

# UPWIND PROJECT UPDATE SPRING 2012

Anu Pramila  
Andrei Vainik  
Juha-Matti Hurnasti  
Tomi Sarni

## Table of Contents

- [1. Introduction](#)
- [2. Development Environments](#)
  - [2.1 Setting Up Linux Development Environment](#)
    - [2.1.1 32-bit vs. 64-bit](#)
    - [2.1.2 Gdal and Postgre libraries](#)
    - [2.1.3 QT Creator and QT Libraries](#)
    - [2.1.4 Git](#)

- [2.2 Windows \(32-bit & 64bit\)](#)
- [2.3 Mac](#)
  - [2.3.2 QT Creator and QT Libraries](#)
- [3. QT Creator](#)
  - [3.1 Debugging](#)
  - [3.2 Folders and relative paths](#)
  - [3.3. Advice to Mac users](#)
- [4. Git](#)
  - [4.1 About GIT in general](#)
  - [4.2 Branches](#)
  - [4.3 Command line usage](#)
  - [4.5 Misc stuff](#)
  - [4.4 SmartGit](#)
- [5. Database](#)
  - [5.1 pgAdmin](#)
- [6. UpWind Third Party Libraries](#)
  - [6.1 GDAL - Geospatial Data Abstraction Library](#)
- [7. Architecture](#)
  - [7.1 UWCore](#)
  - [7.2 UWPlugins](#)
- [8. Known Issues](#)
  - [8.1 Linux](#)
  - [8.2 Mac](#)
  - [8.3 Advice](#)
- [9. Suggestions for Future Development](#)
  - [9.1 Implement XML Readers and Writers](#)
  - [9.2 Improving efficiency](#)
  - [9.3 Correcting functionality](#)
  - [9.4 Short term route planning](#)
- [10. Appendix](#)
  - [10.1 Database description](#)

# 1. Introduction

The UpWind project is about creating a sailboat navigation software. The project has been running since around 2006 and has had multiple development teams mainly consisting of students. As expected the legacy is colorful to say the least. There are two versions of the software, in this document we refer to them as “the old” and “the new”. The old code version is no longer being developed. The new version is being developed based on the old code but with different architecture. In the new architecture features and functionality have been divided into components, plugins.

This document is meant as a sort of starter pack for next team who will be working with the project (fall 2012). Even if you are familiar with C++ and QT, we suggest that at least browse through this document.

## 2. Development Environments

UpWind can be developed with all of the following operating systems. The following chapters describe some known issues which to consider when developing with these operating systems.

### 2.1 Setting Up Linux Development Environment

#### 2.1.1 32-bit vs. 64-bit

For sake of compatibility, install the development environment to 32-bit linux systems. The gcc compiled files are not compatible between 32-bit and 64-bit systems. All-in-all, it is more difficult to find libraries and programs for 64-bit linux thus not making it worth the 1 GB+ of memory to install 64-bit system.

In case you working with 64-bit linux, a quick google search does give the following answer to a problem:

<http://www.cyberciti.biz/tips/compile-32bit-application-using-gcc-64-bit-linux.html>

#### 2.1.2 Gdal and Postgre libraries

The picture below represents a view from “.pro” -file. These files are tracked by git so you should install the system using the paths mentioned below but those paths can be changed too, just don't commit the .pro files then so that others won't have problems with the paths. To download and install the gdal 1.7.0 library, use the Synaptic Package Manager and quick search for “gdal”. You have to install following libraries: *gdal-bin* , *libgdal1-dev* and *libgdal1-1.7.0* . By default it should install them to */usr/include/gdal* path. For Postgre you need the following libraries installed: *libpg-dev*, *libpg5*, *postgresql-client*, *postgresql-client-9.1* .

For the old version to compile, you might need to get OpenGL libraries as well. Use the Synaptic Package Manager as well to install these. You need at least the following: *libqt4-opengl*, *libglu1-mesa*, *libglu1-mesa-dev*, *libgl1-mesa-dri*, *libgl-mesa-dev*, *libgl-mesa-glx*, *mesa-common-dev*.

Try not to get the latest versions of the libraries as they might not be compatible with the code.

#### 2.1.3 QT Creator and QT Libraries

You can use the latest version of QT IDE Creator (2.4.1 +) but try to install QT libraries version 4.7.4 . You do not need the other stuff, just the IDE and libraries.

Download link:

<http://qt.nokia.com/downloads>

## 2.1.4 Git

Whether you use the command line version or some other graphical version (Smartgit) or integration (eGit for Eclipse), you need to install git first. You can do this easily with opening Software Manager or Synaptic Package Manager and searching for “git” and install the plain “git” which should automatically install *git-core* and *git-gui* also.

Install git from command line:

```
$ sudo apt-get install git
```

```
DESTDIR += ../plugins
```

```
mac {
    INCLUDEPATH += /Library/Frameworks/GDAL.framework/Headers/
    LIBS += -L/Library/Frameworks/GDAL.framework/unix/lib -lgdal -lpq
}

else:unix {

    INCLUDEPATH += /usr/include/gdal/ \
        /usr/include/postgresql
    LIBS += -lpq -lgdal1.7.0
}

win32 {
    INCLUDEPATH += D:\\gdal\\include
    INCLUDEPATH += "C:\\PostgreSQL\\8.3\\include"
    LIBS += -LD:\\gdal\\bin
    LIBS += -llibgdal-1 \
        -llibpq
    LIBS += -L"C:\\PostgreSQL\\8.3\\bin"
}
```

Notice the different way how files are accessed in different operating systems.

Here's another way to link libraries on Windows. This way you can define the .dll file explicitly.

```
win32 {
    INCLUDEPATH += "C:\\gdal\\gdal\\include"
    LIBS += "C:\\gdal\\gdal\\bin\\libgdal-1.dll"

    INCLUDEPATH += "C:\\Program Files (x86)\\PostgreSQL\\8.4\\include"
    LIBS += "C:\\Program Files (x86)\\PostgreSQL\\8.4\\bin\\libpq.dll"
}
```

## 2.2 Windows (32-bit & 64bit)

Notice the win32 in above picture.

## 2.3 Mac

The UpWind project can be developed with Mac Os X also. The author of this chapter has used Snow Leopard but there is no reason to believe that other versions would not work as well. If something has not been explained in this chapter, then check also the instructions for Linux system. They may help.

### 2.3.1 Gdal and Postgre libraries

You will need to install Gdal library. Gdal package can be found from the internet. For example <http://www.kyngchaos.com/software/frameworks> has a download link for package named Gdal\_complete.dmg. I used version 1.8 but the newest version might work as well. Install the package and you should have GDAL.framework folder in Macintosh HD -> Library -> Framework.

