
DashT - OpenCPN Overlay Plug-in

Release v2.0.3

Petri Mäkijärvi

May 12, 2024

CONTENTS:

1	Introduction	1
1.1	Disclaimer	1
2	Installation	3
2.1	Mandatory requirements	3
2.2	OpenCPN Plug-In Manager	3
2.2.1	Reporting issues	4
2.3	DashT Windows installer	4
2.3.1	Inspection (Win)	4
2.3.2	Install (Win)	4
2.3.3	Uninstall (Win)	7
2.3.4	Troubleshooting (Win)	8
2.4	DashT Linux (deb) package	9
2.4.1	Inspection (Linux)	10
2.4.2	Install (Linux)	10
2.4.3	Uninstall (Linux)	11
2.4.4	Troubleshooting (Linux)	12
2.5	Optional platforms	13
2.6	Installed helpers	13
2.6.1	Windows scripts	13
2.6.2	Linux scripts	13
2.7	wxWebView and GTK2	14
3	Getting Started	15
4	NMEA-0183 Data	19
4.1	Data source priority	19
5	Dashboard	21
5.1	Configuration	21
5.2	Air temperature	22
5.3	AWA/TWA Dial	23
5.4	AWA	23
5.5	AWA/AWS Dial	23
5.6	AWS Dial	24
5.7	AWS	24
5.8	Baro History	24
5.9	Barometer	24
5.10	Baro Dial	25
5.11	COG	25

5.12	Cursor	25
5.13	Depth Graph	25
5.14	Depth	26
5.15	From Ownship	26
5.16	GPS Clock	26
5.17	GPS Compass	26
5.18	Satellites in view	27
5.19	Satellite Status	27
5.20	Heel	27
5.21	CPU clock	27
5.22	GPS local time	28
5.23	(local) Sunrise/Sunset	28
5.24	Magnetic COG	28
5.25	Magnetic HDG	28
5.26	Moon Phase	29
5.27	Pitch	29
5.28	Position	29
5.29	Rudder angle graph	29
5.30	Rudder angle	30
5.31	SOG	30
5.32	SOG Speedometer	30
5.33	STW	30
5.34	Sum Log	31
5.35	Sunrise and Sunset	31
5.36	Trip Log	31
5.37	True Compass	31
5.38	True Heading	32
5.39	TWA	32
5.40	TWS	32
5.41	TWA/TWS Dial	32
5.42	TWD/TWS Dial	33
5.43	VMG (route)	33
5.44	Water temperature	33
5.45	Wind History	34
6	Tactics	35
6.1	Introduction	35
6.2	What Tactics can do?	35
6.3	Prerequisites	38
6.3.1	CMG/VMG abbreviations	39
6.4	Without Polar	39
6.4.1	Calculate true wind data	39
6.4.2	SOG instead of STW	40
6.4.3	Heel corrections	40
6.4.4	Calculate Leeway	40
6.4.5	Calculate the surface current	42
6.4.6	Current arrow on chart	42
6.4.7	Boat laylines	43
6.4.8	Wind barbs	44
6.4.9	Current Direction and Speed	44
6.4.10	Leeway	44
6.4.11	Average Wind Instrument	45
6.4.12	TWA to Waypoint	46
6.4.13	Odograph	47

6.5	With Polar	48
6.5.1	Display polar on chart	50
6.5.2	Performance data	51
6.5.3	Vector graph	51
6.5.4	Polar speed	52
6.5.5	Actual VMG	52
6.5.6	Target VMG Angle	53
6.5.7	Target VMG	53
6.5.8	Actual CMG	54
6.5.9	Target CMG Angle	54
6.5.10	Target CMG	55
6.5.11	Polar Performance	55
6.5.12	Bearing compass	56
6.5.13	Polar Compass	58
6.5.14	Temporary waypoint and Target-TWA laylines	60
6.5.15	NKE-style Perf.Records	61
6.5.16	Data Export	64
6.6	Tactics terminology	66
6.7	Align your compass	66
7	Signal K In	69
7.1	Signal K server node	70
7.2	NMEA-0183 data type	70
7.3	NMEA-2000 data type	71
7.4	Other data types	71
7.5	Signal K data interchange	71
7.5.1	How it works	72
7.6	Configuration	73
7.7	Troubleshooting	73
7.7.1	No connection	74
7.7.2	No data	74
7.7.3	HALT state	74
7.7.4	Confusing timestamps	75
8	Engine/Energy	77
8.1	Introduction	77
8.2	Installation/Launch	78
8.2.1	Use Node.js	78
8.2.2	Engine Gage script	78
8.3	Configuration	79
8.3.1	Adding new dials	79
8.3.2	Subscribe to data	80
8.3.3	Search again	81
8.3.4	Change Display	81
8.3.5	Change Subscription	82
8.4	Customization	82
8.4.1	File location	82
8.4.2	Language file	84
8.5	Rich infras	84
8.5.1	pub Linux	84
8.5.2	pub Windows	85
8.6	Troubleshooting	86
8.6.1	Data is bad	86
8.6.2	After config, all dead	86

8.6.3	Data “X” not available	86
9	InfluxDB / Grafana	89
9.1	Introduction	89
9.2	Docker InfluxDB	90
9.2.1	Windows scripts	90
9.2.2	Linux scripts	91
9.3	Set up InfluxDB	91
9.3.1	InfluxDB storage	92
9.3.2	InfluxDB backup	92
9.4	Grafana	93
9.5	HTML/JS updates	98
9.6	Under the hood	98
9.6.1	Docker Dashboard	98
9.6.2	Container creation	99
9.6.3	nginx	100
9.6.4	InfluxDB read-back	102
9.6.5	Troubleshooting	102
10	InfluxDB Out	105
10.1	Line Protocol File	105
10.1.1	File content	106
10.1.2	File location	107
10.1.3	Backup copies	107
10.1.4	Importing to InfluxDB	107
10.1.5	Configuration file management	107
10.2	HTTP Streamout	107
10.3	Conversions	108
10.3.1	Stream from VDR	108
10.4	Debugging	109
10.4.1	Level 4	109
10.4.2	Level 5	109
11	Line Chart	111
11.1	Introduction	111
11.2	Setting up	114
11.3	Numerical values	117
12	DashT Race Start	119
12.1	Introduction	119
12.2	Define Line	120
12.2.1	Wind shift direction	121
12.2.2	Pre-determined markers	121
12.3	Approaching Start Zone	122
12.3.1	Open application	123
12.3.2	Drop marks	125
12.4	Start Zone	125
12.5	The Heat Is On!	127
12.6	The Race Is On!	129
13	DashT Race Mark	131
13.1	Introduction	131
13.2	Define Race Course	132
13.2.1	Open application	134
13.3	Start The Course	135

13.4	Windward leg	135
13.5	Reaching leg	137
13.6	Leeward leg	137
13.7	Final leg	137
13.8	Troubleshooting	138
13.8.1	Missing data	138
13.8.2	Route messed up	138
13.8.3	Application No Show	138
14	Tweaks	139
14.1	OpenCPN	139
14.1.1	PlugIns	139
14.1.2	AUI	140
14.2	DashT	140
14.2.1	Dashboard	140
14.2.2	Fonts	141
14.2.3	Instrument fonts	142
14.2.4	Colors	143
14.2.5	Extra dial tweaks	143
14.2.6	Dashboard index	143
14.3	DashT/Tactics	145
14.3.1	Performance	147
14.3.2	AverageWind	149
14.3.3	BaroHistory	149
14.3.4	PolarPerformance	149
14.3.5	Windhistory	149
14.3.6	Odograph	149
14.3.7	Streamout	150
14.3.8	StreaminSk	150
14.4	DashT/WebView	150
14.4.1	Race	151
15	Security	153
15.1	Data flow	153
15.2	Security Policy	154
15.3	Security Implementation	155
16	Privacy	157
16.1	Privacy policy	157
16.2	Position	157
16.3	Personal information	157
16.4	Local storage (plug-in)	157
16.5	Local storage (browser)	158
16.6	Cookies (browser)	158
16.7	Calling back home	158
17	Indices and tables	159
	Index	161

INTRODUCTION

Welcome to *DashT*, an advanced overlay plug-in for the popular [OpenCPN](#) chart plotter.

In its simplest form, *DashT* can be used as the OpenCPN's built-in plug-in Dashboard. In this case one can stop reading this document now since the user interface remains quite similar for Dashboard instruments - maybe some *Tweaks*?

But stopping here would be a pity, since underneath that familiar first impression given by *DashT* hides a real powerhouse, providing instruments and functions meeting the requirements of wide variety of users, ranging from cruisers to regatta and offshore racers. Motor boat owners have not been forgotten.

HISTORY: Not a “yet-another-fork of Dashboard”, *dashboard_tactics_pi* v1.5.111 had a goal to combine the built-in *dashboard_pi* and the popular *tactics_pi* (a fork of the former) by merging them while retaining all of their functions unchanged but making it available to share Tactics algorithms plug-in wide with timestamped data. v1.5.111 was a forerunner in *OpenCPN* world by embracing the open marine data format [Signal K](#) which is now recognized by *OpenCPN*, v5.2 onwards.

DashT builds on this heritage, but is expanding further to meet the modern, interconnected ecosystem which is today present both on the boat and once back on *terra firma*.

1.1 Disclaimer

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

INSTALLATION

Go and grab an installer package for your operating system and distribution from this [distribution's Latest Release page](#), expand the *Assets* tab below the description if you do not see any.

NOTE: The documentation you are reading is generated from the distribution. If you are adventuring and taking a beta version, make sure that you are reading also its documentation. See the version information on the page (depends of the platform where it can be found, sorry), available also in the page URL.

The platforms and operating system supported:

- Windows 10 (.exe installer - fully tested platform)
- MacOS (.pkg - **not tested as platform**, helper scripts missing)
- Ubuntu 20.04LTS Focal Fossa and distributions similar to it (.deb - fully tested platform)

2.1 Mandatory requirements

- OpenCPN v5.2 or greater
- Linux: wxWidgets run-time support with wxWebView run-time library (GTK3, not available on GTK2 based Linux distributions by default)

2.2 OpenCPN Plug-In Manager

The plug-in manager of *OpenCPN* is a great tool to unify the plug-in installation for the best user experience. The tool's catalogue and the plug-ins are pulled from a centralized data retrieval system, proper to OpenCPN.org. Whenever you can, use the OpenCPN Plug-In Manager.

Please be reminded that the OpenCPN Plug-In Manager is not installing the software from the repository from which this document is generated. Therefore you will need to pay attention which version you get and compare that to this document's version.

The features of *DashT* described in this document are those obtained only - in a guaranteed manner - using an installer provided with a same version number and from the same repository this document has been generated from. You can reach that repository using the version navigation buttons visible somewhere around this page - depending of the format and the platform on which you are reading it.

2.2.1 Reporting issues

If you experience issues with the version of *DashT* installed by the OpenCPN Plug-In Manager from the OpenCPN.org repository, please report about your issues in corresponding [DashT discussion thread](#) - **not** into the repository where this document has been generated from. Check carefully the version of *DashT* which has been installed and the source of installation before reporting, no matter where you do report!

Before reporting for this particular software, see the troubleshooting sections for the corresponding software module or instrument. If you think you have found an issue, please describe it in the source software's repository [here](#) - make also a last attempt to search for your problem's keywords from the issue list, thank you.

Should you decide to open an issue to author's attention, please collect all information requested on the issue template *before* pressing that submit-button, thus avoiding time wasting messaging to collect the requested background information from you, one by one. All information requested is necessary, not to mention the screenshots alone! This is **FOSS** and the community who has contributed to this project is proud to bring this software to you, free of charge. But the level of support can be only proportional to the **free** word of the definition - please contribute by making a considerable effort to explain your problem clearly.

2.3 DashT Windows installer

NOTE: Unlike the *OpenCPN Plug-In Manager*, the *.exe installer creates a Windows uninstaller. This is required to uninstall also the *helper scripts*. If this is an issue, do not execute the installer but use the *OpenCPN Plug-In Manager*. Of course, you may not necessarily get the same software version than this document is describing and the connections to third party systems managed by those helper scripts will not be available.

2.3.1 Inspection (Win)

The installer is an *.exe installer which you have downloaded from the repository's *Releases* (see [Installation](#)). It can be inspected (if not yet done during the download) by scanning it with your Windows 10 Microsoft Defender antivirus program or other such product. More antivirus programs can be found and used to scan the file in one go at [virustotal.com](#).

Finally, to inspect what exactly is contained inside the *.exe installer, one can use, for example 7z archive manager.

You are also invited to read the [privacy statement](#) and [security policy](#) in this documentation. [DashT security page on GitHub](#) provides the latest security status and provides links to open issues.

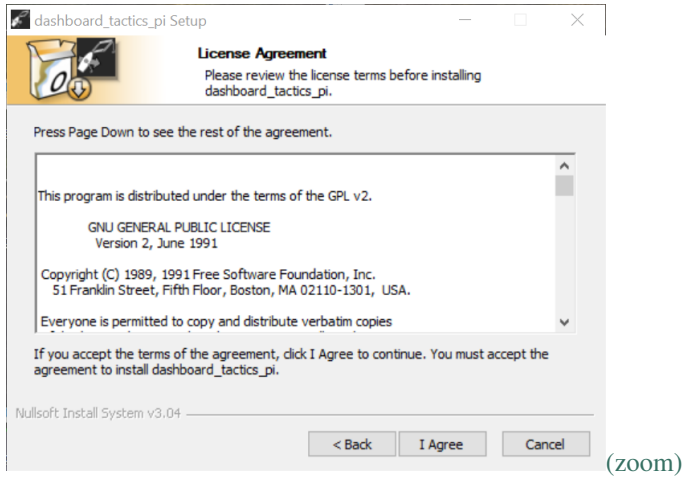
2.3.2 Install (Win)

Once you launch the *.exe installer you need to get through the Windows security messages. They may vary and are not explained here. Once you get through, the [Nullsoft Scriptable Install System](#) - another great free and open source program - will be launched to install *DashT* and its *helper scripts*.

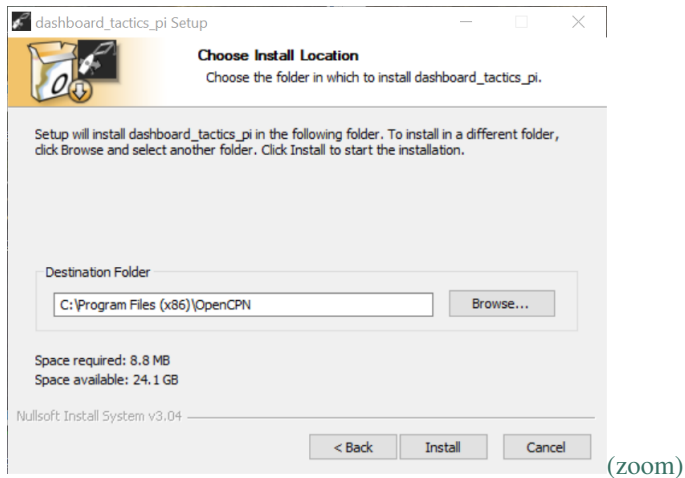
UPDATING your *DashT*: If you are using InfluxDB / Docker you must make sure all Docker services are stopped - it may well be that you are running them with automatic start (if you have shut down or rebooted your system with them running, the services are persistent). In order to **keep your database and other Docker settings safe** and, at the same time make sure that the installation does not stop because of a locked file system please rename the following directory: \Users\Public\DashT. This is where all Docker-linked files are located, including your database. Keep it safe! Do not start the installation/upgrade if you have issues with this point, the installation would also fail in the end.



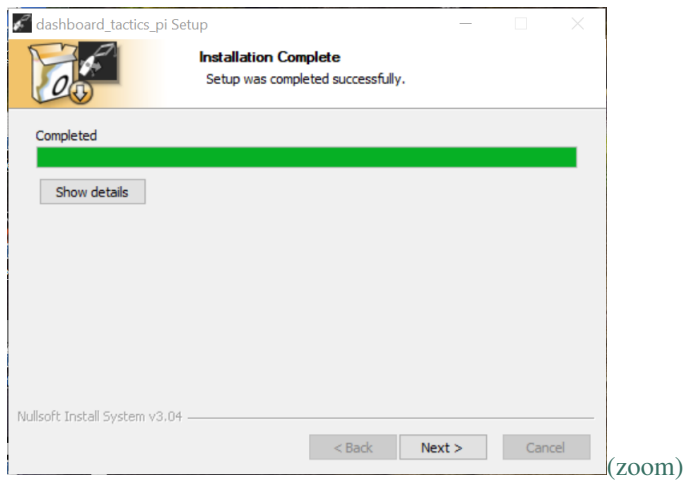
NOTE: The installer does **not** install *OpenCPN*, it *requires OpenCPN*! It is an installer which installs an *OpenCPN* plug-in. Get yourself one from OpenCPN.org if you do not have this great open source chart plotter yet.



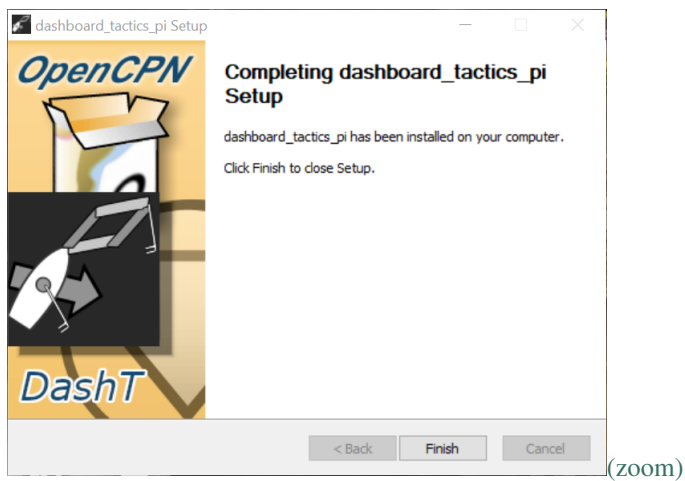
The license of the installed software is naturally the same as is it with the *OpenCPN*, *GPL v2* - i.e. free and open source. There are some dynamic components based on web technologies and third party open source software not using this particular license but they have adapted, and *DashT* with them the even more permissive *MIT license*. If you use the *DashT* units which are under MIT-license, you will be asked to accept that license, but only if you actually use those instruments, and only once per instrument.



There is most likely no reason to change the default *OpenCPN* installation location. If you do, probably you know better and do not need to read further.



Installation is pretty fast, but you have still time after the installation to review, with *Show details* button which files were installed and where, before terminating the installation with **Next >**.



2.3.3 Uninstall (Win)

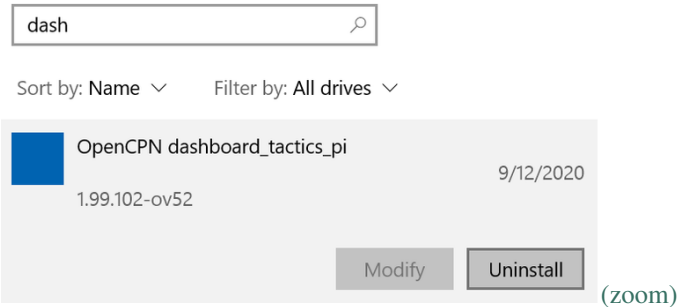
NOTE: The Windows uninstaller is registered in Windows under the name *dashboard_tactics_pi* which is the catalogue name for it in *OpenCPN*.

Apps & features

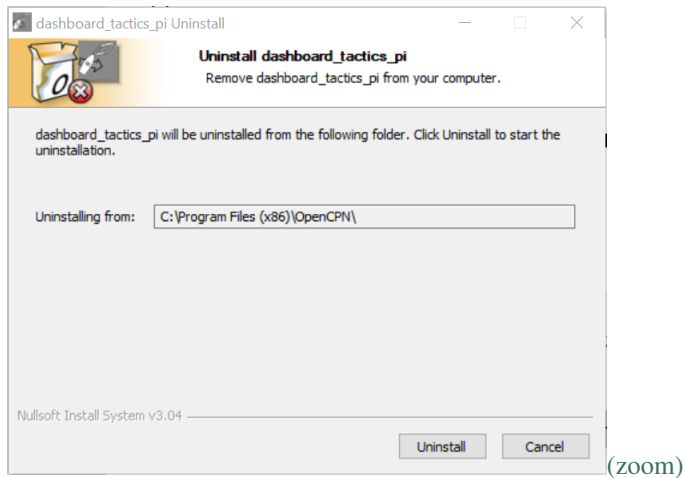
Optional features

App execution aliases

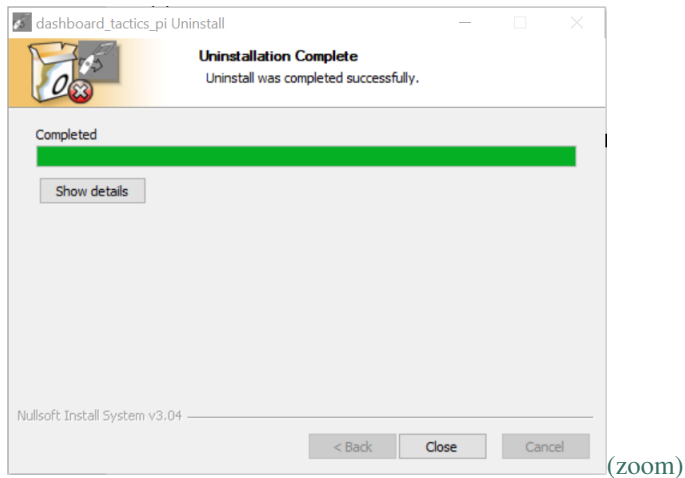
Search, sort, and filter by drive. If you would like to uninstall or move an app, select it from the list.



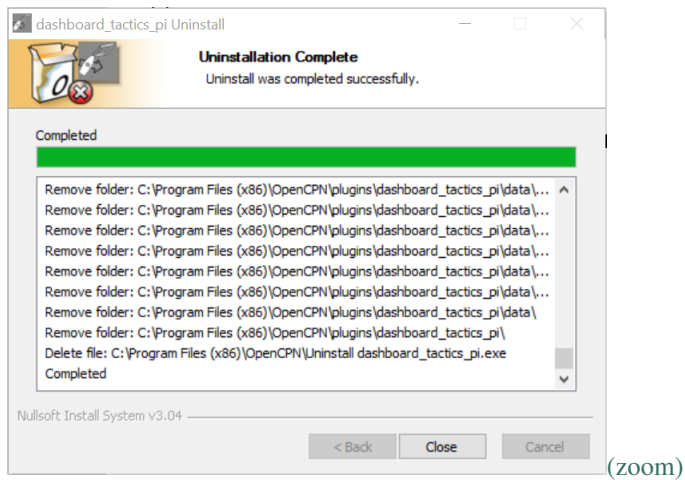
Search for the uninstaller and launch it using Windows Settings -> Apps -> Apps & features.



It is very unlikely that you need to change anything in the above first uninstall dialog which appears.



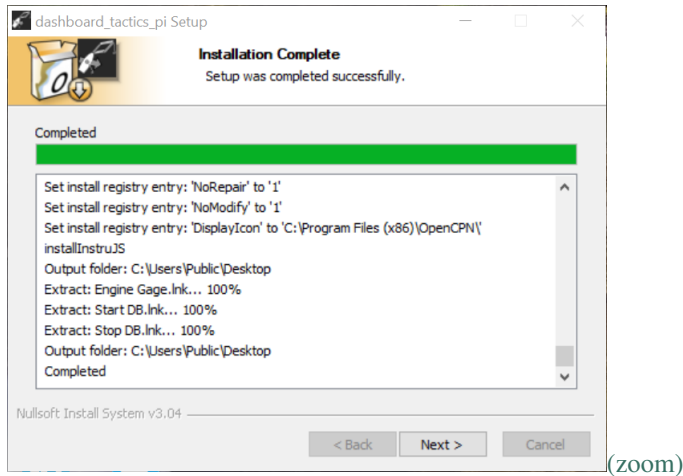
Uninstallation should be really fast. This is not a big program, just a plug-in.



You can see which files were actually uninstalled and from where by pressing **Show details** button before pressing **Close**.

2.3.4 Troubleshooting (Win)

Installation is pretty fast, and not very verbose if something goes wrong. That is why it is always a good idea use that *Show details* button and to scan which files were installed and where, before terminating the installation with **Next >**.



There may be some errors along the way and the installation never finishes. This can be checked by comparing your installation's end to the expected, few last files depicted in the above screenshot.

The completion of the installation is best inspected by observing that a new directory, `\Users\Public\DashT` has appeared. Also, you should have three new desktop buttons for all users, defined in `\Users\Public\Desktop`.

If everything looks right but you cannot find *DashT* from *OpenCPN* as described in [Getting Started](#) chapter, you should get familiar with the *OpenCPN* log file. On a Windows system it is located in: `\ProgramData\opencpn\opencpn.log`

Take a look at the end of the log file where the most recent start-up logs are located. You can search lines having string `"dashboard_tactics_pi"` which is used by *DashT* for its log messages. There are usually only few such log messages containing this identifier, but if you do not see any it means quite likely that *DashT* never gets loaded by *OpenCPN* and that either the installation has completely failed to copy files into right locations, or that there is some compatibility issue - have you [checked the versions](#)? Otherwise the log messages can give you a hint what went wrong. Anyway, the log file is always requested (zipped, thanks!) if you contact no matter who for community help.

2.4 DashT Linux (deb) package

DashT local installation package contains post-installations scripts (see section below). Since the *Ubuntu Software* tool in v20.04LTS seems to ignore `.deb` the suggested tool on it, and in general is `gdebi(1)` to install / uninstall the *DashT* local file `.deb` package. If you do not have `gdebi` in your system, install it with `sudo apt install gdebi-core`. Another tool we are going to use below is `dpkg(1)` which is available by default.

NOTE: There is a graphical tool available for `gdebi`, named `gdebi-gtk`. However, to make the document most suitable for all platforms without explaining the differences with distributions the choice is to concentrate on command line tools only. However, if you install the *DashT* with `gdebi-gtk` the result will be the same - it a GUI to `gdebi`. Use the method which best suits your way of working with your Linux system, of course; you are not **obliged** to follow all verification steps below. They are provided only for total transparency - we respect your system's integrity and want to keep **you** on the driver's seat!

2.4.1 Inspection (Linux)

After the download of the *.deb file from the distribution repository we can list its contents to see what we can expect from the installation. To get the list without installing, we use:

```
dpkg -c dashboard_tactics_pi_1.99.102-ov52-1_amd64.deb
~/Downloads$ dpkg -c dashboard_tactics_pi_1.99.102-ov52-1_amd64.deb
drwxr-xr-x root/root      0 2020-09-08 23:35 ./usr/
drwxr-xr-x root/root      0 2020-09-08 23:35 ./usr/lib/
drwxr-xr-x root/root      0 2020-09-08 23:35 ./usr/lib/opencpn/
-rw-r--r-- root/root 2206200 2020-09-08 23:35 ./usr/lib/opencpn/libdashboard_tactics_
pi.so
...
-rw-r--r-- root/root    25190 2020-09-08 23:30 ./usr/share/opencpn/
plugins/dashboard_tactics_pi/data/streamout_template_http.json
```

What about the post-install scripts, how I can study them before launching the installation?

```
dpkg -e ../dashboard_tactics_pi_1.99.102-ov52-1_amd64.deb
cd DEBIAN
ls
control md5sums postinst postrm
cat postinst
#!/usr/bin/env bash
set -e
# /* $Id: postinst, v1.0 2019/11/30 VaderDarth Exp $ */
#
# Post-install script for Debian package dashboard_tactics_pi<version>.deb
...
```

You can observe that the post-install scripts are only addressing the helper scripts, no third-party programs. Nor do they modify the configuration of the system in any way. They are only needed for configuration control.

You are also invited to read the [privacy statement](#) and [security policy](#) in this documentation. [DashT security page on GitHub](#) provides the latest security status and provides links to open issues.

2.4.2 Install (Linux)

Ready to install? It is simple with gdebi:

```
sudo gdebi dashboard_tactics_pi_1.99.102-ov52-1_amd64.deb
[sudo] password for myname:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading state information... Done

dashboard_tactics_pi PlugIn for OpenCPN
dashboard_tactics_pi PlugIn for OpenCPN
Do you want to install the software package? [y/N]: y
/usr/bin/gdebi:113: FutureWarning: Possible nested set at position 1
  c = findall("[[(\\S+)/\\S+[]]]", msg)[0].lower()
(Reading database ... 285733 files and directories currently installed.)
```

(continues on next page)

(continued from previous page)

```

Preparing to unpack dashboard_tactics_pi_1.99.102-ov52-1_amd64.deb ...
Unpacking dashboard_tactics_pi (1.99.102-ov52) over (1.98.1061-ov51) ...
postrm - Removing DashT scripted helper launchers from
/usr/local/bin:
- dashtengine
- dashtdb
postrm - Done.
Setting up dashboard_tactics_pi (1.99.102-ov52) ...
postinst - Installing DashT scripted helper launchers into
/usr/local/bin:
+ dashtengine
+ dashtdb
postinst - Enabling the execution of DashT system analysis script:
/usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs/scripts/
linux/o-platforminfo.sh
(enabled)
postinst - Done.

```

You may notice that the deb-package manager updated the previous version - the post-installation scripts did the same for the helper scripts.

Now you can launch your *OpenCPN* and look for *DashT* plug-in in its plug-in manager.

2.4.3 Uninstall (Linux)

GUI Uninstall: If you change your mind during the “try-and-buy” period, *i.e.* you want to uninstall immediately *DashT* and you have still the *.deb open under the *graphical gdebi-gtk* you can use it to uninstall the package with a simple click. In Ubuntu 20.04LTS the *Ubuntu Software* center seems to lack (“*Show N technical items*”) option, so it cannot be used for uninstall. However, below we explain the command-line method to be compatible with all systems:

Search for the *dashboard_tactics_pi* package:

```

apt search dashboard_tactics_pi
Sorting... Done
Full Text Search... Done
dashboard_tactics_pi/now 1.99.102-ov52 amd64 [installed,local]
  dashboard_tactics_pi PlugIn for OpenCPN

```

Then remove completely the package from the system with `dpkg --purge` command:

```

sudo dpkg --purge dashboard_tactics_pi
[sudo] password for myname:
(Reading database ... 285744 files and directories currently installed.)
Removing dashboard_tactics_pi (1.99.102-ov52) ...
postrm - Removing DashT scripted helper launchers from
/usr/local/bin:
- dashtengine
- dashtdb
postrm - Done.
Purging configuration files for dashboard_tactics_pi (1.99.102-ov52) ...
postrm - Removing DashT scripted helper launchers from

```

(continues on next page)

(continued from previous page)

```
/usr/local/bin:
- dashtengine
- dashtdb
postrm - Done.
```

2.4.4 Troubleshooting (Linux)

If everything looks right but you cannot find *DashT* from *OpenCPN* as described in [Getting Started](#) chapter the same applies to Linux, Windows or any platform with *OpenCPN* - first take a look at the end of the log-file, which contains all sorts of interesting information. In Linux systems it is located in: `~/ .opencpn/opencpn.log`. Please skip back for a moment now in [Windows installation troubleshooting](#) to read about what to look from that file...

Did not find *any* trace of *dashboard_tactics_pi* in the log file?

This perhaps means that *OpenCPN* has find out that *DashT* plug-in distribution which has been installed is not compatible both with it and the system on which it installed. This is usually caused by incompatibilities in the libraries, mainly *wxWidgets* version - for which *GTK*-library version it has been compiled for.

NOTE: DashT is supported only on GTK3-based systems and it needs *wxWebView* library. *OpenCPN* installed on older GTK2 Linux distributions is naturally supporting GTK2 and would, correctly refuse GTK3-based *DashT* - it would not work! However, with some basic Linux skills it is possible to make *DashT* to work on GTK2 systems with the help of the kind author who has put such a “unofficial” version (handmade) into the distribution. This is the case for *Raspberry Pi/Buster*, which is a GTK2 based system. So is, by default its 64-bit version while it has both GTK2 and GTK3 as alternatives installations. By default, *OpenCPN* v5.2.4 is build GTK2 if it is available. In this case, *DashT* is incompatible with that version of *OpenCPN*.

You can use the following script now on your system to dig out all the above information and figure out why the *DashT* does not get loaded in your system:

```
cd /usr/share/opencpn/plugins
cd dashboard_tactics_pi/data
cd instruks/scripts/linux
./o-platforminfo.sh
```

The print is long and meant for an expert (see instructions at the end of output) but let’s take out a few lines which gives you immediately knowledge will it work for you, or not:

```
...
>>>> Displaying OpenCPN's dependencies on wxWidgets libraries <<<<
...
libwx_gtk3u_core-3.0.so.0 =>
/lib/x86_64-linux-gnu/libwx_gtk3u_core-3.0.so.0
...
libwx_gtk3u_webview-3.0.so.0 =>
/lib/x86_64-linux-gnu/libwx_gtk3u_webview-3.0.so.0
...
```

The above means that this is a GTK3 system and there *is* *wxWebView* installed for it. If GKT2 is indicated, or if *wxWebView* is not found, there is not much hope to get prebuilt *DashT* *.deb package to work on this system.

Maybe your system is Ubuntu 18 or older, or one of their derivations? Please consider upgrading. Or [make the effort to compile everything from the source](#), including the *wxWidgets* with *wxWebView* library. It has been done, it works. But one can ask is the advanced, web technology enabled *DashT* the right tool for your trusted but a bit outdated system?

Of course, in navigation one seeks stability, but *DashT* is not a navigational tool. It is mainly intended to increase the performance of you and your boat so it needs to follow its time.

If you still think that *DashT* **should** be recognized by *OpenCPN* in your system, but it does not please use the following command in the above directory to make a reasonable size report file which you can attach to your issue post:

```
./o-platforminfo.sh 2>&1 | gzip >~/Downloads/report.txt.gz
```

2.5 Optional platforms

Other platforms where installation is possible and an installer is provided:

- Raspberry Pi4B Raspian Buster (*.deb installer provided for **64-bit** Ubuntu 20.04LTS Focal Fossa version, not compatible with RaspberryOS64 which by default works with GTK2. GTK3 is available with `update-alternatives` but will the OpenCPN be GTK3, probably not: see below, *will not work out of the box*.

2.6 Installed helpers

2.6.1 Windows scripts

The helper scripts and the supporting configuration files are installed in \Users\Public\DashT folder. Three buttons to use those will be installed on the Desktop:

1. Start web server (for EngineDJG only)
2. Start DB (database *and* web services for everything)
3. Stop DB

These buttons will do nothing, however, if you do not install yourself the third party software they are referring to. No third party software is pulled in without you wanting it to happen. See corresponding sections in this document

2.6.2 Linux scripts

The Linux version installs just two helper scripts: **dashtengine** and **dashtdb** - try them out from the command line.

NOTE: we let fans of GNOME3, KDE, LXQT, xfce4 and alike to create their own desktop buttons!

Respecting the principle “only if you want it you will get it”, no third party software is declared as automatic dependency in the *DashT* package installation. Only if you launch one of the scripts, and if the supporting software is not there, you will be asked do you want to get it installed. This way, a person who is not interested in database functions will not get database software installed for nothing.

Third-party programs installed are, for

- *dashtengine*
 - nodejs, npm
 - * signalk-server, http-server
- *dashtdb*
 - docker, docker-compose
 - * InfluxDB v2.0, Grafana, nginx web server

NOTE: If you are running *dashtdb* script on a *arm64/aarch64* platform, and it proposes to install Docker with InfluxDB v2.0 it may fail to do so - it is not necessarily available for *arm64/aarch64* - only *amd64*. But the situation may evolve and you can try the installation.

2.7 wxWebView and GTK2

If you have a recent distro at level of Ubuntu 20.04 LTS (*Focal Fossa*) you do not have to read this section. For older versions, and upgrade is recommended to a GTK3 enabled version.

