]:1, part);
     line:= line + part;
End;
SaveEquipmentMessage( 'SM: ' + line);
End;
{------------------------End procedure Execmlsreccommand------------}


{------------------------Start procedure Closemlsrec------------}
Procedure CloseMLSrec;

Var
  setupfile    :        Text;
  value        :        String;

Begin
     Ar429comm.CloseAR429;
     Ar429.CloseAr429;
     Remove_ADW_int;

     OpenConfigWrite( setupfile, MIAScfgname);
     Writeln( setupfile, 'MLSBENDIX');

     Str( completeinfo, value);
     Writeln( setupfile, #9'completeinfo = ', value, ';');

     Str( cardaddress, value);
     Writeln( setupfile, #9'cardaddress = ', value, ';');

     Str( irq, value);
     Writeln( setupfile, #9'irq = ', value, ';');

     CloseConfig( setupfile);
End;
{------------------------End procedure Closemlsrec------------}


{------------------------Start initialising---------------------}
Begin
     For x:= 1 To 4 Do
     Begin
          ADW_A_pres[x]:= False;
          ADW_B_pres[x]:= False;
          ADW_C_pres[x]:= False;
     End;
     For x:= 13 To 76 Do
     Begin
          ADW_A[ x]:= 0;
          ADW_B[ x]:= 0;
          ADW_C[ x]:= 0;
     End;

     With TempMLSint Do
     Begin
          Bas1_flag     := True;
          Bas2_flag     := true;
          Bas3_flag     := true;
          Bas4_flag     := true;
          Bas5_flag     := true;
          Bas6_flag     := true;
          auxa1_flag    := true;
          auxa2_flag    := true;
          auxa3_flag    := true;
          auxa4_flag    := true;
          ELangle_flag  := true;
          AZangle_flag  := true;
          BAzangle_flag := true;
          DME_flag      := true;
          discretes_flag:= true;
          leftclr       := true;
          rightclr      := true;
          flag          := true;

          ErrorTime( Bas1_time);
          ErrorTime( Bas2_time);
          ErrorTime( Bas3_time);
          ErrorTime( Bas4_time);
          ErrorTime( Bas5_time);
          ErrorTime( Bas6_time);
          ErrorTime( AuxA1_time);
          ErrorTime( AuxA2_time);
          ErrorTime( AuxA3_time);
          ErrorTime( AuxA4_time);
          ErrorTime( AuxB_time);
          ErrorTime( AuxC_time);
     End;

     ErrorTime( Tangle);
```

```
     ErrorTime( Valid_Tangle);                   { angles are no longer}
     Valid_Tangle.sec100:= 8;                    { valid than 80 msec}

     anglebegin:= False;
     completeinfo:= 1;
End.
{-------------------------End initialising---------------------}
{-------------------------End Unit MLSBENDIX---------------------}
```