MGL Avionics

Navidata file format

Navidata file version 5

Draft document dated March 20, 2014

Data items

Byte Unsigned 8 bit value

Word Unsigned 16 bit value

Smallint Signed 16 bit value

Longint Signed 32 bit value

Date/Time 32 bit number of seconds since a given date and time

PascalString ASCII string. First byte is length of string. String follows. Note: This is not a “C” style string, there is no termination character as it is not needed.

REL File pointer Signed 32 bit value. Relative pointer (relative to some other file location)

The header

The Navidata file starts with a fixed header.

```
TNaviDataHeader= Record
        FileID: array[0..7] of char; //Text "NAVIDATA"
        Vendor: longint;
        NaviDataDate: longint; //creation date
        MagicNumber1,MagicNumber2: longint;
        Serials: array[0..31] of string[11]; //This is 384 bytes in size
        NumberOfSections: byte;
        EncryptionIdentifier: byte;
        Version: word;
        NumberOfAirports: longint;
        AirportIndex: longint;
        Airports: longint;
        NumberOfAirspaces: longint;
        Airspaces: longint;
        NumberOfAirways: longint;
        VictorAirways: longint;
        NumberOfSids: longint;
        SIDS: longint;
        NumberOfStars: longint;
        STARS: longint;
        NumberOfObstacles: longint;
        Obstacles: longint;
```

```
  NumberOfWaypoints: longint;
  Waypoints: longint;
  //Items from here onwards are version 5
  WaypointAllocationTable: longint;
  StartDate,EndDate: longint; //Date range for validity of this data
  CycleStr: longint; //4 characters
  NumberOfHoldingPatterns: longint;
  HoldingPatterns: longint;
end;
```

Open Navidata uses a vendor ID of “0”. The vendor ID is assigned by MGL Avionics to a vendor of data. The vendor ID is used to tell a system what style of encryption or copy protection is used. This is hardcoded into the systems. The following items up to and including serials is vendor specific and can be ignored for open data. It should be set to all zeros.

NumberOfSections contains a number showing how many data sections follow. This is set to the value “10”.

NavidataDate – Date of creation of this navidata file (number of seconds since start of 2000)

EncryptionIdentifier should be set to zero (vendor specific).

Version should be set to “5”.

NumberOfAirports – number of airports in the airport information section.

AirportIndex – REL file pointer to sorted index to the airports. This allows the system to perform a fast binary search to find a particular airport without having to scan the entire data. Pointer is relative to the byte following the header (add size of header to get file position).

Airports – REL file pointer to airports information.
Pointer is relative to the byte following the header (add size of header to get file position).

NumberOfAirspaces – number of airspace definitions

Airspaces – REL file pointer to airspace data
Pointer is relative to the byte following the header (add size of header to get file position).

NumberOfAirways – number of airways definitions
Please set this number to zero. Final format of this data has not been decided.

VictorAirways – REL file pointer to airways data.
Please set this pointer to zero. Final format of this data has not been decided.

NumberOfSIDS – number of SIDS in data
Please set this number to zero. Final format of this data has not been decided.

SIDS – REL file pointer to SID data
Please set this pointer to zero. Final format of this data has not been decided.

NumberOfStars – number of STARS in data
Please set this number to zero. Final format of this data has not been decided.

STARS – REL file pointer to STARS data
Please set this pointer to zero. Final format of this data has not been decided.

NumberOfObstacles – number of Obstacles in data

Obstacles – REL file pointer to Obstacles data
Pointer is relative to the byte following the header (add size of header to get file position).

NumberOfWaypoints – number of waypoints in data

Waypoints – REL file pointer to Waypoint data
Pointer is relative to the byte following the header (add size of header to get file position).

WaypointAllocationTable – REL file pointer to waypoint allocation data. This data is used to locate the airport waypoint associated with an airport data record.
Pointer is relative to the byte following the header (add size of header to get file position).

StartDate, EndDate – inclusive date and time range for validity of this data. (number of seconds since start of 2000)

Cycle – optional 4 character cycle code (year/month usually)

NumberOfHoldingPatterns – number of holding patterns in data
Please set this number to zero. Final format of this data has not been decided.