DashT requires *wxWebView*, which requires *libwx_gtk***3*u_webview_3.0* or similar. Helas, on older, GTK2 based installations one cannot find anymore an equivalent *libwx_gtk***2*u_webview_3.0* or such, for security reasons (*webkit2gtk* is not maintained anymore).

Therefore, the logical decision is to maintain *DashT* only for GTK3 based systems.

However, it is possible to make it work by installing your own *wxWebView* and compile it with *wxWebView*.

NOTE: you would be using a system with a **potentially insecure** *wxWebView* - make your own security assessment and assume your own risk before applying blindly the below method.

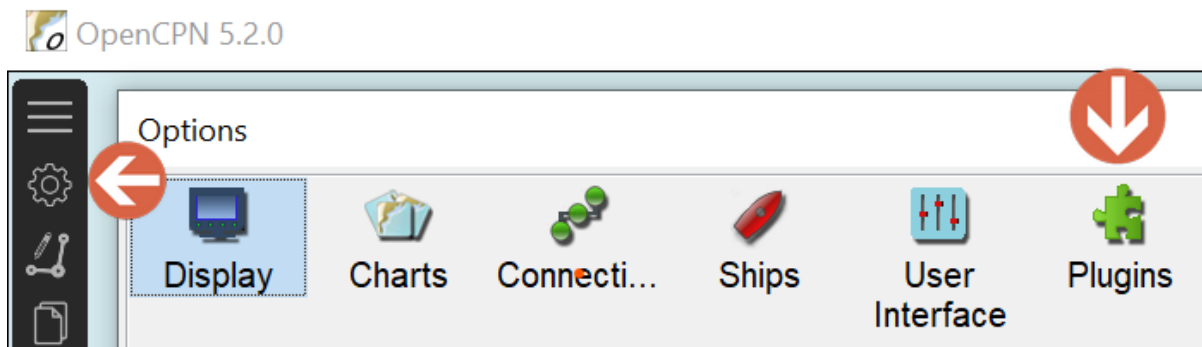
As an example, Raspberry Buster: compile your own *wxWidgets* stable installation from [source](#) and [install it](#). Make, say a *gtk2_build* folder for the build. In it, `./configure --with-opengl --with-gtk=2 --enable-debug`. Then `make -j4`. When you say `sudo make install` after the build, all the libraries will be go into */usr/local/lib*, and will not be clashing with your distribution's *wxWidget* libraries, they will prevail. Only that the missing libraries, amongs them *libwx_gtk***2*u_webview_3.0* is now found, after you run `ldconfig`.

Note: Even with the above working, you would be limited to *EngineDJG* dials and *Race DashT* functions: the InfluxDB v2.0 does not yet exist for *armhf* architecture in their docker repo. One needs to wait final release to see how this evolves. Meanwhile, if you are interested in the time series database functions you need to provide the InfluxDB v2.0 service from another computer and use the HTTP method to write into it, see [InfluxDB Out](#).

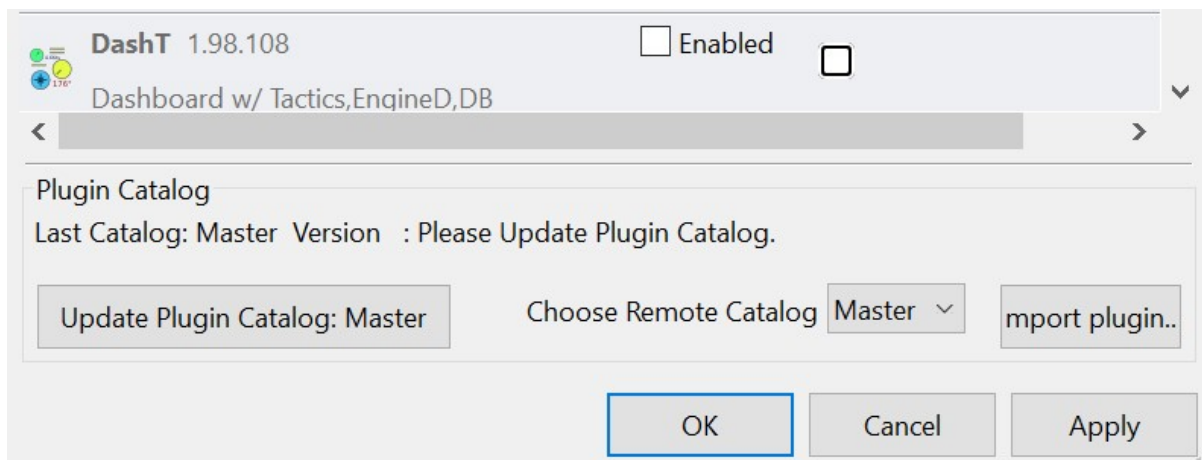
GETTING STARTED

Start (or restart) OpenCPN v5.2 or superior after the installation of the *DashT* with a local installer package.

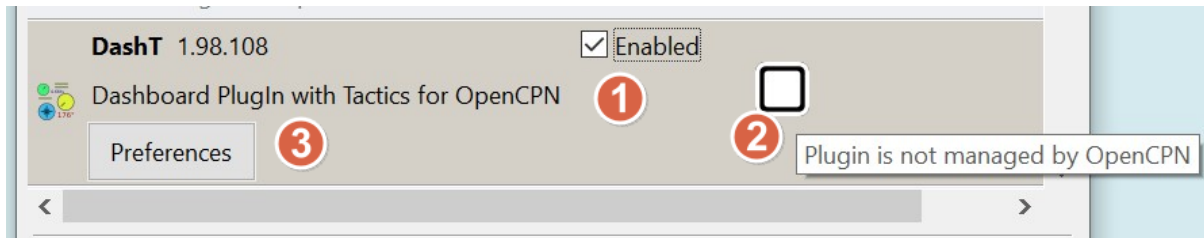
Start the OpenCPN plug-in manager from the OpenCPN options menu:



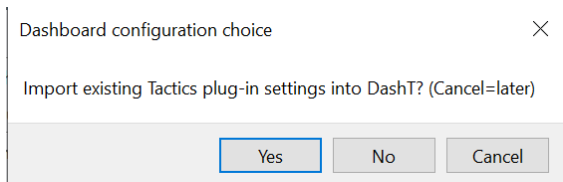
Search for the *DashT* plug-in. As a new plug-in, it is disabled. Verify that its version is the same as this document's version, *i.e.* they are likely coming from the same distribution.



(1) Enable DashT plug-in.



If you have Tactics plug-in installed, you will be suggested to import its settings. Make your choice:

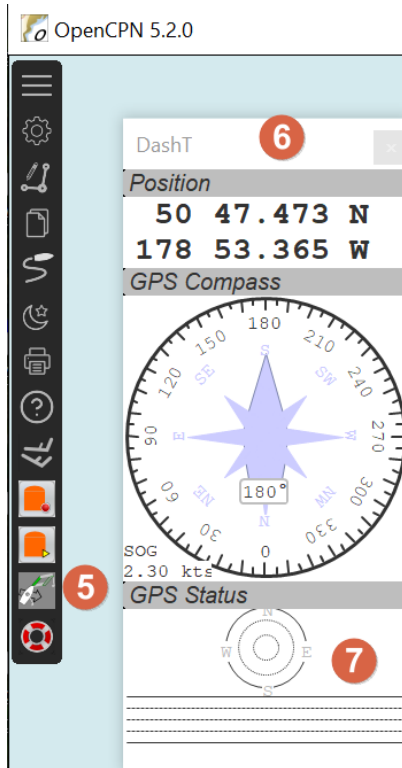


(2) is needed for this distribution: this is a local installation, not from the centralized OpenCPN.org repository.

(3) You may want to notice that one has access to *DashT* preferences from here. Right now we are not going to set them.

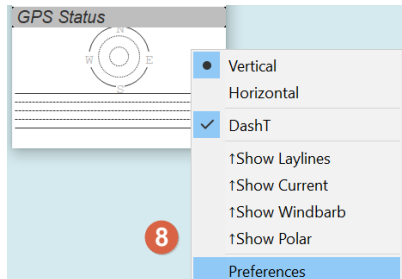


(4) Apply your choices and leave the OpenCPN options dialog. *DashT* is enabled but not visible.



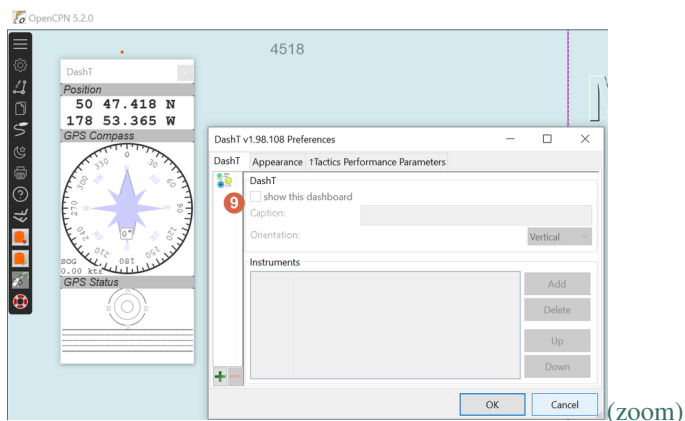
(5) Click on the *DashT* icon to make its default instrument panel (6) visible. The default panel's instrument are exactly the same than on built-in Dashboard. Right-click on an instrument (7):

(8) Select Preferences.

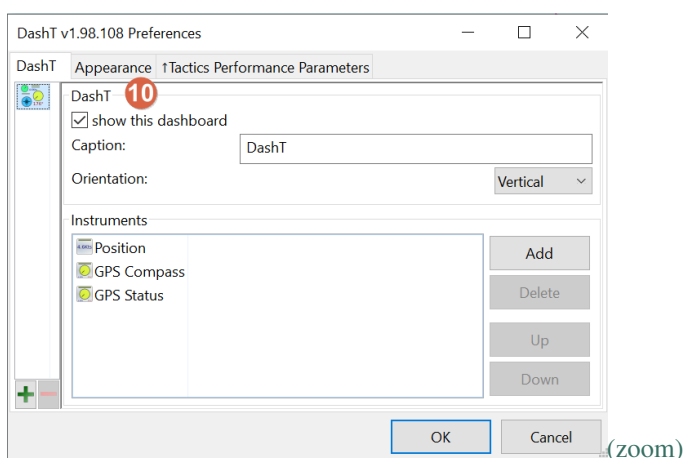


(9) *DashT* preference dialog's main tab is used to manage dashboard panels and the instruments in those. By default only one dashboard panel is visible. Click on its symbol to show the instruments in it.

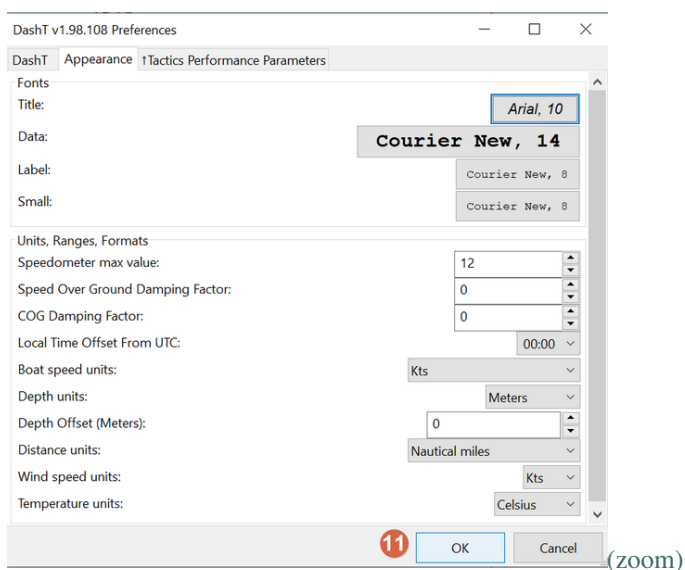
You got it right, you can add new instrument panels with + (plus) and remove them with - (minus) signed buttons on the left, and populate them with the instruments you like with the **Add**-button on the right. There are quite a few instruments, so they would be better explained in dedicated chapters later on in this document.



(10) Finalize the initial settings by clicking on “Appearance” tab.



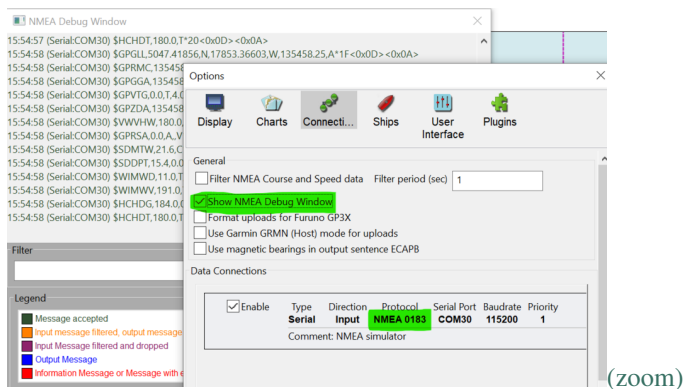
(11) Set the appearance and conversion values to your preferences. Note that these values, unlike those from the Tactics plug-in are not imported from the OpenCPN built-in Dashboard plug-in.



NMEA-0183 DATA

Like the built-in Dashboard, *DashT* receives all the same NMEA-0183 data OpenCPN receives, via the OpenCPN.

If you see the NMEA-0183 data coming in with the OpenCPN NMEA-0183 data stream debugger, the same data is sent to *DashT*.



In addition to the sentences it receives, OpenCPN generates a few sentences of its own. For example, when following a route, it generates RMC sentences. Please refer to OpenCPN's documentation. Even these values end up to *DashT* plug-in.

4.1 Data source priority

DashT has two sources for instrument data and it uses them in this priority order:

1. *Signal K server node's delta channel* - which can be of *NMEA-0183* type or *NMEA-2000* type
2. OpenCPN provided NMEA-0183 data

This means that if the same data is available from both channels, the priority is given to the one coming from the *Signal K server node's delta channel*.

NMEA-0183 data sentence priority, in case sentences containing same data is strictly the same as in OpenCPN Dashboard plug-in, and this independently is the data source *Signal K server node* or OpenCPN.

In addition to those two instrument data channels, *DashT* can read back any of the data value from the time-based database connector, provided that the streaming into it has been enabled. The instruments taking usage of this are typically dedicated for historical data and the data source in those is clearly indicated.

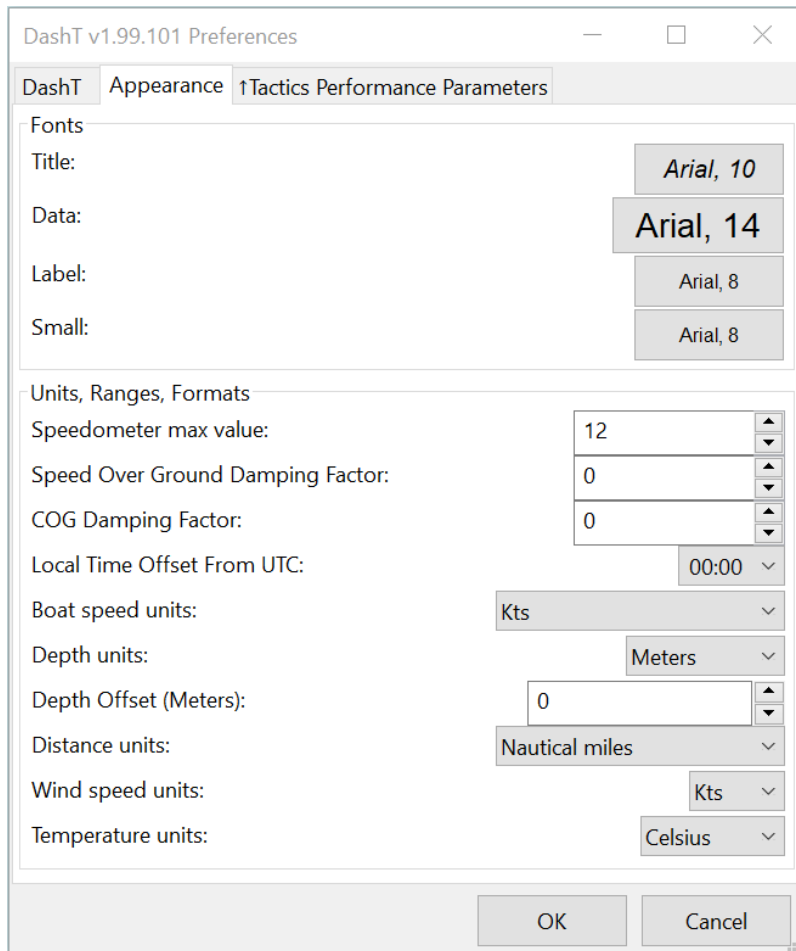
DASHBOARD

This chapter is a list of available standard Dashboard instruments which are the first one in the Preference dialog's **Add** button's list.

In case you are not a longterm user of OpenCPN or [otherwise have not read its documentation](#) you may want to take a look of this list: these instruments you can find also from the built-in Dashboard, the look of them may vary between the two since in *DashT* there are quite a few customization options available, see corresponding chapter for possible tweaks through the .ini/.conf-file.

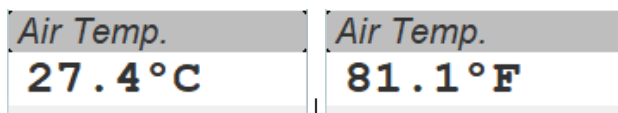
5.1 Configuration

The below screen capture configuration screen of the *DashT Preferences* dialog, its *Appearance* tab. It only a little bit changed - for temperature unit - in *DashT* compared to what is available in *OpenCPN* built-in Dashboard plug-in. Therefore you are invited to consult the *OpenCPN* built-in documentation for their meaning. However, they are quite self-explanatory:



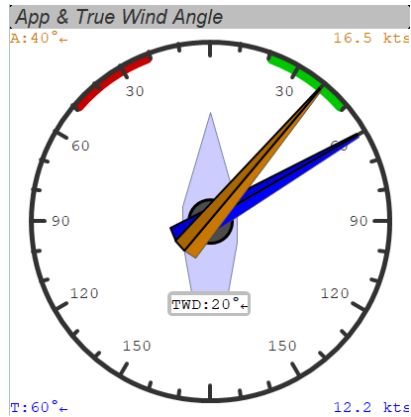
There is more, in-depth analysis made directly from the code base available in *Tweaks chapter, about ini/conf-file settings*.

5.2 Air temperature



Air temperature. Only you know from where and how it is actually measured and how well the instrument is calibrated.

5.3 AWA/TWA Dial



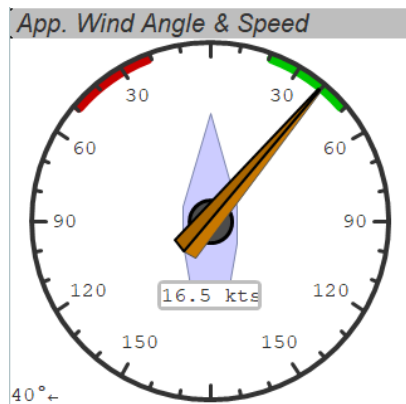
Apparent Wind Angle (upper needle) and True Wind Angle (lower needle) Dial. You get also the speed information and the True Wind Direction as numerical values.

5.4 AWA



Apparent Wind Angle as numerical value.

5.5 AWA/AWS Dial



Apparent Wind Angle (needle and numerical value) and Apparent Wind Speed (numerical value) Dial.

5.6 AWS Dial



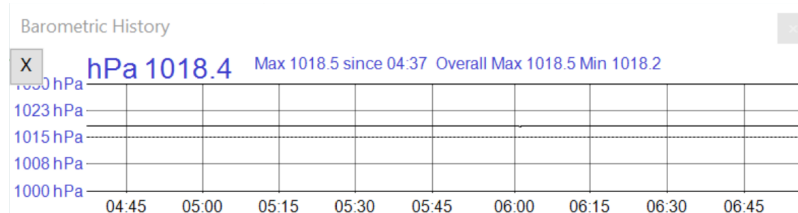
Apparent Wind Speed dial. Numerical value is given, also for the true wind speed.

5.7 AWS



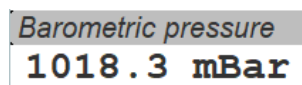
Apparent Wind Speed as numerical value.

5.8 Baro History



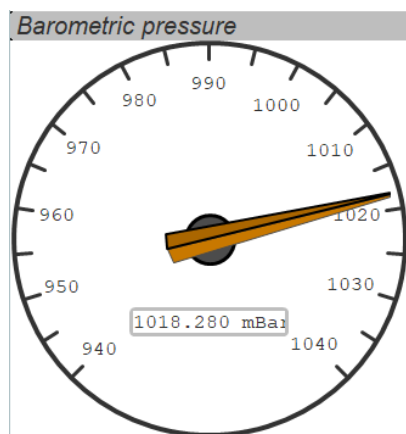
Barometric history. Please note that this instrument allows you to register the data by exporting the values to a CVS-formatted file with a fixed interval. Click on the **X**-button. See also [Tweaks](#).

5.9 Barometer



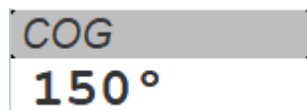
Latest value from the barometer.

5.10 Baro Dial



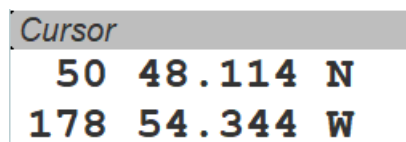
Latest value from the barometer if you prefer a dial. Oh, and you get more decimals as bonus for the numerical value!

5.11 COG



Course Over Ground.

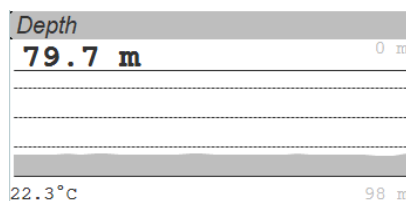
5.12 Cursor



Position of the cursor.

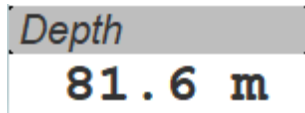
The depicted position is South of the Delarof Islands, U.S.A.

5.13 Depth Graph



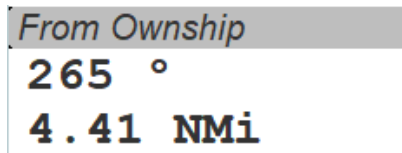
Depth as graph, also as numerical value and unit, also with the temperature of the water.

5.14 Depth



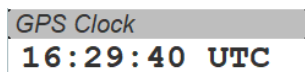
Depth in a numerical instrument.

5.15 From Ownship



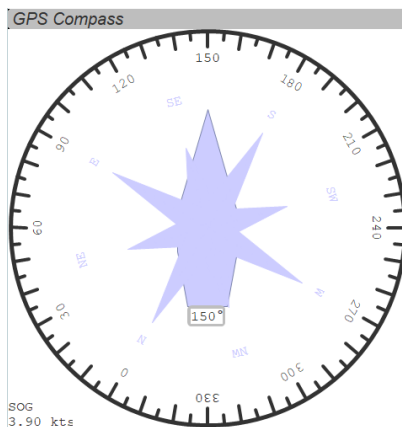
Position and the distance of the cursor from the actual position of the boat.

5.16 GPS Clock



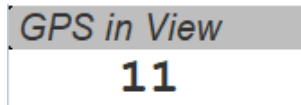
UTC time provided by the GPS.

5.17 GPS Compass



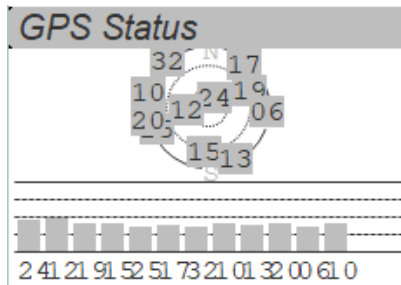
GPS Compass is actually Course on Ground (COG), giving also the numerical value together with the speed on ground.

5.18 Satellites in view



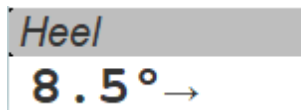
Number of satellites in view.

5.19 Satellite Status



Provides a graphical, condensed summary about the satellites currently in view.

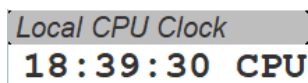
5.20 Heel



Heel angle, aka. "Roll".

Arrow direction? Right = Starboard side, Left = Port side.

5.21 CPU clock



In a long *passage*, if you do not synchronize your computer clock with the GPS somehow, it will be drifting. See below.

5.22 GPS local time

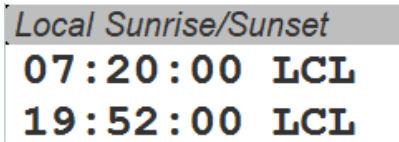


Local GPS Clock
18:41:06 LCL

Local time from GPS satellites.

NOTE: in distributed systems the correct time synchronization between various devices is of uttermost importance! See above: compare your CPU's time to this time (or to the GPS UTC time if you are running UTC time on your navigation computer). Do not let the CPU clock to drift away, readjust if necessary, or better still would be to synchronize all boat's devices to the GPS if that is possible.

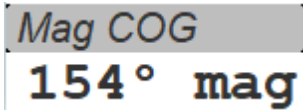
5.23 (local) Sunrise/Sunset



Local Sunrise/Sunset
07:20:00 LCL
19:52:00 LCL

Sunrise and sunset in your current location using the local time.

5.24 Magnetic COG



Mag COG
154° mag

Magnetic Course on Ground.

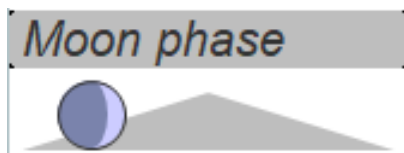
5.25 Magnetic HDG



Mag HDG
152°

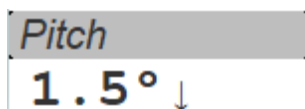
Magnetic Heading.

5.26 Moon Phase



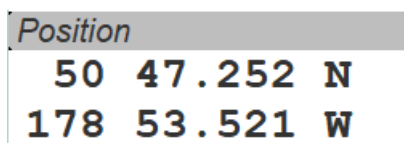
Moon phase in the current position referred to the GPS time.

5.27 Pitch



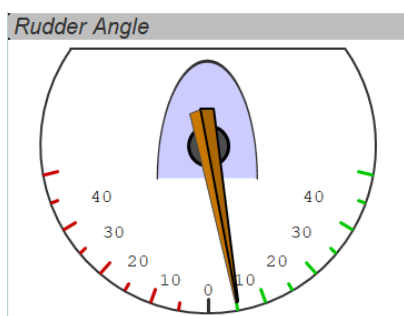
Pitch angle. Arrows are pointing the nose going up or down.

5.28 Position



Current position of the boat.

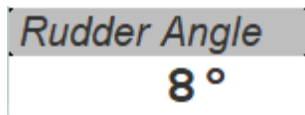
5.29 Rudder angle graph



Rudder angle as graph.

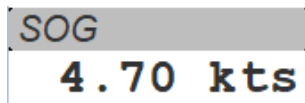
Take the “needle” center point as the rudder shaft. When the rudder turns on the starboard side (= right on the graph = pushing the tiller to the port side), the boat turns to starboard direction.

5.30 Rudder angle



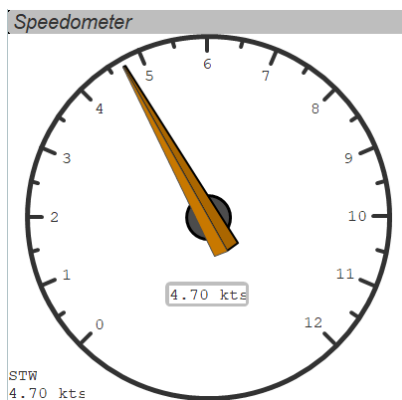
Numerical presentation of the above graph. Negative values are to the port side and positive values to the starboard side.

5.31 SOG



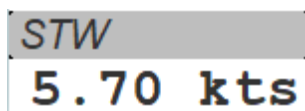
Speed on Ground.

5.32 SOG Speedometer



Speedometer - the classical OpenCPN dial, presenting the Speed On Ground, together with the Speed Through Water as a numerical side value.

5.33 STW



Speed Through Water. Keep your paddle wheel clean. This is an important input to Tactics functions!

5.34 Sum Log

Sum Log
3346.0 NMi

Total log distance given by your navigation system.

5.35 Sunrise and Sunset

Sunrise/Sunset
04:55:00 UTC
18:32:00 UTC

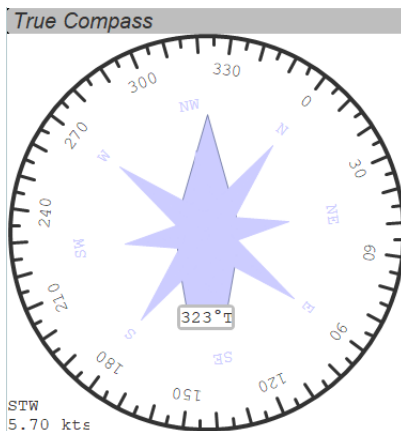
There is the same instrument for the local time in the current location. This is the same but with the UTC time.

5.36 Trip Log

Trip Log
35.1 NMi

Trip distance given by your navigation system.

5.37 True Compass



True Compass shows HDT, true heading direction and its value. Speed through water (STW) is shown as a side value.

5.38 True Heading

True HDG
323° true

Same as True Compass, *i.e.* true heading direction as numerical value.

5.39 TWA

True Wind Angle
57° ←

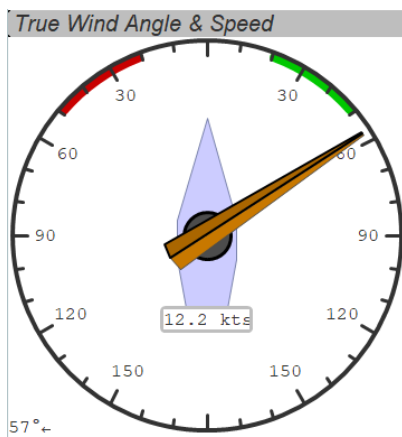
True Wind Angle

5.40 TWS

True Wind Speed
12.20 kts

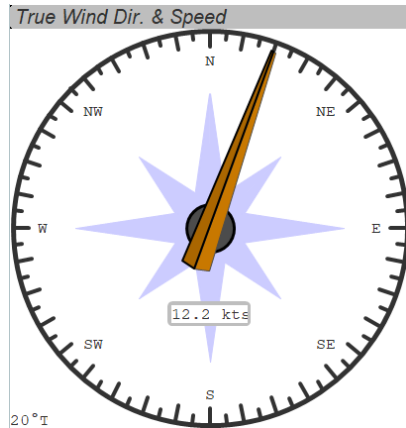
True Wind Speed

5.41 TWA/TWS Dial



True Wind Angle and True Wind Speed dial.

5.42 TWD/TWS Dial



True Wind Direction and True Wind Speed dial.

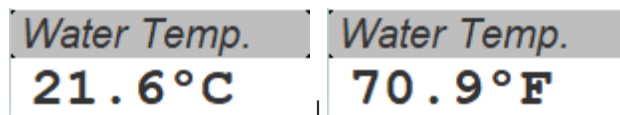
5.43 VMG (route)



Velocity Made Good - **not** indicating speed of a sailboat towards the direction of the wind. See Tactics instrument for it. This value is an OpenCPN term defining the speed of the boat to the direction of the next marker in an **active** OpenCPN route.

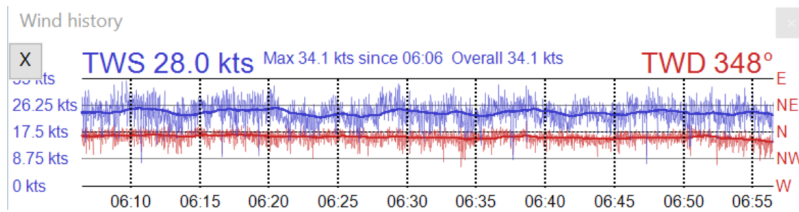
Tactics uses this term also in its perhaps more classical meaning for the sailing boats. Nevertheless, it is flexible enough to use another term since the OpenCPN one cannot be changed. See the *Tweaks*: the abbreviation can be changed to your liking in all Tactics functions.

5.44 Water temperature



Water temperature. Only you know from where the temperature is taken and how well it is calibrated. It is better to get one's feet wet before jumping in!

5.45 Wind History



True Wind Direction and True Wind Speed data history. Please note that this instrument allows you to register the data by exporting the values to a CVS-formatted file with a fixed interval. Click on the **X**-button. See also [Tweaks](#). The [Tactics chapter](#) explains the saving of the data from the GUI more precisely.

TACTICS

This chapter is ©2015-2019 [tom-r](#) and reproduced here from the [original source](#), together with the source code containing tactic_pi algorithms and methods under the GNU General Public License v3 (GPL v3) – Likewise, modifications in [DashT software](#) into those algorithms and their description in this chapter and in this document are ©2019-2020 [canne](#), and published under the same license. Please feel free to fork and copy this work but if you plan to publish it, please keep this notice unmodified in the document together with any copyright statement in the source code.

6.1 Introduction

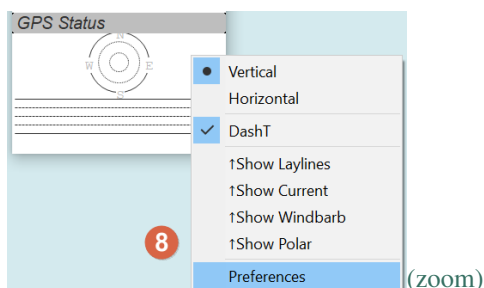
Everything started with the question : *“Do we make it around the edge of that island when we tack now and sail the same apparent wind angle on the other tack?”*

Disclaimer: This is still alpha code (not even beta), and you should not use it for live – real sailing. I will not be liable for any harm, damage or whatever strange things happen if you use this plugin and rely on its data.

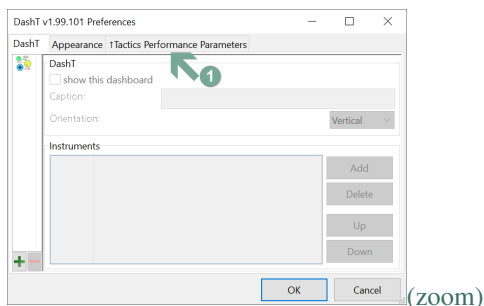
6.2 What Tactics can do?

