CUB File Format specification

Table of Contents

File Structure

Cub format contains airspaces and NOTAMS. It consists of 3 parts. At the start of file is CubHeader. Following are multiple CubItem's. Each one can have multiple CubPoint data associated with it. Those are in the last part of file. Positions of first CubItem and CubPoint are stored in header.

Data Structures and Types

CubHeader

First 210 bytes in file is reserved for header.

BYTES	DATA TYPE	NAME	DESCRIPTION
4	UINT32	Ident	Identifier, needs to be 0x425543C2
122	CHAR[122]	Title	Copyright notice
16	UINT16[8]	AllowedSerials	Up to 8 device serial numbers that are allowed to read the file, 0 othervise; used only when IsSecurity equals 2
1	UINT8	PcByteOrder	If 0 than multi-byte integer values need to have their byte order reversed
1	UINT8	IsSecured	Type of encryption for data after header, 0 is unencrypted
4	UINT32	Crc32	Ignored
16	UINT8[16]	Key	Encription key

4	INT32	SizeOfItem	Size of one CubItem data
4	INT32	SizeOfPoint	Size of one CubPoint data
4	INT32	HdrItems	Number of CubItem's in file
4	INT32	MaxPts	Maximal number of points
4	FLOAT	Left	Left value of data bounding box
4	FLOAT	Top	Top value of data bounding box
4	FLOAT	Right	Right value of data bounding box
4	FLOAT	Bottom	Bottom value of data bounding box
4	FLOAT	MaxWidth	Maximal CubItem width in radians
4	FLOAT	MaxHeight	Maximal CubItem height in radians
4	FLOAT	LoLaScale	Used in shape construction
4	INT32	HeaderOffset	File byte offset of first CubItem
4	INT32	DataOffset	File byte offset of first CubPoint
4	INT32	Alignment	Ignored

CubItem

Structure holds basic information about airspace or NOTAM.

Information about their storage is saved in header. First item in file is located at position *HeaderOffset*. There are total of *HdrItems* in file. Each item contains *SizeOfItem* bytes. Next item in file is located right after previous one.

Total size of CubItem structure is 42 bytes. If *SizeOfItem* is smaller than the structure, remaining bytes should be set to 0.

BYTES	DATA TYPE	NAME	DESCRIPTION
4	FLOAT	Left	Left value of item bounding box
4	FLOAT	Top	Top value of item bounding box
4	FLOAT	Right	Right value of item bounding box
4	FLOAT	Bottom	Bottom value of item bounding box
1	UINT8	Style	Airspace type: join highest bit and lowest 4 bits to get CubStyle, bits 5-7 are CubClass)
1	UINT8	AltStyle	Altitude type of MinAlt (lowest 4 bits) and MaxAlt (highest 4 bits)
2	INT16	MinAlt	Bottom altitude bound of airspace
2	INT16	MaxAlt	Top altitude bound of airspace
4	INT32	Data	Relative byte offset from first CubPoint data
4	INT32	TimeOut	If user disabled the airspace
4	UINT32	ExtData	Extra data
8	UINT64	ActiveTime	Encoded NOTAM active time

CubStyle data type

VALUE	NAME	DESCRIPTION
0	asUnknown	
1	asCtr	
2	asRarea	
3	asParea	
4	asDarea	

5	asTra	
6	asTma	
7	asTiz	
8	asAwy	
9	asCta	
10	asGliderSec	
11	asTmz	
12	asMatz	
13	asRmz	
14	/	Non-existing, falls back to aaUnknown
15	asNOTAM	
16	asAdvisoryArea	
17	asAirDefIdZone	
18	asFlInfoRegion	
19	asDelegatedFIR	
20	asTrafficInfoArea	
21	asSpecialRulesZone	
22	asTempFlightRestriction	
23	asAerodromeTrafficZone	
24	asFlInfoServiceArea	
25	/	Non-existing, falls back to asRmz
26	asAerialSportAndRecreatArea	
27	asTransponderRecommendedZone	
28	asVFRRoute	
29	asAlert	
30	asTempReserved	
31	asWarning	

CubClass data type

VALUE	NAME	DESCRIPTION
0	acUnknown	
1	acClassA	
2	acClassB	
3	acClassC	
4	acClassD	
5	acClassE	
6	acClassF	
7	acClassG	

AltStyle type

VALUE	NAME	DESCRIPTION
0	aaUnknown	
1	aaAgl	Above ground level
2	aaMsl	Medium sea level
3	aaFl	Flight level
4	aaUnl	Unlimited
5	aaNotam	NOTAM