HoldingPatterns – REL file pointer to holding pattern data
Please set this pointer to zero. Final format of this data has not been decided.

Airports and airport index data format

Within this file an unrestricted number of airport records are stored.

Each index entry contains a type identifier, a six character airport identifier (1 to 6 characters in size), a 32 bit pointer to the start of the airport record in the file and finally a geographic position of the airfield (latitude and longitude).

The six character airport identifier is also found in the Waypoint file format and this should be considered as the key to link the two databases. For example, the waypoint file format has the long name “Cape town international airport” while the short name (or identifier) is “FACT”. “FACT” will then be used as identifier in the airport data file index.

Kind:	byte (See airport types further on)
Identifier:	1 byte length of string (1..6), followed by six ASCII characters
Pointer:	32 bit signed integer
Latitude:	32 bit signed integer
Longitude:	32 bit signed integer

Each index entry is exactly 20 bytes in size.

Please note: The index must be sorted by identifier from lowest to highest using common ASCII sort criteria. (For example the identifier “ABC” is lower in value than “DE”). The sorting is vital as binary searches are prescribed to quickly locate a particular entry.

Strings are stored starting with a length byte, followed by a field that is the size of the maximum number of characters that can be stored in the string. Unused characters in a string are “don't care” values.

All integer values are stored in “little endian” format (also known as Intel style). The LSB occupies the lowest address location in the file.

Airport types

Defined airport types are currently any of the following:

0 – unspecified type

1 – 'AIRPORT' Typical assignment for medium sized airports

2 – 'MAJOR AIRPORT' Typical assignment for large and international airports

3 – 'SEAPLANE BASE'

4 – 'AIRFIELD' Typical assignment for smaller municipal airfields, glider fields etc

5 – 'PRIVATE AIRFIELD'

6 – 'ULTRALIGHT FIELD'

8 – 'HELIPORT'

These type identifiers are compatible with the waypoint type identifiers used in the Waypoint file format

Position data format

The position format is used in all data files. This format allows usage of 32 bit integers while allowing position resolution to less than 3 feet.

Position latitude and longitude are stored as signed 32 bit integers.

Degrees are multiples of the value 180000 and any degree fraction is a multiple of 1/180000.

Degrees North and East are positive while degrees South and West are negative.

Example: North 45 degrees, 59 minutes, 30 seconds is 8278500.

Example: North 0 degrees, 30 minutes, 0 seconds is 90000.

Airport record data format

Each airport record consists of a fixed length data section followed by a flexible length list of frequencies applicable for this airport, followed by a flexible length of runway information for this airport, finally followed by an optional data section that can contain almost any type of data. This is intended to store image data such as approach plates and airport layouts (for example a graphic showing runways, taxiways and parking areas), however, it is flexible enough to store any kind of data.

Airport record fixed portion

Runways Pointer:	32 bit signed integer	Pointer to runway data
Data Pointer:	32 bit signed integer	Pointer to other data
Altitude:	16 bit signed integer	Altitude in feet. May be negative.
NumberOfFrequencies: byte		Number of frequencies in this record
NumberOfRunways: byte		Number of runways in this record
NumberOfOtherData: byte;		Number of other data sections in this record

This is immediately followed by the list of frequencies if 1 or more frequencies is defined.

If no frequencies are defined, this is followed by the runway data if at least 1 runway is defined otherwise the “other data” follows here.

Airport record frequencies

The frequency section starts with a list of relative pointers to each frequency defined. Each pointer is relative to the position of the last byte of the fixed record portion +1 (i.e. The target address is the address of the first relative pointer plus the offset found in the pointer of interest).

For each frequency, a flexible length record is inserted. Each record is pointed to by a relative pointer.

Frequency:	32 bit unsigned integer	Frequency in Hz

Type:	1 byte length of string (1..4), followed by four ASCII characters
Description:	1 byte length of string (0..50), followed by 50 ASCII characters

The type string should follow recognizable abbreviations for the type of frequency used, for example “APP”, “TWR”, “CTAF”, “GND” and so forth.

The descriptor may be any text, for example: “Tower, 07:00-17:00, 124.8Mhz other hours”

Airport record runway data

The runway section starts with a list of relative pointers to each runway defined. Each pointer is relative to the position of the last byte of the fixed record portion +1 (i.e. The target address is the address of the first relative pointer in the frequency section (if any) plus the offset found in the pointer of interest).