- Calculate true wind data : TWA, TWD, TWS from true heading (HDT), speed through water (STW) and apparent wind speed (AWS), with optional correction by the heel angle. Calculation is enabled via a preference setting and disables available true wind data from the bus throughout the tactics_pi plugin ;
- Calculate the “leeway”, the boat drift based on heel. A common formula is used for that purpose ;
- Calculate the surface sea current and display it as single instruments (current speed/direction) as part of the “Bearing compass” or as overlay on the chart (semi transparent). The routines take boat heel and leeway into account. If you don’t have a heel sensor, there is a simply workaround, see below. Current display on the chart can be disabled by a preference setting ;
- Calculate and display the boat laylines for the current tack, and the same TWA on the other tack. Sea current is taken into account, if available ! Laylines may be toggled on/off. Adjustable length and max. width (triangle, with one corner at the boat) of the boat laylines. The layline width reflects the boat’s yawing (COG changes over time) ;
- You can load a polar file and calculate/display performance data, like actual VMG (velocity made good up-/downwind), Target-VMG, Target-TWA (the opt. TWA up-/downwind), CMG (course made good towards a waypoint), Target-CMG (opt. CMG angle and speed), polar speed (the speed you should be able to sail at current TWA/TWS based on your polar) ;

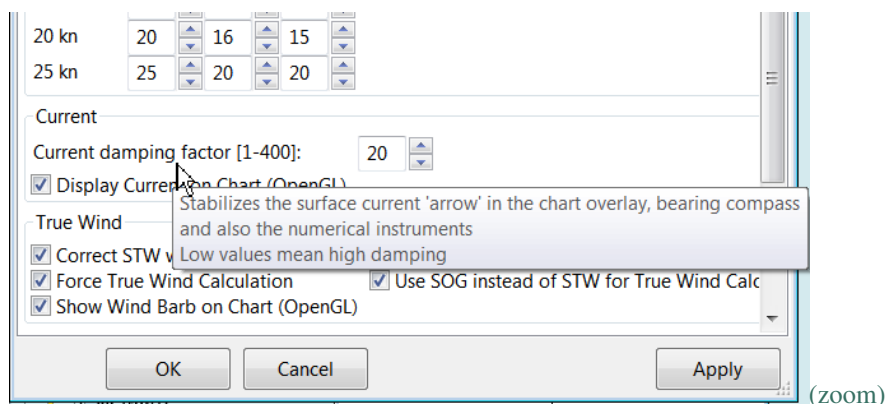
- It can display the actual polar as overlay on the chart, including markers for Target-VMG/CMG angles. Just adopt your course and place your heading pointer on one of the markers, and you sail Target-VMG/CMG based on your polar ;
- You can set a (one !) temporary tactics waypoint and display the laylines to the mark, based on a Target-TWA calculation, while taking your polar into account ;
- It has a “dial instruments” called “Bearing compass”. Boat true heading (HDT) points “up”, it shows the boat laylines as well, the surface current, a pointer to the waypoint (either set manually as the temporary Tactics waypoint or read from a NMEA RMB sentence), needles for AWA and TWA, markers for the Target-VMG/CMG angles ;
- It has a “Polar compass” instrument, graphically displaying the actual polar ring and markers for Bearing to WP, Target-VMG angles and Target-CMG angles ;
- It has an “Average Wind” calculating background process with short and long term time scales. A graphical instrument is provided with an adjustable averaging time, displaying both numerical values and both short and long term deviations to port / starboard
- It can create specific NMEA performance records with the purpose to export them to the displays of your instruments. You can now e.g. calculate the polar target speed in the plugin and send it to your instrument display outside in the cockpit. Currently only available for NKE, but may be enhanced in the future ;
- It can export the data from BaroHistory, WindHistory and PolarPerformance instruments to a flat text file with selectable export rate and column separator (via ini file). The export will either create a new, or append to an existing export file ;
- Various Tactics settings are grouped in a separate tab in the Preferences configuration dialog of *DashT*. To access the preferences dialog, right mouse click on any *DashT* instrument, then select “Preferences” (8);



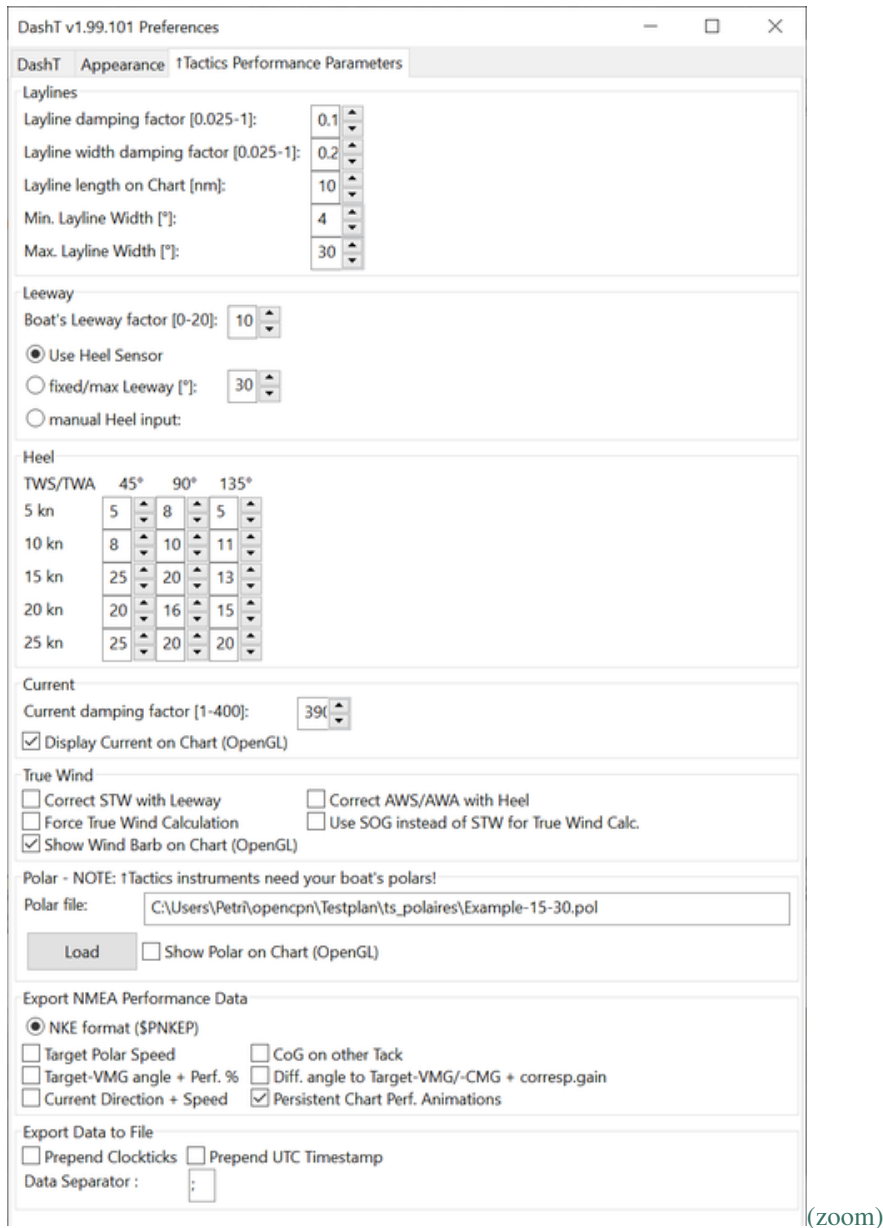
- You’ll find all settings in a separate tab “*Tactics Performance Parameters*” (1):



- There are pop-up *ToolTips* available for almost all of the Tactics preferences. Just hover the mouse over any of the settings to get explanation and usage description.



Below is a screenshot which depicts all the Tactics parameters which can be changed from the Preferences dialog. Hover the mouse over each parameter to get an explanation. To get the same explanation and some additional hints, see the tweaks explained [later on in this document](#) which explains the items which can be changed in the ini/conf-file.



(zoom)

6.3 Prerequisites

- You will need to activate OpenGL, if you want to use the chart overlay functions ;
- SOG, COG from the GPS ;
- True Heading from an electronic compass. If not available, magnetic heading will do, as long as you have magnetic variance available. It can be available from these sources, in this priority order:
 - From *OpenCPN* GPS Fix ;
 - HDG-sentence ;
 - RMC-sentence (likely already used by *OpenCPN* for fix) ;
 - From from the wmm_pi plug-in.

- Boat speed through water **STW** from a log / “paddlewheel” sensor (keep it clean!) ;
- Apparent wind angle **AWA** and Apparent Wind Speed **AWS** ;
- Heel sensor which supplies your boat’s heel angle, also called “roll”. If not available, there’s a workaround with manual input in preferences ;
- You need a polar file of your boat to use all polar based performance calculations, explained in a separate section in this document ;
- Calibrated sensors or correction to AWA, Compass HDG/HDT, STW (Speed through water), and AWS (apparent wind speed), as good as possible. Especially the compass heading calibration tends to be neglected. However, this is vital for a proper surface current calculation. All I can say is : sh*** in – sh*** out ... ;-) .

6.3.1 CMG/VMG abbreviations

The terms CMG and VMG used in this documentation are not unique across instrument manufacturers and other sailing software packages, even in OpenCPN and in Dashboard “VMG” is used in routing functions! This software will still use both terms, familiar to the intended target users, both in the software and here, in this documentation. However you can change both terms by setting two preferences in the ini/conf-file file. See the [corresponding chapter](#) how to do that.

See also the Terminology at the very end for the definition of terms.

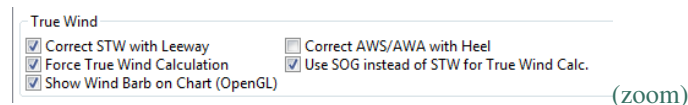
6.4 Without Polar

The instruments and functions in this chapter do not require that you have your boat’s polar.

6.4.1 Calculate true wind data

The plug-in can calculate true wind data (TWA, TWS, TWD) but keeps the calculated data inside the plug-in. (it does not broadcast it to OpenCPN!)

True wind calculation is done silently if you don’t have TWA, TWS, TWD available in your NMEA stream. Furthermore you can force the true wind calculation in the plug-in by a preference setting. If the tick “*Force True Wind Calculation*” is set



It does not matter if TWA, TWS and TWD are already available on the system or not. Calculation is done in the plug-in if you select this option. It does calculate TWA, TWS and TWD.

This is useful in case, for example if you have a heel sensor but which is not integrated in your instrument bus. You can use the corrections to get more accurate true wind data by taking into account the heel (roll).

Input is AWA, AWS, STW, and for TWD also true heading HDT.

NOTE: *DashT* has a special function for a **moored boat** which is enabled by the *Force True Wind Calculation* option: normally, one needs STW (*i.e.* some speed) to create the vector calculations for True Wind. *DashT* detects if the STW is not null - your instruments are awake - but that the speed is less than 0.2 knots. In this case it calculates the True Wind Direction (TWD) solely based on the HDT. This might be useful if you are observing the wind evolution using the wind history instrument in a safe harbor or mooring. But if there is a big swell or a strong current, this function may cut itself off because of sporadic STW values.

If you don't have HDT on your system bus (but only HDG), you can use `wmm_pi` plug-in: it supplies the magnetic variation and if it is enabled, its value is fed into *DashT* by OpenCPN and its value is taken into account to calculate HDT from HDG.

Troubleshooting: if you have issues with the true wind calculation in *DashT*, take a look at the OpenCPN log-file. *DashT* collects the information needed by the Tactics algorithm, reporting the progress in the log file with `dashboard_tactics_pi` identifier to help in the troubleshooting in case of some data is missing, for example.

6.4.2 SOG instead of STW

You can select to replace STW (Speed through water, the “paddle wheel” or the “log”) with SOG (Speed on ground, from the GPS) in the Tactics algorithm true wind calculation.

The purpose of this option is to have a fall-back for the true wind calculation in case your paddle wheel gets clogged.

However, it can be considered also as a way to eliminate the effect of the surface current to the True Wind calculations: Some racers like to refer to the water on which they are gliding, but some other sailors like to refer to the *terra firma* over which the wind is blowing... make your choice with this option!

6.4.3 Heel corrections

If you have a heel sensor in your system, and its data is available to *DashT*, you can use the two following corrections:

1. **Correct STW with Leeway** : the plug-in can calculate your leeway (drift angle) based on your heel sensor (see below). That means your boat is possibly moving sideways, which adds an error to the True Wind calculation. Standard instruments normally do not take this effect into account. For example NKE does this correction in its regatta processor only, but not on their normal instruments. Please take also into consideration the following paradigm: Leeway calculation uses (STW^2) which would lead to a circular definition (cf. Tactics v1.0.009). That is why leeway is not corrected, in circular manner with the corrected STW. However, the corrected STW is used on all other algorithms where it is applicable, for example in the current calculations ;
2. **Correct AWS/AWA with Heel** : This option corrects your AWS and AWA data by the heel angle. Use this option with great care! Manufacturers normally already make this correction if you have a heel sensor integrated in your instrument bus. OpenCPN will simply receive the already corrected data for AWS / AWA in this case. If you apply now the correction the result would be wrong! The option is provided mainly for systems using an external heel sensor, such as built with an [Arduino](#) but which is not recognized by the boat's instrument system. To make sure that you understand the eventual risk of a double correction, a warning pop-up is shown when you set this tick. See [Sailboat Performance Testing Techniques](#), A.Gentry, 1981.

6.4.4 Calculate Leeway

Leeway describes the drift of the boat due to the force of the wind. Leeway is the basic input for the surface current calculation described later on. Input for the leeway calculation is your heel angle. Normally you'd say : the more you heel, the more you drift. But that's only part of the truth. Other significant inputs are boat speed and the shape of your hull.

A widely (NKE, B&G,...) used formula calculates the leeway with 3 input values : heel, boat speed (STW), and hullshape-factor.

$$Leeway = LeewayFactor * Heel / (STW^2)$$

See the [Leeway](#) instrument which uses this formula. Please note that while the *Heel* has a direction, there is no other indication for the actual wind direction.

The *LeewayFactor* attribute in the above formula is for you to provide, for your boat or the sailing condition. It is given in the Tactics Preferences, in the parameter called “*Boat’s Leeway factor [0..20]:*”

The input range is 0...20, 10 is a good value to start with.

If you do not have a heel sensor on board, you can either set a fixed value (e.g. 0 when motoring without sails), or try to set up a very simple “*heel polar*”:



TWS/TWA	45°	90°	135°
5 kn	5	5.1	5
10 kn	8	10	11
15 kn	25	20	13
20 kn	20	16	15
25 kn	25	20	20

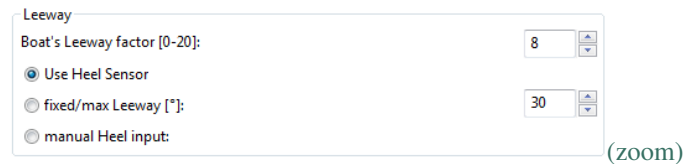
The idea of the “*heel polar*” is that almost every magnetic compass installed in the cockpits has a scale, where you can read the degrees of heel: simply compare the *DashT* Dashboard instruments display, for TWA and TWS, read the values from the scale and put it into the small table above.

NOTE: Please make sure to read the True Wind Angle (TWA) and the True Wind Speed (TWS) and not apparent wind angle and speed (AWA/AWS).

I tried it on my own boat, comparing the heel polar values with those of my sensor. It works astonishingly well.

Even if you use the heel-polar, you have to estimate the “*Boat’s Leeway factor [0..20]:*”

You have 3 choices for heel input, depending on where you set the radio button in the preferences. You can switch the radio buttons forth and back while sailing to compare the results, or as a tactician to take into account different sailing conditions.



The attribute “*fixed/max Leeway [°]:*” has a dual purpose:

1. The given value is always taken into account as maximum possible Leeway value. In the screenshot below, I set it to 30°. If your heel polar or calculation with the formula above outputs values >30°, the program takes 30° ;
2. If you set the radio button here, the routines always take 30°, no matter what your sensor calculates or your heel-polar would tell you.

6.4.5 Calculate the surface current

If you compare your HDT and COG vectors in OpenCPN (the 2 forward vectors on the chart at your boat), the difference between both is a mixture between Leeway (the boat's drift) and surface current. Once we can determine Leeway, the rest is surface current.

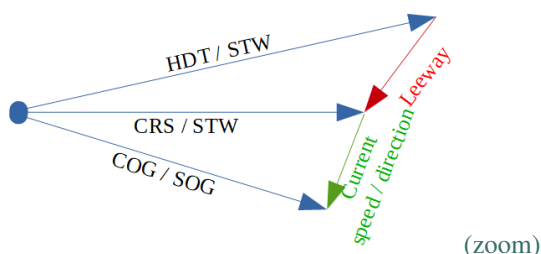
The surface current calculation is simply a triangle calculation with vectors.

Always seen from the current position, the first vector is HDT (degrees) / STW (length).

As your boat drifts with the wind, the second vector is “course through water” (CRS, degrees) and STW (length).

Course through water is actually HDT with applied leeway.

The resulting vector between CRS/STW and COG/SOG is the surface current.



To calculate the current, you need as input HDT, STW, Leeway, COG and SOG and your GPS latitude / longitude.

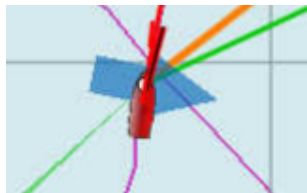
In the preferences you can set the damping factor for the current calculations : the lower the values are the more filtering is applied, and the reading gets more stable. On the other hand, it starts lagging a bit. The lower the value, the more damping is applied. You can do experimenting in the range of 0.001 to 0.025. Keep this value at the lower end, then start to increase, until it gets unstable.



NOTE: Tactics algorithms currently always corrects STW with Leeway for the current calculation, independent of the preference setting “*Correct STW with leeway*”. This may change in a future release but even then the default setting will remain as it is.

6.4.6 Current arrow on chart

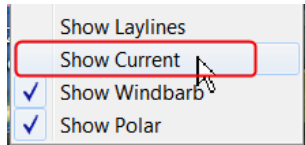
You can show a semi transparent blue current symbol underneath your boat, showing the surface current direction.



To activate the current display on the chart by default, upon program start, navigate to the Preferences dialogue and set the tick “*Display Current on Chart (OpenGL)*”. The preference is only setting the default; while navigating in OpenCPN, you can turn the current overlay display on / off as you like:

Just right-click on any *DashT* instrument and select “*Show Current*” in the

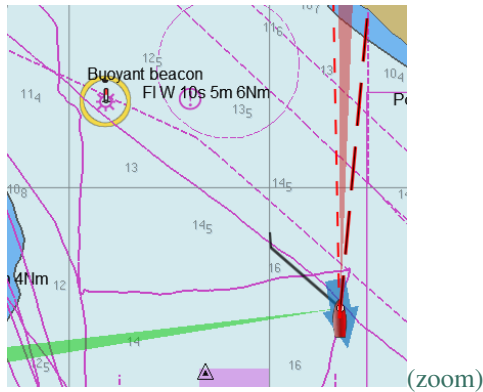
contextual menu



This will toggle the current display arrow on/off.

6.4.7 Boat laylines

You can show the boat laylines on the chart. They refer to COG.



The colours mean:

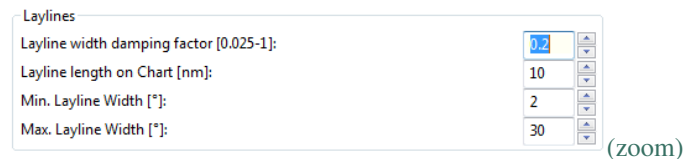
- *red* = wind from port ;
- *green* = wind from starboard.

The width is defined by the yawing of your boat, the more there is yaw, the wider the laylines do get.

The second layline (green in this example) simply shows you where you would get on the other tack / gybe when you sail the same TWA after the tack. Leeway and current are taken into account for the calculation of the second layline:

$$\Delta COG = (COG - HDT) + 2 * TWA + |Leeway| + CurrentAngle$$

In the Preferences, you can set the following options:



The “*Layline width damping factor*” is the rate how fast the layline width reacts on COG changes.

It's done with exponential smoothing, the smaller the factor, the higher the damping rate.

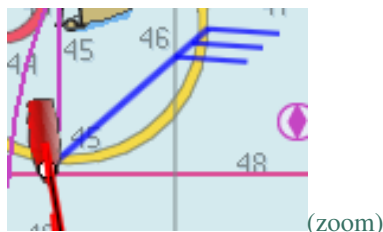
You can define the length of the laylines on the chart, as well as a minimum and maximum width.

If you don't like the yawing effect simply set min and max both to 1 or 2 degrees.

To toggle the layline display on the chart on/off, right-click on any *DashT* instrument and select or unselect “*Show laylines*” from the *contextual menu*.

6.4.8 Wind barbs

You can also show a wind barb at the boat position, showing you direction and speed (feather length) in 5 kt steps.



(zoom)

To activate the wind barb display on the chart by default, upon program start, navigate to the Preferences dialogue and set the tick “*Display Wind Barb on Chart (OpenGL)*”.

The preference is only setting the default; while navigating in OpenCPN, you can turn the windbarb overlay display on / off as you like: to toggle the windbarb display on the chart on/off, right-click on any *DashT* instrument and select or unselect “*Show Windbarb*” from the *contextual menu*.

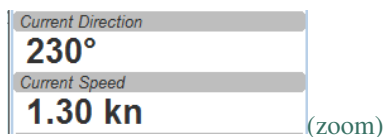
Tactics brings in some improvements to the windbarb presentation, inherited from other OpenCPN plug-ins, it shows the wind barbs up to 47 knots(!) correctly with the following mapping:

symbol	knots	Wind Barb Scale	
	0-2		23-27
	3-7		28-32
	8-12		33-37
	13-17		38-42
	18-22		43-47

(zoom)

6.4.9 Current Direction and Speed

NOTE In *DashT* the Tactics instruments are identified by an up-arrow ↑ - for performance boost they provide!



(zoom)

Current direction means: “current sets into”.

6.4.10 Leeway



(zoom)

Boat drifts 0.5° to starboard due to heel or roll, see *Heel* instrument.

NOTE: See the *formula for leeway's calculation*: it does **not** take into account the direction of the wind. Therefore the **direction arrow displayed is that of the direction of the heel**, positive values received

from the instrument are for the starboard heel and negative values for the port heel. In very light winds this logic may be flawed and the value itself based on a hull factor may not make sense.

6.4.11 Average Wind Instrument

The graphical instrument shows the average wind and its deviations to both sides.

This instrument is also used to control and display the result of a background processing average wind calculation using Tactics double exponential smoothing algorithm. It is recommended to have it in your instrument catalog if you plan to use any other instrument requiring the information about the average wind.

DashT has a continuously running background process keeping these values available for all instruments, such as *TWA to Waypoint*, *Race Start* and *Race Mark* even if this instrument is not active or shown.



The curve is centered on the average wind, green means the wind is on starboard of the actual average wind, red means it blows more from port.

One can adjust the averaging time in steps of 1 [min] between 4 and 30 mins.

“Short” integration time is from 10 .. 50 % of that time, default is 25 % integration time. That value has its own slider which is depicted with a red or green thick bar and the corresponding, sliding minimum and maximum value lighter bars in the upper part of the diagram. This feature can be turned off in the ini/conf-file. See the *corresponding chapter* explaining the parameters.

The red number in the center is the average wind direction, left and right is the min and max (unfiltered) wind angles to either side.

The very thin lines are the unfiltered wind direction input from the instruments.

To adjust the time average, just pull the slider left / right.

The vertical scale is [minutes], short dashes every minute, full horizontal line every 5 minutes.

The instrument has its own timer, so it's independent of the data connection speed or rate.

The idea of this instrument is following: if you sail in puffy, changing winds it allows you to see graphically when the wind changes to the other side. In theory, you should tack, as soon as the wind veers away and crosses the average wind direction

Race Mark is another instrument which can be used to determine if a tack imposes or not. It provides your location on the ladder rungs, giving also the idea is the wind shift a lifter or a header.

6.4.12 TWA to Waypoint

To use this instrument, you need to drop a Tactics waypoint on the chart, using the right-click context menu on the chart canvas. Alternatively, you can place a waypoint in GPS so that it appears to OpenCPN in an RMB-sentence. If you define both, Tactics WP takes the preference.

This instrument requires also the averaged wind data processing running continuously in the background. How to obtain it will be explained below.

TWA to Waypoint
113° - 129° - 151° (zoom)

This instrument is meant to check the TWA on the (new) tack prior to sailing the tack/gybe manoeuvre.

Once you know the TWA angles of your sails, this is very useful e.g. to see which of your downwind sails to use on the next tack.

The instrument shows three data values:

1. minimum instantaneous or short term average TWA
2. long term average TWA
3. maximum instantaneous or short term average TWA

All 3 values are exactly the same data as used in the Average Wind Direction instrument, just recalculated to TWA towards a waypoint.

NOTE 1: one cannot refer from and to port / starboard side, values are simply sorted in the display the low and high.

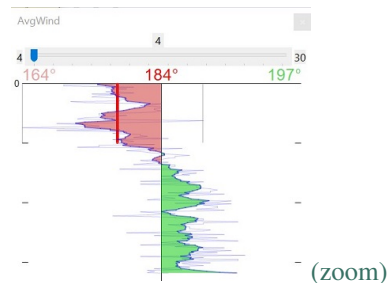
NOTE 2: This instrument is controlled and is thus depending of the settings of the Average Wind Direction instrument. In *DashT* it has been extended to include two range of averaging time, “normal” and “short” time span. This particular instrument using that data is set, in *DashT* to use the “short” time range (like for regatta) by default. Since this feature can be turned off in the Average Wind Direction instrument, this instrument automatically returns also back to “normal”, i.e. to long average wind integration and smoothing time like in the Tactics v1.0.11. It is also possible to select the long integration time permanently for this instrument only, in the ini/conf-file. See the [corresponding chapter](#) explaining the parameters.

Once the wind veers from port to starboard side on full downwind courses the display can look like this:

TWA to Waypoint
154° - 178° - 160° (zoom)

That means, average TWA is 178° and the wind deviates to 154° to one tack and 160° to the other tack.

Default time is 4 minutes, maximum 30 minutes. “Short” integration time can be adjusted (in the *ini/conf-file*) from 10 .. 50% of that time, default is 25% of the overall integration time.



To use Average Wind, both “Short” and “Long” in all *DashT* instruments needing that information, just define the ‘Average Wind Direction’ instrument in your list of instruments, open it once (per session) and move the slider to the integration time of your preference. You can close the instrument afterwards, just keep it in the list of instruments.

More use cases how to use this instrument to select which gennaker/spi or sail for the next tack:

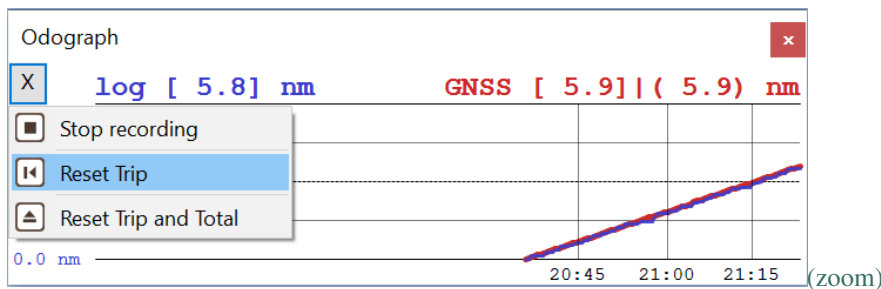
Let’s say we have 2 gennakers on board, the first one start to work at 105°TWA, has its best performance at 120-125° and drops out at 135° TWA.

The second gennaker starts at 115°, has its best performance at 130-140° TWA and can be used all down to 160° TWA.

With the example values (113 – 129 – 151) from the first screenshot above you cannot stay on the direct course line to the WP with the second gennaker all time as it starts at 115° (whereas we recorded 113° TWA...), but overall it fits with its optimum performance (130-140°) much better to the average wind direction (129° TWA) than the first gennaker.

NOTE: *DashT Race Mark* is another instrument which can be used for this type of optimization process in case the race route can be marked in advance: it calculates the TWA values for two legs ahead.

6.4.13 Odograph



This registering odometer is a graphical history instrument like *Wind History* or *Baro History*. Like in all similar *DashT* instruments based on its enhanced *Tactics* race engine, storing the collected data is possible into a comma separated value (CSV) file for later analysis - in this particular instrument all the positions and the calculated distances and the direction between those will be stored.

NOTE: If you find it cumbersome to collect and combine data from various CSV-files after the race, please consider using *DashT time series database streaming* which allows you to collect **all** data for post-race analysis.

The instrument provides two level of trip counters. The first one, illustrated on the upper right corner is persistent, allowing to record and visualize the entire distance of the race course. The second one is a volatile trip counter which is intended to measure the progress within the current leg.

Registration and reset functions are controlled from the upper left corner button which has a symbol > when it is armed, and X when it the instrument is registering data in a CSV file. The above picture depicts the menu items.

By default the odograph is displaying also the boat’s own log data based trip information, *i.e.* a third trip data value is calculated from the boat navigation system’s total log, if it is available. Boat’s navigation system’s eventual trip counters are not read.

A racer is interested in evaluating the performance over the element on which he or she is moving, the water.

With this instrument being ground position based, one can use - or set - boat’s navigation system to get the boat’s speed as STW. One need to check this, of course but typically also the log data would be based on the paddle wheel rotation in this case.

With paddle wheel based log data the instrument allows a graphical comparison over a long period of time between the distance over the water and the distance over the ground. This can give indication about the changes in the water

element on which we are moving, like the strengthening of the current or boat having an excessive leeway because of getting overpowered.

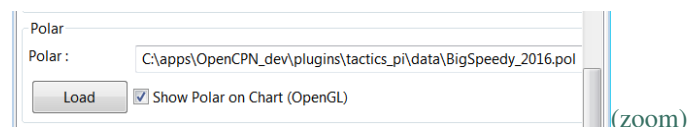
If the boat's navigation system's log is also based on GNSS position data and it cannot be configured differently, then that information has little added value in this instrument, of course. The collection of boat's log data can be disabled permanently, see *Tweaks*.

Cruisers: While this instrument is targeting racers, it can be a handy assistant for any navigation with its two levels of accurate trip counters, frequent measurement based on GPSS data and *OpenCPN*'s own and proven Mercator projection distance calculation.

6.5 With Polar

The instruments and functions in this chapter require your boat's polar.

You can load a polar file by defining one in the Preferences dialog:



Click on the load button and select a polar file.

To clear the selection - to continue without a polar file - one needs to write a text NULL in the *Polar:* field and press *Load* button. An empty field means that you want to select a new polar file.

The format is the same that the *polar_pi* plug-in uses. Use that tool to display the polar, there is no polar visualization in *DashT*.

NOTE: If you do not have a polar, the Weather Routing plug-in of OpenCPN comes with quite a few polars of popular boats. Maybe you can find a polar close enough of your boat in there.

For racers, it is a good idea to prepare a few alternative polars simply by reducing the the polar speed by, say 10 percent: they can be used in bad weather conditions or by a single hand sailor who is not able to push the boat continuously to its limits.

When Tactics is loading the polar file, it is writing it into a two-dimensional static array having 181 lines for each TWA degree (0...180°) and 61 rows (0...60) for each knot of windspeed. It is a simple lookup table for the algorithm usage, having the purpose to reduce the processor load and increase access speed.

Polar Loading Procedure :

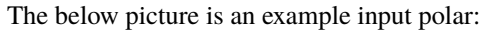
1. The TWA = 0 row and the TWS = 0 column is filled with zeros
2. The rest of the array is filled with NAN values.
3. The values from the polar file are placed at their corresponding spots in the array
4. The missing data in between given values (= not NAN) is then filled with average values.

NOTE: only polars with TWA / TWS / STW, where TWS and STW are in knots do make sense

While there is averaging to fill in the missing values, there is no extrapolating beyond the last valid values.

If you run in a 30kt wind, and your polar ends at 25 kts, then the performance instruments will give you a “no polar data” text. Please consider at that moment (if you have time) to turn on *polar_pi* / *vdr_pi* plug-ins to record new data since you obviously are a serious racer!

The only exception of the extrapolation is the range between the 0 kts windspeed and the first given value, where a simple averaging between the two is done.

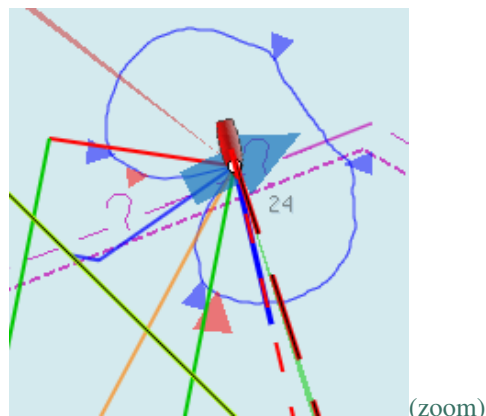


(zoom)

(zoom)

49

6.5.1 Display polar on chart



(zoom)

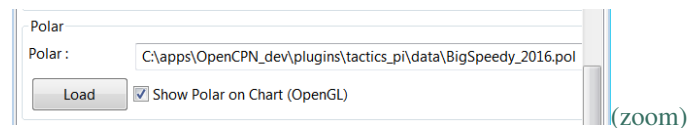
You can display the polar as overlay on the chart (OpenGL is required). The size of the different polar “rings” is normalized, they all have the same size. Nevertheless, the plug-in always shows the current / correct polar ring. The polar orientation is related to the true wind direction, and it shows blue markers for the Target-VMG angles up- and downwind, and red markers for the Target-CMG angles (if you have an active NMEA-RMB-sentence or a Tactics_pi waypoint set).

There’s also a small blue HDT line displayed from the boat. The additional marker line for HDT is only to help you to distinguish which of the two OpenCPN red, default marker lines is for heading and which one is for the course over ground.

To sail Target-VMG / Target-CMG angle, simply steer the blue HDT pointer on one of the Target-VMG / - CMG markers.

As illustrated in the above screenshot, there can be 2 red CMG markers, the preferred one has a bigger size!

To activate the polar display on the chart by default, upon program start, navigate to the Preferences dialogue and set the tick “Show polar on chart (OpenGL)”.



(zoom)

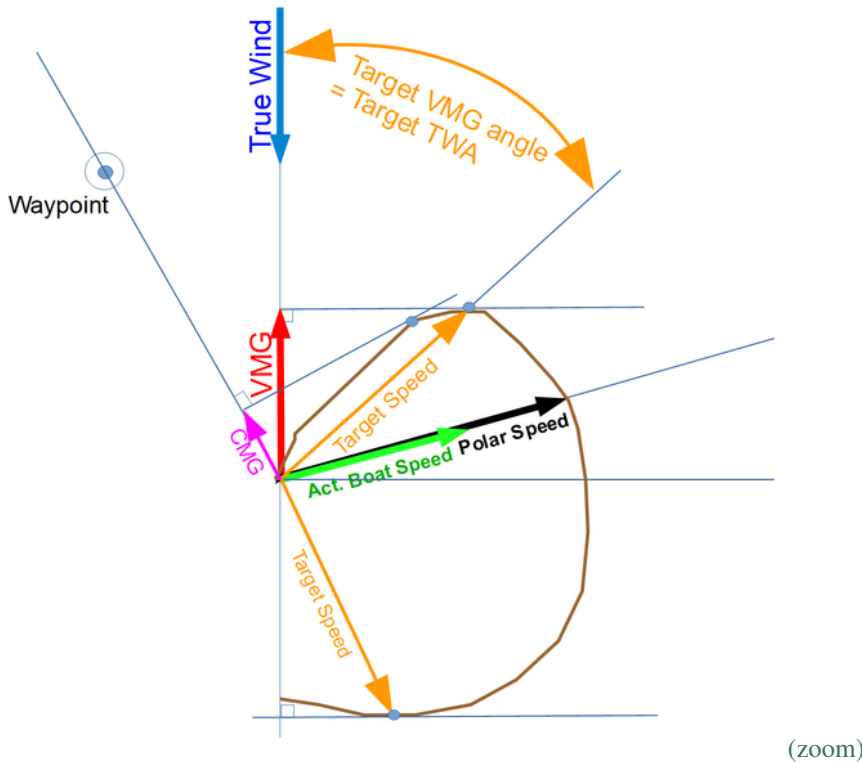
The preference is only setting the default; while navigating in OpenCPN, you can turn the polar overlay display on / off as you like: to toggle the polar display on the chart on/off, right-click on any *DashT* instrument and select or unselect “Show Polar” from the *contextual menu*.

6.5.2 Performance data

Tactics performance algorithms are available plug-in wide and used, among the other *DashT* instruments in following original Tactics instruments.

6.5.3 Vector graph

Let's first take a look the following vector chart as for reference on the different terms in relation to the polar curve:



NOTE: We need to presume that you are familiar with the above terminology - if not, read about **VMG** to start with - but if you do not agree, say with the **CMG definition** you are free to change the abbreviations used in the application using the ini/conf-file settings, see the *corresponding chapter* explaining the parameters. All requests to change the nomenclature will be rejected since one has this option at his/her disposal. The vector algebra does not move.

