

UPDRAFT

THE PROGRAMMERS DOCUMENTATION

Team¹ Leader :

Tomas Zamecnik

Team Members :

Cestmir Houska

Jakub Marek

Bohdan Maslowski

Maria Vamosova

Ales Zita

Project Supervisor :

Alexander Wilkie

Faculty of Mathematics and Physics,

Charles University,

Prague,

Czech Republic

07-May-12

¹ glideplan_swproj@freelists.org

TABLE OF CONTENT

TABLE OF CONTENT.....	2
1 INTRODUCTION	4
1.1 GLIDE PATH VISUALISATION AND PLANNING	4
1.2 THE UPDRAFT PROJECT - OPRAVIT	4
2 THE UPDRAFT PROJECT	6
2.1 THE TEAM	6
2.2 TECHNICAL DETAILS OF DEVELOPMENT.....	7
2.3 REPOSITORY STRUCTURE	8
2.4 PROJECT SPECIFICATION.....	8
2.4.1 Environment.....	8
2.4.2 Supported languages	8
2.4.3 Negative specification.....	8
2.4.4 List of features	8
2.4.5 GUI principles.....	9
3 THIRD PARTY SOFTWARE	10
3.1 NOKIA QT.....	10
3.2 OSG.....	10
3.3 OSG EARTH	10
4 THE THIRD PARTY DATA	12
4.1 INTRODUCTION	12
4.2 MAP IMAGERY SOURCES	12
4.3 ELEVATION DATA SOURCE	13
5 PROGRAM ARCHITECTURE	14
6 THE CORE.....	15
6.1.1 Plug-in Interface.....	<i>Chyba! Záložka není definována.</i>
7 THE PLUGINS	19
7.1 INTRODUCTION	19
7.1.1 Plug-in creation.....	19
7.1.2 Minimal Plug-in Example	19
7.2 AIRSPACES	20
7.2.1 Introduction	20
7.2.2 Overview	21
7.2.3 Design	21
7.2.4 OpenAir(tm) parser	22
7.2.5 OpenAir(tm) drawing engine	24
7.2.6 3 rd party software used	26
7.2.7 The Airspaces plug-in diagram	27
7.3 TURNPOINTS.....	28

7.4	TASK DECLATION.....	31
7.5	IGC VISUALISATION.....	31
8	PROGRAM INSTALLATION.....	32
8.1	WINDOWS.....	32
8.2	LINUX.....	32
8.3	MAC.....	32
9	CONCLUSION.....	33
10	FUTURE WORK.....	33
11	REFERENCES.....	34
12	TABLES.....	36
13	FIGURES.....	37
14	APPENDIX A – PEOPLE.....	38
15	APPENDIX B – OPENAIR(TM) AIRSPACE FORMAT DEFINITION.....	39
16	APPENDIX C – ELEVATION DATASET USAGE CONFIRMATION.....	43
17	APPENDIX D – DVD CONTENT.....	45

1 INTRODUCTION

1.1 GLIDE PATH VISUALISATION AND PLANNING

There is more to gliding then just to fly the airplane. For instance every competition flight needs to be planned in advance as well as analysed after the flight is done.

The post-flight analysis or so-called de-briefings, is the process during the pilot takes the stored flight path recordings and scrutinises the data or re-plays the flight he made to further analyse the flight or learn from it. As for planning of the competition flight there are sets pre-defined *turn-points* from which usually the three turn-points are chosen to form a competition triangle. Every turn-point is defined by its world coordinates. During the competition flight, the pilot needs to navigate his airplane through the defined set of points in pre-defined order for his flight to be valid. Although this planning can be done by-hand, the use of computer program is very convenient. Therefore a need of software that can be used for these tasks has arisen. To our knowledge there are few free and commercial solutions on the market. However the free software doesn't match the quality of the commercial applications and the commercial applications are expensive. One of the widely spread commercial software for this purposes at the present time is a application called *SeeYou* [1] created by the *Naviter* company, which we use as an inspiration.

Our aim is to create similar (in functionality) open-source product under the GNU license and therefore made this product available to wider glider pilot communities. We try to improve on some of the SeeYou features, which we believe can be done in more user friendly manner. We would not like our product to be compared to SeeYou, rather taken as a free-of-charge alternative for non-heavy users. To achieve this uneasy task we setup the group of skilled programmers and flight enthusiasts, who has spent many nights and day to make this achievable. We call our project **The Updraft**².

1.2 THE UPDRAFT PROJECT - OPRAVIT

to create such complex software, which could be used as a previously mentioned glider pilot tool and all the basic procedures can be done. The tools we want to be contained within the final product are :

- flight planning (called task declaration)
- airspace visualisation
- turnpoints and airfields visualisation
- flight path visualisation with colour-coded information about
 - speed
 - vertical speed
 - altitude
 - airspeed

² <http://updraft.github.com/>

We call this project **The Updraft**.

2 THE UPDRAFT PROJECT

2.1 THE TEAM

This subject was originally proposed by our Project Leader *Tomas Zamecnik*. He, as an enthusiastic glider pilot, has pointed out the lack of non-commercial glider-path planning and visualisation software. Group of people fond of flying was established with *Tomas Zamecnik* as a project leader being the glider pilot himself and a group of people being fond of flying. Finally the team consists of 6 people.

Team Members :

- **Tomas Zamecnik**
- **Cestmir Houska**
- **Jakub Marek**
- **Bohdan Maslowski**
- **Maria Vamosova**
- **Ales Zita**

See the “*Appendix A – People*” for further details.

Programming of the Updraft project was divided into several partially separable parts with each of the part dedicated to one team member. Obviously it was impossible to split the workload evenly, so most of the people ended up working on parts of the project together and tight cooperation between team members proved crucial. However there always was a tendency for having one responsible person for each of the project component. The table below shows every member main responsibilities. (The work was well coordinated and the Meetings were held on a week basis as well as cases of team brain-stormings. Edit.) For e-mail communication we used services of *freelists.org* [2] for its accessibility and transparency. The group e-mail address is glideplan_swproj@freelists.org.

Developer	Updraft component	Main responsibilities
Tomas Zamecnik	Turnpoints plugin, TaskDecl plugin	Leadership and management, Turnpoints plugin, User documentation, TaskDecl plugin, Importing files
Cesmir Houska	Core, GUI	Updraft core, Settings, TaskDecl. GUI, Mouse picking
Jakub Marek	IGC plugin, Core	Core, IGC visualisation, IGC parsing, on-demand rendering (not yet finished), Translations (not yet finished), Bottom panel interface, OpenFile dialog
Bohdan Maslowski	GUI	Mac at all, Installation for all platforms, 2D Map mode
Maria Vamosova	IGC plugin, GUI	OSG Earth implementation, Map layers visualisation, IGC graph drawing, IGC statistics, Camera manipulation, Map Layer groups + left pane interface, Scene manager
Ales Zita	Airspaces plugin, documentation	Airspace plugin, documentation, turnpoint labels, airfields visualisation

Table 1 : Team members main responsibilities.

2.2 TECHNICAL DETAILS OF DEVELOPMENT

The programming language we choose for the Updraft implementation was C++ for both the object oriented characteristic of the language and overall advancement of this language's features and possibilities. To keep the complexity of the language on a manageable level, we decided to use a Google C++ style guide [3]. Google C++ style guide is one of the widely used open-source set of suggestions and recommendation on how to maintain the readability of the code. The style checker was incorporated to the build of the project to ensure the rules would be followed.