Designation:	16 bits Designation code (see description below)	
Length:	unsigned 16 bits	Runway length in feet
Width:	unsigned 16 bits	Runway width in feet
Bearing:	16 bits unsigned	True GPS approach bearing relative to first designation heading. 0XFFFF if no GPS approach data
Surface:	1 byte length of string (0..8), followed by eight ASCII characters	
Latitude1,Longitude1:	32 bit signed position of first designation threshold	
Latitude2,Longitude2:	16 bit signed relative position of second designation threshold (relative to first designation threshold)	
Altitude1,Altitude2:	signed 16 bits, altitude of threshold 1 and threshold 2 in feet	

Surface is a ASCII string describing the surface of the runway. Examples: “TAR”, ”SAND”, ”GRASS”.

Bearing bits 0-8 contains a bearing value of 0-359 true which is the exact bearing of the runway centerline to be used for GPS aided approaches. The bearing is the bearing onto the first designation runway heading. Reciprocal bearing is used for the second designation heading. If no GPS approach data is available, this value is to set to 0xFFFF.

If bit 9 of the Bearing value is set, the 16 bytes from Latitude1 to Altitude2 contain localizer and glide slope data. The format of this has not been decided on at time of writing this document.

Latitude1/longitude1 contains the exact position of the runway threshold of the first designation heading in Enigma position format. Latitude2/Longitude2 specifies a relative offset for the second designation runway heading. Simply add these values to latitude1/longitude1 to obtain the position for the second threshold.

Altitude1 and Altitude2 specify the threshold altitudes in feet.

If GPS approach data is included, this data can be used by the instrument to guide the pilot

down a glide slope path towards the runway.

Runway designation data format

The designation is stored as a 16 bit value.

If the value is greater or equal to 0x8000, the heading is expressed in cardinal points (often used for water “runways” for seaplanes). In this case the least significant 3 bits are used as an index to the heading as:

'N','NE','E','SE','S','SW','W','NW'.

The resultant designation is taken as index to the heading +'/'+ reciprocal heading. For example 'NE/SW'.

If the value is less than 0x8000 then bits 12,13 and 14 define a designation type from 0 to 7:

Type 0: designation is formed from bits 0..5 and a reciprocal is calculated. Result is for example: 12/30. This designation is stored as value 0x000C.

Type 1: unused

Type 2: similar to Type 0 but used with dual runways with the first heading being the left runway. For example: 12L/30R

Type 3: As type 2 but first heading is the right runway. For example: 12R/30L.

Type 4: unused

Type 5: unused

Type 6: Helicopter pad. Bits 0..11 is the number of the pad from 1 to 4095. The value 0 is not used. Example: H1

Type 7: Water runway for seaplanes. Similar to type 0. Example: 12W/30W

Data sections

Each airport record may have zero to many data sections. Data sections may contain any data. Data types are identified by a single numeric in the index.

If at least one data section is identified, the data section starts with a table of a 16 bit type identifier followed by a 32 bit pointers to each data section. Please note that these pointers are absolute addresses in the data file and not relative as with frequencies and runways.

Each index entry for a data section is defined as follows:

Pointer:	32 bits unsigned	Absolute pointer to data
Type:	word	Type of data in this section

Currently the following types are defined:

0	Data is ASCII text. Characters 0x20 to 0x7F. Ox7F is the degree symbol, all other characters are standard ASCII. 0x0C,0x0A is used to terminate lines. Font is non-proportional allowing simple ASCII style layout. Any other value is interpreted as end-of-text.

Please note: Not more than 40 characters per line. Fixed 24 lines per page. Multiple pages may be used.

This data is mainly intended to be used as a brief description of the airport, cautions and other information relevant to a pilot using the airport.

No other types are currently defined.

Waypoint format

Within this file an unrestricted number of waypoint records are stored. Each waypoint record is exactly 48 bytes in size, starting at the first byte in the file. An empty waypoint file has zero records.