Some of the instruments are not showing a single value like on Dashboard, both the value is split in two, displaying first a percentage value and then the value from which the percentaget is taken. Let's take an example of the polar performance, displaying the percentage of your current speed of the polar speed of your boat at that moment:

6.5.4 Polar speed



This instrument is useful in crosswind / reaching conditions without a waypoint.

It shows the optimum speed for the given wind conditions.

This is simply the reference of what speed we should be able to sail based on our current TWA / TWS values. The % value is the reference to STW. In this example, we are currently doing (only) 51%, of what the polar has stored as optimum value. According to the polar we should be able to do 12.17 knots.

This is “*Act.Boat Speed*” compared to the “*Polar Speed*” in the [vector graph](#).

Values below 100% mean that we are slower than the polar says, above 100% means that we are faster than the polar - if that persists, we should run the *vdr_pi* plug-in or *DashT* database streamer to record the new data since obviously our polar is not good.

NOTE: while the actual boat speed can be greater than the polar speed, the percentage value shown is limited to maximum of 150 % - if you see values like this, there is something wrong in the polar file, in the STW/TWA/TWS values or both.

DATA: the calculated data is made available to other instruments as OCPN_DBP_STC_POLPERF and OCPN_DBP_STC_POLSPD: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process. However, the generation of *NKE-Style performance records* takes place every second based on the current data at that moment even if this instrument is available or not.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record names are `performance.polar.polarSpeedRatio` and `performance.polar.polarSpeed`, respectively.

6.5.5 Actual VMG



The actual VMG is the “*Velocity made good*” referring to the wind direction. The means we’re moving with 6.27 kts into wind direction. Also works when we sail downwind, in case of which it is off the wind.

$VMG = STW * \cosine(\text{True Wind Angle})$

DATA: the calculated data is made available to other instruments as OCPN_DBP_STC_POLVMG: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.actual.velocityMadeGood`.

6.5.6 Target VMG Angle



Also known as “*Target TWA*”; this is the optimum TWA (True Wind Angle) when sailing upwind or downwind for a given wind speed, based on your polar data. Very useful when sailing up-/downwind without a waypoint.

The program simply searches the polar with a given TWS for the optimum TWA up-/downwind. It is defined as the tangents on the polar.

DATA: the calculated data is made available to other instruments as OCPN_DBP_STC_POLTVMGANGLE: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.target.targetAngle`.

6.5.7 Target VMG



Also known as “*target boat speed*” (target speed in the diagram above), this value is the reference to the Target VMG-Angle. In our example it means : If we would sail with 164° TWA (from the example above), then we could make 11.95 knots according polar), but currently we’re doing only 51% of that.

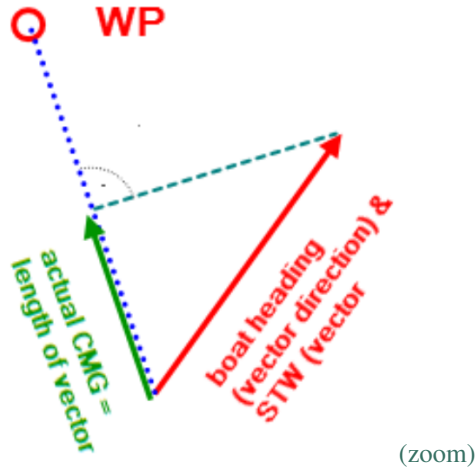
DATA: the calculated data is made available to other instruments as OCPN_DBP_STC_POLTVMG: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.target.velocityMadeGood`.

6.5.8 Actual CMG

Actual CMG
8.36 (zoom)

Actual Course Made Good, aka VMC: the component of your boat speed towards a waypoint. We're moving with 8.36 knots towards that waypoint.



Quite valuable to know on reaching courses towards a waypoint.

$$CMG = STW * \cos(Heading - MarkBearing)$$

DATA: the calculated data is made available to other instruments as `OCPN_DBP_STC_POLCMG`: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.actual.courseMadeGood`.

6.5.9 Target CMG Angle

Target CMG-Angle
130° (zoom)

Optimum angle to sail fastest to a waypoint, based on your polar data (Like VMG, but not up-/downwind but towards a waypoint).

DATA: the calculated data is made available to other instruments as `OCPN_DBP_STC_POLTCMGANGLE`: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.target.courseAngle`.

6.5.10 Target CMG



Same as Target VMG, but towards a waypoint. Means : “If we would sail 130° (Target CMG Angle, from ex. above), we would move towards the waypoint with 11.98 knots, but currently we’re only doing 64% of that.

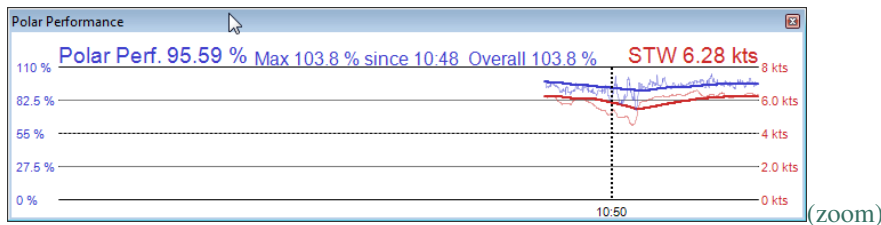
NOTE: Calculation is verified, but it doesn’t tell you if you are on the correct tack.

DATA: the calculated data is made available to other instruments as OCPN_DBP_STC_POLTCMG: The instrument must be running (but not visible) - this is asynchronous, data arrival driven calculation, not a background process.

RECORDING: in *InfluxDB Out* by default is **disabled**, when *enabled*, the default record name is `performance.polar.target.courseMadeGood`.

6.5.11 Polar Performance

A great sail trimming aid:



A graphical history instrument like *Wind History* or *Baro History*.

The history instrument plots the STW (speed through water) as percentage of the polar speed data (=100%) for the actual true wind speed TWS and true wind angle TWA. It is this comparison in the polar chart above, plotted as percentage.



The idea is to provide a simple sail trimming aid, as the percentage value is quite stable in comparison to the real speed values. The TWA / TWS is constantly adjusted while reading the polar data.

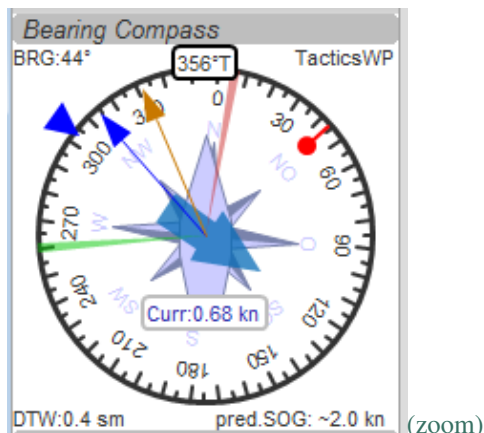
As long as the filtered curve continues to climb or is 100%, or even better your trim adjustments are correct! But if the curve points down, you’re sailing is getting worse and you better do something for it!

NOTE: while the actual boat speed can be greater than the polar speed, the percentage value shown is limited to maximum of 150 % - if you see values like this, there is something wrong in the polar file or in STW/TWA/TWS values.

DATA: the graphical history instrument is not sharing its data with other instruments.

RECORDING: the data can be recorded only in CSV files as with other graphical instruments. If you find it cumbersome to collect data from various CSV-files, please consider using *DashT time series database streaming* which allows you to collect **all** data for post-race analysis or even to display it on-line, when underway. It is enough to have the performance and other instruments in their single line, numerical format and to activate their recording in *InfluxDB Out* to save the raw values calculated by this instrument and to do the statistical calculations using the various Flux mathematical functions either in InfluxDB or on the Grafana dashboard, both available off-line or on-line.

6.5.12 Bearing compass



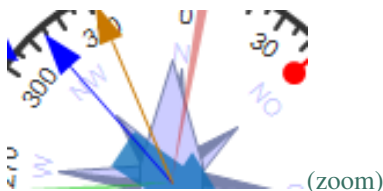
This instrument is inspired by NKE's tactics page, and is called here as "*Bearing Compass*". Let's walk through this feature rich instrument:



The uppermost, numerical embedded display shows the true heading (HDT), in the above example 356°T.



The center part of the instrument depicts a blue surface current arrow, based on boat heading (HDT) and shows the current speed, “*Curr: 0.68 kn*” in our example.



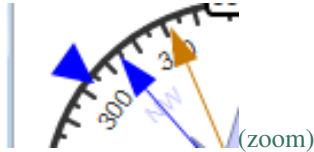
Furthermore we have the blue, thin arrow, which is TWA on boat axis. It also shows the TWD on the degree scale (315° in the example) and the AWA arrow in orange/yellow.

NOTE: Many colors, like the orange/yellow are inherited from the Dashboard dials and defined in OpenCPN - please not that where *DashT* provide a way to customize the Dashboard dials, the customization will take also place in Tactics dials for this reason. However, blue and red arrows, layline and red dot colors cannot be changed. Up to you to find the right combination of colours from the OpenCPN color palette.

You see the red/green laylines, which are based on COG. As with the laylines on the chart, the second layline shows you where you end up sailing the same TWA on the other tack. Leeway and current are taken into account. Use the second layline together with the waypoint marker described below.

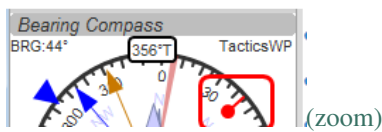
NOTE: The calculation is based on TWA. Especially when sailing downwind with a gennaker, your apparent wind angle depends very much on the speed of the boat. As soon as the gennaker start working

aerodynamically, it'll speed up the boat and your AWA will show lower values (points more forward). If you gybe now, your boat speed will drop, and although sailing the same TWA, your app. wind angle will be higher than before. You will have to bring you boat back up to speed to see the same AWA than before the tack. This can be tricky when you're close to a buoy and don't have much space/time to speed up your boat again.



The blue triangle outside the degree scale is the Target-VMG Angle (Target TWA).

Simply adopt your course to place your blue TWA-arrow on the Target-VMG pointer, and you sail optimum (polar based) speed up-/or downwind.



If a waypoint is active, either by a NMEA-RMB sentence from your GPS or the temporary tactics WP which you can place on the chart, you will see the WP as a red dot.

NOTE: The manually placed tactics waypoint overrides any simultaneously available RMB-sentence defined waypoint, including those RMB sentences created by the OpenCPN routing functions.

Change your course and place it towards the red dot and you will directly bump into your waypoint marker.

You can also use the second layline to determine when it is time to tack so that you are sure to make it around the waypoint: the red dot should be outside the second layline before doing so.

The upper part of the bearing compass will give you

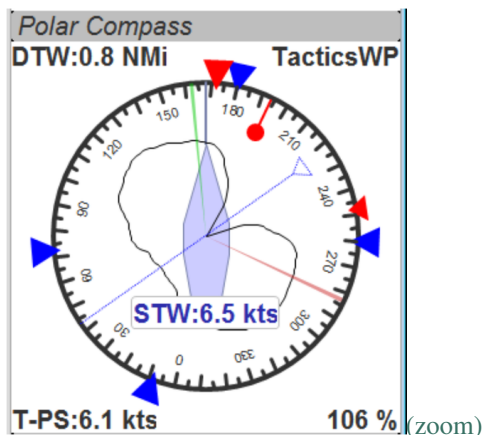
- BRG bearing angle to the waypoint
- TacticsWP = name of the waypoint (if Tactics Waypoint dropped, always this)



In the lower part of the Bearing Compass instrument will give you

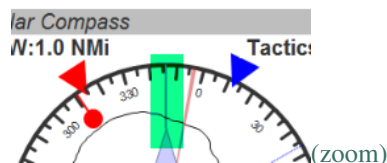
- DTW = distance to the waypoint
- predicect speed over ground (SOG) *on the other tack*.

6.5.13 Polar Compass



This instrument is derived from Bearing Compass. If you a polar loaded, it shows you the actual polar ring. The size of the ring is normalized, *i.e.* it has always the same size.

The polar is rotated with the TWD, which is shown as thin blue line here. In this example True Wind Direction is 226°, the wind is blowing from port aft, boat's true heading being 168°.



The thin blue vertical line, highlighted in green color in the above picture is indicating the true heading, HDT.

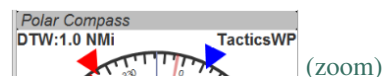
The 4 blue markers (triangles) are showing the Target-VMG angles, up- and downwind. They are based on the actual polar ring and are moving with the polar.

The red marker(s) (triangles) are the Target-CMG angles towards the active waypoint. They are only shown if you have an active waypoint set either via NMEA-RMB or by dropping the Tactics waypoint on the chart.

In contrary to the Bearing Compass, the VMG/CMG markers are shown in conjunction with the polar and are rotating with the polar/true wind angle.

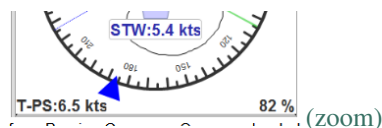
The red dot inside the compass shows the bearing to the waypoint, like in the Bearing Compass.

To sail optimum VMG- or CMG-Angle, change your course in a way that the boats heading line (the thin blue line from the bow) points on the marker of your choice.



Top left and top right data fields are showing:

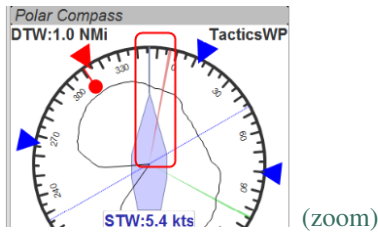
- DTW = Distance to Waypoint
- TacticsWP = name of the waypoint (if Tactics Waypoint dropped, always this)



The lower center numerical data is speed through water (STW).

The two bottom fields are

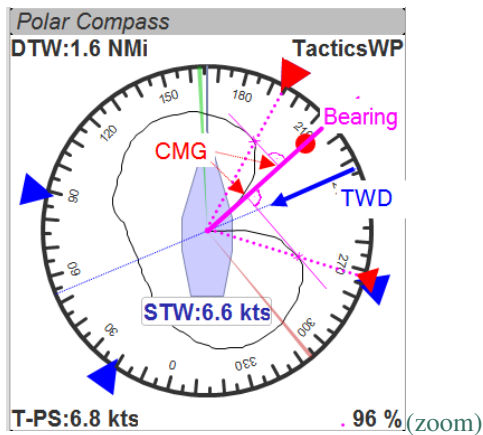
- T-PS : Target Polar Speed
- PolarSpeed-% = the percentage of you actual STW compared to the Target Polar Speed



Furthermore you see the laylines which are based on COG. In this example, there is a angle between the HDT line and the layline, meaning that we have a significant drift!

NOTE: There may be two red CMG pointers on the outer ring. They are based on Bearing and True Wind direction. Generally one of them is preferred, because you approach the waypoint faster following it. The preferred one arrow has a bigger size!

Let's walk through the below example use case in the Polar Compass:



We have TWD, shown with the blue arrow. The polar is rotated with TWD.

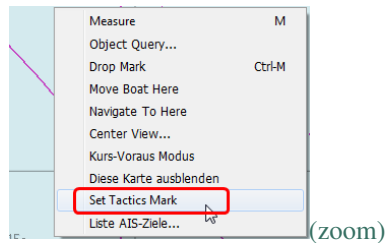
Next, we have the bearing to the waypoint, see the solid purple line “Bearing”.

From this solid purple bearing line, we have to find the tangent on the polar curve to both sides. The tangent is - in relation to the purple bearing line – the highest point of the polar curve.

Graphically one draws perpendicular lines from the purple bearing line to either side until it just touches the polar curve.

The length on the purple bearing line, measured from the 0-point of the polar, to the perpendicular intersections (marked CMG) corresponds to the Target CMG speed.

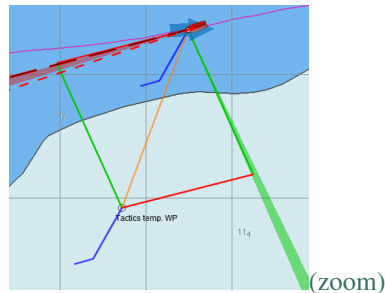
6.5.14 Temporary waypoint and Target-TWA laylines



You can right-click on any place on the chart and drop a temporary Tactics waypoint (exactly one).

NOTE: this OpenCPN function can be a bit tricky and the point does not always go where you think you clicked (but you can move it by dragging it). You can attempt to keep the cursor strictly within the limits of the dialog depicted above: OpenCPN remembers the last cursor position on the chart when you finally click the menu item, which is quite far from the original click position.

As soon as you activate the layline display - or if it is already activated, the plug-in will do a Target-TWA calculation to that WP, based on the current TWD and your boat polar. Surface current is taken into account.



You can delete that waypoint as any other WP. Select it with right click and choose “Delete”. You can also drag the waypoint on the chart, it behaves like a normal waypoint and appears in the waypoint manager.

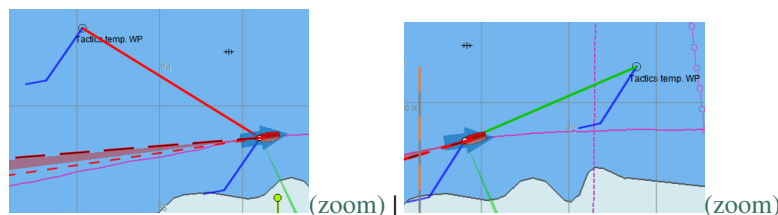
The chart functions takes the Target-VMG (Target-TWA) angle up-/downwind and applies it to our boat as well as to the temporary Tactics waypoint mark.

If there is a line intersection, it chops off the lines at the intersection so that you remain strictly in the perimeter defined by the temporary Tactics waypoint mark.

Colours green and red are again the wind directions green = wind from starboard, red = wind from port.

Additionally, Tactics is doing a polar based calculation to see if the direct course would be faster compared to the Target-VMG calculation.

In that case you'll get a red or green line directly to the waypoint. Colour depends on the side the wind is blowing from. Wind from port → red, wind from starboard → green



You may have noticed that Tactics is placing a wind barb also on the temporary Tactics waypoint mark.

NOTE: In the contrary to the weather routing plug-in, Tactics is not, explicitly, using grib files for current/wind info. The temporary Tactics waypoint mark is meant for a quick, near run around a buoy, cape of an island, etc. using the surface current and the live wind data we currently are experiencing. Just

drop a mark on the chart and off you go. Delete it, drop it somewhere else, and boom, off you go again in agile sailing! Maximum one tack/gybe per mark dropped, not more.

6.5.15 NKE-style Perf.Records

NKE supports the upload of specific performance data to their instrument bus, which can be shown in their display devices. These records are polar based and unless you're using their (quite expensive) regatta processor, Tactics gives us an easy way to display e.g. the *"Target Polar Speed"* outside in the cockpit on the standard instrument displays.

NOTE: Due to the lack of information on other manufacturers' systems interfaces or capabilities, Tactics implements this for the NKE systems only for now. Also, be aware that OpenCPN can only import/export NMEA183 right now, but not NMEA2000 or SeaTalk. Any use cases allowing to overcome these limitations will be welcome!

The following five NKE-records are implemented :

Speed and performance target

```
Speed and performance target
$PNKEP,01,x.x,N,x.x,K*hh<CR><LF>
|           \ target speed in km/h
|           \ target speed in knots
```

Course on the next tack

```
$PNKEP,02,x.x*hh<CR><LF>
|           \ Course (COG) on other tack from 0 to 359°
```

Opt. VMG angle and performance up and downwind

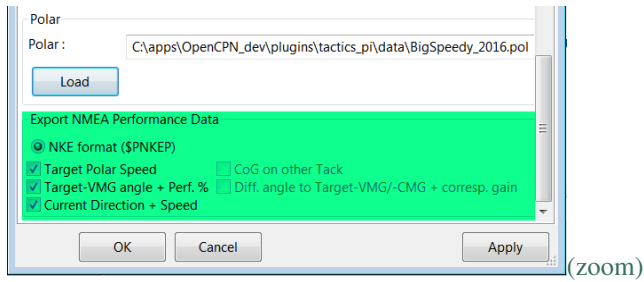
```
$PNKEP,03,x.x,x.x,x.x*hh<CR><LF>
| |           \ polar speed performance TWA/TWS from 0 to 99%
| |           \ performance upwind or downwind from 0 to 99%
| |           \ opt.VMG angle 0-359°
```

Angles to optimise CMG and VMG and corresponding gain - available but has not been verified due to lack of a compatible NKE-instrument:

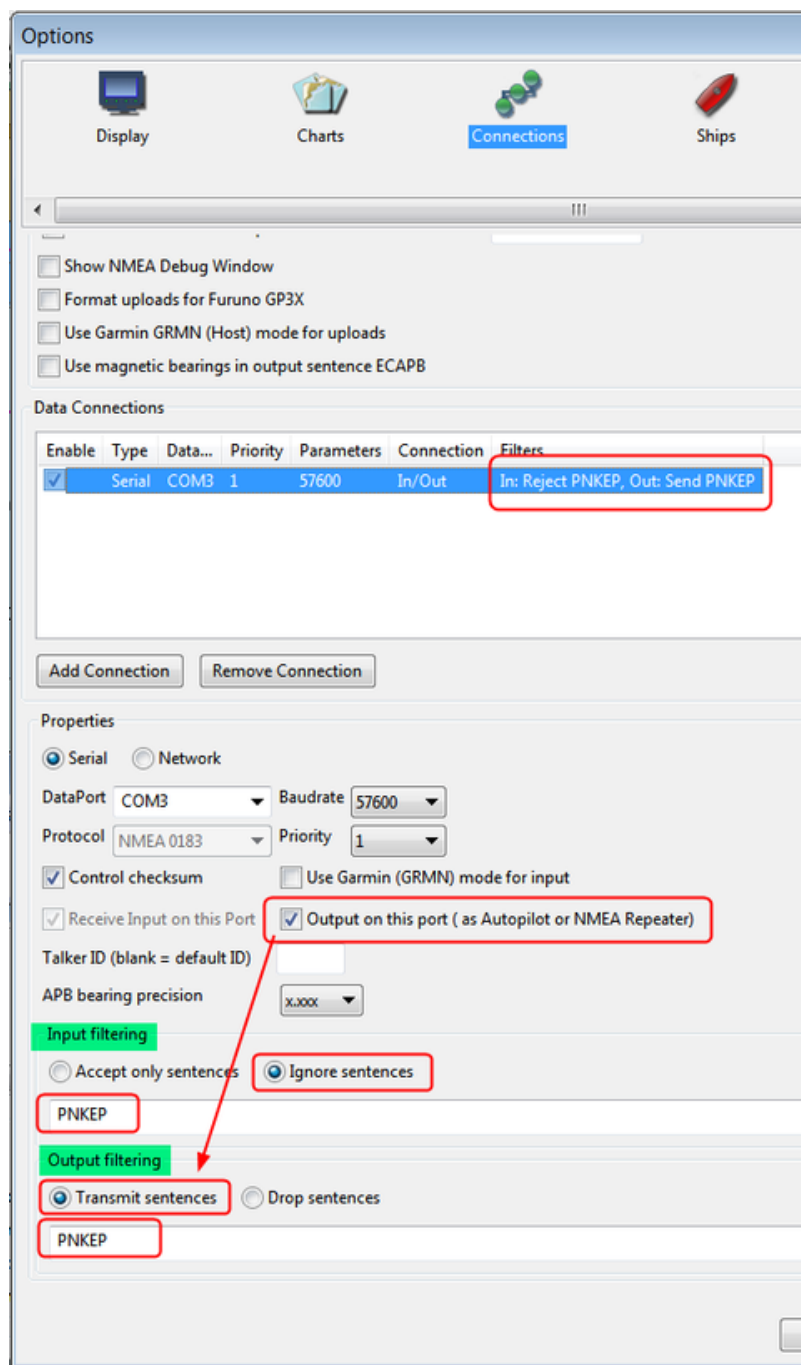
```
$PNKEP,04,x.x,x.x,x.x,x.x*hh<CR><LF>
| | |           \ Gain VMG from 0 to 999%
| | |           \ Angle to optimise VMG from 0 to 359°
| | |           \ Gain CMG from 0 to 999%
| | |           \ Angle to optimise CMG from 0 to 359°
```

Direction and speed of sea current

```
$PNKEP,05,x.x,x.x,N,x.x,K*hh<CR><LF>
| |           \ current speed in km/h
| |           \ current speed in knots
| |           \ current direction from 0 à 359°
```



These five NMEA183-records all begin with a **\$PNKEP** and are created on the fly using the data calculated in the plug-in and are sent to OpenCPN's NMEA stream. To send the records to your instruments, you have to define an outgoing connection in your Interface connections, e.g. like this :



Set an output filter as shown above, filtering for PNKEP. After the set up, you should see records beginning with \$PNKEP, in your NMEA debug window.

NKE exports the \$PNKEP sentences as soon as they are available on the topline bus. Normally they are calculated in their regatta processor and then exported to the PC.

Therefore ignore all incoming \$PNKEP-sentences! Click on Input filtering (see screenshot above), select Ignore sentences and add PNKEP.

Do not forget to re-init your NMEA data stream in your instruments, to make sure the new records are accepted.

NOTE It can be quite challenging to teach to a NKE TL25 mast display these new sentences since their frequency is not very high. So you need to be patient and try several times.

6.5.16 Data Export

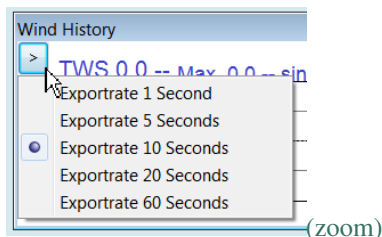
This function which allows to write data in CSV (comma separated values) file is originating from Tactics, therefore it is explained here. It has been implemented also in Wind History and Baro History instruments, which are under Dashboard instrument category. Here we use the Polar Performance instrument for an example.

NOTE: the preferred method to save data in *DashT* is *InfluxDB v2.0 time series database*, but if one does not want to use it the exported CSV-file recordings allow information limited to a single historical data instrument saved for for importing to a spreadsheet, Jupyter Lab, on-line services, etc. Please see the [distribution repository's examples for developers using CSV-files](#).

For these three instruments, there is a button for this purpose, placed on the upper left corner:



Once you click the '>' button, a menu pops up where you can select the export rate (the latest setting is saved to `opencpn.ini` and will be set as default at startup for the next usage).



After selecting one of the export rates, a file selection dialog is opened asking from you for an export file name.

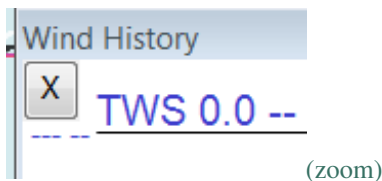
If you give it an existing name, the data will be appended (added) at the end of the file.

If the file does not exist, it will be created.

Like the export rate, also the filename is stored in `opencpn.ini` and set as default for the file selection dialog.

There is NO logical check on the content of the file if you decide to use an existing file!

Once the export is running, the button changes to X :



To stop the data export, click the X . Export stops immediately.

The export files always contains the current date, local time (incl. sec.), and the instruments data; the first line in the file is always the headline with the field description. A standard wxWidgets function is used to format date and time, taking your local date/time settings into account. You can use Dashboard instruments to check the local time of your computer and compare it to the stored values.

The data separator can be set, it does not have to be a comma, in this Polar Performance export example it is a semicolon ;

```
Date;Time;AvgTWA;AvgTWS;smoothed BoatSpd;Percent
26.07.2019;18:11:04;159;11.8;4.30;60.78
26.07.2019;18:11:05;159;11.8;4.31;60.87
26.07.2019;18:11:06;159;11.8;4.31;60.91
26.07.2019;18:11:07;159;11.8;4.32;60.93
26.07.2019;18:11:08;159;11.8;4.32;60.97
26.07.2019;18:11:09;159;11.8;4.32;60.97
26.07.2019;18:11:10;159;11.8;4.32;60.96
26.07.2019;18:11:11;159;11.7;4.32;60.96
```

(zoom)

NOTE: To reduce the system load, the internal timer for the baro history instrument is set to 5 seconds, i.e. only every 5 seconds, the display is updated. This should be good enough for barographs which generally export their data at even lower rates.

(zoom)

“Prepend Clockticks & Prepend UTC Timestamp” : With these ticks you can add Clockticks and a UTC Timestamp to the data export, at the beginning of each line of exported data.

This can be useful if you intend to process the data afterwards, e.g. via InfluxDB, Grafana, etc.

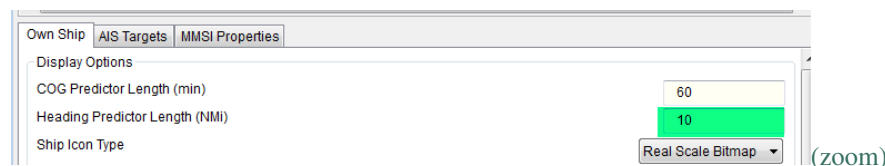
“Data Separator” : You can define and set your own column separator here, instead of the ini-file method described above. A Tab is represented by \t if it is required as separator.

6.6 Tactics terminology

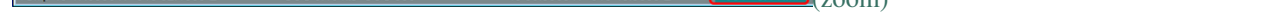
Term	Description
AWA	Apparent Wind Angle; the relative wind angle measured by your wind sensor, related to the boat axis(0°...180°)
AWS	Apparent Wind Speed; the relative speed of the wind measured by your wind sensor
CMG	Course Made Good; the relative speed approaching a waypoint. This term may be customized to whatever you like. See the <i>ini/conf-file chapter</i> for “ <i>CMGSynonym</i> ”
COG	Course Over Ground; generally supplied by the GPS
HGD	Magnetic heading of your compass; not compensated with mag. variation
HDT	True heading of your compass. “True” means compensated with magnetic variation
Heel	The angular degrees how your boat is heeled (leaning) sideways due to any force from outside (wave, wind, water ballast on one side of the boat, swing keel etc....)
CRS	Course through water; HDT + Leeway, but without currents
Lee-way	The drift of your boat caused by the wind. As soon as the wind is blowing it implies a force on your boat, the boat starts drifting. Leeway is not including any drift due to surface currents ! That’s actually the challenge :-) you can manage with Tactics!
SOG	Speed Over Ground; generally supplied by the GPS
STW	Speed Through Water; the info that is returned by your “paddlewheel” sensor
Tar-get CMG	The optimum speed / angle towards a waypoint; aka VMC
Tar-get VMG	The optimum speed / angle up- or downwind with reference to the true wind direction (without a waypoint)
TWA	True Wind Angle; the angle of the true wind relative to the boat axis (0°...180°). The unit then gives you the direction as “>”=port, “<”=starboard
TWD	True Wind Direction; true wind direction related to the compass rose (0°..359°)
TWS	True Wind Speed; the speed of the wind in the atmosphere
VMC	Velocity Made on Course; same as CMG
VMG	Velocity Made Good; the relative speed up-/downwind with reference to the true wind direction. This term may be customized to whatever you like. See the <i>ini/conf-file chapter</i> for “ <i>VMGSynonym</i> ”

6.7 Align your compass

- Swing your compass as described by the manufacturer
- Connect your GPS to O to get a stable position
- Make sure you have true heading available (use wmm_pi, in case you don’t get the mag. variation from the GPS)
- Directly in OpenCPN, set your heading predictor to a high value, e.g. 10 miles



Put the mouse onto the (thin HDT) predictor line towards the end of the line (the long line reduces the error).

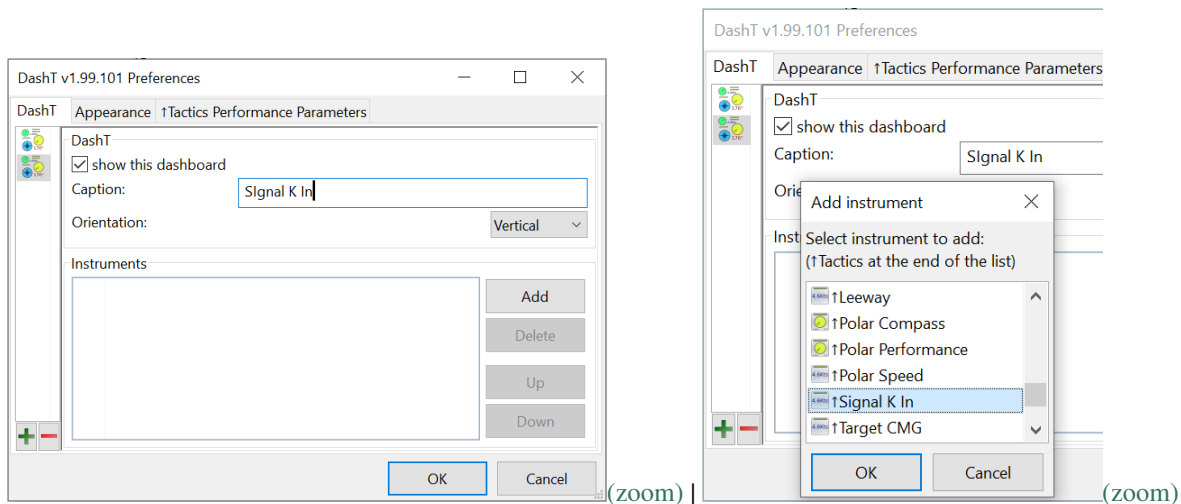


SIGNAL K IN

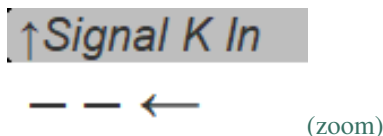
One single instrument but so important that it deserves its own chapter!

The instrument is not reading one single instrument but it is actually controlling a data streamer which reads data from a *Signal K server node*. This alternative data source to *OpenCPN* is feeding *DashT* and its instruments with Signal K data from a delta channel (delta for changed data). The fast, time stamped data source is an alternative to *OpenCPN* to get data normally not visible to *OpenCPN*.

This is how you start Signal K input stream, by creating *Signal K In* instrument. It is recommended to have its own, dedicated window pane with only one of these single line instruments in it (see below the troubleshooting section to find out why). You can have multiple instances of this instrument type but only the first one is going to do anything else but showing the data rate.



This is what you are going to see when the *Signal K In* instrument is starting, it means that it is waiting for a connection to a *Signal K server node*:



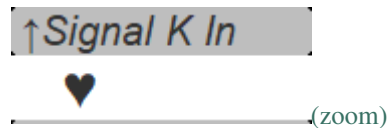
7.1 Signal K server node

As you may have learned by now, if there is no *Signal K server node* the *Signal K In* instrument will remain forever in the waiting state.

OpenCPN v5.2 and superior supports Signal K data and therefore also the *Signal K server node* as data source. If you have set *OpenCPN* for Signal K data and you still do not get any data in *DashT*, please read the operation principles below to understand that *DashT* asks only NMEA-0183 data from *OpenCPN* but for Signal K data it requires a direct, dedicated connection to a *Signal K server node*.

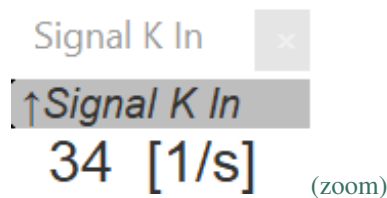
NOTE: *Signal K* is a data format. *Signal K server node* is a data processing and interchange server which is delivering data in Signal K data format. *DashT* requires a *Signal K server node* for Signal K data.

In case you do not have a *Signal K server node* please read the *OpenCPN* documentation for supplementary software, which has a section for [Signal K server node](#) which gives you the right pointers for your operating system installations.



If there is a *Signal K server node* available in the local network of your computer, you will see the above heartbeat pulsing on the instrument display. Data is coming in already, at this point.

In case there is an issue with this and you find out that all components are there, please see the troubleshooting section further down of this document for instructions.



Once enough data has been received to make meaningful statistics, the data rate is displayed. It is showing data values received, in average and per second. The value shown is not corresponding the data rate value which one can observe on the *Signal K server node* dashboard. It counts messages *Signal K server node* receives but *Signal K In* streamer counts valid data **values** it has subscribed to and which it effectively receives.