Next, in the GDAL.framework folder, go to unix -> lib. There you should have libgdal.dylib. In this same folder we should move also the postgre library. (There could be a better way to do this but this way works). Download from the Postgresql's web site the package for mac os. Install the package but there is no need to run it. We only need a library. Find library called libpq.dylib and copy it to GDAL folder as described earlier. The file is somewhere in Macintosh HD -> Library -> PostgreSQL folder.

In the code itself, the libraries are included in .pro-files. E.g. the part of a .pro file which is important for a mac user for CharProviderInterface plugin looks like

```
mac {  
    INCLUDEPATH += /Library/Frameworks/GDAL.framework/Headers/  
    LIBS += -L/Library/Frameworks/GDAL.framework/unix/lib -lgdal -lpq  
}
```

This plugin is the one where these libraries are needed. PostgreChartProvider plugin is another where code lines like these are found. most of the other plugins have no need for these libraries.

If you wish to have the old project up and running, you will also need some OpenCV libraries and therefore install OpenCV. OpenCV is a collection of libraries used in machine vision. Installing the OpenCV for mac could be a bit tricky but there are quite good instructions in the OpenCV web site e.g. <http://opencv.willowgarage.com/wiki/InstallGuide>

The .pro file for the old project should look according to the mac version something like this

```
mac {  
    INCLUDEPATH += /Library/Frameworks/GDAL.framework/Headers  
    INCLUDEPATH += /Library/PostgreSQL/9.1/include  
    INCLUDEPATH += /Users/apra/opencv/include/opencv  
    LIBS += -L/Library/Frameworks/GDAL.framework/unix/lib -lgdal -lpq  
    LIBS += -L/Users/apra/opencv/build/lib -lopencv_highgui -lopencv_core -lopencv_imgproc  
}
```

You will need to define paths for GDAL, PostgreSQL and OpenCV, as well as define some paths to some libraries

### 2.3.2 QT Creator and QT Libraries

You can use the latest version of QT IDE Creator (2.4.1 +) but try to install QT libraries version 4.7.4 . You do not need the other stuff, just the IDE and libraries.

Download link:

<http://qt.nokia.com/downloads>

### 2.3.3 Git

Git can be used from the command line but I would recommend using **SmartGit** especially if you are unfamiliar with git.

Git can be found from <http://git-scm.com/download> and SmartGit <http://www.syntevo.com/smartgit/download.html>

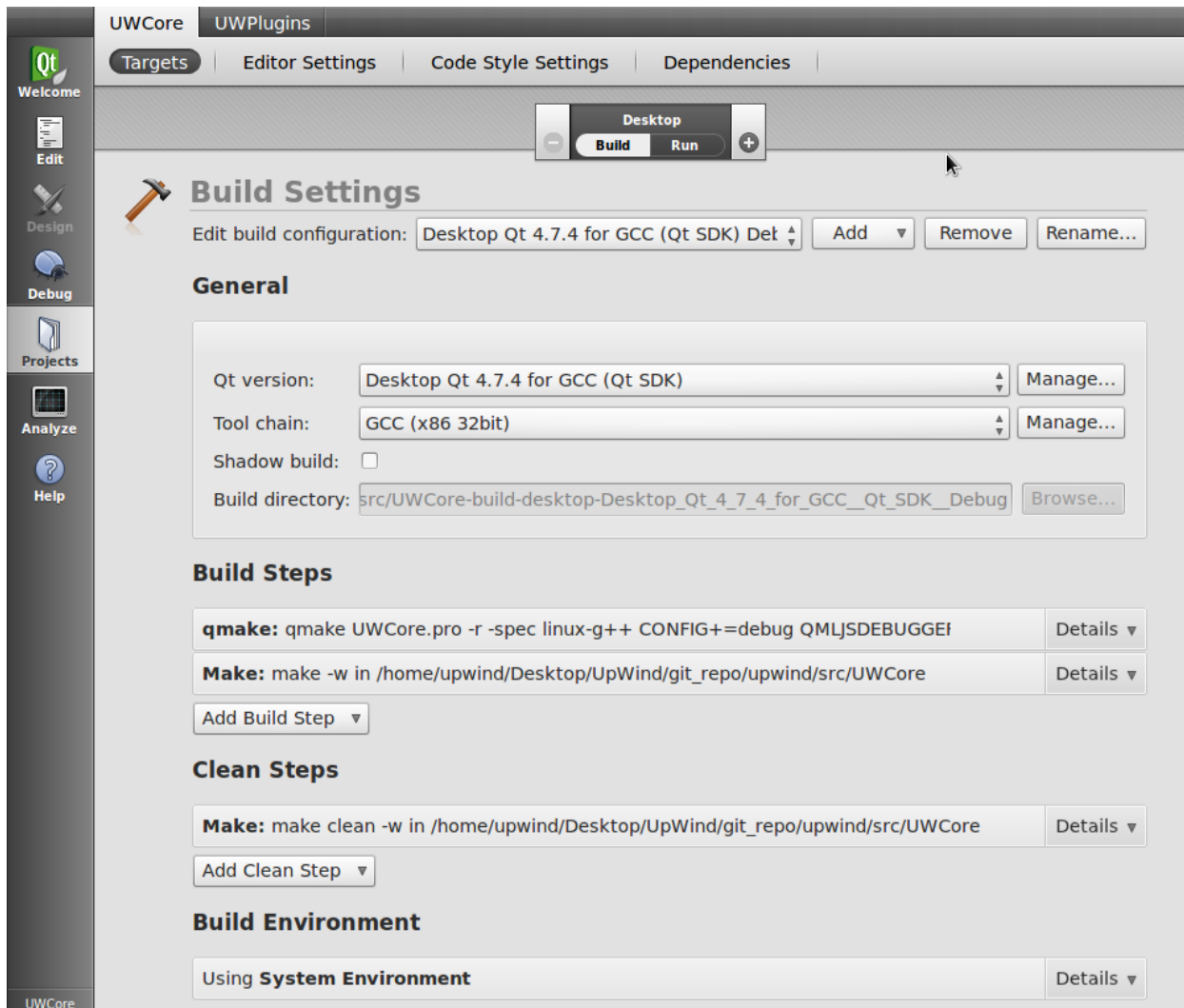
If you download Smartgit from web, take a version > 3. Versions below 3 are not good.

Do not use the git accompanied by the QT Creator. It doesn't work properly, especially with mac.