Each waypoint record has the following format:

Latitude:	32 bit signed integer
Longitude:	32 bit signed integer
Data field:	32 bits, contents depend on waypoint type
Waypoint type:	byte
Short name:	1 byte length of string (1..6), followed by six ASCII characters
Long name:	1 byte length of string (0..27), followed by 27 ASCII characters

Strings are stored starting with a length byte, followed by a field that is the size of the maximum number of characters that can be stored in the string. Unused characters in a string are “don't care” values.

The short name entry must have at least one character and may be up to six characters in size. This entry is used as “key” and in aviation terms would contain the short identifier of an airfield or navaid beacon. This key is used to lookup related data in other databases, for example an airport data base containing details of an airport such as frequencies and runway headings.

The long name contains a human readable description of the waypoint. For example: “Los Angeles International”. The long name is arbitrary and not used as search key for any purpose.

All integer values are stored in “little endian” format (also known as Intel style). The LSB occupies the lowest address location in the file.

Waypoint types

Bits 0-6 of the waypoint byte field specifies the waypoint type. Bit 7 (msb), if set specifies that steering is allowed for this waypoint. In this case the waypoint, if used in a route, might not be overflown as the next leg may be intercepted using steering commands. If bit 7 is cleared, the waypoint will always be overflown before a course change will be commanded.

Note: Overfly behavior is subject to waypoint intercept radius as selected in the instrument.

Defined waypoint types are currently any of the following:

0 – 'WAYPOINT' Waypoint of unspecified type

1 – 'AIRPORT' Typical assignment for medium sized airports

2 – 'MAJOR AIRPORT' Typical assignment for large and international airports

3 – 'SEAPLANE BASE'

4 – 'AIRFIELD' Typical assignment for smaller municipal airfields, glider fields etc

5 – 'PRIVATE AIRFIELD'

6 – 'ULTRALIGHT FIELD'

7 – 'INTERSECTION' (reporting point, boundary crossing)

8 – 'HELIPORT'

9 – 'TACAN'

10 – 'NDB/DME'

11 – 'NDB'

12 – 'VOR/DME'

13 – 'VORTAC'

14 – 'FAN MARKER'

15 – 'VOR'

16 – 'REP-PT'

17 – 'LFR'

18 – 'UHF-NDB'

19 – 'M-NDB'

20 – 'M-NDB/DME'

21 – 'LOM'

22 – 'LMM'

23 – 'LOC/SDF'

24 – 'MLS/ISMLS'

25 – 'OTHER NAV' Navaid not falling into any of the above types

26 – 'ALTITUDE CHANGE' Location at which altitude should be changed. Note: This item is “bidirectional” and contains two altitude values, one for each direction. The altitude value is the new target altitude (typically sets the altitude bug).
This waypoint type is normally only used in routes/flight plans.

27 – 'Start accent/descent angle' Location at which vertical angle should be changed. Note: This item is “bidirectional” and contains two values, one for each direction. The value is degrees in steps of 0.1 degree. Negative values are for descent. This waypoint is typically followed by another of same type to set the angle to “0”.
This waypoint type is normally only used in routes/flight plans.

28 – 'Start accent/descent rate' Location at which vertical rate should be changed. Note: This item is “bidirectional” and contains two values, one for each direction. The value is in feet/minute. Negative values are for descent. This waypoint is typically followed by another of same type to set the rate to “0”.
This waypoint type is normally only used in routes/flight plans.

29 – 'SPEED CHANGE' Location at which speed should be changed. Note: This item is “bidirectional” and contains two speed values, one for each direction. The speed value is the new target speed (typically sets the target speed).

This waypoint type is normally only used in routes/flight plans.

30 – 'NOTIFY'. This waypoint, when encountered will cause the EFIS to display the text of the long descriptor to the pilot. This can be used to inform the pilot to perform a specific task, for example take a photo or contact a flight controller.
This waypoint type is normally only used in routes/flight plans.

Data field

The data field contains waypoint type related information.

Waypoint types 0-6,8: Field contains waypoint altitude in feet. Negative altitudes are supported. Value is a 32 bit signed integer

Waypoint type 7 does not use the data field and the contents are “don't care”

Waypoint types 9-25: Field contains frequency in steps of 1000Hz. For example a frequency or 118.00 Mhz would result in the value 118000. 32 bit unsigned integer.
Note: for NDB and low frequency stations the value is to be interpreted in Hz.