On top of the main programming language we decided to use Nokia Qt framework [4] for its multiplatform user interface capabilities. Nokia Qt is a development framework used for creation of applications for many platforms including Windows, Linux, Mac OS, Symbian and many others. One of the goals we set was to have Updraft usable under not only Windows, but under MacOS and Linux operating systems as well, because these are 3 most common operating systems nowadays.

The program was developed on Windows and Linux platforms and tested on Mac. As a development environment we used the Microsoft Visual Studio 2008 [5] under student license on Windows and on Linux.

CMake [6] was used as a cross-platform project build system.

2.3 REPOSITORY STRUCTURE

We used the *Git* [7] version control software and hosted the source code on the *GitHub* [8] web hosting service. We choose GitHub offers free accounts and storing space for open-source project, such as ours, and for very convenient possibility of creating the web page for our product. The web-page address of our product can be found on <http://updraft.github.com>. The project repository is accessible via internet on <https://github.com/updraft>. The repository structure of our project is as follows:

- updraft.github.com - Project web page files.
- updraft - Project main repository folder.
 - Docs - Documentation related files
 - Experiments - Project experiments with 3rd part SW installation.
 - SourcesInstallation - Test files for source code installation.
 - Updraft - Source code repository.

2.4 PROJECT SPECIFICATION

2.4.1 ENVIRONMENT

The goal of this endeavour is to create multiplatform desktop application capable of serving as a glider pilot tool for flight planning and post-flight visualisation. The target platforms are Linux, Microsoft Windows or Mac OS. Connection to the internet is not required, but enables some of the functionality.

2.4.2 SUPPORTED LANGUAGES

Application is distributed in Czech and English localisations.

2.4.3 NEGATIVE SPECIFICATION

The software is not intended to be used by competition directors or referees or for judging the achievements of any official goals. Application will not be focused on detailed flight planning like calculating optimal speed, water ballast, final glide etc. The application will not run on mobile devices and will not serve as an on-board computer.

2.4.4 LIST OF FEATURES

The features include the visualisation of 3D maps including the terrain height, visualisation of flight relevant data, such as airspace divisions, turn-points, airfields, etc., next to the flight planning and visualisations of recorded IGC files.

The tools we contained within the final product are :

- flight planning (called task declaration)
- airspace visualisation
- turnpoints and airfields visualisation
- flight path visualisation with colour-coded information about
 - speed
 - vertical speed
 - altitude
 - airspeed

2.4.5 GUI PRINCIPLES

Most of the action is done by mouse.

3 THIRD PARTY SOFTWARE

As this is an uneasy task, we made our work a little bit easier by utilizing several 3rd party frameworks, which help us to create the application. These are :

1. *Nokia Qt* [4] – the multiplatform User Interface
2. *OSG* [9] (the open scene graph) – as a OpenGL interface for 3D geometry visualization
3. *OSG Earth* [10] – as a tool for the world map visualization
4. *GeographicLib* [11] – for exact elipsoid distance calculation

3.1 NOKIA QT

Nokia Qt [4] is a cross-platform development framework used for creation of applications with graphical user interface (GUI) for many of the operating systems including Windows, Linux, Mac OS, Symbian and others. Nokia Qt is a free and open source software distributed under the terms of the GNU Lesser General Public License. As it has been always our goal for Updraft to be multi platform application, we decided to use the Nokia Qt framework as it ideal for this task. Moreover, the Nokia Qt is known to be able to incorporate the OSG efficiently. Nokia Qt uses standard C++ but makes extensive use of special code generator called the *Meta Object Compiler*, or *moc* together with several macros to enrich the language. [11]

3.2 OSG

The *OpenSceneGraph* [9] is an open source cross-platform 3D graphics application programming interface used in fields such as visual simulation, games, virtual reality, scientific visualisations and modeling. The OSG is distributed under OpenSceneGraph Public License based on LGPL licence. It uses the *OpenGL* [12] for 3D graphics acceleration interface. We use the OSG as a base for the OSGEarth framework, which we decided to use for the worldwide map visualisation.

3.3 OSG EARTH

The *OSGEarth* [10] is a terrain rendering toolkit written in C++ and distributed under LGPL license. It is developed by Jason Beverage and Glenn Waldron as a visualisation toolkit for the map, elevation and vector data. OSGEarth utilises the OSG framework as the 3D graphics interface.



Figure 1. The OSGEarth framework example.

3.4 GEOGRAPHICLIB

The GeographicLib is a small C++ project for solving geodesic problems. The library is licensed under the [MIT/X11 License](#). In our project we use a small part of their code for very exact computation of distance on the geode surface.

4 THE THIRD PARTY DATA

4.1 INTRODUCTION

So far we described the environment we chose for our project development. OSGEarth is a superb framework for visualising the map imagery and elevation data of various sorts. Now for data we decided to use the online/caching strategy. It means that we use the data sets, which are available online for reasons of minimal necessary installation packaging, but cache the data we once downloaded from the internet source to limit the bandwidth requirements to minimum level. In this way the application is capable of visualising the data from all over the world without the necessity of all the data being stored locally and distributed with the application. Note that these data packages are enormous in size for obvious reasons. Therefore the internet connection during the application run is highly advisable but not essential.

Two types of data sources are necessary for visually pleasing display of the world, the map imagery data and the elevation data. There are several free data sources available online for the map imagery, but not so much for the elevation data.

4.2 MAP IMAGERY SOURCES

In order to display the world map we need to use a free database of map imagery data. This data set needs to be in one of the formats supported by the OSGEarth framework. The complete list of the OSGEarth drivers capable of processing particular data sources are listed on the OSGEarth web page³. The configuration of the particular data sources is written in the `initial.earth` file stored in `/bin/data` folder.

In our case we wanted to have at least two basic views of the virtual world. We pick the sources in following manner: the first one to be very simple for the sake of the visualisation clarity and the second one to be the “realistic” view, i.e. the photomaps with the elevation displayed as a terrain displacement.

Map imagery data sources:

- OpenStreet map [13] - free worldwide map tiles database hosted on under open licence
- ArcGIS online map service [14]- free worldwide satellite tiles imagery database

³ <http://osgearth.org/wiki/TileSourcePlugins>



Figure 2. OpenStreet map source.



Figure 3. ArcGIS online map service.

4.3 ELEVATION DATA SOURCE

As for the elevation (so-called heightfield) data source we use the Ready Map elevation data source [15]. For this data source is designated for development purposes only, we asked the author for permission to use this data set for our project. The permission was granted and the corresponding e-mail is attached to this document. See Appendix C – Elevation dataset usage confirmation.

5 PROGRAM ARCHITECTURE

The application is designed as a core/plugin architecture with the core being responsible for user interface, map visualization and plug-in handling. The plug-ins are linked as a dynamic libraries to the main project and forms the main features functionalities. Plug-ins are capable of drawing to the map layer, creating plug-in related menus, mouse click actions, etc.