## 3. QT Creator

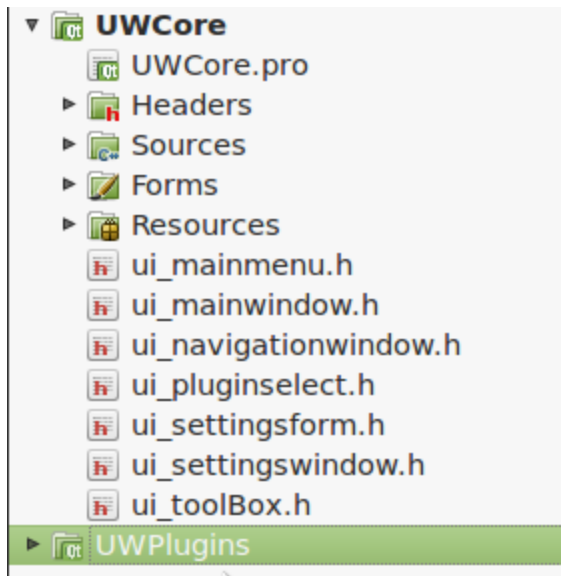
### 3.1 Debugging

In order to use breakpoints and debugging features, you need to make sure you have set up QT to make debug builds.



In the picture above, you can see how the project settings should look like. Remember that you have to do this for each project. You can either use shadow build or don't but you have to make the decision with everyone to avoid problems. We would suggest that you do not use shadow build. You do not need to define "Dependencies". You can change to debug build by switching the correct choice from "Edit Build Configurations".

## 3.2 Folders and relative paths



In the above picture the forms folder contains .ui files which are auto-generated files by the UI editor. those files can be loaded by QT UI editor and edited. Each .ui file has corresponding .h and .cpp file in header and source folders. The resources folder is a generic place to load all the image resources used by the project. The .pro file holds the project settings such as what other libraries or files are being included to the project and some build instructions. Headers hold all the .h files and sources all the .cpp files.

For example when using include a file, you should relative path to file which you want to include. “../” is reference to parent folder. Specially if you use shadow build to different location than beside the source files, you should remember this when including these libraries.

### 3.3. Advice to Mac users

On top of the Project view, there is a selection to change setting for the build configuration and run configuration. Go to each of your projects run configuration settings and change the working directory to something like:

`/Users/username/Desktop/UpWind/git_repo/upwind/src/UWCore/`

or:

`%{buildDir}`

This should be done because otherwise QT Creator tries to find some project specific files (e.g. files that contain database login information so that it doesn't need to be inputted every time you start the program) from .app folder and they are not there but in the build directory.

## 4. Git



## 4.1 About GIT in general

None of our team members had previous experience with GIT, thus we had problems in the beginning. If you have experience with other version control systems like SVN or CVS, do not assume you will know how to use GIT.

GIT was developed for Linux kernel development, that's why it has different logic. The main idea in GIT is to work with branches. A branch is an independent copy of source code, which may be local (your computer) or remote (UpWind server) or both. Just like in any other VCS (version control system) you will have local copy and remote copy and you can commit changes and synchronize your local copy with remote copy. Note that **commit** command will write changes to local copy only. If you want to update changes to remote copy you need to use **push** command. You can get updates from remote copy with **pull** command.

## 4.2 Branches

A good practice is to create a separate branch for each developer, this way you will not mess up each others code, and when you will you can revert changes in that particular branch. All developers will work on their own branches and when you think your code is good, you can merge it with main branch. You can also merge with any other branch you like. Note that you always can and should merge local branches. After you are done merging, you should test that everything works, and after that you can push your changes to GIT server. This way you can make sure that, if you mess something up, you will mess up only your local copy, and not the one on the server.

A piece of advice. **Never push/commit code that doesn't compile or doesn't run!** This will save you and your co-developers a lot of time and trouble.

Another handy feature in GIT is switching between branches. You can pull the latest changes from the server (update all branches). Then just switch to your co-developers branch, test it out and if you like, you can merge it with your own branch. Note that you may not have all branches locally, you will need to check which branches exist on the server, and then pull them to local repository. Vice versa you may have a branch only in local repository.

## 4.3 Command line usage

There are a lot of tutorials on the Internet, if you don't know what to do just Google. In this chapter we present the most common actions you will need.

**Get project from GIT server.** Note that this command will pull only the Master branch. After this you might want to create a new branch for yourself, or pull some other existing branch.  
*git clone ssh://username@10.20.207.204:2222/opt/git\_repo*

**List existing branches.** -a (all) will list local and remote branches.  
*git branch -a*

* antti	← this is local branch, and we are currently working in it
master	← this is local branch
remotes/origin/HEAD -> origin/master	← this is remote branch
remotes/origin/Jussi	← this is remote branch
remotes/origin/Tomi	← this is remote branch
remotes/origin/antti	← this is remote branch

### Create new branch.

*git branch branch\_name*

**Switch to another branch.** Assuming that, branch\_name already exists locally.

*git checkout branch\_name*

**Pull a branch from remote server.** Note that this command will first create a local branch by name *Mikes\_branch*, and then pull source from remote to local copy.

*git checkout -b Mikes\_branch origin/Mikes\_branch*

**Merge branches.** Note that merge will always merge to active local branch, the one you are working on. Also make sure that your co-developer has updated his latest changes to the remote copy. Merging will also produce a lot of conflicts, resolving them is tedious. For this reason you might want to consider using SmartGit which has a nice GUI for this purpose.

*git merge origin/Tomi* ← origin/Tomi indicates a remote branch

**Undo merge.** If you get stuck and want to undo last merge.

*git reset --hard HEAD*

**Commit changes.** -a stands for all files. Note that this will commit changes only to local branch.

*git commit -a -m "my comment"*

**Put changes to GIT server.**

*git push*

## 4.5 Misc stuff

When you commit your project, not all files are added to VCS. There is a hidden file **.gitignore** in your project folder. This file list those files that should not be put into VCS. Here's an example:

```
UWCore.app
*.pro.user
*.o
moc_*.cpp
qrc_resources.cpp
ui_*.h
qrc_resource.cpp
plugins/
```

## 4.4 SmartGit

SmartGit is a graphical user interface for git. It is available to many operating systems. It can be downloaded from here <http://www.syntevo.com/smartgit/index.html>

In order to begin, a new project can be cloned from Project->**Clone**. There you can input remote git repository URL `ssh://username@10.20.207.204:2222/opt/git_repo`. You can also open an existing project from Project->**Open Repository**, and input the path e.g. `/Users/username/Desktop/UpWind/git_repo`.

Select **Pull** and “Just fetch changes”. With this command, you will get the information about the branches on the server. You need this whenever you need to switch branch or merge your branch with someone else’s, or whenever you wish to push your own code to the main branch.

With the command **Commit** you can commit your changes to the local branch.