Waypoint type 26: Field contains two altitude values in steps of 10 feet. Altitudes are positive values, negative altitudes are not supported. The LSB 16 bits are relevant for forward execution of a route, the MSB 16 bits are relevant for reverse execution of a route.

Waypoint type 27: Field contains two angle values in steps of 0.1 degree. Angles are signed 16 bit values. Negative values are for descent. For example: a value of 30 means ascent at 3.0 degrees, a value of -22 means descent at 2.2 degrees. The LSB 16 bits are relevant for forward execution of a route, the MSB 16 bits are relevant for reverse execution of a route.

Waypoint type 28: Field contains two rate values in feet/minute. Rates are signed 16 bit values. Negative values are for descent. For example: a value of 500 means ascent at 500 feet/minute. The LSB 16 bits are relevant for forward execution of a route, the MSB 16 bits are relevant for reverse execution of a route.

Waypoint type 29: Field contains two speed values in MPH. Speeds are unsigned 16 bit values. For example: a value of 200 means a speed of 200 MPH. The LSB 16 bits are relevant for forward execution of a route, the MSB 16 bits are relevant for reverse execution of a route.

Position data format

Position latitude and longitude are stored as signed 32 bit integers.

Degrees are multiples of the value 180000 and any degree fraction is a multiple of 1/180000.

Degrees North and East are positive while degrees South and West are negative.

Example: North 45 degrees, 59 minutes, 30 seconds is 8278500.

Example: North 0 degrees, 30 minutes, 0 seconds is 90000.

Comments

The fixed length entry waypoint and route file makes it very efficient for a micro controller to access a particular entry as the location of the entry is a simple calculation. This also makes it possible to easily edit individual entries or move their position.

There is no demand that waypoints are stored in any particular order in the waypoint file. They might be sorted by short name or waypoint type but this is of little consequence.

Airspace format

The airspace file contains regions, controlled airspaces and special use airspaces.

Within this file an unrestricted number of airspace records are stored.

Each airspace record contains a fixed size data portion, a flexible size data portion and a list of points that form a regular, non-self intersecting polygon describing the location and outline of the airspace. Each airspace may be defined using more than one polygon. A typical example of where this is used is if an airspace crosses the date line (E180 to W180). In order to simplify processing of such airspaces, two airspaces are defined, one in the eastern, the other in the western hemisphere. Please note that your software must be able to handle the definition of the bounding rectangle crossing the date line. This is indicated by the North West longitude having a positive value while the South East longitude has a negative value.

Each airspace record has the following format:

Type	longint. The type of airspace for this record. Format see below. Note: also used at the start of the file to indicate optional file options. See text on file options in this document.
North West Lat	longint. North West latitude of a bounding rectangle.
North West Long	longint. North West longitude of a bounding rectangle.
South East Lat	longint. South East latitude of a bounding rectangle.
South East Long	longint. South East longitude of a bounding rectangle.
Pointer Next	longint. Pointer to next record in file. (Zero for last record)
Pointer Points	longint. Pointer to points of the polygon. (polygon size below)
Frequency 1	longint. Contact frequency in Khz.
Frequency 2	longint. Alternate contact frequency in Khz.
Upper Altitude	longint. Upper altitude limit for airspace. Format see below.
Lower Altitude	longint. Lower altitude limit for airspace. Format see below.
ICAO	string. ICAO designation for airspace.
Name	string. Name of airspace.
Class	string. Airspace class if not special use airspace.
Exception	string. Optional qualification for airspace.
Comm-name	string. Radio contact name or phrase.
Level	string. Qualifier for upper/lower altitude.
Times	string. Operating time or time based restrictions.
Weather	string. Weather related restrictions.
-start of polygon points data-	
Size	longint. Total number of points in polygon(s).
Latitude[Size],Longitude[Size]	array of longint pairs describing points of polygon.

Data types

Type

If the type identifier is the first item in the file (at file address bytes 0-3), the type may be a standard type identified by bits 8-31 all zeros. In this case the file contains a linear list of airspaces.
If the type identifier equals the value 0xFFFF0001 the file is a tiled airspace file. See text on tiled airspace files in this document.