Extra Data

If highest 2 bits are 0, than ExtData encodes some NOTAM data:

BITS	DATA MEANING
30-31	00
28-29	NotamType (ntNone=0, ntCancel=1, ntNew=2, ntReplace=3)
23-27	First letter of NOTAM subject
18-22	Last letter of NOTAM subject
13-17	First letter of NOTAM action
8-12	Last letter of NOTAM action
7	0
4-6	NotamTraffic (ntMisc=0, ntIFR=1, ntVFR=2, ntIV=3, ntCHK=4)
0-3	NotamScope (nsUnknown=0, nsAero=1, nEnRoute=2, nsAE=3, nsWarn=4, nsAW=5, nsChk=8)

Letters are uppercase ASCII characters. They are encoded as integers starting with 1 representing 'A' and up to 26 representing 'Z'.

Active Time

BITS	DATA MEANING
63-52	DaysActive flags (daSun=0x001, daMon=0x002, daTue=0x004, daWed=0x008, daThu=0x010, daFri=0x020, daSat=0x040, daHolidays=0x080, daAUP=0x100, daIrregular=0x200, daNOTAM=0x400, daNotSet=0x800)
26-51	Encoded NOTAM start date and time
0-25	Encoded NOTAM end date and time

Unpacking NOTAM Time

Time is packed into minutes. It is used in CubItem ActiveTime and CubPoint *cdiInserted* type. To unpack individual parts:

```
minutes = time%60;       time /= 60;
hours   = time%24;       time /= 24;
days    = (time%31) + 1; time /= 31;
months  = (time%12) + 1; time /= 12;
years   = time+2000;
```

CubPoint

It contains information about airspace shape, name, frequency and other optional information.

Structure have total of 5 bytes. First byte is always flag and remaining 4 bytes are interpreted differently depending on flag.

When lowest bit is set to 1, data is interpreted as shape point. Before using x and y integers, they must be multiplied by *LoLaScale* from header.

For every CubItem, origin is reseted to item's *Left* and *Bottom* bounding box value.

BYTES	DATA TYPE	NAME	DESCRIPTION
1	UINT8	flag	0x81
2	INT16	x	Move origin x-axis for x*LoLaScale
2	INT16	y	Move origin y-axis for y*LoLaScale

BYTES	DATA TYPE	NAME	DESCRIPTION
1	UINT8	flag	0x01
2	INT16	x	Add new point with x' = originX + x*LoLaScale
2	INT16	y	Add new point with y' = originY + y*LoLaScale

First CubPoint with 7th bit in flag set to 1 encodes airspace name. Lowest 6 flag bits represent length of the string (up to 64 characters/bytes). String starts right after the 5 byte structure.

After airspace name, there is saved frequency and its name. Length of name is encoded in lowest 6 bits. String is located right after the structure.

BYTES	DATA TYPE	NAME	DESCRIPTION
1	UINT8	flag	0xC0
4	UINT32	freq	airpsace frequency

Last optional part of data should be interpreted individually byte by byte depending on 2nd byte named *id* that represents type of following data.

BYTES	DATA TYPE	NAME	DESCRIPTION
1	UINT8	flag	0xA0
1	UINT8	id	CubDataId type
1	UINT8	b1	Meaning of value depends on id
1	UINT8	b2	Meaning of value depends on id
1	UINT8	b3	Meaning of value depends on id

CubDataId

When composing bytes to an integer they always follow in order of highest to lowest byte. After some types, string is following the CubPoint structure.

VALUE	NAME	DESCRIPTION	DECODING
0	cdiICAO	ICAO code of the airport	b3 represents length of string that follows after structure
1	cdiFreq2	Second Frequency integer	b1, b2 and b3 are frequency
2	cdiClassExcRmk	Exception rules to airspace class	b2 and b3 are length of following string
3	cdiRemarks	NOTAM Remarks	b2 and b3 are length of following string
4	cdiIdent	NOTAM Identification string	b3 represents length of string following the structure
5	cdiInserted	NOTAM insert date/time	b1, b2 and b3 + 1 byte after the structure