NOTE: In some of the *Signal K* data structures there are more than one data value per message, for example in the position data latitude and longitude: *Signal K In* streamer counts each of the data value since this is the way they are internally streamed to the instruments, one by one. Therefore the value is presenting the effective data rate all the way to the instruments.

While you are all set now, it would be perhaps interesting to understand more about the Signal K data and how it arrives in *DashT* and how it is used there, so please continue reading.

7.2 NMEA-0183 data type

As discussed in [NMEA-0183 Data](#) chapter, the origin of data in *Signal K server node* can be also a [SignalK data source](#) with NMEA-0183 data type.

In this case the data arrives to your instruments exactly the same way through *Signal K In* streamer than it would come via the *OpenCPN* plug-in interface. Only that you get it faster and with timestamps set by the *Signal K server node*, a feature mandatory for the usage of any time based database or analysis software.

Logically speaking, there is nothing to gain to get this same data via OpenCPN - one would just get an extra delay in the data path. Even if one has a NMEA-0183 based navigation data system, *Signal K In* streamer is the preferred way to get this data in *DashT*.

7.3 NMEA-2000 data type

To get access to a rich and flexible data source is the main motivation for *DashT* to interface with *Signal K server node*. If available, it provides much higher data rates and wide variety of new data parameters. The *Signal K data* format unifies the access to this data source.

NMEA-2000, a derivation of the CAN data bus is the most likely commodity off the shelf (COTS) source for engine and energy data on boats with recent electronics (albeit it may have a different commercial name for © and ® business reasons). Shortly, what the CAN-bus is doing in your car, NMEA-2000 is used to the same in your boat. NMEA-0183 remains, of course a data source for *DashT* instruments but EngineDJG instruments, for example require Signal K data which simply is not available in NMEA-0183 coming from OpenCPN.

7.4 Other data types

Currently, *DashT* does not support GPIO, I^2C , BLE or other *Signal K* data types other than NMEA-0183 and NMEA-2000.

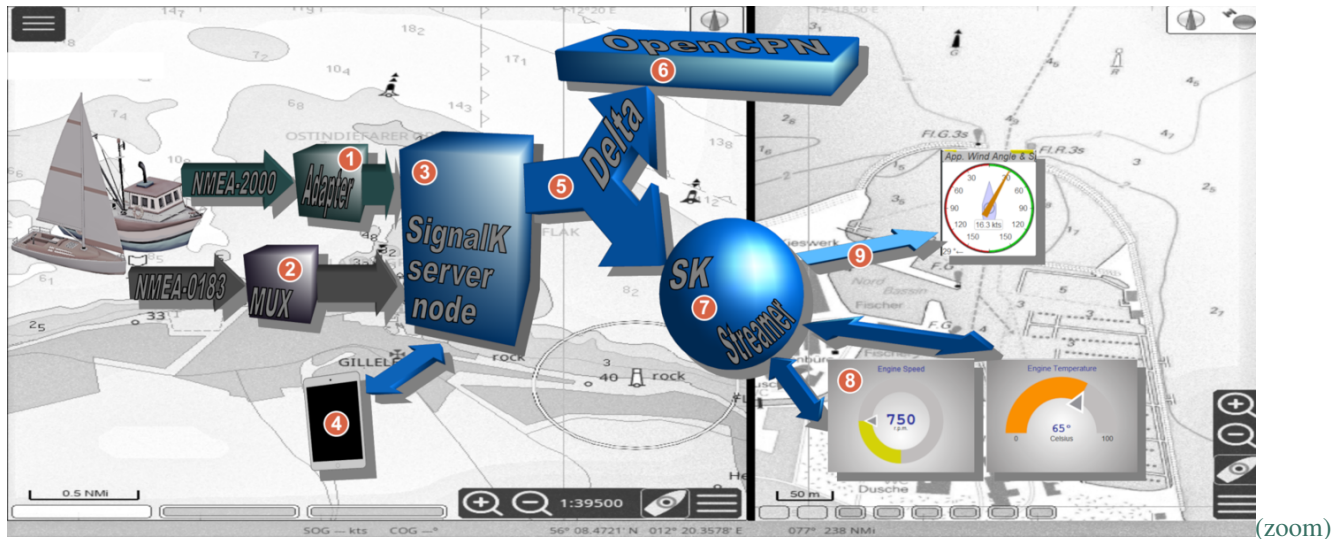
There is no other particular reason for this other than there is not enough information and use cases available for other data formats which would allow testing of them. From the *DashT* instruments' point of view the data can be from any source and any type. However, *SignalK server node* sets, in *delta sentence* the data source and the type, which also changes the data structure: currently *DashT* has been taught to parse only NMEA-0183 and NMEA-2000 type of sentences. If a new interesting data source and type emerges, it would be quite straightforward to add parsing of its data structure into *Signal K In* streamer, for example for energy data and similar.

7.5 Signal K data interchange

Signal K, a modern and open data format for marine use is a protocol understood by *DashT* for data interchange over the network. *Signal K keys* are defined for each data source, called below **data paths**. A *DashT EngineDJG* instrument, for example **subscribes** to the data coming from one of the data path using a Signal K key.

NOTE: Unlike with OpenCPN, where all data is pushed to everybody who has asked for NMEA-0183, with *Signal K server node* and *DashT* support for it, the instruments are subscribing to data. In other words, if you have only one instrument, showing one data value, *Signal K server node* needs to send only that data to *DashT*, nothing else. This, obviously reduces the power consumption and allows your computer to run more efficiently.

7.5.1 How it works



- (1). NMEA-2000 databus is the source of the engine and energy data for the EngineDJG instrument.

There may be other, potential data sources which can be enabled in the future, such as IoT capable sensors, Bluetooth Low Energy (BLE), I^2C and General Purpose I/O pins (GPIO). For now, only NMEA-2000 is discussed as data source and type but *DashT* does not require NMEA-2000 in particular, just *Signal K* data.

- (2). In case the NMEA-2000 databus does not provide navigational data, needed by *OpenCPN* and displayed by other *DashT* instruments, your boat probably sports also a classic NMEA-0183 wired sensors and instruments. They end up, typically into a multiplexer (MUX), which interfaces simply with the *Signal K* server node either by a USB connection, Ethernet connection or WiFi.

- (3). *Signal K* server node, or a commercial *Signal K* data enabled router / multiplexer is entirely network enabled and can locate anywhere in your boat, not necessarily on the same computer where one is running *OpenCPN* and *DashT*.

Signal K data format is a standard for data interchange. A server implements and a client uses that data format. For example “*Signal K* server node” is not “*Signal K*”, it is one of its implementations but there are others. See [this list](#) of open source and commercial products available.

- (4). For example, one can have immediately access from the cockpit tablet to the rich set of instruments, plug-ins and features *Signal K* server node provides

- (5). *Signal K* standard defines and a server node provides, among its other networked data interchange interfaces, so-called Delta-channel, where all the changes in the boat data is available. When the boat has a NMEA-2000 databus this usually means a lot of data and many *Signal K* keys. Clients can subscribe to all of this data or to some selected keys only.

- (6). *OpenCPN* has developed an interface to *Signal K* data, using a different interface than *DashT* is using. Of course, *OpenCPN* is not a consumer of the engine data, and it is not even remotely interested in knowing if the ice cube machine is still working. But it needs the time, position and other navigational data for its routing and map functions. Also, it needs to feed the majority of its plug-ins with NMEA-0183 or *Signal K* data. For that it is using an internal (*i.e.* fast) multiplexer gateway.

NOTE: Even if you decide not to use *Signal K* data in *OpenCPN* but NMEA-0183, please be aware that a *Signal K* server node is able to provide all the NMEA-0183/AIS data to the chartplotter (and other clients) also in NMEA-0183 format - and still continue to feed *DashT* with *Signal K* data only for those values *DashT* subscribes to.

(7). *DashT* is a plug-in for *OpenCPN* chartplotter, containing an efficient, built-in Signal K data streamer. It subscribes to *all* data a Signal K server sends over the Delta channel to start with, then asks the instruments what of this data available they are interested in, subscribing to those data keys only. When data arrives, it is distributed to subscribers over a C++ method call-back mechanism (*i.e.* very efficiently). This way, *DashT* is making gain in speed and lowers the number of network connections to the Signal K server node, reducing its workload as well as that of your computer.

(8). The *DashT EngineDJG* instrument comes only in one flavour, unlike other *DashT* “traditional” OpenCPN Dashboard instruments which are hard-coded for their intended usage. An EngineDJG instrument is configured after its creation. The user of the instrument is provided with a list of available Signal K keys. Only one of the keys can be selected per instrument. The origin of the data is not required to be NMEA-2000 data source, but it most probably is. Once set, the instrument is subscribed to a Signal K key and show, say port engine rotation speed in r.p.m. There is no limitation other than the screen size for the number of these instruments.

(9). The traditional Dashboard instruments, such as wind data and similar are subscribed automatically to the corresponding Signal K key. If it is not available, they will receive the data as before, from the *OpenCPN*’s NMEA-0183 distribution channel.

7.6 Configuration

The default configuration file is in JSON-format (like Signal K data). The default values are good for normal, local-only operation. It may require changes in case when things are not working. It is located:

Windows \ProgramData\opencpn\plugins\dashoard_tactics_pi\streamin-sk.json

Linux ~/.opencpnplugins/dashboard_tactics_pi/streamin-sk.json

Typical changes would be to change port or the location of your *Signal K server node* and its delta channel.

```
"streaminsk" : {
  "source"      : "localhost:8375", // not limited to localhost
  "api"         : "v1.20.0",       // version of Signal K server
  "connectionretry" : 5,           // [s](min.=1s to reduce CPU load)
  "timestamps"  : "server",       // Signal K "server" or "local"
  "verbosity"   : 1               // 0=no, 1=events, 2=verbose, 3+=debug
```

NOTE: The configuration file is read only in during the startup so you would need to restart OpenCPN after modifying this file.

7.7 Troubleshooting

When debugging or searching for a probable communication issue you would set the verbosity parameter to a value between 2... 5, the 5 being really verbose; so talkative that it would actually affect the performance and the OpenCPN log file will get really big, really fast.

Typically, you would stop *OpenCPN*, set the debug value and, before starting OpenCPN you would set a line-by line observation to its log file. With **grep** command you can further reduce the filtering if it is too verbose.

On Windows using PowerShell:

```
PS C:\ProgramData\opencpn>
Get-Content ./opencpn.log -Wait -Tail 20
```

On *nix systems:

```
tail -f ~/.opencpn/opencpn.log
```

or with filtering

```
tail -f ~/.opencpn/opencpn.log | grep dashboard_tactics_pi
```

7.7.1 No connection

If the arrows keep on moving forever from right to left, it indicates that the TCP/IP cannot get connection. Check the configuration file's parameter, which is by default:

```
"source" : "localhost:8375", // not limited to localhost
```

If your *Signal K server node* is located in another computer, replace the `localhost` with the computer's name or if that does not work, with its IP-address (numerical).

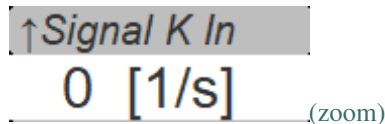
In case the remote Signal K server is still not answering it may be that it does not implement the delta service in the port 8375, or on any other port perhaps; it is not a mandatory requirement for a Signal K server to provide this service.

NOTE: the implementation of Signal K data delta channel is the reason why only *Signal K server node* is supported in *DashT* - other servers have simply never been tested.

If you know that the local *Signal K server node* is there, that it is based on *Node.js* and it is equal or greater to version 1.19, there is indeed no reason why the port 8375 would not be served. In this case you may try simply to use IP-address 127.0.0.1 instead of `localhost`, maybe there is an issue with your systems' Domain Name Service (DNS) settings.

7.7.2 No data

This is indicated by the following condition in the *Signal K In* throughput display:

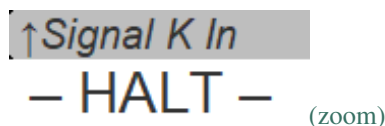


The screenshot shows a digital display for 'Signal K In' with an upward arrow icon. The main display shows '0 [1/s]'. Below the main display, there is a smaller text '(zoom)'.

First, see the *Signal K server node*'s dashboard and verify that it gets indeed some data in - if not, nothing will come out either...

If all looks good both for *Signal K server node*'s input and also the Dashboard's instruments keep hopping around as they normally do, there is perhaps simply no data available in 8375 port. See above, that's not a mandatory requirement for a Signal K server, maybe you are using some commercial implementation of it?

7.7.3 HALT state



The screenshot shows a digital display for 'Signal K In' with an upward arrow icon. The main display shows '- HALT -'. Below the main display, there is a smaller text '(zoom)'.

The message indicates that the continuously running communication thread has been stopped. There is no other remedy for this condition but to stop gracefully OpenCPN and restart it.

To avoid this to happen one should keep both the Signal K input stream *instrument* and the Influx DB output stream *instrument* both in their own, distinct instrument windows. In other words, they shall be separated from other instruments but also from each other. This is to avoid that the communication thread would get orphan when instrument windows get reorganized.

If one attempts to change the orientation of the *Signal K In* input stream's carrying instrument display pane, the communication thread will be detached from it and the instrument itself indicates halted state. If you absolutely need to change the orientation of the single *Signal K In* instrument pane (perhaps you want to dock it - otherwise the orientation has no meaning for a single instrument), you need to remove it from the list of instruments and create a new one with the desired orientation.

7.7.4 Confusing timestamps

One may experience difficulties to find recorded data from *InfluxDB v2* which, as a time series database requires that you know at what time range your data was recorded. There is one, potential source for this and that is the clocks not running the same time!

For NMEA-0183 data coming from OpenCPN, timestamps are generated by *DashT* on-the fly at the reception using the local, CPU clock.

For any data coming from *Signal K server node* the timestamps coming with the data are used. If you are repeating a recording, or if your CPU's time is different than that of the *Signal K server node* there is a chance that you get quickly confused.

If not, this is what *DashT* will do: if there is more than five seconds difference between the GNSS (GPS) data provided by the `navigation.datetime` and your CPU (computer) time, *DashT* starts to use exclusively that GNSS (GPS) originated time, **also for the Tactics' generated regatta processor data**, but with offset from the local CPU clock. This, because the `navigation.datetime` messages do not arrive with frequency high enough for the very fast Tactics regatta processor. Without this measure, one would never be able to match the derived performance data with the input data.

Briefly, if you can, synchronize your CPU (computer) clock with the GNSS (GPS) when underway and over the network when not.

ENGINE/ENERGY

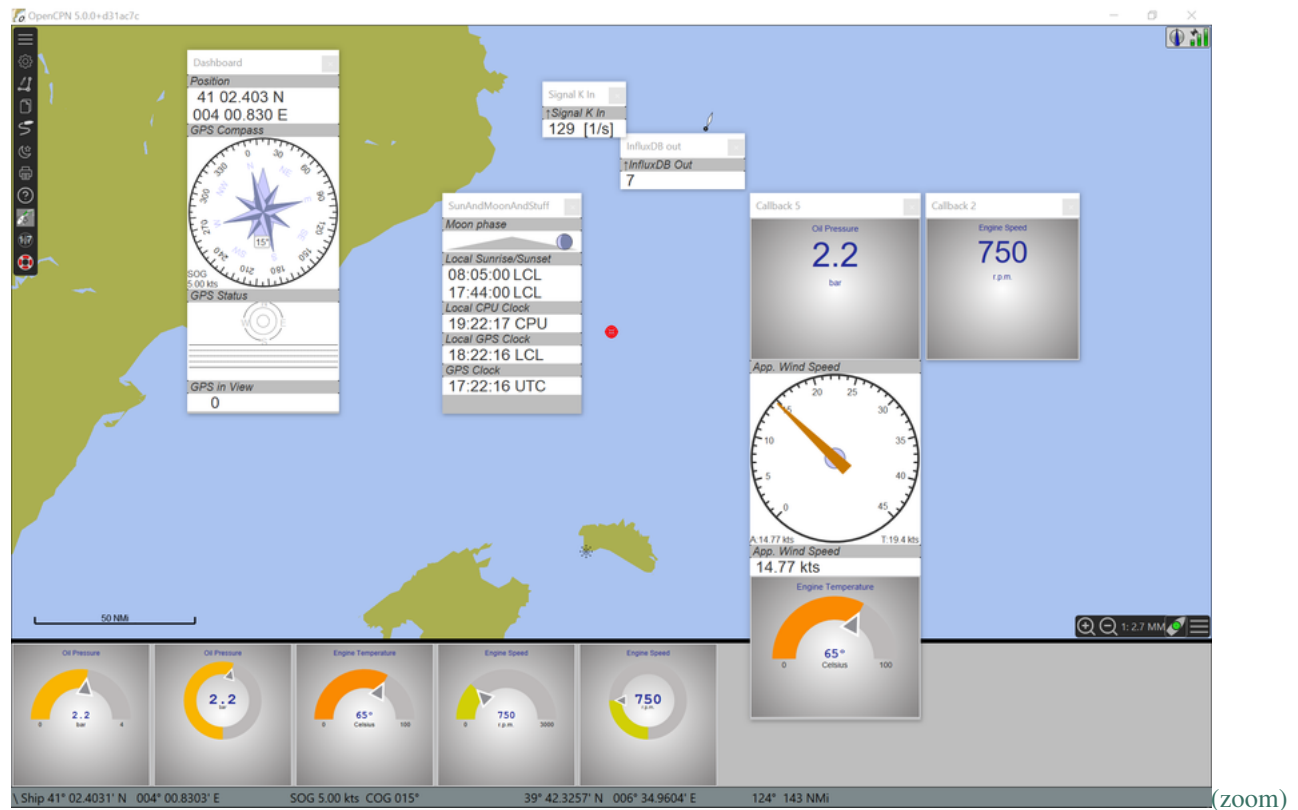
DashT EngineDJG Dials are intended for engine and energy data originating typically from NMEA-2000 databus (or its commercial namesakes) of your boat. The data is read from a [Signal K server node](#) by *Signal K In communication instrument*.

“D” as dial, “JG” as [justGage](#) - thanks for the great dial, folks!

8.1 Introduction

DashT EngineDJG Dials are created using modern web techniques and the resulting application is incorporated in *OpenCPN* dashboard:

- A dedicated instrument for Engine and Energy data in Signal K data format
 - Gets data from DashT plug-in by subscription
- Data obtained from a single Signal K server node interconnection
 - Allowing multiplication of instruments *ad infinitum*
 - Instrument is not making data connections, connects only to DashT
- Does not interfere with NMEA-0183 data push toward the “traditional” Dashboard instruments
 - Your good old instruments receiving their data from OpenCPN will continue to work as before
- HTML5 and JavaScript based
 - Plain text customization files provided
 - Loading from the same or from a remote computer



8.2 Installation/Launch

The EngineDJG gauge does not require any additional installations, all the components are incorporated in the plug-in and ready to use.

8.2.1 Use Node.js

Since you have already a *Signal K server node* up and running and the network connections all set, we just need to use those connections and use their settings and launch both *Signal K server node* and an *http-server* on *Node.js*. This sounds difficult but there is a helper scripts for that, explained below:

8.2.2 Engine Gage script

Windows: a button is installed on your desktop, called *Engine Gage*.

Linux: on command prompt, type `dashtengine` (create a desktop button, if you like)

Both are serving the same purpose: launching a script, which starts the *Signal K server node* on *Node.js* (if not yet running) and launching a *http-server* on port 8080, which is serving *DashT* with the *EngineDJG* web instruments: the instruments are executed then on “mini-browsers”, incorporated in Dashboard instrument panels.

Later on this document, the usage in *feature rich infrastructures* will be explained, for now we expect that the instruments are available for *DashT*.

8.3 Configuration

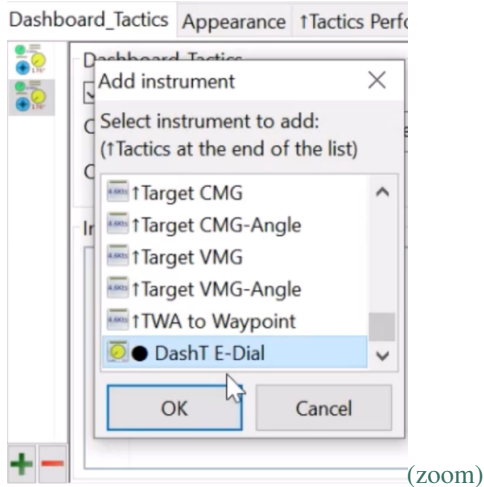
This chapter is addressing the first time configuration - *i.e.* make instruments to learn what data is available in your boat and select the ones you are interested in.

NOTE: Configuration can take place only when the instruments receive data from the *Signal K server node*.

8.3.1 Adding new dials

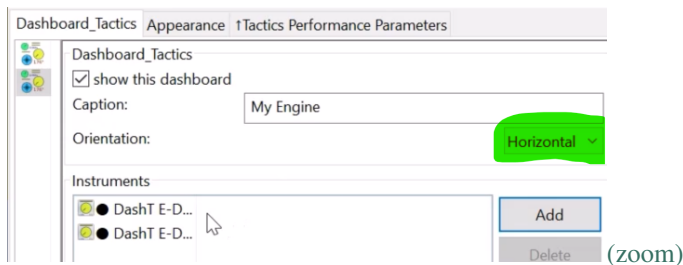
DashT EngineDJG instruments are compatible with all Dashboard instruments and they can be added in any Dashboard instrument cluster window pane. However, since the number of engine parameters, for example is important it is suggested that you create instruments clusters like for “Engine” data and for “Energy” data.

The adding of instruments is like adding any other Dashboard instrument in DashT’s preferences: scroll all the way down of the list to find the single instrument. Add as many instances of it as you estimate you are going to need to show the data parameters you are interested in.



(zoom)

It is suggested than before the configuration the instrument cluster is in “Horizontal” mode to allow sometime very long menu list presented during the configuration phase to roll out vertically. Once the instrument cluster window pane is configured, one can change, of course to “Vertical” mode if needed.

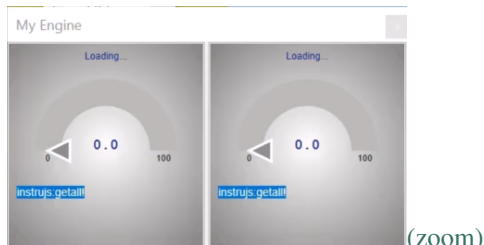


(zoom)

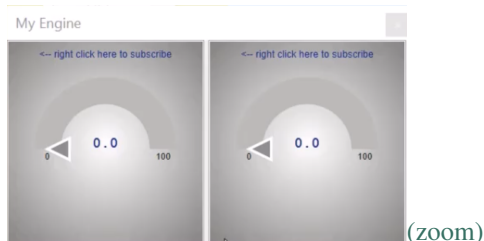
How-to video in [vimeo.com](https://www.vimeo.com)

8.3.2 Subscribe to data

Once a new DashT EngineDJG instrument has been created it asks from *DashT Signal K In* streamer what available data paths there are in your boat. It requires a complete list and, depending of your boat's instrumentation this can be a pretty long one... Normally, the inquiry should be finished in less than 10 seconds, though.



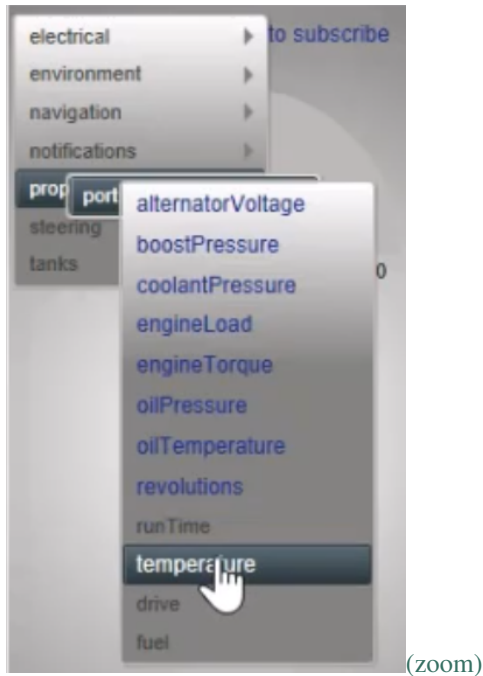
Once a list of all available data paths has been received, DashT EngineDJG instrument builds a selection menu out of them and invites you to make your selection using the context menu which is activated by a **maintained right click** on the **upper left** hand corner of the instrument.



In the context menu, all available data paths are listed a hierarchical order. The names of the data paths are usually clear enough to understand the nature and the origin of the data. A full list of [Signal K keys](#) is also available.



The data path values where the last element is marked in blue means that the DashT installation package has a pre-defined setting for it and it can be shown by simply selecting that data path.



Most common data path values have been included in the DashT installation by default for engine and some most obvious data paths for the energy. If the data path is grayed out, it means that it cannot be selected since no data path rule has been defined. This does not mean that the value cannot be shown, it is just that the development and testing has not been able to test it or is not aware of it - there is really a lot of data paths of all sort, such as status data. A way to define your own configuration for data paths is discussed *later in this document*.

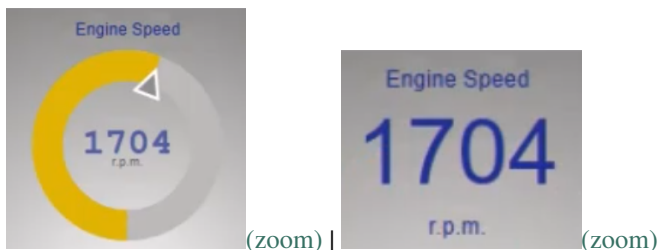
8.3.3 Search again

If you cannot find a data path you are expecting to find from the menu, you can ask for a quick rescan by selecting any of the grayed-out items from the menu, *i.e.* a menu item which is not in blue color. You may get an alert of a non-existing path, depending of the common configuration file settings. Resulting action is, anyway the same as in *Changing EngineDJG instrument's data path subscription* and the data sources are scanned again for new paths.

NOTE: The available data scanning is cumulative and if the data path you are expecting was not available from your data bus within the previous scan's time window frame (in less than 10 seconds), it may have been omitted. Rescan may help. Once the path is recongnized and configured, the subsription to it will be persistent.

8.3.4 Change Display

You can scroll the display types with Ctrl+↑ and Ctrl+↓ (Ctrl-key kept down and press arrow keys up or down): apart the default 180-degree dial type, there is also a 360-degree 'donut' and a simple numerical display type available.



[How-to video in vimeo.com](#)

8.3.5 Change Subscription

While the EngineDJG is running (*i.e.* data is coming in by an existing subscription a data path from the Signal K streamer) right-click on the upper left corner in order to get the context menu which allows you to stop the data display and force a complete re-initialization of the data path subscription as explained in the previous section.

[How-to video in vimeo.com](#)

8.4 Customization

It goes without saying that nobody else but **you** will be able to find out the particular data sources available in your own boat! *DashT EngineDJG* reports you all the data paths but the *DashT* distribution can make but a modest guess what might interest you in the first place. It is likely that we are missing something.

Unlike the most of the *DashT*, which is written in C++/wxWidgets requiring compilation/build etc., the *EngineDJG* instrument is a HTML5/JavaScript program which means there are plain text files which you can modify, allowing you to customize the data paths displayed according to your requirements.

Modern web development techniques have been used which are resulting in a very compact and thus non-human readable run-time program execution format, but configuration files are provided also in plain text without merging and compression, allowing you to make your own customizations.

8.4.1 File location

Defaults are for the input JavaScript files:

Windows:

```
\Users\Public\DashT\www
```

Linux:

```
/usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs/www/
```

In case you have a boat full of electronics, computers and servers alike, you know what you are doing but you may still want to follow the instructions for *feature rich environments*, in which case the above paths would probably be best known by yourself.

NOTE: Make sure to take backups before modifying anything! JavaScript is “easy” but does not have any pity to syntax errors: it is a grinding halt without a visible error message...

Add data paths

You would need to modify a configuration file named `common.js` - here is an excerpt of the instructions in that file

Contribute/report here, please:

<https://git.io/JejKQ>

- with a screenshot and a short description of your installation, thanks!
SignalK Path keys:

<https://git.io/JvsYw>

The Signal K values are always in SI units (like m/s, not knots).
 Conversion to a wanted unit is made with multiplier/division/offset.
 Avoid using floating point values like 0.0000000003 in JavaScript!
 UTF8 - do not change encoding, cf. degree character. Notepad++ recommended.
 Usage example: enginedjg/index.html loads a minimized version, common.min.js

- make a copy of common.min.js by renaming it;
- make a copy of this one with name common.min.js and modify it;
- (no need for compression with this non-executing file!)
- or, modify enginedjg/index.html to load your own file, no problem!
- issues? open the index.html in a browser, hit Shift+Ctrl+I and reload;
 - * Console gives you the reason why it does not load anymore:
 - * Look for messages in red, a typo, missing comma?
- note: next update/reinstallation overrides this file, keep backups!

Typical change that you may want to do is to have different titles for each battery parks, instead of having a * wildcard. You would copy-paste-modify the below definition by replacing the wildcard with two (or more) individual records, both with a full data path (Signal K key) name and a title for your liking, and this for as many times your system is reporting about those values being available:

```
{
  version      : 1,
  path         : 'electrical.batteries.*.current',
  title        : 'Battery Current',
  symbol       : '',
  unit         : 'Amps',
  display      : 'dial',
  decimals     : 1,
  minval       : -20,
  loalert      : 0,
  hialert      : 0,
  maxval       : 20,
  multiplier   : 1,
  divider      : 1,
  offset       : 0
},
```

Of course, you may want to simply change units in some entries, like from Celsius to Fahrenheit. Just be careful to preserve the UTF-8 degree sign, will you.

The header part of the file contains some common customizations, like turning off the alerts or giving more time for them to set in. One can also increase the debug level in case the EngineDJG HTML/JS code needs to be inspected in an external browser.

NOTE: the common.js file is **not** used by default but one needs to replace the compressed version of it with this human readable one in instrujg/index.html. Follow the instructions in common.js for that. If a need arise, do not hesitate to do the change - there is no performance penalty since the file is read only during the startup of instruments. See also

8.4.2 Language file

Unfortunately, it is self service, no community support. Yet! You can participate with your native tongue by submitting your translations here: <https://git.io/JejKQ>

Meanwhile, you can just replace the `lang.js` file with yours. Keep a copy of the original, though: **JavaScript does not make any gifts** but goes to a grinding halt for any syntax error (like a missing comma) and none of the DashT EngineDJG instruments will start! There will be no fancy warning message, just a gray background staring at you... - check this [troubleshoot section](#).

8.5 Rich infras

The instrument is composed of a HTML file and numerous JavaScript files, like any other browser based application. If you do not run a *Node.js* on your local computer, the configuration task consist of making them available on your boat's network infrastructure. It contains many elements being platform specific, explained in below sections.

One needs to explain *DashT* where it should fetch the *EngineDJG* components. This is done in *ini/conf-file*.

NOTE: one might be tempted to use `file://` protocol. Please note that in Windows, starting from [security update of 2020-01-14](#) local storage capability, including cookies is disabled for `file://` protocol in the back-end the wxWidgets is using on Windows (IE). Therefore we address below only the usage of `http://` protocol to retrieve files from your files infrastructure to enable persistent EngineDJG configuration settings across platforms. You can set `file://` protocol in *ini/conf-file* but you would probably need to reconfigure your instrument at every restart. In older operating systems without security updates the `file://` protocol may work but *DashT* is not addressing those system configurations so you would experience other issues.

8.5.1 pub Linux

First, you have to identify where are the DashT EngineDJG files after the installation and export the directory:

```
/usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs/
```

Apache2 server

Most of the Linux systems provide Apache2 server by default and even start it by default. Navigate to your Linux box's network IP-address and you will soon find out if this is the case. Otherwise, it is not a big task to get it running but for clarity we presume now that it is already active. You would say:

```
sudo ln -s /usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs /var/www/html/instrujs
```









Check that it works:

NOTE 1: The IP-address below is an example only, author's Raspberry: use your own

NOTE 2: The actual content of the directory changes with more DashT *instrujs* instruments becoming available

Not Secure 192.168.8.103/instrujs/

Index of /instrujs

Name	Last modified	Size	Description
 Parent Directory	-		
 LICENSE	2020-01-26 22:29	1.0K	
 common.js	2020-02-09 22:22	13K	
 common.min.js	2020-02-09 22:23	4.5K	
 confstat.js	2020-01-26 22:29	2.4K	
 enginedjg/	2020-02-09 22:23	-	
 lang.js	2020-02-09 10:53	1.6K	
 scripts/	2020-02-09 22:32	-	

Apache/2.4.38 (Raspbian) Server at 192.168.8.103 Port 80 (zoom)

http-server node

NOTE: If you are running a *Signal K server node* on your local computer, you do not follow these instructions, the helper script explained above is doing all this work for you. Follow these instructions only if your server is located on another computer.

Since you already have a `_Signal K server node_` you have thus *Node.js*. If the above Apache2 method seems like an overkill to you, there is this dead simple solution:

```
npm install -g http-server
```

You would start the service (and can make a shell script for it):

```
http-server /usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs/www -p 8080
```

Or from where ever you have put the `instrujs` directory on this server.

Execute the script, leave the `http-server` running and verify that files are available on the port `8080` (you can modify the port, of course). The verification, like usual with a normal browser, as above with the Apache server - no need to start *OpenCPN* for that.

The new server and the port need to be described to *DashT* in *ini/conf-file*.

8.5.2 pub Windows

NOTE: We suppose below that you are running also the *Signal K server node* on Windows but locally, *i.e.* not where *OpenCPN* is located. With local-only installation, the helper script is doing all the below work.

In addition to the *Signal K server node*, install also the following server node:

```
npm install -g http-server
```

You will find the files to share from

```
C:\Program Files (x86)\OpenCPN\plugins\dashboard_tactics_pi\data\instrujs\www
```

You would start the service (and can make a batch or PowerShell script for it):

```
http-server "C:\Program Files (x86)\OpenCPN\plugins\dashboard_tactics_pi\data\instrujs\
↪www" -p 8080
```

Or from where ever you have put the `instrujs` directory on this server.

The new server and the port need to be described to *DashT* in *ini/conf-file*.

8.6 Troubleshooting

8.6.1 Data is bad

It is always a good idea to go back as close as possible to the data source, which is Signal K server node. What does it say about that data? If the SI unit value it displays in its own plug-ins and log files makes sense to you, then the issue is probably with the multiplier, divider or offset, or all of them used to convert it (see above sections about the customization).

The floating point data which is coming from Signal K server node is given as such to the *DashT EngineDJG* instrument, but as a string (we are talking to a JavaScript program) in non-scientific notation, in decimal format. For example, if DashT receives 0.24444444, that usually is alright. But a general rule is that values with too many zeros are not advisable in JavaScript. Avoid calculations where any result, including intermediate values would lead to values like 0.0000003 or similar bunch of zeros.

If things need deep understanding of data which is coming in from your boat's databus, maybe the easiest place where you can look at it is the log files of Signal K server node's databus connector. You need to activate them explicitly. The *instrujs* Developer's Guide in *DashT* documentation (in the repository) provides some use cases of deep packet analysis but be warned, it is not for faint hearted!

8.6.2 After config, all dead

The good and bad of JavaScript is that as a scripted language it allows you to make as many syntax errors you like. When you are editing, that is... During the development we use TypeScript to avoid this. But when you are editing JavaScript directly, errors are fatal. Since there is no compilation or transpiling, eventual errors are detected only during loading and even worse, sometimes during the run-time when the code execution passes in the faulty section!

Your best friend is your ordinary browser. Point it to the data directory where the *EngineDJG index.html* file is located and see if you can see an empty dial. If you do not see, then hit **Shift+Ctrl+I** (Shift and Ctrl-keys hold down and press key "I") to open the developer tools. Select the Console tab to see the quite verbose *DashT InstruJS* debug messages and hit **Ctrl+F5** to reload the page. Usually, the point where the syntax error is located is clearly shown in red color.