Application Architecture

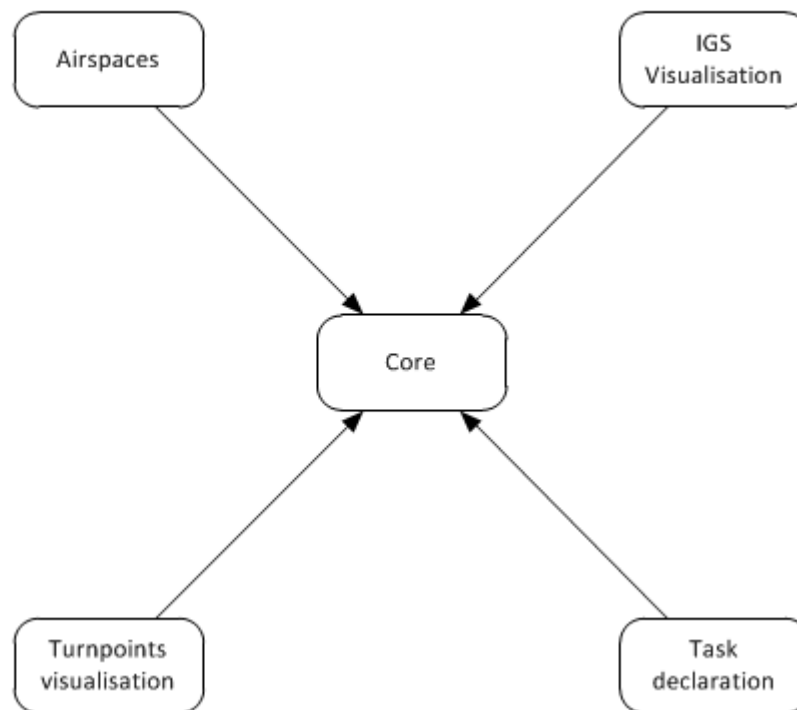


Figure 4. Application architecture diagram.

6 THE CORE

6.1 OVERVIEW

The core provides the basic functionality of Updraft - maps, settings, menus, GUI, etc., whereas the plug-ins handle everything else. Therefore, the core has to provide everything that the plug-ins could possibly need. This includes map mouse picking, handling the GUI and menus, saving and loading the plug-in's settings, loading the application data from the correct directory and registering a known file type. All of this functionality can be accessed via core interface. Each plug-in has access to this core interface and can thus use its functions. For loading the plug-ins dynamically at runtime, Updraft uses the Qt's plug-in system.

6.2 TOP LEVEL STRUCTURE

Top level object of the application is class `'Updraft'` defined in `Updraft/src/core/updraft.h` and `Updraft/src/core/updraft.cpp`. This class inherits from *QApplication* and is available for the whole core as *updraft* macro. It is responsible for creation of the main application structure.

Below the `'Updraft'` object there are five managers that handle separate aspects of the application functionality and the main GUI window.

Managers have dependencies between themselves, that force the order in which they are created. Additionally some of the dependencies are circular (see Figure 5) and need to be broken by dividing the initialization of these objects into smaller parts.

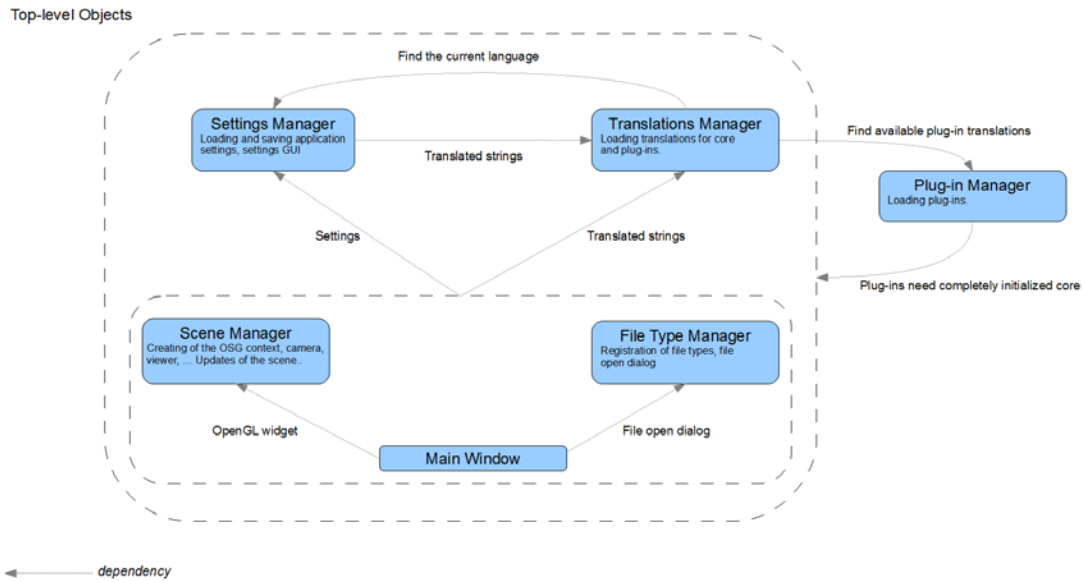


Figure 5. Top level structure diagram.

The order of initialization is following:

- Settings Manager needs to be initialized first, because it contains paths to data files and selected language. However it can not be completely initialized until the translation manager is ready to translate settings dialog and until the main window is ready to add the main menu items.
- Next the plug-in manager is created. It finds and loads all available plug-ins, but because plug-ins need a core interface to work, plug-in manager has to wait with calling their *initialize* method until all other managers are completely initialized.
- Third in the order of creation is the translation manager. It only depends on settings for the current language setting and on plug-in manager for a list of loaded plug-ins (to find their available translations).
- After these are created other application global objects are prepared and then the start up continues with creation of main window, file type manager and scene manager.
- Finally the OpenGL display widget is added to main window and initialization of settings manager and plug-in manager is finished.

Apart from initializing the managers, class `Updraft` also provides some application wide values, like path to current data directory and selectable ellipsoid models for distance calculations.

6.3 PLUG-IN BASE CLASS

Every plug-in in Updraft has to be derived from the *PluginBase* class. This class provides several handler methods that can be overridden by each plug-in to provide the desired behavior. These methods allow the plug-ins to initialize data upon load, de-initialize data when the plug-in is unloaded, and add context-menu items into *MapObject*-related context menus (more information about map objects in section Map Objects), react on left mouse clicks on map objects and also process a file that is being opened by the core. Each plug-in should also provide its name by overriding the method *getName()*.

The *PluginBase* class as a Qt plug-in interface using `Q_DECLARE_INTERFACE`.

6.4 CORE INTERFACE

Interface *CoreInterface* enables the plug-ins to communicate with Core. It is passed to the plug-in during its initialisation phase (every plug-in gets its own instance) and it contains pointer to the plug-in. This makes it possible to attribute all calls to the source plug-in (although this option is currently not used). *CoreInterface* is implemented by a core class *CoreImplementation*, however this class contains only thin wrappers around various parts of the high level managers.

6.5 PLUGINMANAGER

On the side of Updraft Core, the class *PluginManager* takes care of loading plug-ins. The public interface of plug-in manager is fairly limited, only listing all loaded plug-ins and retrieving plug-in instance or plug-in directory by name.

6.6 TRANSLATIONS

Updraft uses *Qt Linguist* for translations. The source strings in the application are written in English, and every other language gets its own translation file.

6.6.1 TRANSLATION FILES

Using the *lupdate* tool, Linguist keeps `*.ts` files generated from the source code of the application. These files can be edited using the translation tool and during build they are converted to binary `*.qm`.

In our application both the updates of `*.ts` files and the generation of `*.qm` files is controlled by CMake scripts.

6.6.2 LOADING OF TRANSLATIONS

There is a specialized *QTranslator* instance for every plug-in and for core. During application start-up the translation manager reads the setting that selects the current language and attempts to load the corresponding `*.qm` files. There is a slight difficulty when registering the language setting, because registration needs a description for the setting and there is no way to translate it at that stage. This problem is avoided by first creating the setting with empty description and setting the translated description after the language is loaded.