Example CUB Parsing Implementation

```
bool CAirspaceContainer::LoadFromCUB(LPCSTR szFile, int &iMsgCode, int iFile)
{
  SCubHeader hdr;
  SCubItem itm;
  SCubPoint pnt;
  CCubItem *asi;
  DWORD dwRead;
  DWORD dwSizeItem, dwSizePoint;

  CFileStream *fs=new CFileStream(szFile);
  if(fs) {
    fs->Seek(0,SEEK_SET,NULL);
    memset(&hdr,0,sizeof(hdr));
    fs->Read(&hdr,sizeof(hdr),NULL);

    if(hdr.Ident==IdentCUB && (hdr.IsSecured>0 || hdr.PcByteOrder==0)) {
      ConvertCub(fs);
      fs->Seek(0,SEEK_SET,NULL);
      memset(&hdr,0,sizeof(hdr));
      fs->Read(&hdr,sizeof(hdr),NULL);

      if(hdr.IsSecured==2) {
        iMsgCode = ERROR_LFF_UNAUTHORIZED;
        delete fs;
        return false;
      }
    }
    delete fs;
  }

  CReadFileStream ms(szFile);
  if(!ms.IsOk()) {
    iMsgCode=-1;
    return false;
  }

  ms.Seek(0,SEEK_SET,NULL);
  ms.Read(&hdr,sizeof(hdr),NULL);

  if(hdr.Ident==IdentCUB && hdr.IsSecured==0 && hdr.PcByteOrder==1 && hdr.SizeOfPoint>1 &&
hdr.SizeOfItem>1) {
    dwSizeItem=MIN((DWORD)hdr.SizeOfItem,sizeof(SCubItem));
    dwSizePoint=MIN((DWORD)hdr.SizeOfPoint,sizeof(SCubPoint));
```

```
for(int i=0;i<hdr.HdrItems;i++) {
  ms.Seek(i*hdr.SizeOfItem+hdr.HeaderOffset,SEEK_SET,NULL);
  memset(&itm,0,sizeof(SCubItem));
  ms.Read(&itm,dwSizeItem,&dwRead);
  if(dwRead!=dwSizeItem) {
    return false;
  }

  asi=new CCubItem;

  asi->m_border.l=itm.Left;
  asi->m_border.r=itm.Right;
  asi->m_border.t=itm.Top;
  asi->m_border.b=itm.Bottom;
  asi->SetTimeout(itm.TimeOut);
  asi->m_style=CubStyle(itm.Style);
  asi->m_class=CubClass(itm.Style);
  asi->AutoClass();
  asi->m_MinAltStyle=CubAltStyle(itm.AltStyle&0x0f);
  asi->m_MaxAltStyle=CubAltStyle(itm.AltStyle>>4);
  asi->m_MinAlt=itm.MinAlt;
  asi->m_MaxAlt=itm.MaxAlt;
  asi->m_datapos=itm.Data+hdr.DataOffset;
  asi->m_iFile=iFile;
  asi->m_iIndex=i;
  asi->m_nSize=0;
  asi->m_dLoLaScale=hdr.LoLaScale;
  asi->m_SizeOfPoint=hdr.SizeOfPoint;
  if((itm.ExtData>>30) == 0 && itm.ExtData !=0) {
    asi->m_NotamType = (eNotamType)(itm.ExtData>>28);
    asi->m_NotamTraffic = (eNotamTraffic)((itm.ExtData>>4)&7);
    asi->m_NotamScope = (eNotamScope)(itm.ExtData&0xF);
    uint16_t val;
    val=(itm.ExtData>>23)&0x1f;
    asi->m_NotamSubject[0]=val?('A'+val-1):'\0';
    val=(itm.ExtData>>18)&0x1f;
    asi->m_NotamSubject[1]=val?('A'+val-1):'\0';
    val=(itm.ExtData>>13)&0x1f;
    asi->m_NotamAction[0]=val?('A'+val-1):'\0';
    val=(itm.ExtData>>8)&0x1f;
    asi->m_NotamAction[1]=val?('A'+val-1):'\0';
    // 00TTSSSS SssssAA AAAaaaaa 0RRRCCCC - Type, Subject; Action, tRaffic, sCope
  }

  asi->m_DaysActive=(eDaysActive)(itm.ActiveTime>>52);
```

```
Uint32 packed = (itm.ActiveTime>>26)&0x3FFFFF;
asi->m_ActiveFromDT = (packed==0) ? NODATA : unpackDateTime(packed);
packed = itm.ActiveTime&0x3FFFFF;
asi->m_ActiveUpToDT = (packed==0x3FFFFF) ? NODATA : unpackDateTime(packed);

double ox,oy;
ox=asi->m_border.l;
oy=asi->m_border.b;
CLoLaPoint lola = { 0, 0 };

ms.Seek(asi->m_datapos,SEEK_SET,NULL);
while(1) {
  char cptr[64];
  int len;

  ms.Read(&pnt,dwSizePoint,&dwRead);
  if(dwRead!=dwSizePoint)
    break;
  if((DWORD)hdr.SizeOfPoint!=dwSizePoint)
    ms.Seek(hdr.SizeOfPoint-dwSizePoint,SEEK_CUR,NULL);
  if(pnt.flag&0x40) {
    if(pnt.flag&0x3f) {