NOTE: Windows owners should not throw away their beloved (?) Internet Explorer. In fact, the wxWidgets WebView back-end library on Windows port is as old as IE8! In that case, having the IE11 is not a bad idea for testing: you would hit key F12 to get the developer tools visible in this case. To reload a page it is F5.

8.6.3 Data "X" not available

The *DashT EngineDJG* dial is configured by selecting from data available on your boat's NMEA-2000 instrumentation bus and referred as Signal K data key. It may occur that after a reconfiguration of *Signal K server node*, or from some other reason a data path (a data source) is not available anymore. If there is a *EngineDJG* instrument on your Dashboard configured to display that data path, it remains waiting for that data forever, in vain.

The simplest remedy for this is to destroy the instrument and create a new one if needed.

But supposing that you are attached to this particular instrument position for some reason (!) there is another way which requires that you shut down OpenCPN first:

Open the OpenCPN *ProgramData/opencpn/opencpn.ini* file (*~/.opencpn/opencpn.conf* in Linux) and find the instrument declaration section, easiest by using the dashboard's title for your search. There is a UID-field with a long, arbitrary string for each instrument based on web-techniques. Now, **without altering the total length or the format** change one single arbitrary number in the UID-field of the instrument you want to reconfigure and save the file.

When you restart OpenCPN, the instrument has forgotten its configuration and you can reconfigure it again to one of the data paths in your new NMEA-2000 configuration.

INFLUXDB / GRAFANA

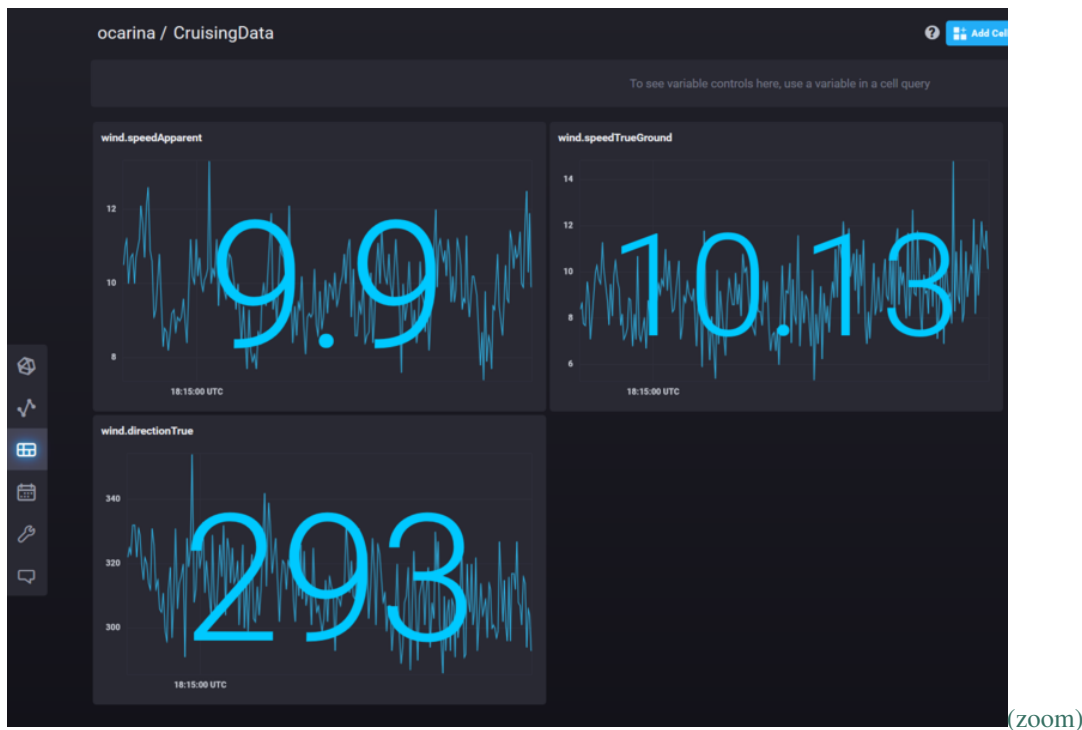
In this chapter we explain first *DashT* helper-script usage of Docker container based server execution to make InfluxDB time series database easily available. In the second part we give necessary information for those who want to do the installation their way and give an overview to the usage of an external time series visualization tool called Grafana.

NOTE: You are not obliged to run InfluxDB time series database while underway - you can also stream all data into a so-called *Line Protocol File* which can be loaded into an InfluxDB DB back home. If only few data sources are needed, one can *record some data sources in a CSV-file with history graph instruments* for off-line analysis. But the provided helper script method based InfluxDB / Docker makes it so easy to use that you do not want to miss the new services which can be built around having near-term historical data - **all data** - at your disposal, like on Grafana dashboards.

9.1 Introduction

Influx data's InfluxDB 2.0 is time series database which can be used directly for data analysis or used as a middleware for data analysis or monitoring software. A data streaming *connector* has been developed between *DashT* for OpenCPN v5 plug-in and this popular time-series database platform. Most natural usage is to export **all** data received by the plug-in into Influx DB 2.0 with the millisecond time stamps of *DashT*.

Once the all data is stored, it can be used to wide variety of usage, either immediately or extracted for off-line analysis. Here is an example of dashboards, familiar from Grafana but here provided by the InfluxDB 2.0 offering instant retrieval and analysis services for live data:



9.2 Docker InfluxDB

Docker ([Docker Desktop](#) for Windows and Mac) is a popular framework to containerize applications. Shortly, a “container” is a mini-operating system, often Linux based running one single service, usually network based. The container is running under Docker or Docker desktop and you would communicate with the application over the network. The application can be made to see your file system so that you can share data with it without going through the network.

NOTE: Why there is no *DashT* Docker container? Simply because it would make yet another build to maintain and yet another dependency. Instead, standard and latest service containers can be used and configuration scripts for those are provided with *DashT* installation package and explained below.

TIP: You can try and *not* to stop InfluxDB using the helper scripts below when you stop your system; normally, at next restart Docker will restart all the services. But on Windows systems, do not leave your system too long time off, without restarting the InfluxDB using the provided script - it synchronizes the clock only this way on Windows.

9.2.1 Windows scripts

The helper scripts to start and stop the Docker based InfluxDB, Grafana and nginx web server are provided with *DashT*. The supporting configuration files are installed in \Users\Public\DashT folder. Two buttons are installed on the Desktop:

1. Start DB (database *and* web services)
2. Stop DB

All you need to do to make the buttons work is to **install** yourself [Docker Desktop](#) and **start it** - following the *DashT* maxim “if you do not want it, you will not get it”, no third party software is installed by *DashT* installer so that you can keep the control.

NOTE: the helper scripts have been developed and tested for Docker Desktop v3.3.3. By default the latest InfluxDB is installed, while the tests have been carried out with InfluxDB v2.0.6.

TIP: If Docker Desktop announces that our system is **Hyper-V compatible** (Windows 10 Pro only) but suggests to use [WSL 2](#) it is not recommended for these applications: InfluxDB, Grafana and nginx will all have part of their filesystems mapped back to local Windows file system in the Public folder. This has been reported to work in less optimal way and even a warning about it given by Docker Desktop v3.3.3. **Hyper-V is preferable here.**

9.2.2 Linux scripts

The Linux version of the above is the helper script **dashtdb** - you would use it from the command line:

```
you@yourlinux:~$ dashtdb
```

```
dashtdb - Launching Docker based services for DashT: nginx, infludb, grafana
```

```
Usage:
```

```
    dashtdb [start|up]
```

```
    dashtdb [stop|down]
```

NOTE: we let fans of GNOME3, KDE, LXQT, xfce4 and alike to create their own desktop buttons!

The same way, like on the Windows counterpart the principle “*only if you want it you will get it*” is respected; no third party software is declared as automatic dependency in the *DashT* package installation. Only if you launch the **dashtdb** script and if the supporting software is not there, you will be asked do you want to get it installed. This way, a person who is not interested in database functions will not get database software installed for nothing.

Third-party programs installed by the script - with your permission:

- dashtdb script needs:
 - docker, docker-compose
 - * InfluxDB v2.0, Grafana, nginx web server

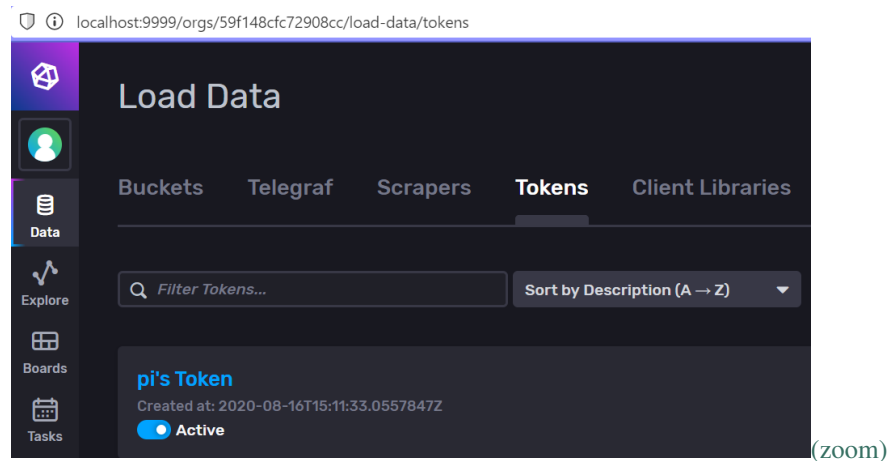
9.3 Set up InfluxDB

There is no reason to repeat [InfluxDB v2.0 documentation](#) here. But setting up the database for your boat is dead easy already from the welcome screen of InfluxDB 2.0, which is now at <http://127.0.0.1:9999>, or in <http://host.docker.internal:9999> if you prefer. The guidance for the first time configuration is excellent! You will need to give:

- User name and password
- Organization name - like the name of your boat
- “Bucket” name - this is where the data is going to be dropped into, give it a name like “nmea”, why not

If you plan not to use streaming or reading back of data, just file-based data storage when underway and its feeding into the database when back in the safe harbour, you are done. You would use, later on this very same interface to upload the data file into the database, into the bucket you have created. Database service itself does not need to run, in this case when you are underway, *DashT InfluxDB Out* is enough.

However, streaming data in live into the database and reading it back live requires a token (your username and password are never asked in data communication):



Give the token in *InfluxDB Out* instrument's configuration to enable HTTP-based real-time write/read operations between *DashT* and InfluxDB v2. You would need to keep the database container running, in this case when underway.

9.3.1 InfluxDB storage

In this solution with Docker, with *DashT* configuration scripts data is stored on your computer, the same one which is hosting Docker.

Windows: C:\Users\Public\DashT\influxdb2

Linux: ~/.opencpnplugins/dashboard_tactics_pi/instrujs/influxdb2

You may want to back up or otherwise keep these directories safe if you want to keep long-term archives of the data. *DashT* cannot guarantee the integrity of the data since this directory is totally under control of Docker/InfluxDB.

TIP: On Linux you can relink this directory to an external device, such as USB3-connected disk to reduce the load to your system disk against repetitive small chunks of writes (if your system disk a semiconductor storage device) and for backup purposes. (On Windows, Docker will most likely have no permission to write to the device which is out of Public or user's folders.)

9.3.2 InfluxDB backup

By nature, the InfluxDB time series database is considered to have a retention policy which can be set for each bucket - or set to keep the data "forever". It can be interesting to have backups if one accidentally does not deal with or does not have time to analyze the data before it is deleted. One can take the advantage of the local file systems mounted by the *DashT* scripts and take a backup on that file system. The backup is taken with the InfluxDB CLI, which on Windows can be opened using a container controller button on the Docker Dashboard.

```
# influx backup \  
> /var/lib/influxdb2/backup_$(date '+%Y-%m-%d_%H-%M') \  
> -t 4YSbL.....DOAo09A==
```

Where the `-t` is your InfluxDB administrator's token and not only a token for a single bucket since you are making a backup of all data here.

To restore our (example) bucket `nmea` (which cannot exist in the target database) with data from a previous backup:

```
influx restore \  
  /var/lib/influxdb2/backup_2021-06-20_18-04 --bucket nmea \  
  -t 4YSbL.....DOAo09A==
```

For more information about backup and restore, see [InfluxDB v2.0 documentation](#).

9.4 Grafana

Grafana is an open source analytics and monitoring solution for almost every database based data, including *InfluxDB v2.0*. *DashT* bundles this popular and easy-to-use visualization solution in its Docker ready-to-launch configuration albeit it does not use it, in any way in *OpenCPN*. For your convenience, *DashT* sets up you few essentials needed to work both with Docker and with a InfluxDB v2 container.

With *Grafana* we want to use a separated volume to share it settings and other parameters in persistent manner, provided that we need to delete and restart the container. The volume is located in local file system as defined in `docker-compose.yml` (see above).

NOTE: To help with the data retrieval from the InfluxDB v2 time series database, *DashT* Docker helper scripts are installing in the mounted, local file system for Grafana, in its `plugins` folder *InfluxDB (Flux) Datasource* (`grafana-influxdb-flux-datasource`) to make the data retrieval possible by a simply data source selection.

Grafana needs two network connections now: one to talk with you (or rather with your browser). All the settings are done in the above configuration file but perhaps it is more clear to look at the resulting container settings with Docker: `docker inspect dasht_grafana`:

```
"Ports": {
  "3000/tcp": [
    {
      "HostIp": "0.0.0.0",
      "HostPort": "30000"
    }
  ]
},
```

This is the port Grafana is listening. Point your browser, in this case to `http://localhost:30000` (chosen not to get mixed with *Signal K server node* port 3000).

Second interface is to Docker: please Docker, let me talk to the InfluxDB database container... Again, this is set in the above configuration file as `db` link, but let's see what Docker reports about in with the above `docker inspect` command:

```
"Networks": {
  "dasht_default": {
    "IPAMConfig": null,
    "Links": [
      "dasht_influxdb:dasht_influxdb",
      "dasht_influxdb:db"
    ],
    "Aliases": [
      "graphs",
      "e965d09373b1"
    ]
  },
}
```

Therefore, in Grafana container we can reach the neighbouring InfluxDB v2 container either with a URL `http://dasht_influxdb:9999` or with its alias `http://db:9999`.

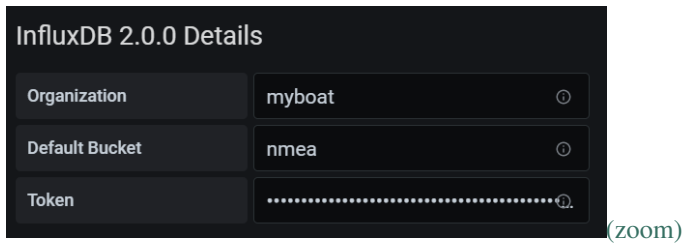
To make confusion total, the newer version of Docker Desktop 3.x requires a pseudo host name `http://host.docker.internal:9999`.

Armed with this knowledge, we can configure Grafana's *InfluxDB (Flux) Datasource*.

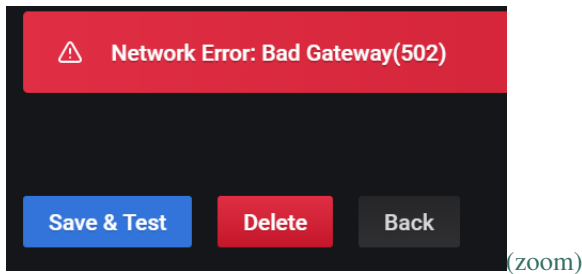
NOTE: (When this is written May 2021) Grafana's InfluxDB data source plug-in says that Flux support is still in beta. Therefore we use Influx's own (beta) which DashT installs by default. The situation is likely to change, hopefully the interface will not change too much and the below remains applicable.



Enter the settings you need to collect from InfluxDB v2 (you can have another tab/window open in <http://localhost:9999>):



If you use the default URL, like `http://127.0.0.1:9999` to connect to InfluxDB v2, you will get an error:

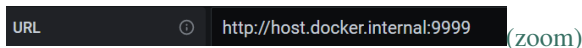


... or similar, depending of the Docker Desktop version you are using (2.x or 3.x).

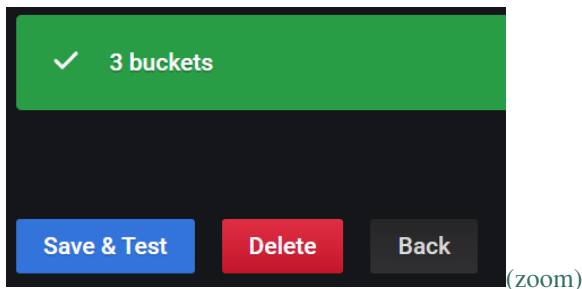
You need to access the InfluxDB v2 through the Docker provided container-to-container link. *DashT* script has given it an alias `db` (of course!). Port remains the same 9999 (we are not going through the proxy server.)

Docker Desktop 2.x: Give URL `http://db.9999`, the alias

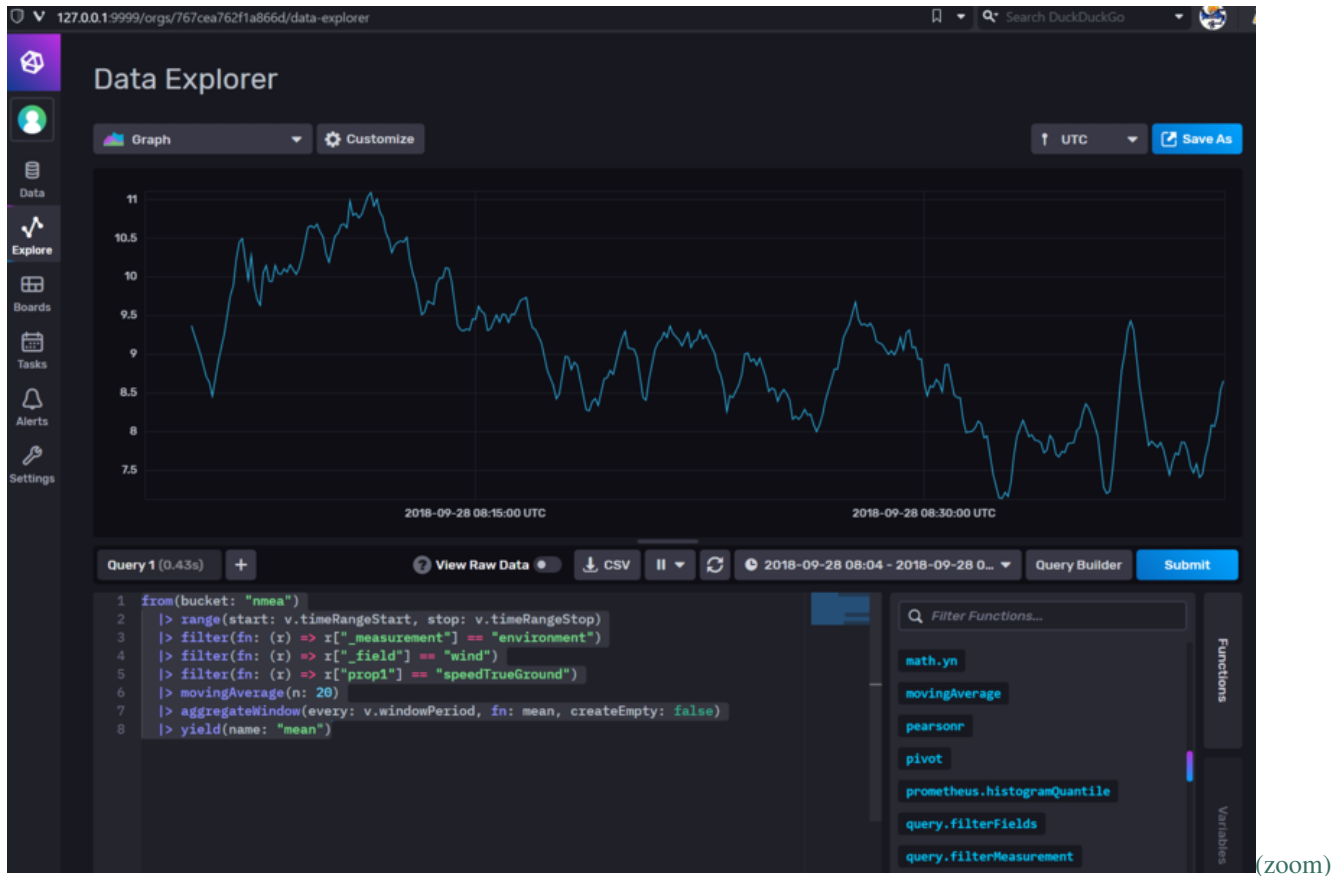
Docker Desktop 3.x: Give URL `http://host.docker.internal:9999`, which then translates in Docker as 'local host'.



Success.



We need some data now and some Flux code to request for it. Let's first collect some data in InfluxDB. Here we select TWS.



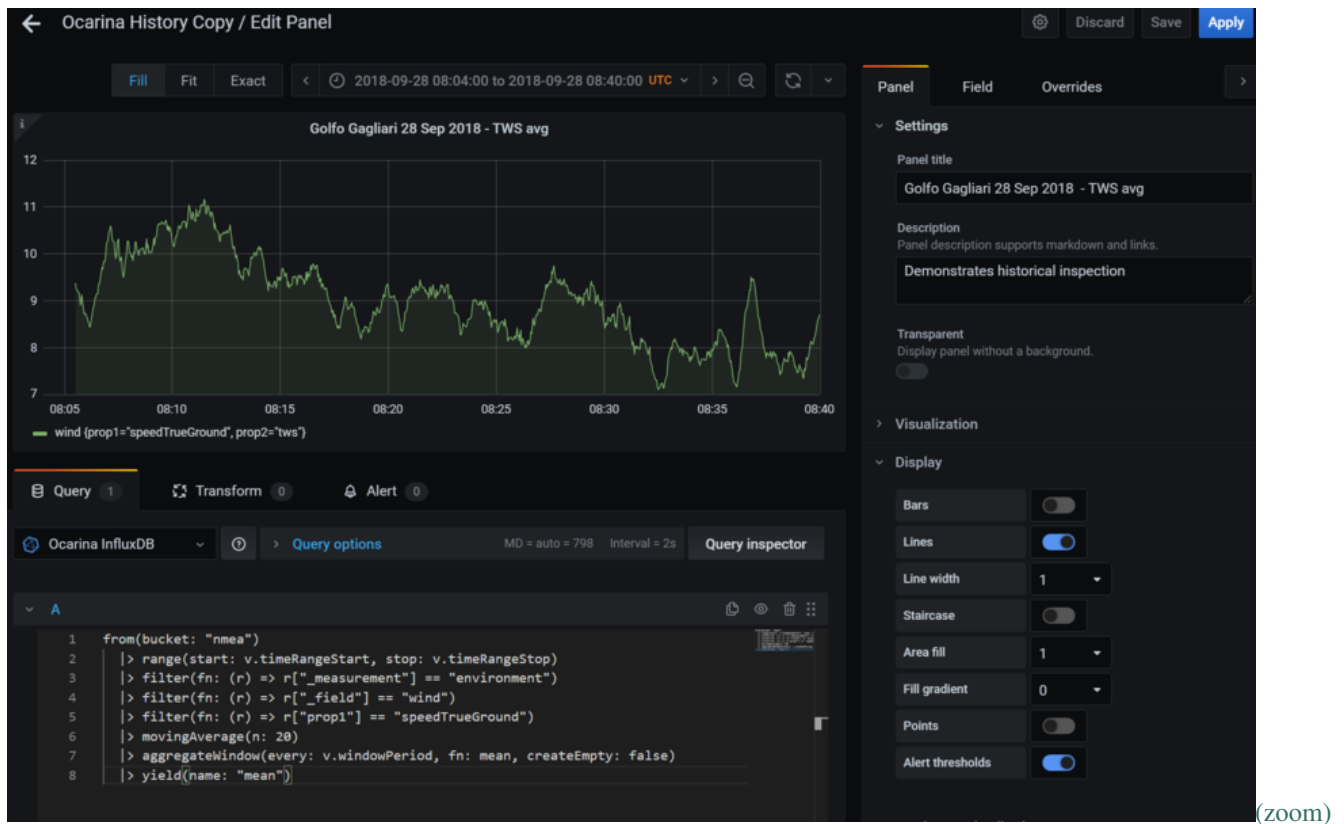
If not familiar with Flux language (who is?) one can easily get a good example with the InfluxDB Script Editor, here for the above screen, one just switches from the click-based selection for inquiry builder to the Script Editor in the above example.

NOTE: there are variables used by InfluxDB which one cannot copy as such to the other program, such as Grafana. We need to remove those and either use Grafana's variable or avoid using them. Here's the code with only one variable in Grafana, n for the limit of the samples to collect.

```
from(bucket: "nmea")
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
|> filter(fn: (r) => r["_measurement"] == "environment")
|> filter(fn: (r) => r["_field"] == "wind")
|> filter(fn: (r) => r["prop1"] == "speedTrueGround")
|> movingAverage(n: 20)
|> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
|> yield(name: "mean")
```

The above code has been generated in the *InfluxDB Explorer* and it can be cut and pasted to *Grafana Dashboard*. It is used, almost the same but with shorted sliding time in the *DashT* instrument (*Line Chart*). You can therefore make your choice for the presentation tool, the data and its retrieval remains the same!

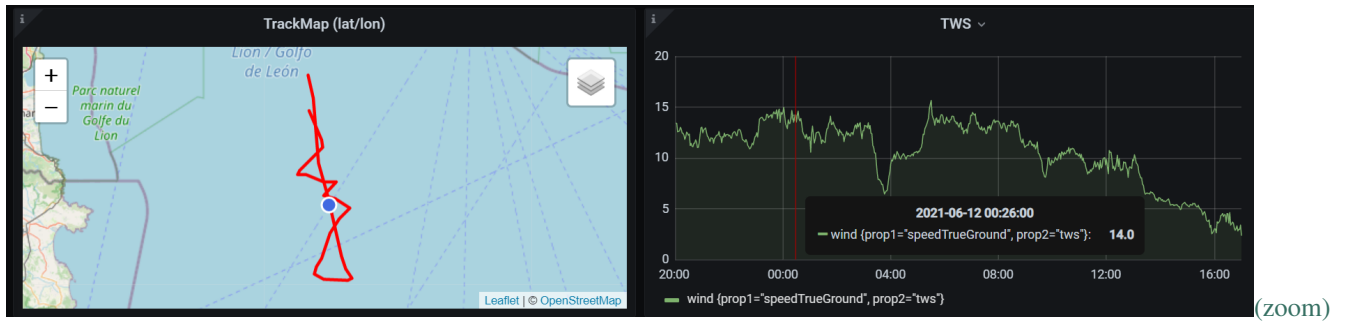
Grafana provides all sorts of debugging tools, such as Query Inspector which are extremely helpful if something goes wrong. It is unfortunately impossible to rewrite here all the knowledge one can certainly find in *Grafana* documentation or just by trying things out. *Grafana* is some pretty powerful piece of software, which is used for Data Center management etc. so do not get frustrated too quickly but experiment. Some nice winter fun!



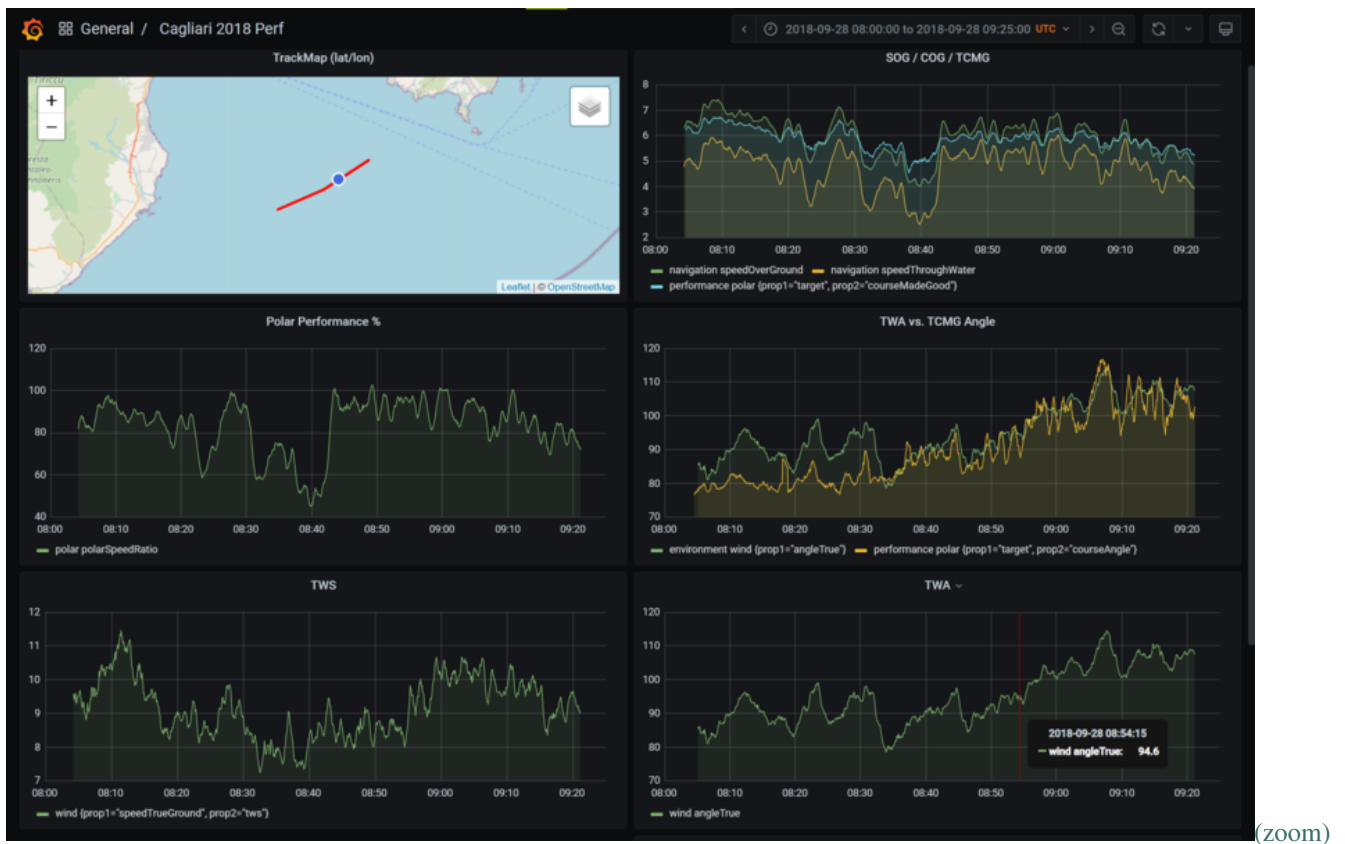
Finally, if you have a big screen in your boat, nothing prevents to put you tens of graphical instruments on it's mighty real estate area. In this example there is only one humble Grafana dashboard instrument:



With Grafana plug-ins, the possibilities are endless. In the below example, we have one panel which makes inquiry for position data but also for True Wind Speed (TWS). When browsing historical data, one can position the pointer on the wind data and, using [TrackMap](#) plug-in for Grafana, show the position on which the wind condition occurred.



Going back in the history, even before the times the *DashT* existed is possible with *InfluxDB Out* tools for *conversions*, such as *conversion of a VDR-playback stream*, allowing *DashT Tactics Polar* calculations to be applied, even retrospectively! Let's move a few years back and use a Grafana dashboard for performance analysis to wonder it took us so long to react to an important wind shift by rolling back that part, from InfluxDB v2.0 with the following Grafana dashboard:



There is a [dedicated developer's corner](#) which collects information how the above panels have been set up, including the Grafana dashboard's JSON-configuration files.

9.5 HTML/JS updates

HTML and JavaScript for the Engine/Energy and other web based instruments are available, in default *nginx* configuration in port 8088. This way, you do not need necessarily run a specific server on port 8080 for Engine and Energy instruments using its *own helper script*. Of course, you need to change the port also in the configuration file of OpenCPN, see *Tweaks*.

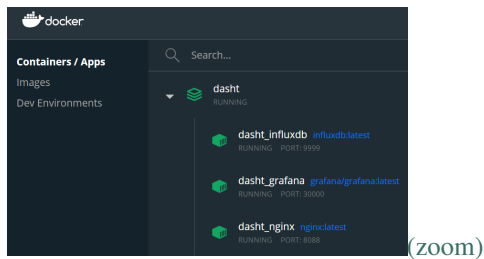
Windows: Update of *DashT* (i.e. reinstalling with installation .exe updates the folder C:\Users\Public\DashT\www from where the *nginx* is serving the HTML/JS files for the instruments.

Linux: Update of *DashT* does **not update HTML/JS files** in ~/.opencpnplugins/dashboard_tactics_pi/instrujs/www - they are copies of the distribution's files. After an update of *DashT*, it is enough to remove the folder www in the above user path and launch the *dashtdb* scripts, which will recreate the folder and copy the updated HTML and JavaScript files into it.

9.6 Under the hood

This section you may want to read only when things do not work for you, for simple curiosity or if you want to do the things yourself - maybe you have a Mac and the above scripts are not available for you (but you can maybe contribute by modifying the Linux scripts as a starting point).

9.6.1 Docker Dashboard



Use Docker Desktop's Dashboard to view the three containers created by the above helper scripts, providing three useful services:

- **nginx** - this is the HTTP / proxy server, a Swiss knife providing both files to OpenCPN's *DashT* but also connecting it to other network based services, and interconnecting those services
- **InfluxDB v2** - time series database which both collects data from *DashT* but which is also available to feed it back to *DashT* but also to other useful services:
- **Grafana** - A monitoring solution which allows you to create more complex dashboards that would be possible with *DashT* or with *InfluxDB v2*.

NOTE: We do not run a *Signal K server node* in Docker - it may require some physical connection like USB and it is better done with *Node.js* which is also network performance-wise a better solution than running it in a Docker instance. For testing and learning purposes it is, of course, possible to run *Signal K* as a Docker instance as well.