6.6.3 AVAILABLE LANGUAGES

When the translation manager lists available languages it lists all `*.qm` files in a directory of each plug-in and in the directory of the main updraft executable. Base names of these files and "english" are used as names of languages.

6.6.4 RUNTIME TRANSLATIONS

Linguist supports changing translations during runtime using the *languageChange* Qt event. However every string has to be translated as a reaction to this event which is problematic especially for *QAction*'s and for strings generated during initialization of plug-ins. For these reasons Updraft only loads translations on start-up.

7 THE PLUGINS

7.1 INTRODUCTION

7.1.1 PLUG-IN CREATION

A new plug-in must contain the main class inheriting from *PluginBase* and *QObject*. Additionally, the class must be marked as *Q_DECL_EXPORT* and *Q_INTERFACES*(*Updraft::PluginBase*), and its implementation must contain *Q_EXPORT_PLUGIN2*(*pluginName*, *PluginClass*) outside all functions.

As a convention, we manually add a global variable *g_core* to all plug-ins to simplify access to the core interface in other classes of the plug-in.

7.1.2 MINIMAL PLUG-IN EXAMPLE

This example shows the mandatory steps to be taken for successful plug-in creation.

In file `/Updraft/src/plugins/foo/fooplugin.h`

```
#include "../../pluginbase.h"

class Q_DECL_EXPORT FooPlugin
    : public Updraft::PluginBase, public QObject {
    Q_OBJECT
    Q_INTERFACES(Updraft::PluginBase)

public:
    QString getName();
    void initialize(Updraft::CoreInterface *coreInterface);
    void deinitialize();
};
```

In file `/Updraft/src/plugins/foo/fooplugin.cpp`

```
#include "fooplugin.h"

Updraft::CoreInterface *g_core = NULL;

QString FooPlugin::getName() {
    return "foo";
}

QString FooPlugin::initialize(Updraft::CoreInterface
*coreInterface) {
    g_core = coreInterface;
}
```

```
QString FooPlugin::deinitialize() {}  
  
Q_EXPORT_PLUGIN2(foo, FooPlugin)
```