If bits 8 to 31 are zero, we have a standard airspace type identifier as follows:

Regions, controlled and uncontrolled airspaces

01 - ADVISORY AREA (ADA) OR (UDA)
02 - AIR DEFENSE IDENTIFICATION ZONE (ADIZ)
03 - AIR ROUTE TRAFFIC CONTROL CENTER (ARTCC)
04 - AREA CONTROL CENTER (ACC)
05 - BUFFER ZONE (BZ)
06 - CONTROL AREA (CTA) (UTA)
  SPECIAL RULES AREA (SRA, U.K. ONLY)
07 - CONTROL ZONE (CTLZ)
  SPECIAL RULES ZONE (SRZ, U.K. ONLY)
  MILITARY AERODROME TRAFFIC ZONE (MATZ, U.K. ONLY)
08 - FLIGHT INFORMATION REGION (FIR)
09 - OCEAN CONTROL AREA (OCA)
10 - RADAR AREA
11 - TERMINAL CONTROL AREA (TCA) OR (MTCA)
12 - UPPER FLIGHT INFORMATION REGION (UIR)

Special use airspaces

32- ALERT
33 - DANGER
34 - MILITARY OPERATIONS AREA
35 - PROHIBITED
36 - RESTRICTED
37 - TEMPORARY RESERVED AIRSPACE
38 - WARNING

Latitude and longitude

Latitude and longitude follow the standard MGL format as follows:

Position latitude and longitude are stored as signed 32 bit integers (little endian).

Degrees are multiples of the value 180000 and any degree fraction is a multiple of 1/180000.

Degrees North and East are positive while degrees South and West are negative.

Example: North 45 degrees, 59 minutes, 30 seconds is 8278500.

Example: North 0 degrees, 30 minutes, 0 seconds is 90000.

Frequencies

Frequencies are stored in a longint in units of Khz. For example: The frequency 124.8Mhz is

stored as 124800.

String

In this file a string is always stored. An empty string is stored as a single byte of value zero.

A string with content is stored as the first byte containing the length of the string, immediately followed by the contents of the string. Another string or other data may follow in the byte immediately above the last character of the string. A string can never be more than 255 characters in length. This form of string is compatible with the Pascal “shortstring”.

Level

This one character string may contain either one of the following characters:

B – Airspace is relevant both for high and low level flight operations.

L – Airspace relevant for low level only

H – Airspace relevant for high level only

Altitude (high,low)

Altitudes consist of a three bit code in the lower three bits of the longint value while the remainder contains a value where required (flight level or altitude in feet after shifting right three bits).

Code

0 - Low altitude - > SURFACE, High altitude UNLIMITED

1 - altitude in feet AMSL

2 - altitude in feet AGL

3 - flight level

4 - GROUND (only used with Low altitude)

5 - By NOTAM (low, high altitude meaningless in data file)

6 - Undefined (source data has no definition for altitudes)

Polygon

The array of points starts with a single longint describing the number of points in the polygon. The polygon is always of closed type and the last point in the polygon has the first point as “next”. Software should close any open polygons that have the last vertex missing.

Polygons are separated by the special point value of latitude 200 and longitude 0.

On encountering this value, the drawing routine should close the current polygon and start a new polygon with the next point in the list.

Polygons are always closed with the value latitude 200 and longitude 0. Every polygon defined in the data has this value as last item.

Tiled airspace file

The first value in the file is a 32 bit number of 0xFFFF0001. This signals a tiled airspace file which is the only type supported with current systems (older, non-tiled format is no longer supported).

This value is immediately followed by a list of 648 file pointers. Each file pointer is a pointer to a 10x10 degree tile starting with latitude N90, longitude W180. The second entry contains a pointer to the tile latitude N90, longitude W170. 36 tiles per row and 18 rows. The last pointer points to latitude S80, longitude E170.

Each pointer is a relative pointer and is relative to the Header.Airspaces pointer (add the two pointers to get to the file pointer).

The coordinate refers to the top left corner of the tile.

Each tile contains all the airspace definitions relevant to this tile PLUS an overlap of 5 degrees on each side. Tiles at the outside fringes of the array do not have the 5 degree overlap into what would be illegal coordinates. The tile contains any airspace that is fully or partially contained in this area or completely surrounds it. The individual airspaces follow the same format as is described in this document. The last airspace in the tile has the “next” pointer as zero value.