The command **Push**, tries to update your local branch and committed changes to the remote branch.

**Stage** prepares the files for commit and push.

From Branch->**Merge** you can merge your changes to some other branch. With Pull command you can update the branch information, but after you select merge, there will most probably be some conflicts. These conflicts will show in the main view with red dots. By selecting any of these conflicted files and pressing the right mouse button, you can select “conflict solver”. This too is not perfect, but with this the conflicts can be solved. With “Resolve” it is possible simply to select your copy or the remote copy. After all the conflicts have been solved, test your code and then push changes. The merge will take effect.

From the Branch-menu you can also **Switch** your branch.

## 5. Database

The database holds all the chart objects for one chart. The chart covers water areas west of Oulu including the the island of Hailuoto. Different objects are put as different tables in the database. Common for all tables is the location column named “wkb\_geometry”. This column holds the geo-coordinate location of the chart object in question. To be more precise it holds the information needed to draw that object to a screen. This again means that this wkb\_geometry data can be interpreted in different ways by the code depending on the object. The wkb\_geometry data can be either *point*, *line*, *polygon* or *multi-polygon* type of coordinates. For example for a *rock* the *wkb\_geometry* is in *point* format and for a *deptharea* it is in *polygon* format. Unfortunately the type information is not included in the actual database tables, you have to know the type or test it. More information about the chart objects in Appendix 1: *10.1 Database description*.

### 5.1 pgAdmin

A handy tool for viewing database. You can find it by searching the title from Software manager

for Linux machines (<http://www.pgadmin.org/download/windows.php> for windows machines).

UpWind Database information (april 2012):

IP: 10.20.207.204

PORT: 5432

USERNAME: postgre

PASSWORD: upwind

You can find the tables through following path: DB->"Databases" -> "chart57" -> "schemas" -> "public" -> "tables". The different levels mean the contents of data for different zoom levels. All the data can be found from the original table named without a "level". You can view the data within the table by right clicking the selected table and choosing "View data".

## 6. UpWind Third Party Libraries

### 6.1 GDAL - Geospatial Data Abstraction Library

Holds methods and classes for accessing chart data from database and loading it into an object to be used by a program.

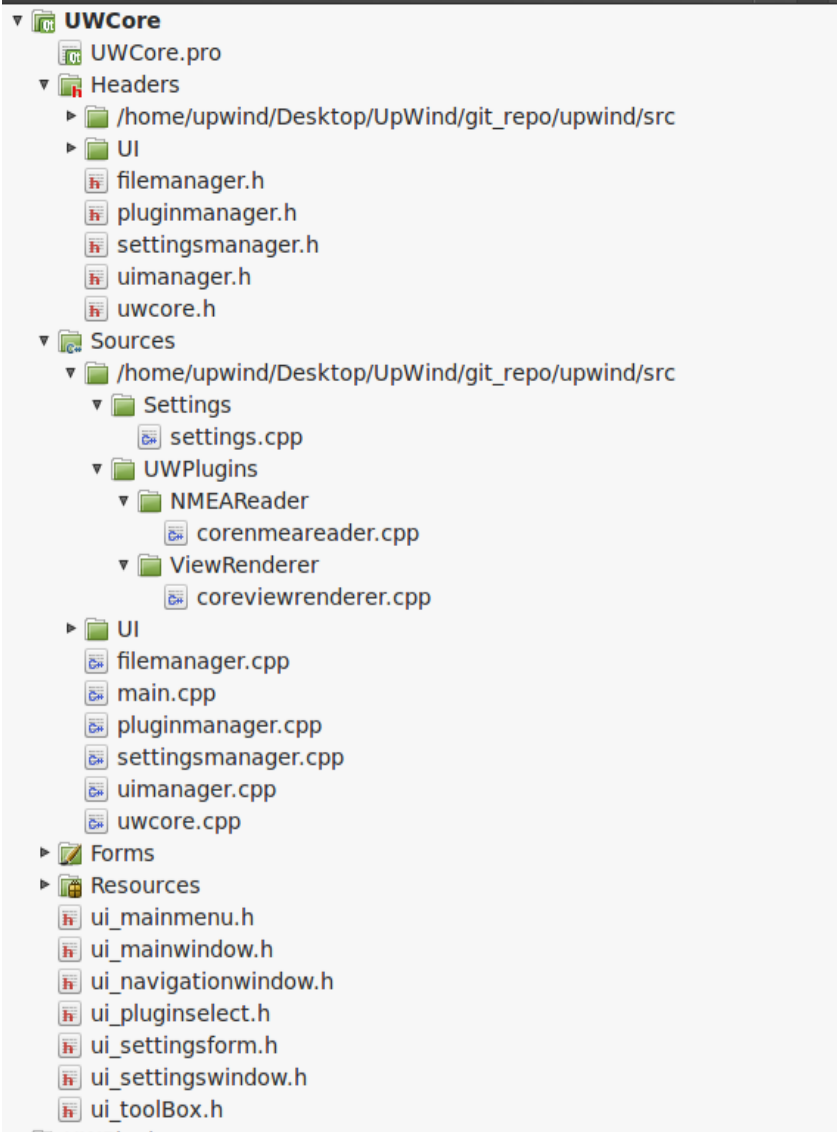
Good guide for accessing for example a column of a database table using GDAL within c++ for instance: [http://www.gdal.org/ogr/ogr\\_apitut.html](http://www.gdal.org/ogr/ogr_apitut.html)

Main page of OGR/GDAL framework: <http://www.gdal.org/ogr/>

## 7. Architecture

The whole reason for the new version of this project is to change the architecture into a plugin type architecture where functionality is being split to components which are loaded as plugins (**UWPlugins**) to the core program (**UWCore**). Each plugin should work as independent as possible from other plugins and they should communicate through the core program and not directly. This chapter describes what do the various plugins and classes mean in the current version.

## 7.1 UWCore



UWCore is the main program that is being launched on startup. The *pluginmanager* class loads the different plugins and connects them if necessary. This would be the place to make connections between plugins if needed. *Settingsmanager* and *Filemanager* classes are still mostly unimplemented. *Uwcore* class is a singleton type class which allows you to access the uwcore instance by using *uwcore::getInstance()* method. This gives you access to plugins through core if you need to access them at runtime directly. Other way is to implement a connection following the examples in *pluginmanager.cpp*. You should familiarize yourself with different ways of instantiating/referencing objects/variables in c++.