In file /Updraft/src/plugins/foo/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)  
PLUGIN_BUILD(foo)
```

7.2 AIRSPACES

7.2.1 INTRODUCTION

One the most important information for a pilot planning his flight is to know where he is actually allowed to fly. The air space is divided into several sectors defined by its shape and a top and bottom altitudes. Each such a defined part of the air space belongs to a particular class, which will tell the pilot more about its availability. There are for example military airspaces, no fly zones or restricted areas, the pilot must be aware of, otherwise risking high fine or even worse kind of countermeasures. This type of awareness is vital for every pilot flying. As these information are so crucial, the pilot usually uses the approved maps, which are costly, and cannot rely on open-source data. On the other hand, if the pilot wishes to use the visualisation tool to check the proximity of any airspace to his flight path after the flight is done, the open-source data are usually sufficient approximation of the reality. These may not replace the approved maps but can rather serve as an eye-candy in post flight visualisation. The files containing such a data are commonly accessible via the Internet. Therefore we like to provide the user with a tool for visualisation of these data. As the most common format used in Europe is the OpenAir(tm)⁴ text format, we decided to incorporate the OpenAir(tm) airspace definition format parser into the application as well as the visualisation algorithm to draw these information on the map surface. For more detail please see the Appendix B – OpenAir(tm) airspace format definition.

⁴ <http://www.winpilot.com/UsersGuide/UserAirspace.asp>

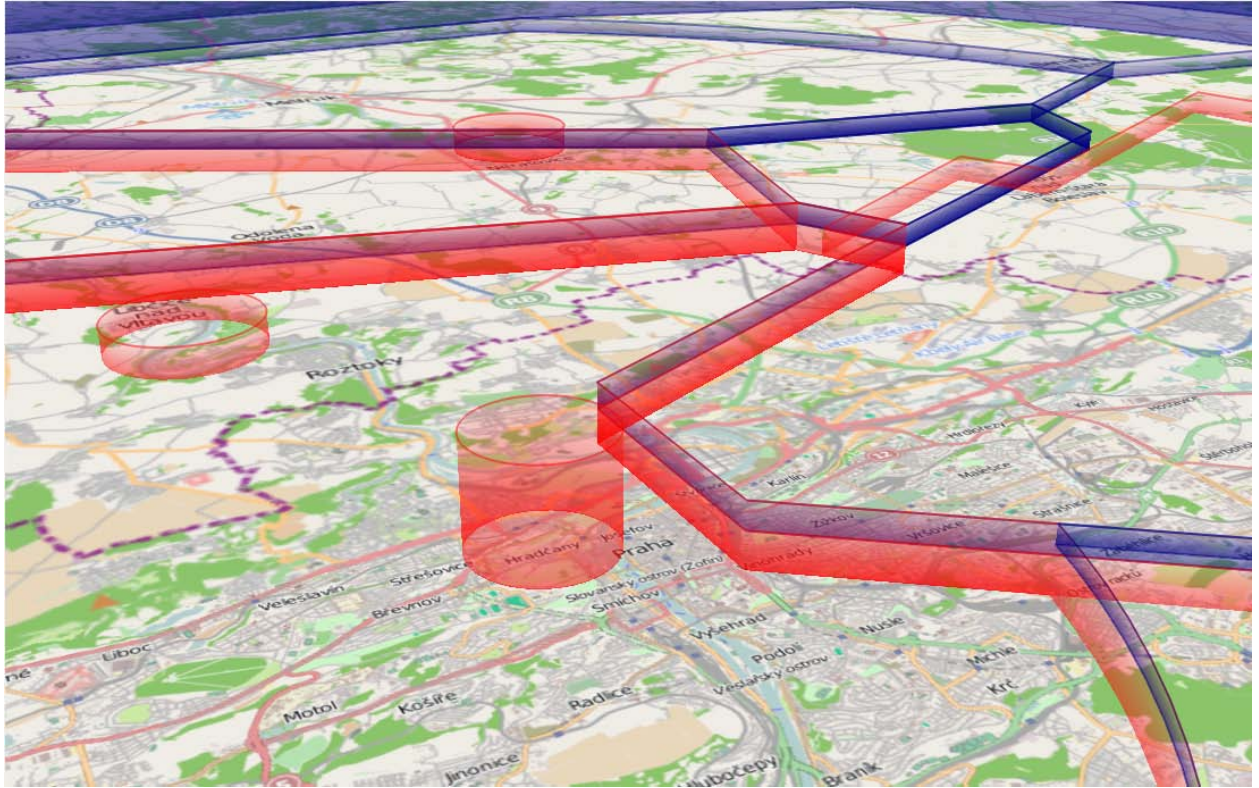


Figure 6. Example of airspace divisions visualisation.

7.2.2 OVERVIEW

The main role of this plug-in is to parse an airspace definition data from the provided file and visualize them.

The basic and at the time of the first program release the only format is the OpenAir(tm) format (see Appendix B – OpenAir(tm) airspace format definition).

7.2.3 DESIGN

The program is divided in two main parts of the airspace processing. The first part consists of the routines which load and parse the text input file containing the provided airspace information. This information is then handed over to the second part of the algorithm, which will draw the defined airspaces into the map layer. We call the first part the *Parser* and the second part the *Airspaces* plug-in. The plug-in is loaded by the core as a part of the application, whereas the Parser is a library called from within the plug-in otherwise forming independent program. A part of the plug-in is the format-specific drawing engine which is called by the plug-in to draw to a particular map layer. The engine is provided with the name of the file the user wishes to process by the *Core* and calls the parser to load the airspaces which the file contains.

The *Parser* as well as the *Airspaces* library is linked to the main project as a dynamic library.

7.2.4 OPENAIR(TM) PARSER

This airspace declaration file is in text format and can contain information about several airspaces usually grouped into one logical or geographical group. Within the file the class, name, altitudes and other information is provided for each of the defined airspace. The airspace itself is defined either as a set of points and arcs forming a polygon or as a set of primitives (i.e. circles). During the start phase of the program, the plug-in parses the information from all the files which are saved in */data/airspaces* folder. Each file typically containing several airspaces within one region (e.g. country, city, etc.).

During the parsing phase of the process the file is opened for reading and the parsing process enters the main parsing loop, which ends when all the file content is processed or if the unexpected error occurs, e.g. the file is of incorrect format. In this loop one by one airspace class member is defined and filled with the data in file. This process is done within the particular Airspace class constructor. As stated above, the airspace is defined by the geometric primitives, such as arcs and points forming a closed polygon or a circle. The special data structures for these primitives were defined in the *geometry.cpp* and *geometry.h* files. These are the *Polygon*, *Arcl*, *Arcll* and *Circle* classes.

The parsing process itself is quite straight forward. The only tricky factoid of which the programmer must be aware of is that some of the variables the airspace definition file contains are state variables. That means that the validity of such a variable reaches the beyond the currently parsed airspace hence is valid until the end of the file or until re-defined. One example of such a variable are the pen and brush definitions, which are often defined in the beginning of the airspace definition file only and are valid for all the airspaces contained in the file. Another example could be the definition of the centre of co-centric airspaces. These are typically the airspaces surrounding one airport, therefore its centre is defined once per airport.

The Parser library is contained in following files located in */Updraft/src/libraries/openairspace/* folder.

- | | |
|---------------------------------|--|
| a. <i>openairspace.cpp</i> | – the OpenAir(tm) format parser main procedures |
| b. <i>openairspace.h</i> | – the OpenAir(tm) parser class declaration and interface |
| c. <i>airspace.cpp</i> | – the single airspace class |
| d. <i>airspace.h</i> | – the single airspace class declaration and interface |
| e. <i>openairspace_global.h</i> | – the parser library main interface |
| f. <i>geometry.cpp</i> | – the airspace geometry primitives class definitions |
| g. <i>geometry.h</i> | – the airspace geometry primitives class declarations |

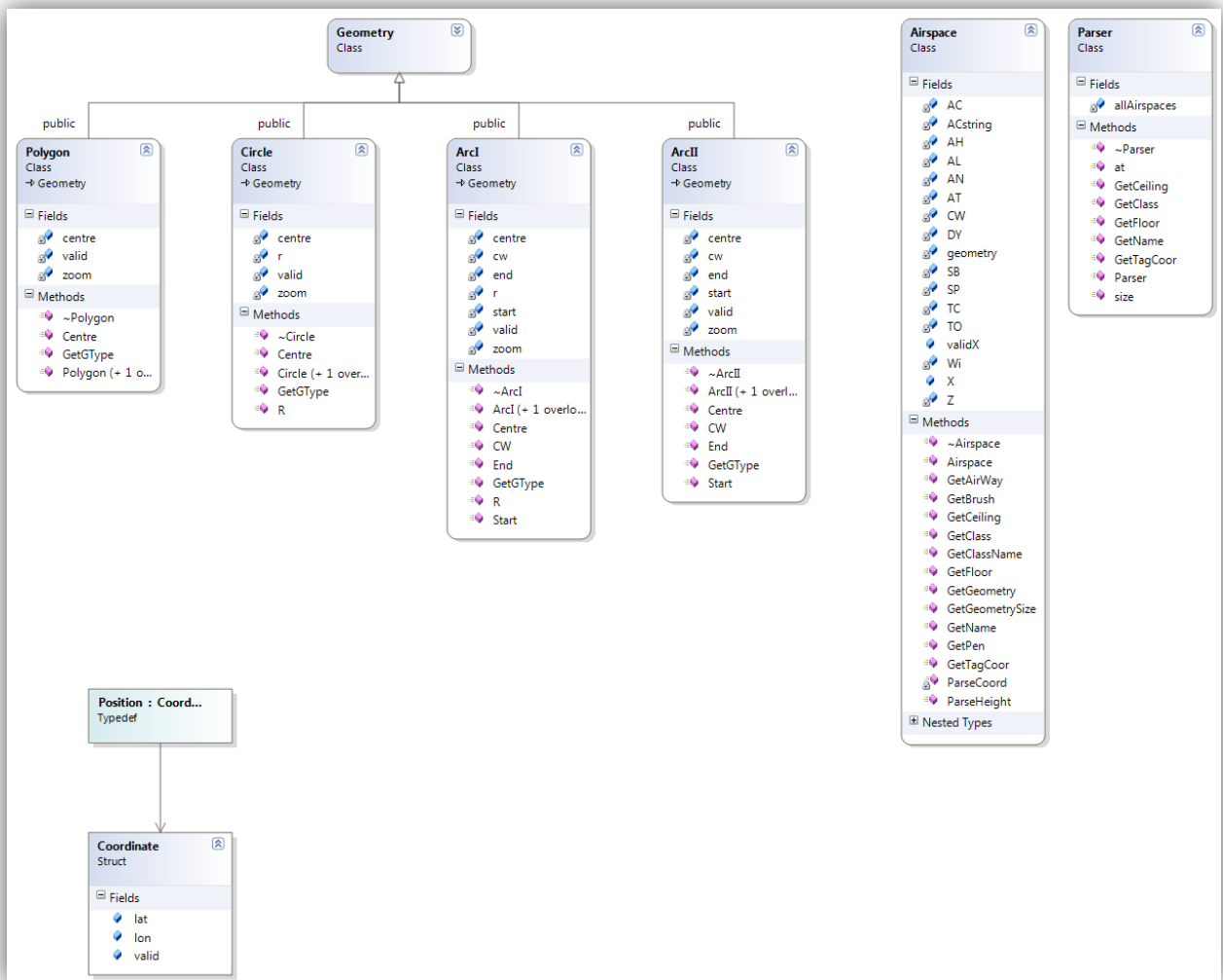


Figure 7 : This graphic shows the classes of the OpenAir(tm) airspace file parser. Each cell shows the variables and method of the particular class.

Output of the parser is the member of *OpenAirspace::Parser* class containing information about all the airspaces described in particular file. The only private variable of this class is the *allAirspaces* array of *OpenAirspace::Airspace* members. Each such a member contains parsed information about one airspace. This concludes the pen and brush colour, name, class, ceiling, floor, geometric primitives forming the airspace boundaries and more.

The OpenAir(tm) *Parser* library is linked to the main project as a dynamic library.



Figure 8. Example of the OpenAir(tm) data visualisations.

7.2.5 OPENAIR(TM) DRAWING ENGINE