Command line is the same both for Linux or Windows (and probably for Mac):

```
you@yourlinux:~$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0cfd91096e25	grafana/grafana:latest	"/run.sh"	5 hours ago	Up 5 hours	0.0.0.0:30000->3000/tcp	dasht_grafana
77084b80da78	nginx:latest	"nginx -g 'daemon of...'"	5 hours ago	Up 5 hours	0.0.0.0:8089->8089/tcp, 0.0.0.0:8088->80/tcp	dasht_nginx
c31e2a3fe3f6	quay.io/influxdb/influxdb:2.0.0-beta	"/entrypoint.sh infl..."	5 hours ago	Up 5 hours	0.0.0.0:9999->9999/tcp	dasht_influxdb

(zoom)

9.6.2 Container creation

Containers are created using the following type of container definition (example from Windows, in Linux version only the file system mount points do change), the file is named in both `docker-compose.yml`:

```
version: '3'
services:
  web:
    image: nginx:latest
    container_name: dasht_nginx
    depends_on:
      - db
    volumes:
      - /c/Users/Public/DashT/nginx/nginx.conf:/etc/nginx/nginx.conf
      - /c/Users/Public/DashT/www:/data/www
    ports:
      - 8088:80
      - 8089:8089
  graphs:
    image: grafana/grafana:latest
    container_name: dasht_grafana
    depends_on:
      - db
    links:
      - db
    volumes:
      - /c/Users/Public/DashT/grafana:/var/lib/grafana
    ports:
      - 30000:3000
  db:
    image: quay.io/influxdb/influxdb:2.0.0-beta
    container_name: dasht_influxdb
    volumes:
      - /c/Users/Public/DashT/influxdb2:/var/lib/influxdb2
    command: influxd run --bolt-path /var/lib/influxdb2/influxd.bolt
      --engine-path /var/lib/influxdb2/engine --store bolt --reporting-disabled
    ports:
      - 9999:8086
```

The description files define all the steps that one would need to either to type manually or set up using Docker Desktop. Now it is automatic every time you use the *DashT* stop/start scripts!

NOTE: On Windows systems, the start-up scripts of *DashT* also synchronize the InfluxDB container's clock with the local CPU clock. This is needed because on Windows, the clock is provided by Hyper-V virtualization. And when you are not running any containers that clock is not running... For this reason, it is extremely important to synchronize your CPU time to a reliable data source (like GPS, when underway).

You can observe that both network ports, dependencies and mount points in local file system have been defined. If you need or want to do the work manually, the above description can be used as a starting point for what is needed.

9.6.3 nginx

nginx engine x is an HTTP and reverse proxy server. It is a true Swiss knife, allowing us to provide and share all services local, like <http://localhost:8088> - we need more services, we just add port numbers. With *nginx*'s rich features and high its reverse proxy we can

- Make *DashT* network based files available as network service from the local file system, the same in which OpenCPN is installed - no copying is needed
- We can make the code in those files to access *InfluxDB v2* **in parallel** with your browser and with *Grafana* by enabling **CORS** (*Cross-Origin-Resource-Sharing*) so that browser security features gets satisfied

NOTE: without a CORS-enabling proxy it is impossible for a web-based application like JavaScript instruments of *DashT* to read/write into *InfluxDB* or other web service since they are not originating from that very same web service. However, *InfluxDB Out* is not affected by this so if you do not use JavaScript instruments, you can omit the CORS-enabling proxy.

We want *nginx* to configure two local port, to act as proxy server and to deal with CORS so that the localhost served JavaScript files can access services such as the *InfluxDB v2* - same thing for *Grafana*, let it access *InfluxDB v2* :

- port 8088 - is mapped in Docker container as port 80, bind back to the host's local file system so that *DashT* provided *www* directory (instrument HTML5/JavaScript) can be served
- port 8089 - is a proxy for files served from 8088 wanting to access *influxdb* server - **CORS Access Control headers** are replied to browsers confirming them that this is OK - tested Chrome 80, Firefox 74 and even IE11 (because *WebView* of *wxWidgets* is using it).

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include      /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile      on;
    #tcp_nopush    on;

    keepalive_timeout  65;

    #gzip  on;
```

(continues on next page)

(continued from previous page)

```

server {
    listen 80;
    location / {
        root /data/www;
        autoindex on;
    }
}
server {
    listen 8089;
    # https://enable-cors.org/server_nginx.html
    location / {
        # InfluxDB query, read
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Credentials' 'true' always;
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
            add_header 'Access-Control-Allow-Origin' $http_origin;
            add_header 'Access-Control-Allow-Headers'
                'Authorization,Accept,Origin,DNT,User-Agent,X-Requested-With,
                If-Modified-Since,Cache-Control,Content-Type,Range';
            add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
            #
            # Tell client that this pre-flight info is valid for 20 days
            #
            add_header 'Access-Control-Max-Age' 17280000;
            add_header 'Content-Type' 'text/plain; charset=utf-8';
            add_header 'Content-Length' 0;
            return 204;
        }
        # InfluxDB query, write
        if ($request_method = 'POST') {
            add_header 'Access-Control-Allow-Credentials' 'true' always;
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
            add_header 'Access-Control-Allow-Headers'
                'Authorization,Accept,Origin,DNT,User-Agent,X-Requested-With,
                If-Modified-Since,Cache-Control,Content-Type,Range';
            add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
        }
        if ($request_method = 'GET') {
            add_header 'Access-Control-Allow-Credentials' 'true' always;
            add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';
            add_header 'Access-Control-Allow-Headers'
                'Authorization,Accept,Origin,DNT,User-Agent,X-Requested-With,
                If-Modified-Since,Cache-Control,Content-Type,Range';
            add_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range';
        }
        proxy_redirect off;
        proxy_set_header host $host;
        proxy_set_header X-real-ip $remote_addr;
        proxy_set_header X-forward-for $proxy_add_x_forwarded_for;
        proxy_pass http://host.docker.internal:9999;
    }
}

```

(continues on next page)

(continued from previous page)

```
}
```

NOTE: To read InfluxDB v2.0 database, the client shall use OPTIONS-method, not GET-method which will be rejected. Therefore the CORS-proxy function is implemented only for OPTIONS-method. It is noteworthy also that *InfluxDB Out* will not need to go through the proxy, it can POST directly the data into the TCP/IP port 9999 of the server. But if a JavaScript client needs to do the same, it needs to go through this proxy and the POST-options needs to be implemented with the CORS-proxy functions.

It is nice to be able edit directly the *nginx*'s configuration file without sometimes complicated tricks. That's why we bind it to a local file system file as well.

TIP: In Windows Docker Desktop 3.3.3. we use `proxy_pass http://host.docker.internal:9999` instead of `proxy_pass http://dasht_influxdb:9999`: while the name resolving worked in earlier versions, in this version one needs to refer to Docker host which then resolves the address. Otherwise a 502 bad gateway error will be returned by nginx in this configuration.

9.6.4 InfluxDB read-back

While *Grafana* is the suggested dashboard tool for on-board and on-line usage of data, the InfluxDB API provides multiple possibilities for developers for on-line and off-line data retrieval for further analysis. While the development details are out of the scope of this document, we can list the following links for those who are interested to get more information:

- JavaScript/TypeScript: DashT *Line Chart* is a Grafana-wannabe line history drawing instrument using InfluxDB's TypeScript API to retrieve the data. You can find the corresponding module [here](#)
- Python Panda DataFrames are most convenient to analyze and plot the data off-line. Please see [this folder in the development repository](#)

9.6.5 Troubleshooting

Troubleshooting is done best with *nginx* server using an ordinary browser, by attempting to open the aforementioned ports 8088 (you would expect to find *DashT* JavaScript instrument's home directory to load the HTML5/JavaScript code) and port 8089 (with InfluxDB v2 running on port 9999 you would expect to drop on it's login page).

All browsers contain debug tools, for example if you want to open any *DashT* JavaScript instrument's `index.html` page you can get good information in case of eventual issues with the page loading like that. Please see the [troubleshoot section of EngineDJG](#).

In case you do not get connected anywhere, it would be worthwhile to open a console on the Docker container which is providing the *nginx* service.

NOTE: On **Windows**, you may have left the services running when shutting down, in which case the containers are started automatically. However, one cannot connect to them with Docker tools unless you have also *Docker Desktop* running. This, you may have selected not to start automatically. If this is the case you need to start it now.

TIP: On Docker Desktop for Windows one has a button Logs which allows to see similar information as below in case one does not like the command line.

Check first that you have, indeed the containers still running:

```
docker container ls
```

From command line, attach the console to the *dasht_nginx* container:

docker container attach dasht_nginx

Try to reach the above connections (ports) again and observe the console. It should say, if working correctly access to all ports (or error messages if an issue):

```
172.20.0.1 - - [07/Sep/2020:09:54:59 +0000] "GET / HTTP/1.1" 200
1301 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/85.0.4183.106 Safari/537.36" "-"
172.20.0.1 - - [07/Sep/2020:09:55:00 +0000] "GET /favicon.ico
HTTP/1.1" 200 1150 "http://localhost:8088/" "Mozilla/5.0 (Windows
NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.106 Safari/537.36" "-"
172.20.0.1 - - [07/Sep/2020:09:55:04 +0000] "GET
/orgs/59f148cfc72908cc HTTP/1.1" 200 313 "-" "Mozilla/5.0 (Windows
NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.106 Safari/537.36" "-"
172.20.0.1 - - [07/Sep/2020:09:55:04 +0000] "GET /5e93c5f5aa.js
HTTP/1.1" 304 0 "http://localhost:8089/orgs/59f148cfc72908cc"
"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/85.0.4183.106 Safari/537.36" "-"
...
```

Stop the console attachment with Ctrl-P Ctrl-Q key-combination - this way the container will continue running. To get back to command prompt after this, use Ctrl-C as usual.

INFLUXDB OUT

DashT InfluxDB Out is an “instrument” which is used to start and monitor a background streamer which takes all or selected data received or created by *DashT* and formats that data so that it is compatible with *InfluxDB v2.0 time series database*.

InfluxDB [Line Protocol to write data](#) is used. Users can select from two different methods: either direct streaming into InfluxDB 2.0 HTTP server or writing to a Line Protocol-format file which is can be later imported in the database using InfluxDB 2.0 web user interface.

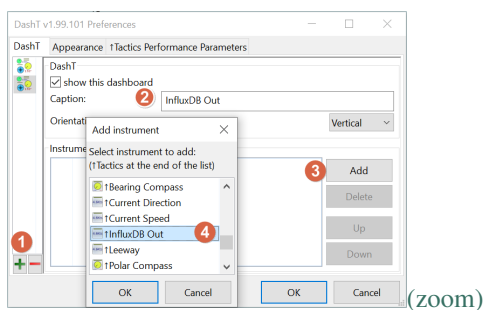
Since the Line Protocol-format file method does not require the presence of *InfluxDB v2.0* server instance while underway, it is easier to understand so let’s explain it first.

10.1 Line Protocol File

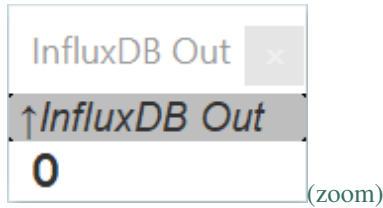
This is the easiest way to get started since no running Influx DB 2.0 server is needed while data is collected. It is also the fastest way to collect data for InfluxDB, limited only by the writing speed of your hard disk. Potentially, but depending of your boat’s instruments, over thirty different types of measurement values received by various Dashboard Instruments can be saved, some of them a few times per second. Since modern disks can easily store data with these rates and volumes, it is the safest option what comes to the performance and the ease of use.

Start your OpenCPN v5.2 or superior with *DashT* plug-in.

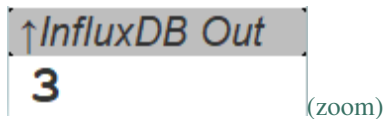
Open the *DashT* Preferences dialog and create a **new instrument panel with a single instrument in it: *InfluxDB Out***:



NOTE: Albeit you can have many *InfluxDB Out* instruments, only the first one is actually doing any streaming - the others would only do monitoring. It can be confusing to remember which one is doing the actual work and one can accidentally delete it... Therefore it is advised to have only one *InfluxDB Out* instrument, alone in a dedicated instrument panel: if you mix *InfluxDB Out* with other instruments, and then reorganize those other instruments, the communication and streaming thread can get orphan below the *InfluxDB Out* while the panel gets reorganized and would halt. There is no way out of the halt condition other than restarting *OpenCPN*.



That's it! Data coming in from OpenCPN (there is always some, like it is giving position data being) is already collected and a background thread is busy writing those into a file.



Looks small? Well, that's hundreds of lines written out, after a minute or so 300 lines have been written...

10.1.1 File content

Below is an example of InfluxDB Line Protocol file (small part of it only after a few seconds of running) created by *InfluxDB Out*. It can be dropped directly into the [InfluxDB v2.0 data collection user interface](#) as it is.

NOTE: *InfluxDB v2.0* will ask you the precision of the timestamps: it is in **milliseconds** as you can see in the data of the excerpt below.

```
...
navigation magneticVariation=-4.000000 1598810821136
navigation headingMagnetic=145.000000 1598810821136
navigation headingTrue=141.000000 1598810821136
navigation speedThroughWater=0.000000 1598810821140
environment,prop1=temperature water=21.600000 1598810821140
environment,prop1=belowTransducer depth=15.400000 1598810821384
environment,prop1=directionTrue wind=20.000000 1598810821384
environment,prop1=speedTrueGround,prop2=twswind=12.200000 1598810821384
...
```

You may be wondering about the format, where there is a resemblance with Signal K data. That is true, Signal K data model is used but what you see is created by a *DashT InfluxDB v2.0 database schema*. The purpose of this schema is to make both writing and reading of data more efficient, one would be drilling first to *environment* data, then *wind* and only after seeking *directionTrue*. This allows data to be organized in continuous chunks, thus speeding the operation and reducing the number of reads needed to reach the value searched for.

NOTE: If you are a database specialist or otherwise do not agree with the *DashT InfluxDB v2.0 database schema*, the good news is that it is totally configurable in the JSON-format configuration file described below. One can, if it is the wish, to move use single Signal K key in CamelCase, in order to be compatible with *Signal K server node* way of writing data in a database. No tests, however, have been carried out without applying a schema in write / read operations.

10.1.2 File location

Windows: C:\ProgramData\opencpn\plugins\dashboard_tactics_pi\streamout

Linux: ~/.opencpnplugins/dashboard_tactics_pi/streamout

10.1.3 Backup copies

At each start of *InfluxDB Out* an existing streamout InfluxDB 2.0 line protocol file is renamed as streamout_bup_YYYY-MM-DD_hh:mm:ss where the YYYY-MM-DD_hh:mm:ss is the time of the backup and **not** the time of the data inside that file:

To find out the period of the data **inside** a backup file open it an use on-line service accepting timestamps in milliseconds such as <https://www.epochconverter.com> to convert the timestamp of the first and last line.

10.1.4 Importing to InfluxDB

InfluxDB has multiple file import sources available in its graphical user interface. Typically you would select Load Data -> Buckets -> Add data and then drag and drop the line data file into the file selection box.

NOTE: The above would work without a problem for a file registered during a regatta or such. Files registered over longer period may grow too big for the InfluxDB network based file loader (limit is, as this is written **10MB** file maximum). In this case one need to split the big file smaller. For Windows, one can use simple [FileSplitter](#), while for Linux one can say `split -l 80000 input_file output_file`. To give you an idea, 48 hours of continuous navigation creates a line protocol file of size 50MB, approximately.

CAVEAT: in splitting a line protocol file by megabytes (size) the last line of the each file and first line of the next one are split blindly in two and, consequently InfluxDB would detect a line protocol error and refuse to upload the file. The solution is to cut the last, incomplete line of each file and complete the first line of the next one with it. This sounds cumbersome but it is not that much of work. However, prefer the per line splitting method if you can.

10.1.5 Configuration file management

The configuration file and the database schema are in a single JSON-file, located in the same data directory indicated above. It can be edited. There is a template file located in the plug-in's program directory dashboard_tactics_pi from where it should be copied (and not modified) if a need arise - see below for HTTP template file for more details.

It is possible to have more than one configuration settings and, consequently more than one database schema - you can add tags, or otherwise modify the schema to facilitate the data browsing. You can also switch between file-based and [http://](#) based streaming operation. But only one configuration file can be used at a time. Select its name in the *ini/conf file*.

10.2 HTTP Streamout

HTTP Streamout into InfluxDB v2.0 is needed if you are using Grafana visualization when underway, or if you want to use *DashT Line Charts* which are Grafana type (modestly imitating) instruments but embedded in Dashboard, using data retrieved from InfluxDB v2.0 database and applying aggregation functions implemented by InfluxDB v2.0 on that data.

Make a copy of a template file streamout_template_http.json from:

Windows: C:\Program Files (x86)\OpenCPN\plugins\dashboard_tactics_pi\data

Linux: `./share/opencpn/plugins/dashboard_tactics_pi/data`

and place it here, with this name (or change the name to your liking):

Windows: `C:\ProgramData\opencpn\plugins\dashboard_tactics_pi\streamout_http.json`

Linux: `~/.opencpnplugins/dashboard_tactics_pi/streamout_http.json`

Select the above path and name in the *ini/conf file*.

Modify `streamout-http.json` file to contain your InfluxDB 2.0 connection parameters, see *Set Up InfluxDB*:

```
"org"      : "myorg",           // HTTP: Influx DB organization name
```

You have selected an organization (boat?) name in InfluxDB 2.0. Type it here in place of `myorg` - exactly as you defined it in InfluxDB 2.0 user interface.

```
"bucket"   : "mybucket",       // HTTP: Influx DB bucket to write
```

Data is written in buckets of InfluxDB 2.0. Create one if not yet done and give its exact name here.

```
"token"    : "ToLdk3DNs3PqbKNS2hdZMure.....E0eu4lE0OUWRt8w=="
```

Tokens are unique, per each server, and sometimes even per each bucket in it. Go to Tokens-tab in Settings of InfluxDB 2.0 user interface and generate one. Copy the new token into clipboard and paste it between string quotes in the configuration file. Do not allow a carriage return, it must be a **single line string**.

Now you are ready to try. But if it does not work? There is debugging information available in OpenCPN log file, read more about debugging below. But before that you must turn the debugging on in the configuration file and restart the *DashT InfluxDB Out* “instrument” (often it is easier to restart the OpenCPN):

```
"verbosity" : 3 // 0=quiet,1=events,2=verbose,3+=debug
```

10.3 Conversions

10.3.1 Stream from VDR

It is possible to convert old OpenCPN *Voyage Data Recorder plug-in* log files into a historically time stamped data in InfluxDB v2.0 time series database by making a VDR data play back.

The following major criteria must be fulfilled:

- The data contains GPS timestamps
- You do not have *Calculate SOG-* option in OpenCPN Preferences checked

The resulting database entries are organized with timestamps generated by *DashT* which gets its time synchronized at every arrival of the GPS data with a time stamp. In between, the time tick is running with the CPU clock, until getting synchronized again. While the intermediate data arrival rate is dictated by the playback speed in the VDR player, the error in successive data sentences is not cumulative throughout the file, only between two GPS time data sentences.

NOTE: The time shift backwards is possible because if *DashT* detects a difference greater than five seconds between the local CPU time and GNSS (like GPS) provided time it switches to use the GPS time for timestamps and calculations. This shift is announced in the OpenCPN log file only, there is no pop-up message - taking it as granted that in off-shore use the racer is not necessarily synchronizing his/her PC with the satellite provided time anyway. Depending of the ratio of the time data arrival, this may lead to a short range with time values being the current (CPU) time before the data is timestamped with the “historical” timestamps. If this is an issue, remove those few tens of lines from the line data file manually, before uploading into the InfluxDB v2.0.

10.4 Debugging

The output file, *streamout* is a text file which can be visualized and its filling can be followed up with the command line tools.

It is possible to increase also the verbosity level of the log messages, values 3 - 5 makes more debug printing out in the OpenCPN log file.

10.4.1 Level 4

Verbosity level 4 is printing out every second the status of the FIFO queue between the streamer instrument and the file writing and data communication thread, which is useful if buffer overrun is suspected

On Windows PowerShell:

```
PS C:\ProgramData\opencpn\plugins\dashoard_tactics_pi>  
Get-Content ./streamout -Wait -Tail 20
```

```
PS C:\ProgramData\opencpn>  
Get-Content ./opencpn.log -Wait -Tail 20
```

On *nix systems:

```
tail -f ~/.opencpnplugins/dashboard_tactics_pi/streamout
```

```
tail -f ~/.opencpn/opencpn.log
```

10.4.2 Level 5

Level 5 is useful to debug issues with the HTTP, printing out the entire header and data POST/OPTIONS message (one may want to **reduce the number of lines** to be sent per message in this case)

LINE CHART

Line Chart instrument is a graphical display tool which brings you *Grafana* type of monitoring tool inside an ordinary Dashboard instrument pane.

Why not Grafana, then? - *OpenCPN* is based on *wxWidgets* cross-platform GUI library. It allows *OpenCPN* to be available on large number of different platforms. The price to pay is that *wxWidgets* needs to be extremely conservative on its implementing back-ends. The *wxWebView* component is needed here and its back-end browser technology is very old. For example, on Windows it is at the level of *Internet Explorer* 8 or, at best *IE11*! One cannot run modern web-developments on it but *DashT* takes a modern line graph library *tui.chart*, *polyfills* it, and writes for you software which allowing to use *InfluxDB* aggregation methods on the data read back from that very same time series database. Resulting function is similar to the to the popular functions provided by *Grafana*.

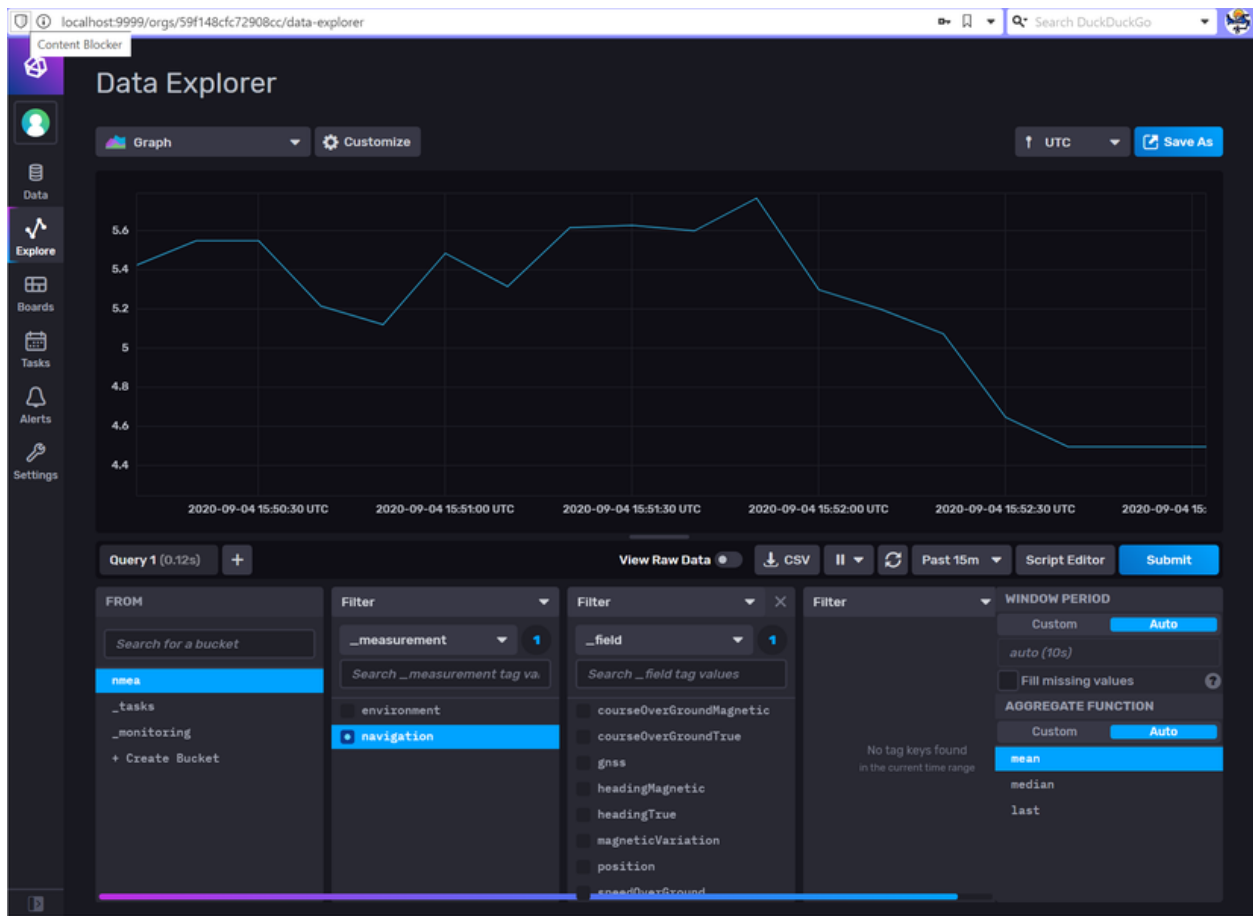
11.1 Introduction

Let's take a look at the data we are recording already with *InfluxDB Out* into the *InfluxDB* time-series database. The database is not only a bunch of buckets where it collects data but it has also its own set of aggregation functions which can be used to analyze and further filter the raw data and return that data to the sender, such as *DashT*. Typical use is with off-line analysis which is described below.

NOTE: *InfluxDB* also sports its own Dashboard layout graphical user interface/application which you are encouraged to study!

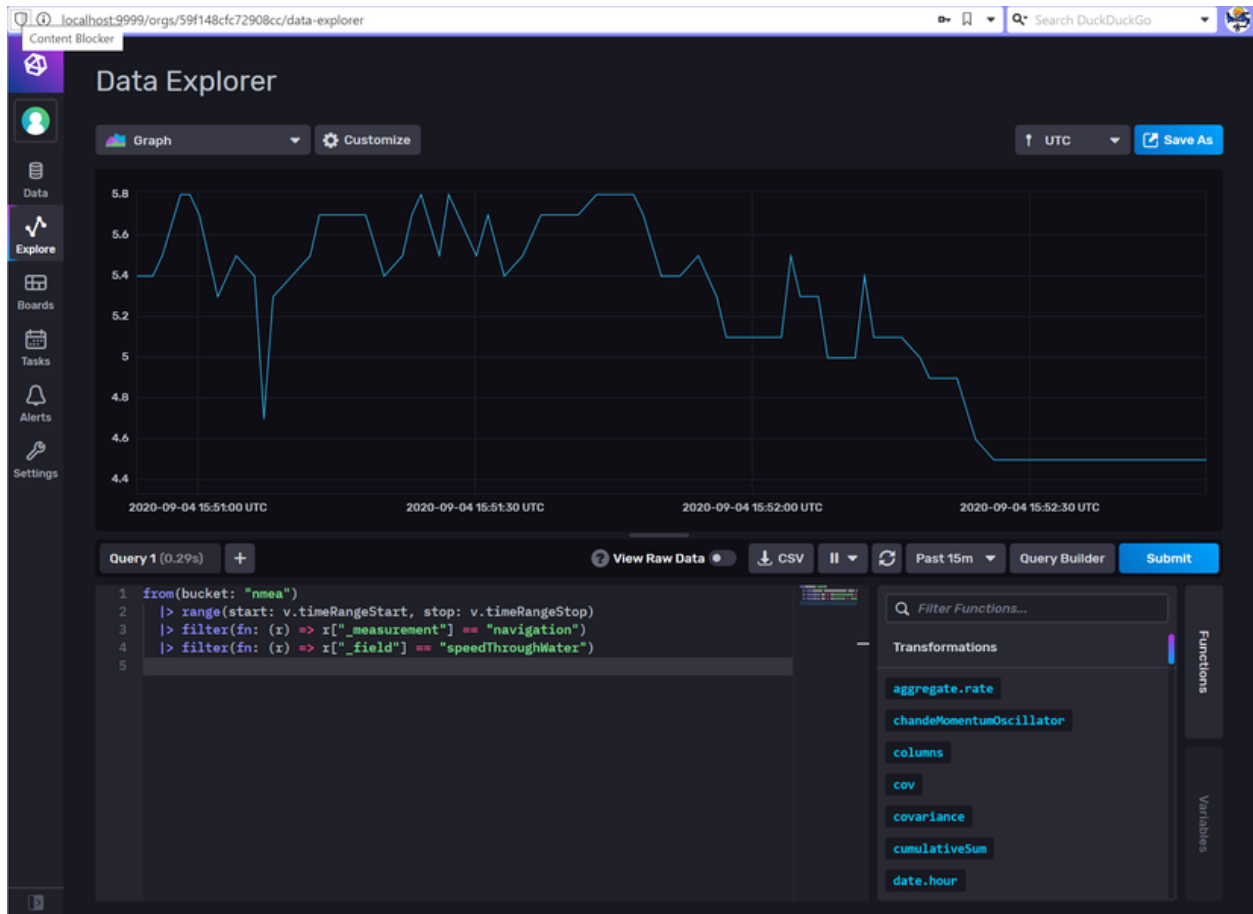
In this use case, we are interested in finding rapidly the tendencies in the Speed Through Water (STW), indicating that we are losing the speed. In our instruments our paddle wheel produced data is going unfiltered and jumping up and down so it is no easy to tell anything about tendencies before they become evident even without instruments...

We open *InfluxDB* and take a look at that data, which in this case *InfluxDB* shows by default with *mean* aggregation filter:



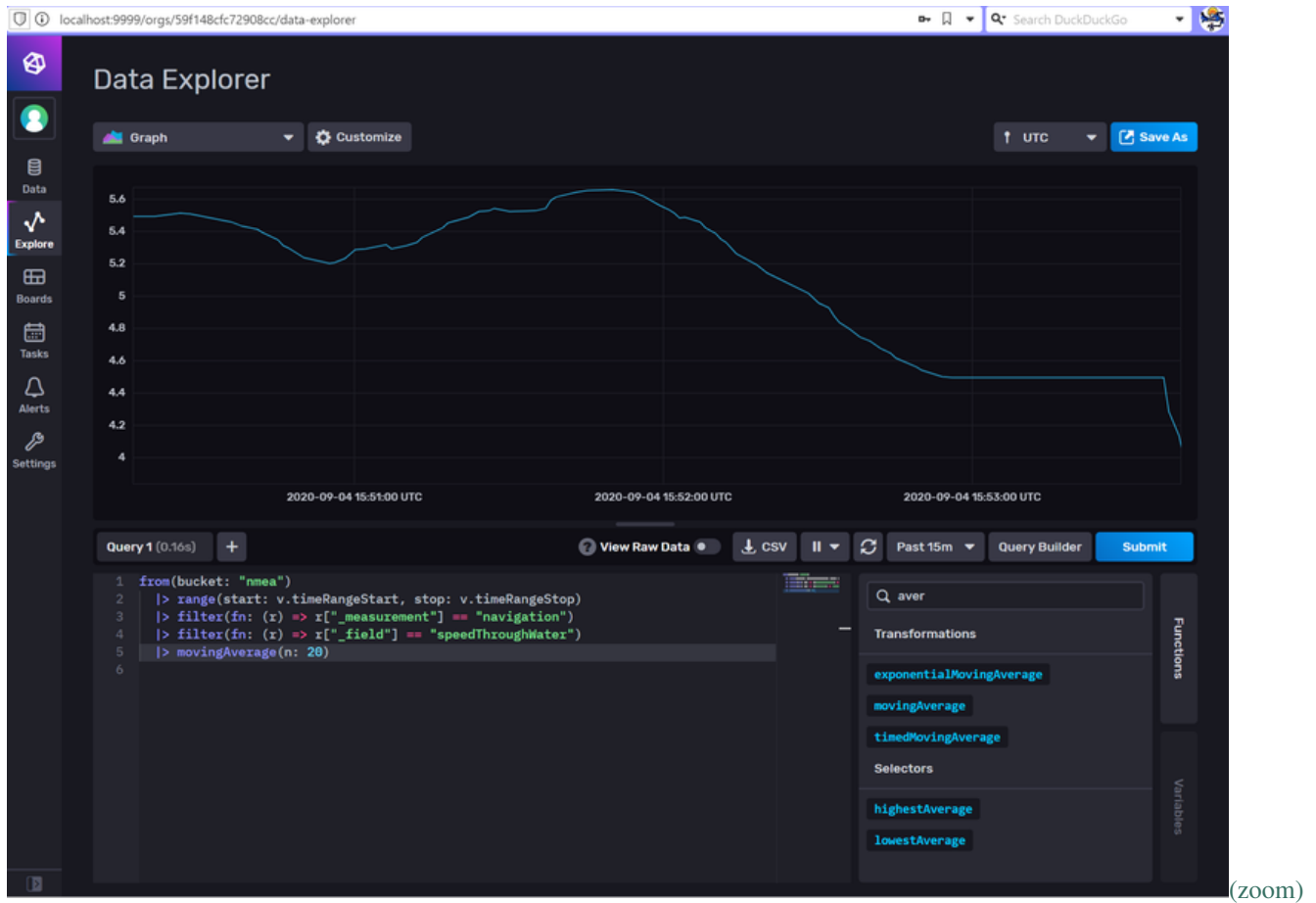
(zoom)

How does the raw data looks like? :Let's remove the *mean* filter:



(zoom)

What about applying Moving Average?

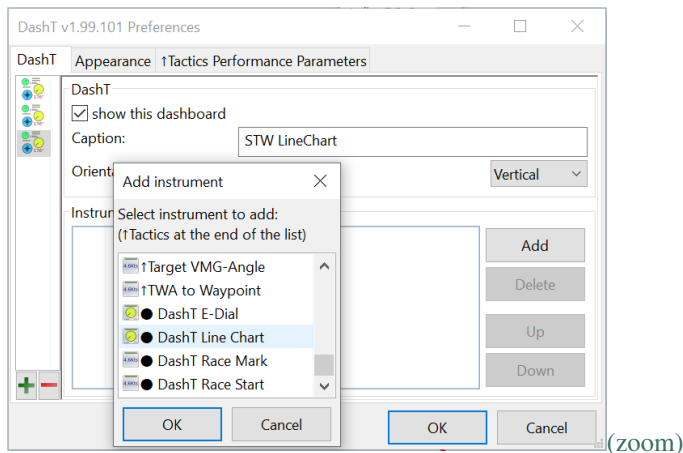


Looks much better.

Now, if we could just read that back to the *DashT* dashboard... No problem!

11.2 Setting up

Line Chart instrument is set up like other *DashT* network augmented instruments, by adding it from the *Preferences* menu into a Dashboard instrument pane:



Selecting the data source is like with an *EngineDJG*, by subscription. Only that we do not subscribe to any data which is coming in, but to the data which is sent into the *InfluxDB* time-series database by *InfluxDB Out* using HTTP protocol. Only this way one has some data in the database which can be read back.

The instrument is listening, for a while the data which is effectively sent to the database and let you select one of those.

NOTE: The below example works out of box so you are not necessarily required to read the description how it is done. But if you want to make changes it is required for understanding where the settings are coming from.

In order to be able to display something reasonably well, the *DashT Line Chart* needs still to know about how you want that data being displayed - *InfluxDB* does not, for example store the units of the values.

DashT Line Chart gets the default values for display formatting from the common configuration file, in this case these are the settings out of the box:

```
{
  version      : 1,
  path         : 'navigation.speedThroughWater',
  title        : 'STW',
  symbol       : '',
  unit         : 'kn',
  display      : 'chart',
  decimals     : 2,
  minval       : 0,
  loalert      : 0,
  hialert      : 0,
  maxval       : 30,
  multiplier   : 1,
  divider      : 1,
  offset       : 0,
  dbfunc       : 'movingAverage(n: 20)',
  dbnum        : 0,
  wrnmsg       : false
},
```

Compare this to the settings in the *InfluxDB Out* configuration file, in:

Windows: C:\ProgramData\opencpn\plugins\dashboard_tactics_pi\streamout_http.json

Linux: ~/.opencpnplugins/dashboard_tactics_pi/streamout_http.json

```
{
  "sentence"    : "OCPN_DBP_STC_STW",
  "mask"        : 4,
  "store"       : true,
  "interval"    : 1,
  "measurement" : "navigation",
  "prop1"       : "",
  "prop2"       : "",
  "prop3"       : "",
  "field1"      : "speedThroughWater",
  "field2"      : "",
  "field3"      : "",
  "skpathe"     : "navigation.speedThroughWater"
},
```

The database streaming configuration sets up database schema, only addressing the data storage breakdown. In addition it defines - internally to *DashT*, not in the database - a Signal K data compatible key to make association with the database schema keys. But it does define in any way what is the unit of the data (knots in our case) or how do you want to call it (STW).

NOTE: *Signal K In* is not required as the data source, it can be also *NMEA-0183*, it is only that a Signal K data key is used to identify the database schema used to store it.

The *DashT Line Chart* configuration file is shared with other similar instruments. It is located in these folders, after installation:

Windows:

```
\Users\Public\DashT\www
```

Linux:

```
/usr/share/opencpn/plugins/dashboard_tactics_pi/data/instrujs/www/
```

It contains instructions how it can be modified and how the instruments are told to take into account your changes.