As you can see from the picture above, the *coreviewrenderer.cpp* and *.h* files are in duplicate

locations. This is due to them being included in the uwcore.pro file as shown below. This kind of linking can cause problems and you should try to achieve the same through different means. In essence the you are linking a class within a separate file to be included in the the build of this file.
































```
SOURCES += main.cpp \
    uwcore.cpp \
    uimanager.cpp \
    pluginmanager.cpp \
    filemanager.cpp \
    settingsmanager.cpp \
    UI/mainmenu.cpp \
    UI/settingswindow.cpp \
    UI/mainwindow.cpp \
    UI/settingsform.cpp \
    ../Settings/settings.cpp \
    UI/ToolBox/toolbox.cpp \
    UI/ToolBox/tab.cpp \
    UI/pluginselect.cpp \
    ../UWPlugins/ViewRenderer/coreviewrenderer.cpp \
    ../UWPlugins/NMEAReader/corenmeareader.cpp \
    UI/navigationwindow.cpp \

#UI/settingline.cpp
```

## 7.2 UWPlugins

All the implemented plugins are sub-projects of the *UWPlugins* project. Each plugin is being compiled into a separate library file. *NMEA* plugins simulate some devices such as compass and GPS. In april 2012 the plan was to override the these plugins with an external simulator which would be connected through serial port.

*QtRenderer* plugin handles everything that is drawn into a chart screen, that is the chart (chartwidget) and calculated navigation paths (*routewidget*). *PostgreChartProvider* -plugin handles the database queries using OGR framework which comes with gdal libraries. *UpWindScene* is where all the calculations for long term/short term route planning should happen. So *PostgreChartProvider* gets the selected tables from the database and stores each table as an object and puts them into a vector. *Chartwidget* inside *QtRenderer* plugin then draws the chart from on vector.

- ▼  UWPlugins
  -  UWPlugins.pro
  - ▼  ChartProviderInterface
    -  ChartProviderInterface.pro
    - ▶  Headers
    - ▶  Sources
  - ▶  Logger
  - ▶  NMEADataSimulator
  - ▶  NMEAINstruments
  - ▶  NMEALogger
  - ▶  NMEAReader
  - ▶  NMEASerialPortReader
  - ▼  PostgreChartProvider
    -  PostgreChartProvider.pro
    - ▶  Headers
    - ▶  Sources
    - ▶  Forms
    -  ui\_settingsui.h
  - ▼  QtRenderer
    -  QtRenderer.pro
    - ▶  Headers
    - ▶  Sources
  - ▼  UpWindScene
    -  UpWindScene.pro
    - ▶  Headers
    - ▶  Sources
    - ▶  Forms
    - ▶  Resources
    -  ui\_settingsui.h
  - ▶  UWPluginInterface
  - ▶  ViewRenderer

## 8. Known Issues

### 8.1 Linux

NMEASerialPortReader plugin doesn't work correctly on Linux. It seems that it misses half of data coming through serial port, and crashes the application after a few seconds.

We haven't been able to virtualize a serial port on Linux, in order to run NMEA simulator and UWCore on same machine. You can find more information on serial port virtualization at [redmine Wiki](#).

**Don't change locale settings on Linux machine.** One time I changed locale setting to Suomi on my Linux Mint machine, just to get right currency, time, and date formats. This surprisingly broke down UWCore renderer! It took me 2 days to make the connection. I still don't know why that happened. Switching back to US English fixed the problem.

### 8.2 Mac

NMEA simulator does not compile/work on mac. The author of the simulator has used some linux/windows specific files that are not available the mac users. If the code is compiled as it is, it tries to use windows files because it doesn't recognize mac. The code can be changed so that it compiles by forcing it to use unix files but it still does not run properly. (At least my code falls into some graphics error. Didn't dig too deep.)

Git version control does not work properly when used through Qt creator. Using git through Qt creator requires for example ask-pass utility. This is available in unix but has been removed from mac. I do not say that it is impossible to get the git work through Qt creator, just that it will be difficult and you may as well save some time and use Smartgit.



## 8.3 Advice

If you start getting weird compilation errors, the first place to look is \*.pro file. Most of the time definition of some header or source file is missing or there is a duplicate.

Linking external libraries is also tedious in Windows. There are many different syntaxes for that. One way to do this is:

```
win32 {  
    INCLUDEPATH += "C:\\gdal\\gdal\\include"  
    LIBS += "C:\\gdal\\gdal\\bin\\libgdal-1.dll"  
    INCLUDEPATH += "C:\\Program Files (x86)\\PostgreSQL\\8.4\\include"  
    LIBS += "C:\\Program Files (x86)\\PostgreSQL\\8.4\\bin\\libpq.dll"  
}
```

Another cause of weird errors, may be because Qt creator does not always clean the project as it should. If you rebuild your project and weird things continue, clean the project once more and check by hand that Qt has cleaned everything as it should. There may be some old object and moc files lingering that will cause your weird problems. Happened to me more than once.

In addition, when you are rebuilding remember now and then run the qmake. It will create new moc files.

## 9. Suggestions for Future Development

### 9.1 Implement XML Readers and Writers

Might be beneficial to import and implement file readers and writers for temporary data storage. In the old version temporary tables were created into the database. These temporary tables are not being currently being used in the new version. For example in the case of long term route planning, the old version used the information about the boat (height and keel depth) to parse the suitable navigation lines from all available navigation lines to speed up the computation of routes. As one can expect the user to be sailing with the same boat a lot, these parsed navigation lines were saved into separate table in database and loaded next time the software was started. Currently the corrected navigation lines are being formed but not put in to any persistent store. Writing these kind of data as xml or text files could be suitable way of storing this sort of information in temporary but persistent way. This of course depends whether the database is planned being run beside the new version of the software or not. A good place to implement such writer/reader would be *FileManager* under *UWCore*.

### 9.2 Improving efficiency

In the new version the code does a lot of *for* loops to go through vectors to find bits and pieces of information. For example, the data is stored as geometric coordinates in the database. It needs to be transformed to graphical pixel coordinates in order to be drawn to the screen. Route

calculus again needs start and end point as geometric coordinates and returns the path as geometric coordinates which again need to be transformed into graphical coordinates. So there are a lot of back and forth *for* loops changing individual values. A lot of similar things happen elsewhere in the code as well. These *for* loops could be optimized by using *break* for instance to stop the execution of for loop when all the criterias have been met. Another way to optimize is to form one vector which holds all the objects needed and each object has both geometric and graphic coordinates included by following the same logic as corrected navigation lines in previous chapter 9.1 .

All in all, currently even with dual core processor, we are talking about multiple seconds of for looping in the startup of the program and during route calculations.

### **9.3 Correcting functionality**

Some of the required functionalities have not been implemented or work incorrectly. The rotation of the view crashes the code, focusing to the boat does not work, etc.

### **9.4 Short term route planning**

This includes bringing the functions and classes from the old code that are responsible in calculating the short term route. In addition, the route should be drawn to the screen accordingly.