      memset(cptr,0,sizeof(cptr));
      ms.Read(cptr,pnt.flag&0x3f,NULL);
      asi->m_csName=cptr;
      asi->m_csName.MakeUTF8();
    } else {
      asi->m_csName.Empty();
    }

    //reading frequency
    ms.Read(&pnt,dwSizePoint,&dwRead);
    if(dwSizePoint==dwRead && (pnt.flag&0xC0)==0xC0) {
      asi->m_iFreq=pnt.freq;
      if(pnt.flag&0x3f) {
        memset(cptr,0,sizeof(cptr));
        ms.Read(cptr,pnt.flag&0x3f,NULL);
        asi->m_csFreqName=cptr;
        asi->m_csFreqName.MakeUTF8();
      } else {
        asi->m_csFreqName.Empty();
      }
      ms.Read(&pnt,dwSizePoint,&dwRead);
    }
    while(dwSizePoint==dwRead && pnt.flag==0xA0) {
```

```
            switch(pnt.id) {
              case cdiICAO:
                if(pnt.b3 > 0) {
                  memset(asi->m_ICAO,0,sizeof(asi->m_ICAO));
                  ms.Read(asi->m_ICAO,MIN(sizeof(asi->m_ICAO),pnt.b3),NULL);
                }
                break;
              case cdiFreq2:
                asi->m_iFreq2 = (int)pnt.b1<<16 | (int)pnt.b2<<8 | (int)pnt.b3;
                break;
              case cdiClassExcRmk:
                len = (int)pnt.b2<<8 | (int)pnt.b3;
                asi->m_classExcRmk.Empty();
                if(len > 0) {
                  LPSTR str=asi->m_classExcRmk.GetBuffer(len);
                  memset(str,0,len);
                  ms.Read(str,len,NULL);
                  asi->m_classExcRmk.ReleaseBuffer(len);
                  asi->m_classExcRmk.MakeUTF8();
                }
                break;
              case cdiRemarks:
                len = (int)pnt.b2<<8 | (int)pnt.b3;
                asi->m_Remarks.Empty();
                if(len > 0) {
                  LPSTR str=asi->m_Remarks.GetBuffer(len);
                  memset(str,0,len);
                  ms.Read(str,len,NULL);
                  asi->m_Remarks.ReleaseBuffer(len);
                  asi->m_Remarks.MakeUTF8();
                }
                break;
              case cdiIdent:
                len = (int)pnt.b3;
                asi->m_Ident.Empty();
                if(len > 0) {
                  LPSTR str=asi->m_Ident.GetBuffer(len);
                  memset(str,0,len);
                  ms.Read(str,len,NULL);
                  asi->m_Ident.ReleaseBuffer(len);
                  asi->m_Ident.MakeUTF8();
                }
                break;
              case cdiInserted: {
                Uint32 timeHigh = (Uint32)pnt.b1<<24 | (Uint32)pnt.b2<<16 |
(Uint32)pnt.b3<<8;
```

```
                Uint8 timeLow = 0;
                ms.Read(&timeLow,1,NULL);
                asi->m_NotamInsertDT = unpackDateTime(timeHigh|(Uint32)timeLow);
              } break;
            }
            ms.Read(&pnt,dwSizePoint,&dwRead);
          }
          break;
        } else if(pnt.flag==0x81) {
          ox=ox+pnt.x*asi->m_dLoLaScale;
          oy=oy+pnt.y*asi->m_dLoLaScale;
        } else if(pnt.flag==0x01) {
          lola.lo=pnt.x*asi->m_dLoLaScale+ox;
          lola.la=pnt.y*asi->m_dLoLaScale+oy;
          asi->m_nSize++;
        } else {
          break;
        }
      }

      if(asi->m_nSize>0)
        Add(asi);

      UNLOCKRELEASE(asi, LOCKWRITE);
    }
    DISPLAY_LOCKS(this);
    DISPLAY_REFS(this,1);

    return true;
  }

lSafeExit:
  iMsgCode=-2;
  return false;
}
```