Any equipment using this file format may focus on the relevant tile rapidly by consulting the list of tile pointers at the beginning of the file and positioning the file access pointer to the location given in the pointer. If a tile is empty (i.e. Contains no airspaces), the relevant pointer entry at the beginning of the file reads 0 and the file contains no data for this tile.

Example tiled airspace file start:

Location 0x0 0xFFFF0001
Location 0x4 0x00000000 // no data for latitude W180-W170, longitude N90-N80
Location 0x8 0x0002034C // data for W170-W160, longitude N90-N80 starts at file location 0x0002034C

Total 648 pointers

Please note: all 32 bit integers are stored as little endian values (Intel format)

Comments

The airspace vector data specifies only polygons rather than a more complex combination of arcs, circles, ellipses and other shapes. Curved parts of an airspace are reduced to an appropriate series of polygon vertexes that provide a close fit for the airspace boundary.

The immediate advantage of this is fast determination if one is currently within a particular airspace or not using commonly known algorithms.

Each airspace record starts with information on the type of airspace, a bounding box and a pointer to the next record.
While scanning the file, the software need only read this first section of the record to determine if further processing of this record is required. The bounding box describes a rectangular region (ignoring polar distortions) fully containing the airspace under evaluation.

Should the software not use this data, it can immediately perform a seek to the file location in the pointer to the next record. After fully processing the record, the file pointer will be at the

same location.

The airspace file does not dictate the color or line type and thickness of any airspace drawing. This is left to the application to decide.

Obstacle data

Obstacles start with a header

```
Version,Left,Bottom,Width,Height: smallint;
LargestBIN,Check: word;
```

Version is set to “0”.

Left,Bottom,Width,Height specify a geographic rectangle containing the obstacle data. This is organized in “bins”. Each bin is exactly 1 x 1 degree in size. A bin contains the obstacle data for that geographic region only.

Bins are identified by X and Y position from 0 to 359 (X) and 0 to 179 (Y). “0” starts at 180 degrees West and 90 degrees North respectively.

The file starts with an index to the bins in the file. The index is Width*Height in size with every index entry 6 bytes in size:

```
Pointer: longint;
Count: Word;
```

Pointer is a relative file pointer to the start of the obstacle data for this bin. Count is a count of obstacles in this bin.

Typical code:

Let xd be the desired X and yd the desired y position of the Bin (considering Left and Top in the header).

FPTable = file position of start of obstacle data + the header (I.e. Points just after the header).

```
seek(Navidata,FPTable+(yd*OBSHeader.width+xd)*6);
```

Read the 6 byte index contain pointer to the actual Bin and count of items in the Bin.

To get to the start of the Bin (first obstacle in the Bin)

```
FPData:=FPTable+(OBSHeader.height*6*OBSHeader.width);
Seek(Navidata,FPData+FilepointerFromIndexJustRead);
```

Obstacles have a fixed size of 16 bytes:

```
Lat,Long: Longint;
AGL: word;
AMSL: word;
Kind,Lights: word;
```

Latitude and Longitude have the same format as used elsewhere in the navidata file (180000/degree).

Important note: Within each Bin, each entry is sorted by longitude in order westerly items before easterly items.

AGL is the height above ground level in feet.

AMSL is the height above mean sea level in feet.

Kind describes the kind of obstacle (table below).

Lights describes further aspects of the obstacle.

Obstacle types

A total of 46 types are currently defined.

{0-5} 'UNKNOWN','ARCH','BALLOON','BRIDGE','BUILDING','BUILDING/MAST',

{6-10} 'CATENARY','COOLING TOWER','CRANE','CRANE T','CTRL TWR',

{11-15} 'DAM','DOME','ELEVATOR','MONUMENT','PLANT',

{16-20} 'POLE','RIG','REFINERY','SIGN','SPIRE',

{21-25} 'CHIMNEY','CHIMNEYS','TANK','T-L TWR','TOWER',

{26-30} 'TOWERS','TRAMWAY','WINDMILL','WIND TURBINE','SLAG HEAP',

{31-35} 'MAST','WATER TOWER','CHURCH','FLARE','TOWER/MAST',

{36-40} 'DUMP','CABLE','OBSTACLE','INDUSTRY','VEGETATION',

{41-45} 'LIGHTHOUSE','CONTAINER','PYLON');