The main class connected to the *Core* is the *Airspaces* class. This class is responsible for the communication with the core and for calling the airspace drawing engine and parser. In the initial phase of the plug-in load the *Airspaces* all the files in */data/airspaces* folder are processed. During the program run also the user-imported files are drawn and the file is stored in */data/airspaces* directory. Each of the files, which needs to be processed is given to the drawing engine, specifically the *oaEngine* class constructor. This method is then responsible for calling the parser.

As mentioned above the output of the parser is the member of *OpenAirspace::Parser* class containing information about all the airspaces described in particular file. This member contains in fact an array of parsed airspaces for the given region and is the main outcome of the parsing process. Each of the airspaces in array is then processed in such a way, that first the colour specification is read (this is often common for whole the file or at least for group of airspaces) or the default colour is assigned. Next the geometry is created for each of the airspace's set of points and arcs forming a closed polygon or a circle. The final geometry which is to be drawn consists of bottom polygon and the top polygon. Both the polygons being topologically identical each of them is located at height assigned as a floor and ceiling of the airspace respectively. Data defining the polygons were parsed during the process described above. The array of point pairs (bottom polygon point and corresponding top polygon point) defines the triangle strip OpenGL primitive used to visualize the side of the airspace. As mentioned earlier the OpenAir(tm) format defines the

airspace as a circle primitive or as a closed polygon consisting of a set of points and arcs. OpenGL is based on processing the triangle meshes defined as a set of vertices for performance reasons. Thus all the arcs, as well as circles declared must be converted to a set of points allowing the use of OpenGL mesh setup. Granularity of this conversion is defined as a constant within *oaEngine* class. Because all the data defining the airspace are in WGS84 [16] format, many arbitrary algorithms which work with this coordinate system had to be introduced. The main aim of these procedures is to compute the set of points from given airspace geometry. These points can be then used for filling the vertex array of the OpenGL utilising the methods of OSG Earth. These are used to correctly transform the WGS84 coordinates of vertices to the OpenGL world view and for correct evaluation of the terrain height from the height field data provided by OSG Earth.

Such a filled OpenGL vertex array is added to a scene graph node as a set geometry. Each node is now defining single particular airspace. Finally the set of OpenGL switches is used to ensure correct evaluation of transparency effects and visibility of the airspaces. The nodes are accumulated in map nodes array, which is handed back to the *Airspaces* class.

Each of the node in the array is added by *Airspaces* class to the scene tree of OSG as a map layer. In the same time the GUI display tree is created and signals for displaying the defined layers connected.

The Parser library is contained in following files located in
/Updraft/src/plugins/airspaces/ folder.

- a. *airspaces.cpp* - the main interface of the plug-in
- b. *airspaces.h* - the airspaces plug-in interface
- c. *oaengine.cpp* - the main airspace drawing engine
- d. *oaengine.h* - the OpenAir(tm) drawing engine class and methods declarations

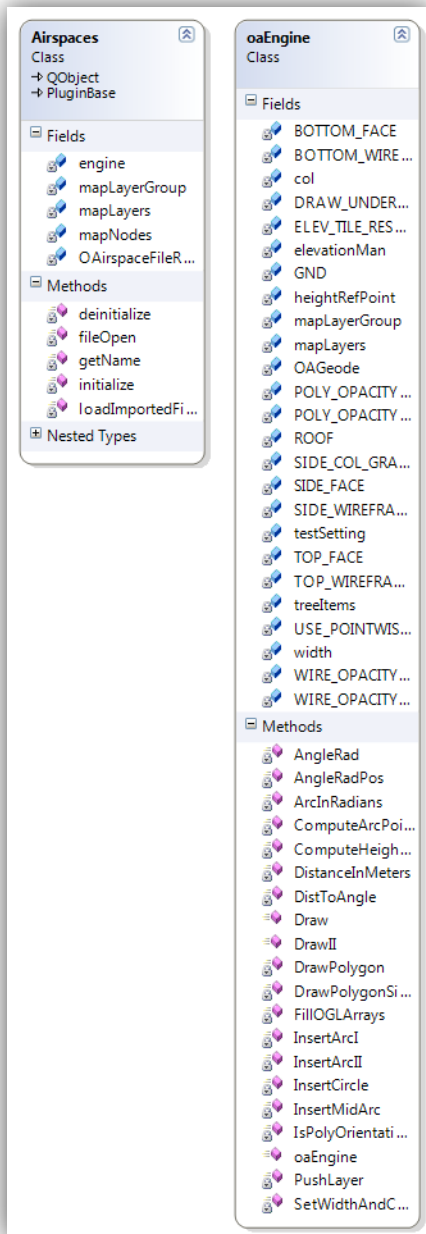


Figure 9 : Airspaces plug-in classes. In each of the cells the variables and methods of the particular class are stated.

The *Airspace* plug-in library is linked to the main project as a dynamic library.

7.2.6 3RD PARTY SOFTWARE USED

- Nokia Qt [4] for the GUI framework
- OSG [9] – as a OpenGL framework, for the mesh visualisation
- OSG Earth [10] – for the world-view matrix transformations

7.2.7 THE AIRSPACES PLUG-IN DIAGRAM

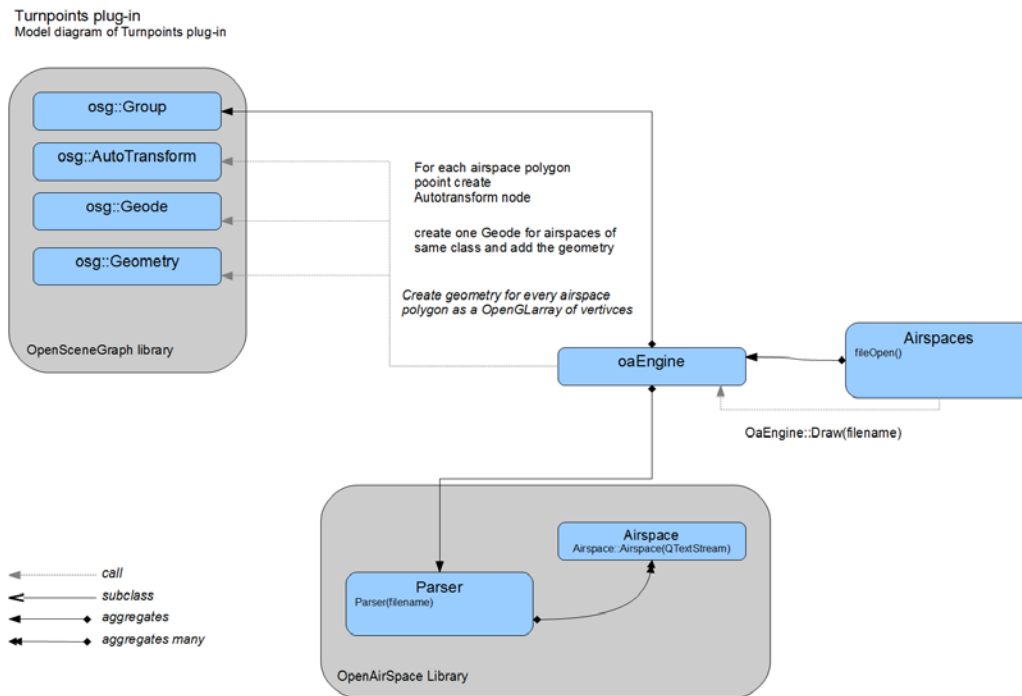


Figure 10 : Airspaces plug-in and OpenAir parser diagram.

7.3 TURNPOINTS

7.3.1 INTRODUCTION

During the glider competition flight the typical task of a pilot is to navigate through the given set of waypoints usually forming a triangle. Each point is defined by its geographical coordinates, typically in WGS84 coordinate system. We will call such a point the *turn-point*. It is common practice (especially during competition) that there is a predefined set of such points. The set is distributed electronically as a file containing all the turn-points (e.g. by competition organizers) concerning the particular area. These turn-points therefore play an important role in flight planning procedure. Hence it is very useful for our software to have the turn-points visualisation capability. It is also very convenient way to visualise the airports whenever is this information provided by the turn-points file.

7.3.2 OVERVIEW

This plug-in is intended to collect and visualize turn-points. It uses Updraft Cup library, which implements parsing and loading turn-points from SeeYou *cup* files. For the cup file format description please see the Appendix D – Cup File Format Description. Loading turn-points from any file format is potentially possible – the plug-in is easily extendable and it doesn't depend on cup file format. It is a good practise that the definition of the turn-point also contains the information about the turn-point type, one of which is the Airport type. Together with this the detailed information about the airport, such as main runway heading or airport type, is often provided. In our application we utilise this information for distinguishing the airport from the common turn-point using different icons. In case of airport visualisation, we also use the information about the runway heading to correctly situate the icon.

7.3.3 DESIGN

The Turn-points plug-in is loaded by the core the same way any other plug-in is. The plug-in register the file type associated with the plug-in in the initial phase. It also loads all the cached files in the designated turn-points directory. Please see chapter _____(file registration) for more detailed description. For each of the files the cup file parser library is called to extract the information about the turn-points from the files. Subsequently the algorithm for visualising the parsed data is called. It processes each of the turn-points and using the information about the type, geographical position and height it creates the icons, which forms the independent map layer. This map layer is then added to the OSG scene with the means described in _____(mapLayer).

7.3.4 TPLAYER CLASS

This class is responsible for creating the geometry from the parsed data utilising the OSG methods for creating the OSG nodes and further using the OSG transformation matrices for positioning the data into the scene. It also encapsulates the geometry to the map layer, which is finally added to the OSG scene to be visible in the map window.

7.3.5 TPFILECUPADAPTER CLASS