There is some preliminary work already started on this. There is a bit of a problem with efficiency as there seems to be a lot of for-loops needed, which could be redone better.

## **10. Appendix**

### **10.1 Database description**

Layers – description of the database file (.db)

#### **Wreck [wreck]**

Definition:

Geometry:

Attributes:

O: Name of the wreck [namewreck] – varchar2(32)

Description: name of the wreck

M: Type of the wreck [typewreck] – number(5)

Description: Type of the wreck

Code list:

0= unknown

1= over the water level

2= wreck under water level; known depth

3= wreck under water level; unknown depth

4= harmless wreck; unknown depth

5= remains of the wreck/bad bottom

O: Depth of the wreck [depthwreck] – number(4,2)

Description: Depth of the wreck under the reference level, unit [m]

M: Identification of the object [objectid] – char(17)

Description: Unique identification of the object in the system

O: Identifier of the oceanographic survey [surveyoid] – char(17)

Description: Identification of the oceanographic survey

O: Generalization [generalize] – number(5)

Description: Generalization

Code list:

0= for all types of products

1= not for product type1

2=... etc

## **Rock [rock]**

Definition:

Geometry:

Attributes:

O: Name of the rock [namerock] – varchar2(32)

Description: name of the rock

M: Type of the rock [FI\_tyrock] – number(5)

Description: Type of the rock

Code list:

0= unknown

1= rock under the water level

2= rock over water level

A sunken or aground ship or part of it

Point

A rock, set of rocks or an islet in water

## Point

3= rock in water's edge

4= set of rocks under water level

5= set of rocks over water level

6= set of rocks in water's edge

7= an islet (little island)

O: Depth of the rock [depthrock] – number(4,1)

Description: Depth of the top of the rock, unit [m]

O: Reliability of the rock [FI\_relrock] – number(5)

Description: Reliability of the rock

Code list:

0= unknown

1= reliable

2= approximate, uncertain

M: Identification of the object [objected] – char(17)

Description: Unique description of the objec in the system

## Navigation line [navigline]

Definition:

Geometry:

Attributes:

M: Type of the navigation line [typenavlne] – number(5)

Description: type of the navigation line

Code list:

0= unknown

1= official

2= directive

M: Category of navigation [categornav] – number(5)

Description: Category of navigation

Code list:

0= unknown

1= navigation line with known depth

2= navigation line with unknown depth

3= old non-surveyed line used for log transportation

4= navigation line for boats

5= ancient navigation line

6= lighted navigation line with known depth in general chart

7= base navigation lane for boats

8= navigation line for boats in coastal charts

O: Depth of navigation line [navlnedep] – number(3,1)

Description: Confirmed minimum depth of the navigation line.

O: Bearing [bearing] – number(5,2)

Description: Geodetic true bearing of the line.

A recommended route dedicated for safe navigation line

O: Radius [FI\_radius] – number(5)

Description: Curve radius of the navigation line.

O: Primary fairway [primfairwy] – number(5)

Description: Primary fairway for which the navigation line has been confirmed.

M: Identification of the object [objected] – char(17)

Description: Unique identification of the object in the system.

### **Daymark [daymark]**

Definition:

Geometry:

Attributes:

M: Identification of the safety mark.

Description: name of the rock

M: Number of the daymark [numdaymark] – number(5)

Description: Ordinal number of the daymark.

O: Height of the daymark's structure [verleng] – number(5,2)

Description: Height of the mark from the surface of water or land, unit [m]

O: Height of the daymark [hgtdaymark] – number(5,2)

Description: Height of the mark from the reference surface, unit [m]

O: Shape of the daymark [FI\_shpmk] – number(5)

Description: Main shape of the mark

Code list:

0= unknown

1= triangular shape

2= shuttle shape

3= square shape

4= circular shape

5= conic shape

6= cylindrical shape

7= spherical shape

8= pole shape

9= barrel shape

10= rectangular shape

11= parallelogram shape

12= pipe shape

13= not specified

14= square on its edge

15= light cabin

16= lighthouse pole  
17= building  
18= mast  
19= pole

Properties of the safety mark used for its identification.  
non-spatial

20= pipe

M: Color number 1 of the daymark [co1daymark] – number(5)

Description: First or main color of the daymark.

Code list:

0= unknown  
1= green  
2= black  
3= red  
4= yellow  
5= white  
6= orange  
7= blue  
8= grey  
9= brown

O: Color number 2 of the daymark [co2daymark] – number(5)

Description: Second or secondary color of the daymark.

Code list:

0= unknown  
1= green  
2= black  
3= red  
4= yellow  
5= white  
6= orange  
7= blue  
8= grey  
9= brown

M: Color combination of the daymark [ccmdaymark] – number(5)

Description: Color combination of the daymark.

Code list:

0= unknown  
1= color 1  
2= color 1/color 2  
3= color 2/color 1  
4= color 1/color 2/color 1  
5= color 2/color 1/color 2

M: Color pattern of the daymark [cptdaymar] – number(5)

Description: Color pattern of the daymark.

Code list:

0= unknown  
1= horizontal cones from top to down  
2= vertical stripes  
3= diagonal stripes  
4= checked  
5= mono color  
6= triangle

7= upper part/lower part  
8= rectangle/triangle  
9= left/right  
10= surround/interior

O: Surface material of the daymark [FI\_surfmat] – number(5)

Description: Surface material of the daymark.

Code list:

1= waved fiberglass plate  
2= aluminium plate with reflector  
3= tree surface  
4= stone surface  
5= concrete surface  
6= steel surface  
7= plywood surface  
8= tile surface  
9= minerit surface  
10= fiberglass surface

O: Mark identification [FI\_dmdsgn] – varchar2(2)

Description: Identification character in the plate shaped mark

### **Coastline [coastline]**

Definition:

Geometry:

Attributes:

M: Type of the coastline [typecoalne] – number(5)

Description: Type of beach

Code list:

0= unknown  
1= surveyed coast  
2= unprecise coastline  
3= steep coast, cliff wall  
4= sand beach  
5= rocky soil beach

O: Height of coastline [elevcoalne] – number(6,2))

Description: Height of coastline, unit [m]

M: Identification of the object [objected] – char(17)

Description: Unique identification of the object in the system.