Obstacle lights

“0”: Unknown (not defined in source)

"1": High Intensity White Strobe Lighting

"2": Medium Intensity White Strobe Lighting

"3": Red Lighting

"4": Dual, Red with HIGH Intensity White Strobe

“5": Dual, Red with MEDIUM Intensity White Strobe

"6": Flood Lights

"7": No Lights

"8": Other, Lighting not listed

Appendix

Date/Time format

Date and time are used for navidata file creation date, validity start and end dates.

This is a 32 bit unsigned number of seconds elapsed since January 1st, 0:00 hours/minutes in the year 2000.

Below is code from Delphi (Pascal) to calculate the number from a standard Delphi TDateTime variable.

From the Delphi help file:

---

type TDateTime = type Double;

Description

Most VCL objects represent date and time values using the TDateTime type. The integral part of a TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of a TDateTime value is fraction of a 24 hour day that has elapsed.

Following are some examples of TDateTime values and their corresponding dates and times:

0 12/30/1899 12:00 am

2.75 1/1/1900 6:00 pm

-1.25 12/29/1899 6:00 am

35065 1/1/1996 12:00 am

```
const
  DaysToMonth: array [0..11] of word = (0,31,59,90,120,151,181,212,243,273,304,334);

function DateToBinary(p: pDateTimeRec): longint;
var
  iday: longint;
  DT: DateTimeRec;
begin
  move(p^,DT,6);
  with DT do
```

```
  begin
    iday:=364 * year + DaysToMonth[month-1] + (day - 1);
    iday:=iday + (year div 4) + 1;
    if year and 3=0 then
      if month<3 then dec(iday);
    result:= second + (60 * minute) + (3600 * (hour + (24 * iday)));
  end;
end;


function EncodeDate(Dt: TDateTime): longint;
var
  DateTime: DateTimeRec;
  wyear,wmonth,wday,whour,wminute,wsecond,w: word;
begin
  DecodeDate(dt,wyear,wmonth,wday);
  if wyear>=2000 then wyear:=wyear-2000 else wyear:=0;
  DecodeTime(dt,wHour,wMinute,wSecond,w);
  DateTime.year:=wyear;
  DateTime.month:=wmonth;
  DateTime.day:=wday;
  DateTime.Hour:=whour;
  DateTime.Minute:=wminute;
  DateTime.Second:=wsecond;
  result:=DateToBinary(@DateTime);
end;
```

About waypoints, airportinformation and airportindex

The navidata contains three important components:

Waypoints

This contains a list of records – each is a waypoint of a specific type. You have various types like airports, VOR, reporting points and many others. Here we are interested in airport types, in particular those where we have more information on like runways and radio frequencies.

Airports can be in any position in the waypoint section of the navidata file, the positions may be fairly random.

Airports

The airports section in the navidata file contains flexible length records on each airport listed in the waypoints section that we have further data on. It is quite OK to have airports in the waypoints section that do not have any entry in the airports section (we might not have any further information on that airport). In this case no mention will be made of that airport in the airports section.

Airport index

The system needs to be able to find out if an airport in the waypoint section has further airport information. It does this by means of finding that airport in the airport index.

Identification is by means of the short waypoint name (up to 6 ASCII characters). This implies that there may not be any duplicate waypoint names of any airport type. The names must be unique for each airport.

The index contains the short name and a pointer to the file location for that airports information.

The index must be sorted by name (ordinary ASCII sort). The system will use a binary search to find the airport name which is quite fast. If the index is not sorted this cannot work.

You will notice that each airport information record also includes a pointer back to the original airport waypoint in the waypointallocation section. This allows the system to find the original waypoint if it only has the airport record.
For this to work, the order of entries in the waypoint allocation table (file pointers) must be the same as the airport index.

Vendors, copy protection and file encryption

For vendor types other than “0” which is a open navidata file, various types of encryption or copy protection mechanisms are employed. The actual method varies by vendor and the corresponding mechanism to read the file is hard coded into the systems.