This class is a part of the Turnpoints plug-in and serves as an intermediate layer between the CupFile class and the Turnpoints plug-in. It prepares the parsed data in such a way, that these can be used by the TPLayer class for visualisation. This class is responsible for loading the cup files by calling the cup parser.

7.3.6 CUP FILE PARSER

Each of the file containing the turn-points needs to be read, data it contains extracted and encapsulated in class data structure to be accessible by the turn-points plug-in. In case of the SeeYou Cup format (see the detailed description in the Appendix D – Cup File Format Description) these data are held in *Updraft::CupFile* class. Data in this format are prepared for use by the Turnpoints plug-in.

7.3.7 TURNPONTS PLUG-IN DIAGRAM

Turnpoints plug-in
Model diagram of Turnpoints plug-in

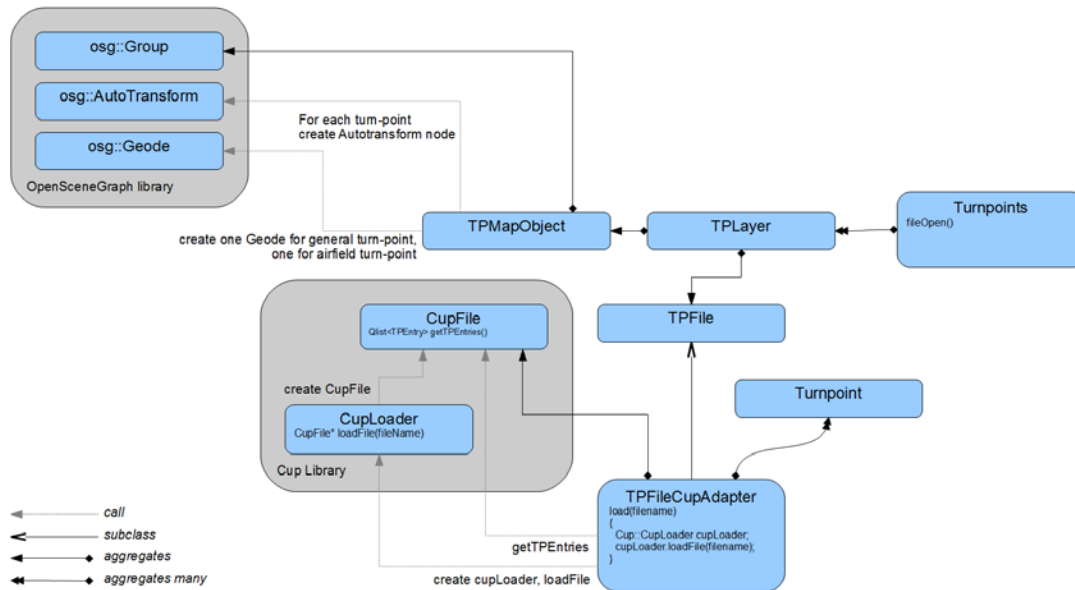


Figure 11. Turn-points plug-in design.

7.4 TASK DECLARATION

7.5 IGC VISUALISATION

8 PROGRAM INSTALLATION

8.1 WINDOWS

8.2 LINUX

8.3 MAC

9 CONCLUSION

We have created this application to serve the glider pilots as a glider path planning and flight visualisation tool. It is a free easy-to use alternative, and we believe even an improvement, to the commercial product SeeYou. The main targets we set ourselves as a goal has been all reached. The application has proven to be smoothly running on all of the designated platforms, i.e. Windows, Linux and Mac OS. The implemented features includes the Airspaces plug-in for airspace division visualisation, the Turnpoints plug-in for the waypoints visualisation, the TaskDeclataion plug-in for the flight planning and the IGC plug-in for the recorded flight path visualisation. All these features are incorporated to the nice looking 3D map environment including several map types and even the terrain elevation visualisation.

10 FUTURE WORK

11 REFERENCES

- [1] Naviter, "SeeYou," Naviter, [Online]. Available:
http://naviter.si/index.php?option=com_content&task=view&id=9&Itemid=213.
- [2] "FreeLists," [Online]. Available: <http://www.freelists.org>.
- [3] Google, "Google C++ Style Guide," [Online]. Available: <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>.
- [4] Nokia, "Qt," Nokia, [Online]. Available: <http://qt.nokia.com/>.
- [5] Microsoft, "Visual Studio 2008," Microsoft, [Online]. Available:
<http://www.microsoft.com/visualstudio/en-us/products/2008-editions>.
- [6] Open-source, "CMake," [Online]. Available: <http://www.cmake.org/>.
- [7] L. Torvalds. [Online]. Available: <http://git-scm.com/>.
- [8] "GitHub," [Online]. Available: <http://www.github.com>.
- [9] Open-source, "The OpenSceneGraph," [Online]. Available: <http://www.openscenegraph.org>.
- [10] G. W. Jason Beverage, "OSG Earth," Pelican Mapping, [Online]. Available: <http://osgearth.org>.
- [11] Wikipedia, "Qt framework," [Online]. Available: [http://en.wikipedia.org/wiki/Qt_\(framework\)](http://en.wikipedia.org/wiki/Qt_(framework)).
- [12] K. Gtoup, "OpenGL," [Online]. Available: <http://www.opengl.org/>.
- [13] Open-source, "OpenStreetMap," [Online]. Available: <http://tile.openstreetmap.org/>.
- [14] ESRI, "ESRI Imagery World 2D," [Online]. Available:
http://server.arcgisonline.com/ArcGIS/rest/services/ESRI_Imagery_World_2D/MapServer.
- [15] G. W. Jason Beverage, "Ready map elevation tiles," Pelican Mapping, [Online]. Available:
<http://readymap.org/readymap/tiles/1.0.0/9/>.
- [16] Wikipedia, "World Geodetic System," [Online]. Available:
http://en.wikipedia.org/wiki/World_Geodetic_System.

12 TABLES

Table 1 : Team members main responsibilities.

7

13 FIGURES

Figure 1. The OSGEarth framework example.	11
Figure 2. OpenStreet map source.	13
Figure 3. ArcGIS online map service.	13
Figure 4. Application architecture diagram.	14
Figure 5. Example of airspace divisions visualisation.	21
Figure 6 : This graphic shows the classes of the OpenAir(tm) airspace file parser. Each cell shows the variables and method of the particular class.	23
Figure 7. Example of the OpenAir(tm) data visualisations.	24
Figure 8 : Airspaces plug-in classes. In each of the cells the variables and methods of the particular class are stated.	26
Figure 9 : Airspaces plug-in and OpenAir parser diagram.	27

14 APPENDIX A – PEOPLE

Alexander Wilkie – Computer Graphics Senior Lecturer on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Tomas Zamecnik – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Cestmir Houska – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Jakub Marek – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Bohdan Maslowski – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Maria Vamosova - Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

Ales Zita – Student of Computer Science master degree on Faculty of Mathematics and Physics by Charles University, Prague, Czech Republic; email:

15 APPENDIX B – OPENAIR(TM) AIRSPACE FORMAT DEFINITION

Following is the definition of the OpenAir(tm) format as provided on the winpilot⁵ web page.

OPEN AIR (tm) TERRAIN and AIRSPACE DESCRIPTION LANGUAGE

Version 1.0

December 10, 1998

Updated October 15, 1999

Send comments to jerryp@winpilot.com

AIRSPACE related record types:

=====

AC class ; class = Airspace Class, see below:

R restricted

Q danger

P prohibited

A Class A

B Class B

C Class C

D Class D

GP glider prohibited

CTR CTR

W Wave Window

AN string ; string = Airspace Name

AH string ; string = Airspace Ceiling

AL string ; string = Airspace Floor

AT coordinate ; coordinate = Coordinate of where to place a name label on the map (optional)

; NOTE: there can be multiple AT records for a single airspace segment

TERRAIN related record types (WinPilot version 1.130 and newer):

⁵ <http://www.winpilot.com/UsersGuide/UserAirspace.asp>

=====

TO {string} ; Declares Terrain Open Polygon; string = name (optional)
TC {string} ; Declares Terrain Closed Polygon; string = name
(optional)
SP style, width, red, green, blue ; Selects Pen to be used in
drawing
SB red, green, blue ; Selects Brush to be used in drawing

Record types common to both TERRAIN and AIRSPACE

=====

V x=n ; Variable assignment.
; Currently the following variables are supported:
; D={+|-} sets direction for: DA and DB records
; '-' means counterclockwise direction; '+' is the default
; automatically reset to '+' at the begining of new airspace segment
; X=coordinate : sets the center for the following records: DA, DB,
and DC
; W=number : sets the width of an airway in nm (NYI)
; Z=number : sets zoom level at which the element becomes visible (WP
version 1.130 and newer)

DP coordinate ; add polygon pointC
DA radius, angleStart, angleEnd ; add an arc, angles in
degrees, radius in nm (set center using V X=...)
DB coordinate1, coordinate2 ; add an arc, from coordinate1 to
coordinate2 (set center using V X=...)
DC radius ; draw a circle (center taken from the previous V X=...
record, radius in nm
DY coordinate ; add a segment of an airway (NYI)

16 APPENDIX D – CUP FILE FORMAT DESCRIPTION

Each line represents one waypoint with these fields, separated by commas. Here is an example:

```
"Lesce-Bled", "LESCE", SI, 4621.666N, 01410.332E, 505.0m, 2, 130, 1140.0m, "123.50", "Home airfield"
```

1. Name

This is the long name for the waypoint. It is supposed to be embraced in double quotes to allow any characters, including a comma in between. This field must not be empty.

2. Code

Also known as short name for a waypoint. Many GPS devices cannot store long waypoint names, so this field will store a short name to be used in various GPS types. It is advisable to put it in double quotes.

3. Country

IANA Top level domain standard is used for the country codes. A complete list is available at <http://www.iana.org/cctld/cctld-whois.htm>

4. Latitude

It is a decimal number where
1-2 characters are degrees,
3-4 characters are minutes,
5 decimal point,
6-8 characters are decimal minutes.
The ellipsoid used is WGS-1984

5. Longitude

It is a decimal number where
1-3 characters are degrees,
4-5 characters are minutes,
6 decimal point,
7-9 characters are decimal minutes.
The ellipsoid used is WGS-1984

6. Elevation

It is a string with a number with unit attached. Unit can be either "m" for meters or "ft" for feet.
Decimal separator must be a point.

7. Waypoint style

It is a digit representing these values:

- 1 - Normal
- 2 - AirfieldGrass
- 3 - Outlanding
- 4 - GliderSite
- 5 - AirfieldSolid
- 6 - MtPass
- 7 - MtTop
- 8 - Sender
- 9 - Vor
- 10 - Ndb
- 11 - CoolTower
- 12 - Dam
- 13 - Tunnel
- 14 - Bridge
- 15 - PowerPlant
- 16 - Castle
- 17 - Intersection

8. Runway direction

It is a string in degrees representing heading of the runway. Only used with Waypoint style types 2, 3, 4 and 5

9. Runway length

It is a string for number with unit representing length of the runway. Only used with Waypoint style types 2, 3, 4 and 5

unit can be either

"m" for meters

"nm" for nautical miles

"ml" for statute miles

Decimal separator must be a point.

10. Airport Frequency

It is a string representing the frequency of the airport. Decimal separator must be a point. It can also be embraced in double quotes.

11. Description

It is a string field with no limitation in length where anything can be stored in. It should be embraced with double quotes.

17 APPENDIX C – ELEVATION DATASET USAGE CONFIRMATION

Od: "Glenn Waldron" <gwaldron@pelicanmapping.com>
Komu: "Ales Zita" <ales.zita@volny.cz>
Předmět: Re: ReadyMap elevation data
Datum: 10.12.2011 - 15:53:03

Ales,

You are welcome to use the server for your project and for a freeware app.
Thanks for asking, and if you have any issues or questions, don't hesitate to email me.

Glenn Waldron / Pelican Mapping / @glennwaldron

On Sat, Dec 10, 2011 at 5:57 AM, Ales Zita <ales.zita@volny.cz> wrote:

```
> From: Ales Zita <ales.zita@volny.cz>
> Subject: ReadyMap elevation data
>
> Message Body:
> Hello,
>
> I'd like to ask for permission to use the world-wide elevation
> data for
> > our school project.
>
> We are a group of 6 people developing the the glider flightpath
> visualisation tool in osgEarth environment as a school project at
> Faculty
> > of Mathematics and Physics by the Charles University in Prague.
> As this is non commercial project, we cannot afford any paid
> elevation
> > data sources. Therefore we would very much appreciate the use of
> your
> > ReadyMap global elevation dataset accessible via the osgEarth api.
>
> Should this project become available as a freeware at some point,
> can we
> > continue to use your server as a souce of the elevation data?
>
> Thank you
>
> Kind regards
>
> Ales Zita
```

>
> --
> This mail is sent via contact form on Pelican Mapping
> <http://pelicanmapping.com>

18 APPENDIX D – DVD CONTENT