O: Vertical datum [verdat] – varchar2(32)

Description: Vertical datum

### **Transmission line [transmline]**

Definition:

Geometry:

Natural or constructed border line of normal water and land.  
line

Air-line used to transmission of electricity or electronic information  
line

Attributes:

M: Type of the transmission line [ypetrnlin] – number(5)

Description: Type of the transmission line

Code list:

0= unknown

1= power transmission line

2= electricity line

3= data cable

4= cable railway

O: Underpass height of the transmission line [vercsa] – number(4,2)

Description: Safe underpass height of the transmission line, unit [m]

O: Owner [FI\_owner] – varchar2(32)

Description: Owner (only for Finnish objects)

M: Identification of the object [objected] – char(17)

Description: Unique identification of the object in the system.

### **Bridge [bridge]**

Definition:

Geometry:

Attributes:

O: Name of the bridge [namebridge] – varchar2(32)

Description: Name of the bridge

M: Type of the bridge [typebridge] – number(5)

Description: Type of bridge

Code list:

1= other than those defined in this list

2= fixed bridge

3= opening bridge

4= rotating bridge

5= flap bridge



7= pontoon bridge  
8= drawbridge  
9= conveyor bridge  
10= pedestrian bridge

O: Horizontal clearance of the bridge [horclr] – number(4,2)  
Description: Width, horizontal clearance of the bridge, unit [m]

O: Height of the bridge, closed [verccl] – number(4,2)  
Description: Free height of the closed or fixed bridge, unit [m]

O: Height of the bridge, open [vercop] – number(4,2)  
Description: Free height of the opened bridge, unit [m]

O: Status of the construct [statuscnst] – number(5)

A construct meant for run over a waterway or other obstacle.  
line, area

Description: Status of the construct.

Code list:

1= permanent, ready  
2= under construction  
3= ruin

O: Owner [FI\_owner] – varchar2(32)  
Description: Owner (only for Finnish objects)

M: Identification of the object [objected] – char(17)  
Description: Unique identification of the object in system.

O: International name [int\_name] – varchar2(32)  
Description International name

O: Road type of the bridge [typeroad] – number(5)  
Description: Type of the road

Code list:

0= unknown  
1= borderline of street or quarter  
2= motorway  
3= trunk road  
4= local road  
5= trail, cart-track  
6= not defined  
7= street  
8= traffic area  
9= railway

### **Depth area [deptharea]**

Definition:

Geometry:

Attributes:

M: Type of the sea area [typedep] – number(5)

Description: Type of sea area

Code list:

0= unknown

1= sea area

2= lake area

3= area not surveyed completely

O: Minimum depth [mindepth] – number(5,1)

Description: Smallest depth of the area.

O: Maximum depth [maxdepth] – number(5,1)

Description: Greatest depth of the area.

M: Identification of the object [objectid] – char(17)

Description: Unique identification of object in the system.

An area defining certain depth area of sea.  
line, area

O: Height of lake [hghtlake] – number(5,1)

Description: Height of the lake from sea level

### **Depth contour [depthcont]**

Definition:

Geometry:

Attributes:

O: Depth value of the depth contour [valdco] – number(5,1)

Description: Depth value of the depth curve.

M: Definition method of the depth contour [methcntrng] – number(5)

Description: Method used in defining the depth curve

Code list:

0= unknown

1= graphical

2= computational method (grid)

3= computational method (triangle model)

4= composed method

5= based on dragging

M: Criteria for defining of the depth contour [contcrit] – number(5)

Description: Criteria for defining the depth contour

Code list:

0= unknown

1= other than those mentioned below

2= confirmed measurement

3= average depth  
4= defined from maximum depths

O: Type of source material [Fl\_tysrcda] – number(5)

Description: Type of source material

Code list:

0= unknown  
1= other than those mentioned below  
2= mixed  
3= sonar dragging  
4= sonar  
5= manual sounding  
6= multi beam sounding  
7= sonar sounding  
8= full coverage sounding  
9= laser sounding

M: Quality of depth contour [qualdepcnt] – number(5)

Description Quality of depth contour

Code list:

0= unknown  
1= reliable

Equilibrium/ isochrone defined based on the survey data.  
line

2= approximate  
3= floating  
4= closing depth area  
5= approximate and floating

M: Identification of object [objected] – char(17)

Description: Unique identification of the object in the system.

O: True depth [truedpth] – number(11,8)

Description: True depth.

M: Identification of the object [objected] – char(17)

Description: Unique identification of object in the system.

O: Height of lake [hghtlake] – number(5,1)

Description: Height of the lake from sea level

Leading line [leadingline]

Definition:

the sea

Geometry:

Attributes:

M: Type of the leading line [typeleadline] – number(5)

Description: type of the leading line

Code list:

0= unknown

1= course line of safe navigation

2= course line

3= connecting line

4= non linked leading line

M: Category of navigation [categornav] – number(5)

Description: Category of navigation

Code list:

0= unknown

1= navigation line with known depth

2= navigation line with unknown depth

3= old non-surveyed line used for log transportation

4= navigation line for boats

5= ancient navigation line

6= lighted navigation line with known depth in general chart

7= base navigation lane for boats

8= navigation line for boats in coastal charts

O: Bearing [bearing] – number(5,2)

Description: Geodetic true bearing of the line.

M: Identification of the object [objectid] – char(17)

Description: Unique identification of the object in the system.

A line, defined by two stationary marks. The marks are visible from

line

O: Identification of known distance [mdistoid] – char(17)

Description: Identification of known distance.

O: Identification navigation line [navlineoid] – char(17)

Description: Identification of navigation line.

### **Navigation aid [navaid]**

Definition:

Geometry:

Attributes

O Type of navigation aid [FI\_typenav] – number(5)

Description Type of navigation aid.

Code list:

0= unknown

1= lighthouse

2= sector lighthouse

3= line mark

4= course light

5= assisting light  
6= other mark  
7= perimeter mark  
8= radar mark  
9= buoy  
10= spar buoy  
11= marker lighthouse  
13= "kummeli" (heap of stones)

M: Navigation type of the navigation aid [ntypnavaid] – number(5)

Description: Navigation type of the navigation aid.

Code list:

0= unknown  
1= left side mark  
2= right side mark  
3= north mark  
4= south mark  
5= west mark  
6= east mark  
7= rock mark  
8= safe water mark  
9= special mark  
10= foremost/lower mark  
11= middle mark  
12= rearward/upper mark  
13= border mark  
99= not applicable

M: Category of navigation [categornav] – number(5)

Description: Category of navigation.

A stationary or floating apparatus constructed for safe navigation.  
point

Code list:

0= unknown  
1= shows the navigation line with known depth  
2= shows the navigation line with unknown depth  
3= shows an old log transportation line which has not been dragged  
4= shows a navigation line for boats  
5= shows an ancient navigation line  
8= shows a coastal navigation line for boats

M: Name of the navigation aid in Finnish [namnavaid1] – varchar2(64)

Description: Name of the navigation aid in Finnish.

M: Name of the navigation aid in Swedish [namnavaid2] – varchar2(64)

Description: Name of the navigation aid in Swedish.

O: Number of the navigation aid [numnavaid] – varchar2(8)

Description: Unique identification of the navigation aid in the system.

O: International number of the navigation aid [inumnavaid] – varchar2(8)

Description: International number of the navigation aid.

M: Status of the navigation aid [statnavaid] – number(5)

Description: Official status of the navigation aid.

Code list:

0= unknown

1= planned navigation aid

2= confirmed navigation aid

3= deleted navigation aid

O: Structural information of the navigation aid [FI\_strcnv] – number(5)

Description: Structural information of the navigation aid.

Code list:

1= buoy mark

2= ice buoy

4= mark buoy

5= tall mark

7= plate shaped mark

8= helicopter plateau

9= radio mast

10= water tower

11= chimney

12= radar tower

13= church tower

14= tall buoy

15= perimeter mark

16= compass checking point

17= border mark

18= border line mark

19= perimeter mark of a canal

O: Fasad light of the navigation aid [FI\_navflht] – number(5)

Description: Defines whether the navigation aid has a fasad light.

Code list:

0= unknown

1= there is a fasad light

2= there is no fasad light

M: Operational status of the navigation aid [ostanavaid] – number(5)

Description: Operational status of the navigation aid

Code list:

0= unknown

1= continuous

2= operates when needed

3= removed from operation

4= restricted operation time

5= temporary operation

M: Top mark of the navigation aid [topmark] – number(5)

Description: Defines whether the navigation aid is equipped with a top mark.

Code list:

0= unknown

1= equipped with top mark

2= without top mark

O: Accuracy [FI\_desaccu] – number(5)

Description Accuracy class of the navigation aid.

Code list:

0= unknown

1= geodetically accurate (0,5 m)

2= indirectly centered or reference transformed (0,5-2 m)

3= mark perhaps moved (2-10 m)

4= mark position in the direction of the navigation line (2-50 m)

5= measured from an air photograph (2-5 m)

6= measured from a chart of 1:20000

7= measured from a chart of 1:50000

8= measured from a chart of 1:10000

9= GPS-positioning (1 - 3 dm)

10= DGPS-positioning (1 - 3 m)

11= measured from a chart of 1:30000

12= measured from a chart of 1:40000

13= measured from a chart of 1:50400

51= geodetic accuracy 1 (0,12 m / 0,3 m)

52= geodetic accuracy 2 (0,2 m / 0,5 m)

53= geodetic accuracy 3 (0,4 m / 1 m)

54= geodetic accuracy 4 (0,8 m / 2 m)

55= geodetic accuracy 5 (2 m / 5 m)

56= theoretic accuracy 1 (planned position)

57= theoretic accuracy 2 (implemented accuracy)

58= theoretic accuracy 3 (object moved)

59= graphical accuracy 1 ( scale>1:1000)

60= graphical accuracy 2 (1:1000=>scale>1:10000)

61= graphical accuracy 3 (1:10000=>scale>1:50000)

62= graphical accuracy 4 (scale = < 1:50000)

63= method accuracy

O: Coordinate verification [FI\_crdvrf] – number(5)

Description: Category of verification made for the position.

Code list:

1= quality class 1

2= quality class 2

3= quality class 3

4= coordinates verified by the Navi-criteria

5= coordinates not verified

O: Method of positioning (VATU) [FI\_vatupos] – number(5)

Description Method of positioning (VATU)

Code list:

0= unknown

1= measured geodetically 1  
2= measured from chart 1:20000  
3= measured from chart 1:50000  
4= radio positioning Syledis  
5= radio positioning Trisponder  
6= optical positioning with sextant  
7= satellite positioning  
8= measured from chart 1:10000  
9= photogrammetry (air photo)  
10= measured from chart 1:30000  
11= measured from chart 1:40000  
12= measured from chart 1:50400  
50= \*new codes from this on\*  
51= measured geodetically 2  
52= GPS-measurement  
53= DGPS-positioning  
54= Decca-positioning  
55= Syledis-positioning  
56= Trisponder-positioning  
57= Autotracker (laser)  
58= tachymeter positioning  
59= theodolite based positioning  
60= optical positioning with sextant 2  
61= graphically from chart  
62= air photo  
63= theoretic positioning  
64= other

(The rest are not yet translated. Perhaps they are not important for our purpose...)

O: Merenkulkupiiri [FI\_mardist] - number(5)

Kuvaus: Merenkulkupiiri (suomalainen luokitus).

Koodilista:

0= tuntematon  
1= Suomenlahti  
2= Saaristomeri  
3= Pohjanlahti  
Järvi-Suomi

O: Tutkaheijastimen tyyppi [typeradrfl] - number(5)

Kuvaus: Tutkaheijastimen tyyppi.

Koodilista:

0= tuntematon  
1= 5-soppiheijastin  
2= Speckter-heijastin  
3= 6-soppiheijastin  
4= AGA-heijastin  
5= 3-soppiheijastin



6= 4-soppiheijastin

M: Kohteen tunniste [objectid] - char(17)

Kuvaus: Kohteen yksilöllinen tunniste järjestelmässä.

O: Liik.jakovyöhtunniste [trafsepoid] - char(17)

Kuvaus: Liikennejakovyöhykkeen tunniste.

O: Karttasymboli [chartsymb] - varchar2(2)

Kuvaus: Karttasymbolin attribuutti

O: VATU muutospäivä [FI\_vatuchd] - varchar2(12)

Kuvaus: VATU muutospäivä.

O: Kansainvälinen nimi [int\_name] - varchar2(32)

Kuvaus: Kansainvälinen nimi.

O: Alkuperäinen latitudiarvo [FI\_origla] - number(17,12)

Kuvaus: Alkuperäinen latitudiarvo.

O: Alkuperäinen longitudiarvo [FI\_origlo] - number(17,12)

Kuvaus: Alkuperäinen longitudiarvo

M: Turvalaitteen maa [country] - number(5)

Kuvaus: Turvalaitteen maa

Koodilista:

0= Tuntematon

1= Suomi

2= Ruotsi

3= Venäjä

4= Viro

5=

Latvia

6= Liettua

7= Puola

8= Saksa

9= Tanska

10= Norja

11=

Muu

O: Yleistys [generalize] - number(5)

Kuvaus: Yleistys.

Koodilista:

0= viedään kaikille tuotetyypeille

1= ei viedä tuotetyypille 1

2= ei viedä

tuotetyypille 2

3= ei viedä tuotetyypille 3

4= ei viedä tuotetyypille 4

5= ei

viedä tuotetyypille 5