Here are some interesting configuration values in our use case:

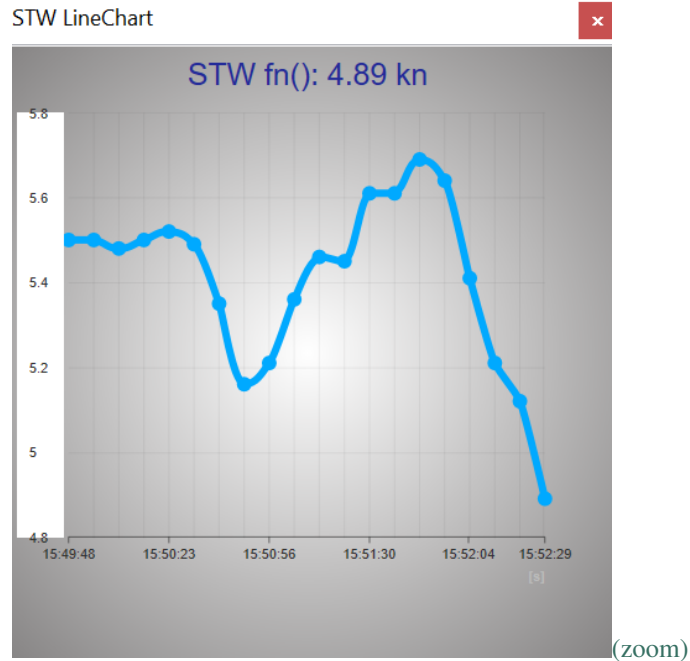
```
dbfunc      : 'movingAverage(n: 20)',
```

We can add a database function to the inquiry for data base data. In this case the database inquiry is build using the database schema and this string is added to the inquiry, and the resulting datapoints are returned with applied moving average function where $n=20$ (see the above screenshot from the InfluxDB, it is exactly the same line we add). The presence of a database function is shown with a symbol `fn()` next to the data title.

```
decimals    : 2,
```

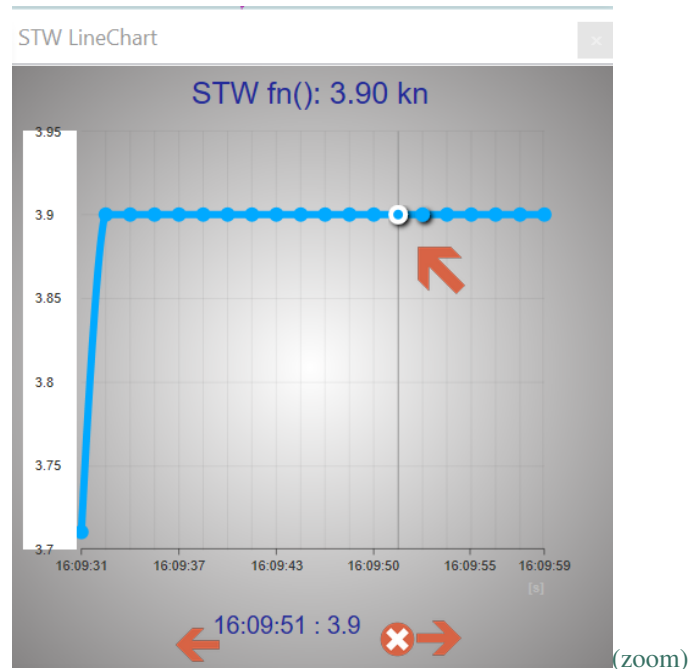
By default one decimal is used but in this case where we use averaging function, it is better to use two decimals. This way, the line chart is moving to the right direction continuously, otherwise the rounding of the values will make line being stable for a while, then jumping to the next decimal, etc. But if your values are two digit values by default, then one decimal is certainly enough. Please feel free to experiment to find a setting which works best with your data.

So finally, here's the result (out of the box):



Pretty cool!

11.3 Numerical values



It is possible to read the numerical value of the historical graph data - the value in the header is the latest received value.

Place the mouse cursor over the bottom part of the graph and move it from left to right. Above the cursor one can observe the numerical value of each data point - in this example they are averaged values returned by the database function.

DASHT RACE START

DashT Race Start provides numerical and graphical functions as an overlay on the OpenCPN chart plotter. It allows you and your team to practice and implement the best strategies on the race start area and optimize the start line crossing.

DashT Race Start is intended to be used in conjunction with *DashT Race Mark*.

The start area and start line functions are mostly applicable both in keel boat and dinghy racing: *DashT Race Start* can be used also in the class room: please see [OpenCPN supplementary software documentation](#).

DISCLAIMER: This software is for educational purposes only. By using it you accept the conditions presented at the first usage of the application, and being equally available in the [Introduction](#) section of this document. The usage for any other purpose is under your entire responsibility and the liability of the author(s) of this software is not engaged. Please respect [Convention on the International Regulations for Preventing Collisions at Sea, 1972 \(COLREGs\)](#) and race's rules at all times.

12.1 Introduction

DashT Race Start is intended for races where the first leg is a windward leg, *i.e.* the start line is heading to the wind.

The start area is defined by the start line. It is either pre-determined using OpenCPN's Route Manager or markers are dropped on the OpenCPN chart canvas by sailing next to the committee boat and the opposite marker.

The start area behind the line is a square, by default 1 x 1 nautical mile - The characteristics can be changed in [Tweaks](#).

The Position and the Course on Ground of the boat in the area are defined by the boat's GNSS system - there is no correction to attempt to set bow's exact position on the chart canvas.

The start procedure is the five minutes, four minutes, one minute to the start.

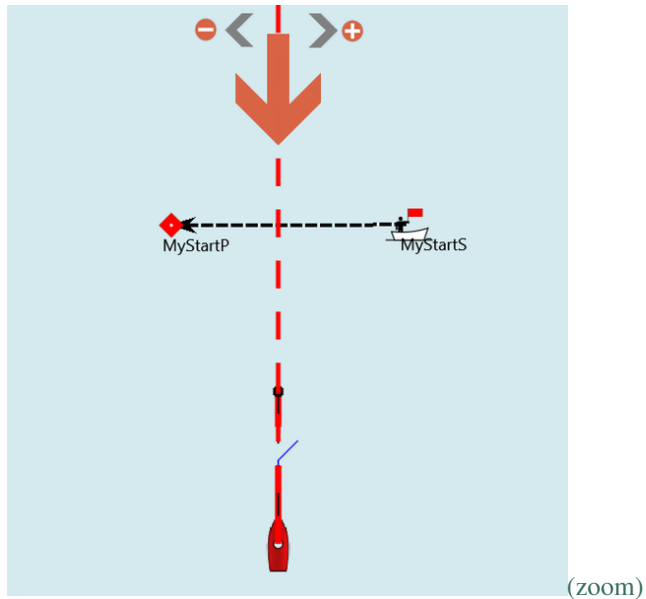
During the start procedure the wind shift is constantly monitored and reported both graphically but also as virtual start line with wind bias.

Laylines are set and attached to the markers. They are turning with the wind shifts.

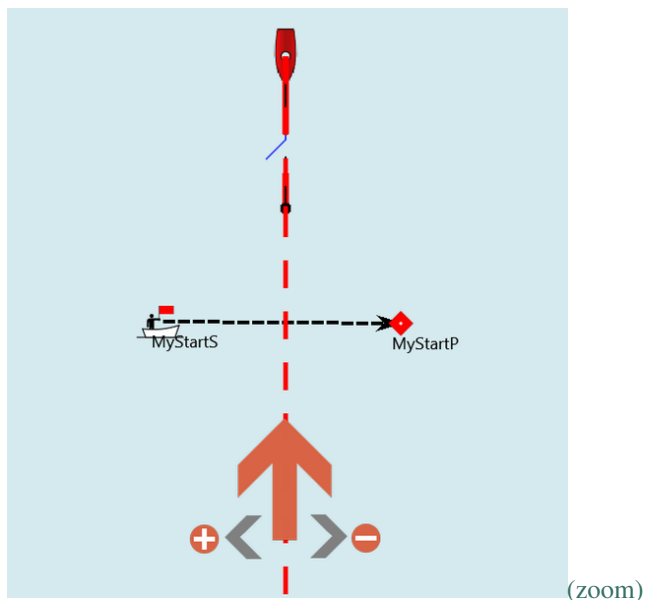
Zero Time To Burn point is shown ahead of the bow when a [Boat Polar](#) is provided.

12.2 Define Line

DashT Race Start is designed for a race start procedure where the start line is set so that the wind is heading it perpendicularly. There can be pre-determined positions for starboard marker and the port marker. If none, you can drop your own virtual marks by sailing next to the markers. One of the marker can be the racing committee's boat, do not sail too close to them! The below diagram depicts a northbound race start.



And the below diagram for a southbound race start, for clarity (actually, you can just turn the diagram above to whatever direction the race line happens to be).



NOTE: In the case of a downwind start procedure (*i.e.* the wind is not behind the start line) nothing will be shown in the start area.

12.2.1 Wind shift direction

Certainly one of the most important aspect to manage before and after the start line is the wind shift. The terminology is the following:

(+) = wind is veering = turning clockwise = turning towards the starboard mark = turning to your right



(-) = wind is backing = turning counter-clockwise = turning towards the port mark = turning to your left

12.2.2 Pre-determined markers

Start-line markers the position of which is provided by the race organizer are convenient, albeit one can ask the question how perpendicular will be the wind to the start line? *DashT Race Start* will help you with that!

NOTE Since the marks can drift in real life, you may want to sail next to them - it will be easy to snag a pre-positioned marker in *OpenCPN* at that moment. In this case you may find that using *DashT Race Start* dropped marks and resulting real-time start line is as convenient way to mark it.

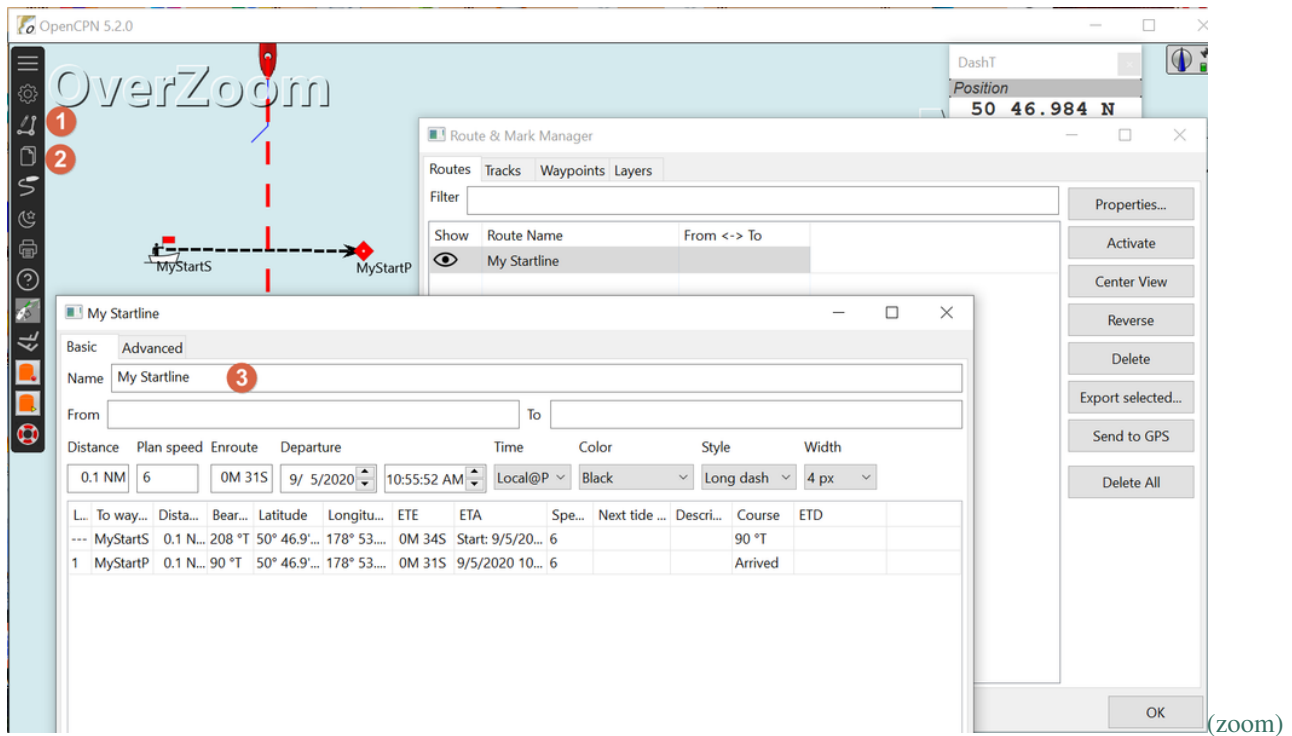
With the data provided by the organizer, use *OpenCPN* route manager and create the two following waypoints - note the names, you must use the names *MyStartP* and *MyStartS* for port and starboard markers, respectively. You do not have to use the fancy icons, any icon will do, only the name and position coordinates will be used.

	Always	MyStartP
	Always	MyStartS

(zoom)

Using *OpenCPN* route manager's create route function (1) set up a route between the two points. The direction of the route is not important. However, when the route manager suggests to "use a nearby waypoint", select "Yes". An *Esc* finishes the route creation.

Now the final of the three identifiers *DashT Race Start* needs to find your start line: give it a name which shall be "My Startline" (3) (evidently...). In the route editor (2) you can set other parameters for your startline (which will not be the case for the *DashT Race Start* created startline):



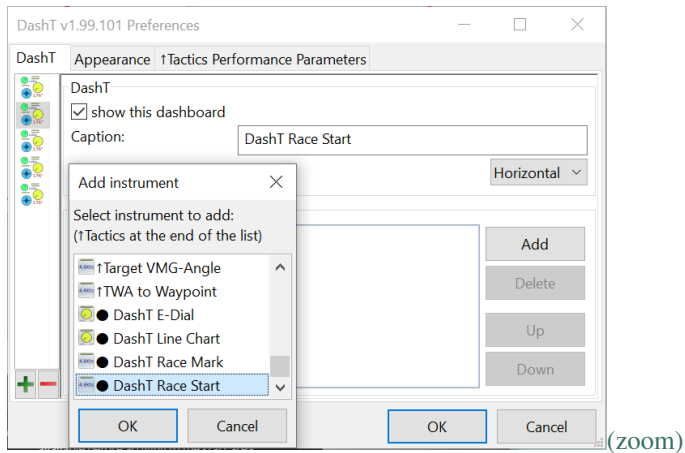
NOTE: When you approach the startline, you need to **activate** your start-line “route”. *DashT Race Start* is searching for an user route by constantly polling for the existence of an active user route with the above characteristics. At this moment you can also hide your start line “route” in order to reduce the number of lines on the chart canvas. Keep the waypoints (markers), though when the route manager asks do you want to do so.

12.3 Approaching Start Zone

After the above (optional) preparation you need to start *DashT Race Start* application. Do not expect the below happening automagically without you making some preparation and trials before: the service uses network software (albeit in the “network” being only your computer) - the network service function part is **not** part of standard *OpenCPN* installation (v5.2 when this is written).

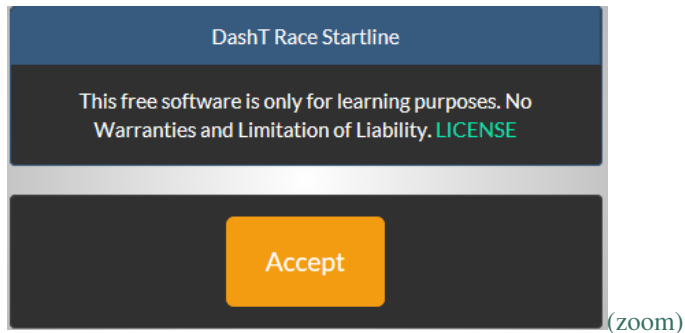
NOTE: Albeit there is no need for any database services or such, please see the corresponding chapter, [InfluxDB / Docker](#) anyway, for the easiest possible way to get all network based services of *DashT* available with a simple push-button start (Windows) or by a simple command (Linux). Of course, you can have your own way to set the required service, something like how it is done in [EngineDJG script](#). But if nothing is set to retrieve the *DashT Race Start* over the network, local or other, nothing explained below will take place!

12.3.1 Open application

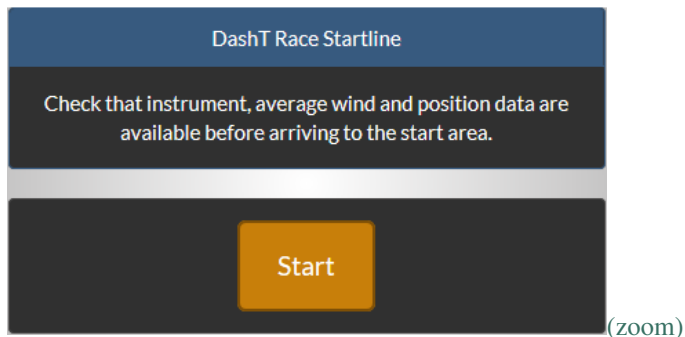


As usual with complex applications with plenty of buttons and large real-estate area, it is better to keep them alone in a dedicated *DashT* instrument window.

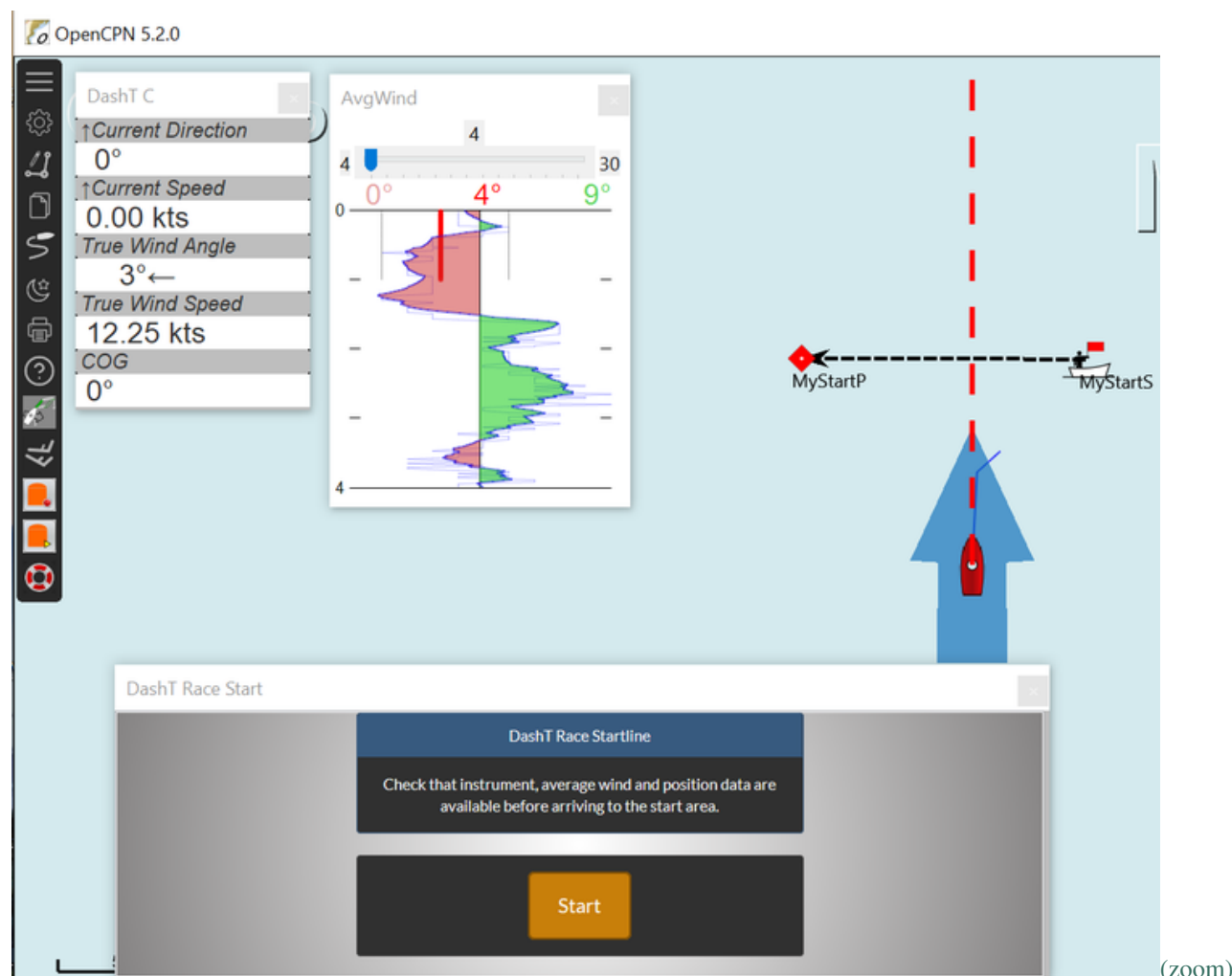
For the first time initialization, you are requested to acknowledge and accept the free software license and limitations of liability. By continuing you give your acceptance. If a new application of the same type is created the acceptance is asked again. For this one, this screen will not reappear unless the ID of the instrument is changed in the *ini/conf-file*.



The *DashT Race Start* will enter the standby state, leaving you time to verify with the other *DashT* instruments that it can receive all data it needs.



Let's check that that we have everything needed for a scientific approach to the start line:



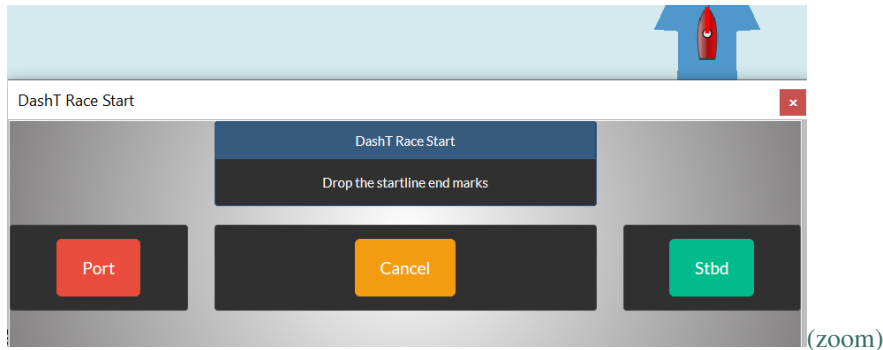
We need all the below, otherwise the *DashT Race Start* is not able to figure out the fast moving situation once in the start zone. Please consider:

1. **Average Wind** - this is the instrument which allow you to manage the wind shifts. This is the key factor to define the end of the start line you should target. Set the integration time according to how nervous the wind is, shorter for continuously turning wind. Read more from *Average Wind Instrument* to complete master this important instrument!
2. **Current Direction** - an other extremely important parameter. Note that in *DashT* Tactics-algorithms are used to calculate the actual current (no predetermined tables are used): it is a vector calculation containing the boat's Leeway and the Speed Through Water (STW), therefore keep your boat moving and your paddle wheel clean! Read more: *Calculate the surface current*
3. **True Wind Angle** - *TWA*
4. **True Wind Speed** - *TWS*
5. **Course Over Ground** - *COG*

NOTE: from the presence of *COG* you can understand - and it goes without saying - that the application is depending on your global navigation satellite system (GNSS) data. Therefore it is not your boat's bow you are following but the position of your antenna (such as GPS antenna) unless your boat's navigation system takes into account its position offset. So far there is no information for that in *DashT Race Start*, so please take care.

12.3.2 Drop marks

NOTE: If you have your own, organizer provided startline (see above) and you have **activated** it, this step will be skipped.

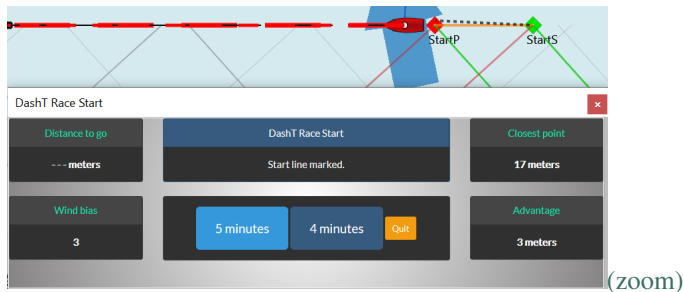


This is (hopefully) self-explanatory. Ask the boat to be passed next to the marks, preferably climbing to the wind so that speed will drop in order to increase the accuracy of the dropped mark. Once the marks have been dropped, they can be readjusted using the *OpenCPN* route manager. You can also decide to start all over if you have time for that but usually this should not be necessary.

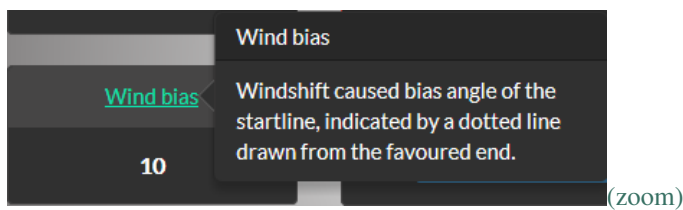
NOTE: Position the *DashT Race Start* application window so that you do not have to move it anymore, if possible: be aware that if you (accidentally) dock it, the application will be restarted. Do not panic since the only thing you can lose is the count-down timer, the application will catch up with the rest but the time is definitely lost and you will lose in concentration, for nothing.

12.4 Start Zone

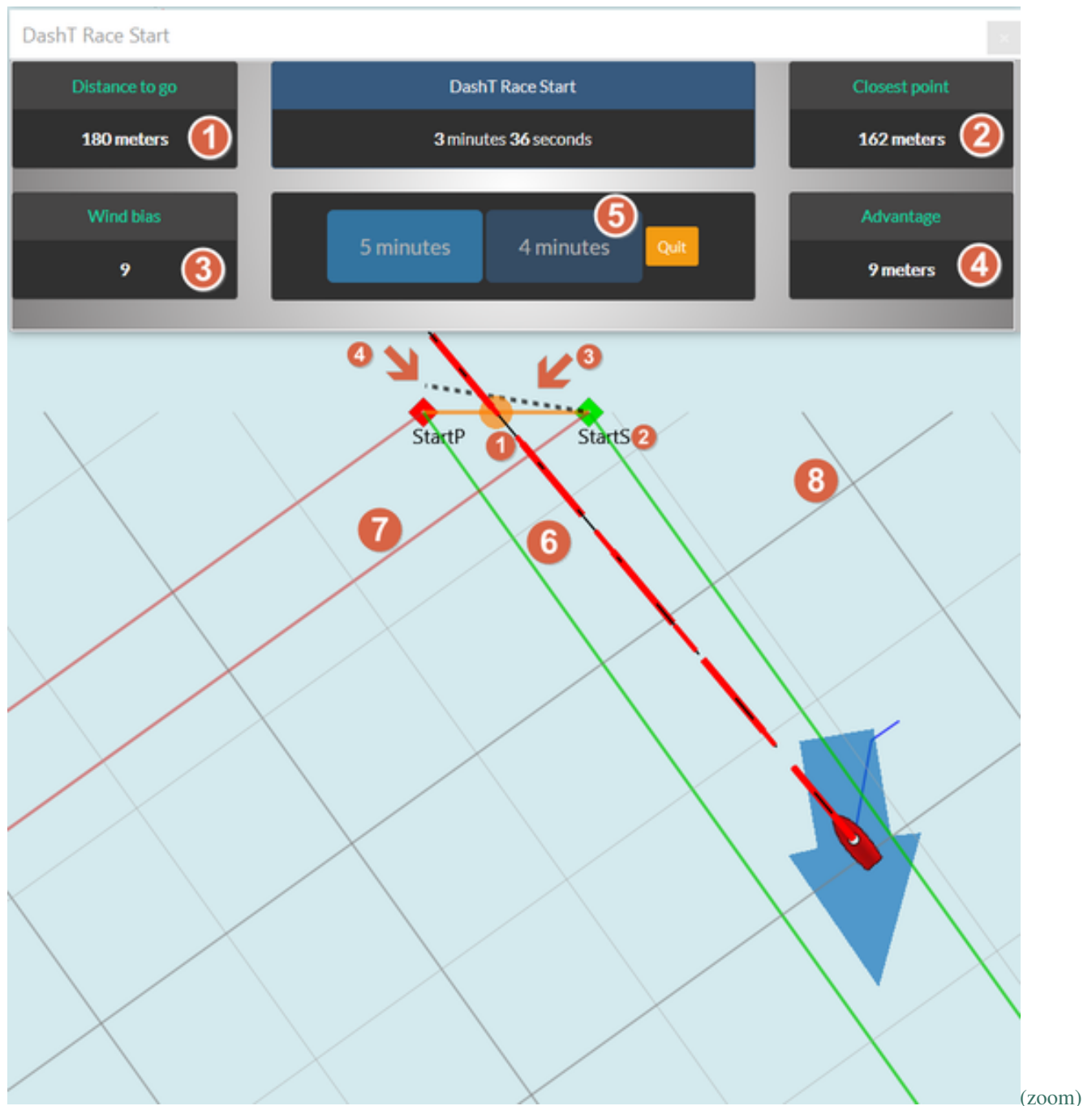
As soon as the start line has been defined either by dropping the second mark or by detecting an active user defined start line, the application calculates all data related to the start zone area, displaying the numerical values in the application window and many useful and perhaps even less useful lines and dots on the chart canvas. Of course, you can control what will be shown to your liking in the *ini/conf-file*.



For the numerical values and what they mean you will get pop-up help by hovering your mouse over the title and keeping it there for a short moment:



We will now move to the start zone, take a breath, point towards the start line to see what he have got over there...



1. **Distance to go** - shown in case the COG indicates you will effectively cross the start line at this point
2. **Closest point** - normally the perpendicular distance to the start line, but in the example case the closest point is the starboard marker, perhaps the committee boat
3. **Wind bias** - Long term average wind angle which indicates the angle the wind has been deviating from the theoretical perpendicularity - positive values = it is veering, negative values = it is backing. The dotted line shown is the first ladder rung on the way to the windward mark. It is attached to the favoured mark. We do not go to the details of the theory of ladder rungs, see for example [this video](#).
4. **Advantage** - the maximum distance the non-favoured end would make you to lose on the ladder rungs compared to the favoured end. With this (demo) short start line the distance is only one boat length and therefore it would

not be the end of world if you cannot scratch the paint out from the committee point. This value is here to give you some perspective to the gains which can be *potentially* obtained.

5. **Timer** - If you miss the five minute signal you can synchroize yourself with the four minute one. Anyway, let's not think that this is your primary startup timer in your boat, so we do not steal real-estate of you chart plotter with BIG NUMBERS. Think it as a nice helper for the tactician to give him/her the Zero Burn notification (see below)
6. **Starboard laylines** to start line - Turning with the average wind, and since the wind bias is positive (wind is veering) the width between the two laylines is increasing. Confirms in this example our intuitive conclusion that the starboard tack will be easier to sail than the port tack.
7. **Port laylines** to start line - Turning with average wind. In this use case of veering wind, the distance between the two lines is getting narrower and the sailing towards the start line is not impossible but getting more demanding
8. **Distance grid** is turning with the laylines (with the average wind) to increase your situation awareness. The distance by default is 50 meters. The grid can be turned off, its appearance can be changed and the distance can be set in the *ini/conf-file*

12.5 The Heat Is On!

The Tactician has made a decision in his incredible wisdom! We are not going to get into the priority sump next to the starboard mark just to lose the speed in an avoidance manoeuvre. Instead, we are heading closer to the port mark and then attempt to optimize our speed by climbing to the wind as much as we can while still keeping up the optimal polar speed. Well, that's the theory, at least, let's see how it will work...

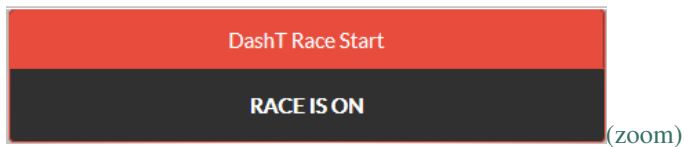
The Tactician has suddenly an information overflow which would make melt an average sailor's brain!



1. The **Bow** is pointing outside of the start line...
2. ... but **Course over ground** indicates nevertheless that we would end up roughly to the position he was thinking we should get...

3. ... certainly because of the strong tide **Current**!
4. But are we doing the optimal speed? For that he drops a *Tactics Waypoint* somewhere next to the start line, a bit more towards the port mark (not necessarily on the line)
5. The **live polar graph** is showing him that we doing about a little bit over optimal angle at least - for polar performance he can take a look at the Tactics instruments. So he decides that we shall get more speed now
6. It is the last minute to accelerate anyway: *DashT Race Start* indicates with a big grey square on the COG-line that there is **Zero Burn position** only a few boat's length's ahead! After that point, one needs to stop thinking and get and keep full polar speed of the boat to reach the start line in time.

12.6 The Race Is On!



We were not the first one to cross the line but certainly the one with the highest speed thanks to the well trained team and the excellent acceleration of our boat!

But no time to waste, the ladder climbing race starts, let's move to *DashT Race Mark* which we have prepared already with the race information.

DASHT RACE MARK

DashT Race Mark provides numerical and graphical racing functions as an overlay on the OpenCPN chart plotter. It allows you and your team to optimize both windward and leeward race legs by increasing the team's situation awareness and by helping the tactician in the decision making.

DashT Race Mark is intended to be used in conjunction with *DashT Race Start*.

The racing functions are mostly applicable both in keel boat and dinghy racing: *DashT Race Mark* can be used also in the class room: please see [OpenCPN supplementary software documentation](#).

DISCLAIMER: This software is for educational purposes only. By using it you accept the conditions presented at the first usage of the application, and being equally available in the [Introduction](#) section of this document. The usage for any other purpose is under your entire responsibility and the liability of the author(s) of this software is not engaged. Please respect [Convention on the International Regulations for Preventing Collisions at Sea, 1972 \(COLREGs\)](#) and race's rules at all times.

13.1 Introduction

DashT Race Mark helps you and your team to climb ladder rungs of a windward leg or to descent those of a leeward leg in the most efficient way. It provides you the numerical true wind information about the upcoming legs which allow you to plan well ahead and prepare the most efficient head sail to use for each leg.

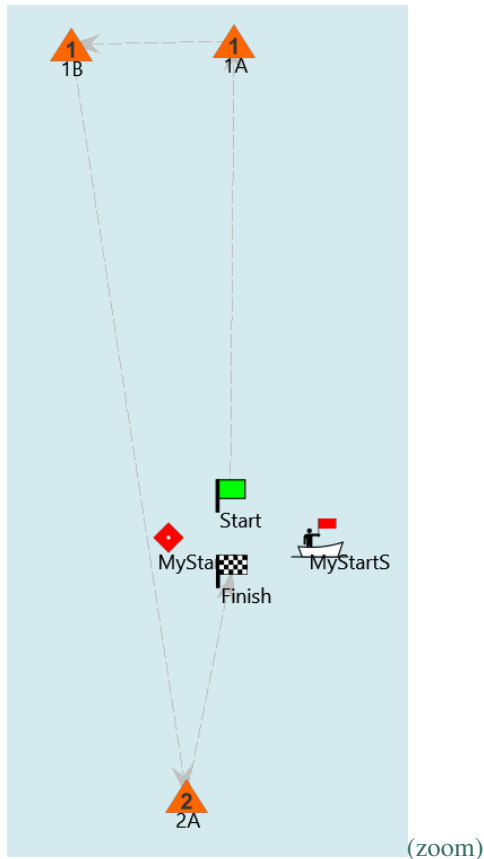
The race marks are defined and managed using *OpenCPN* Route Manager. It would be useful to get familiar with it before using *DashT Race Mark*. The skills needed are adding, naming and moving waypoints. Of course, one needs to be able to make route between those waypoints, activate/deactivate and modify the route which presents the race course.

TIP: Learn to use [Layers](#) feature of *OpenCPN* Route Manager. “Layers” are presented by GPX-files. Provided that you can get your race marks from the organizer as a GPX-files, import it as a “layer”, either a temporary or a permanent one. You can also include only the wanted marks before importing to avoid congestion. Briefly, “layers” are a great way to keep your race marks separated from the navigation data.

Without pretending to be able to present all the world's race courses, we concentrate here to explain an example based on simple windward/leeward race with a spreader. The distances are reduced in sake of the presentation on a single screenshot.

13.2 Define Race Course

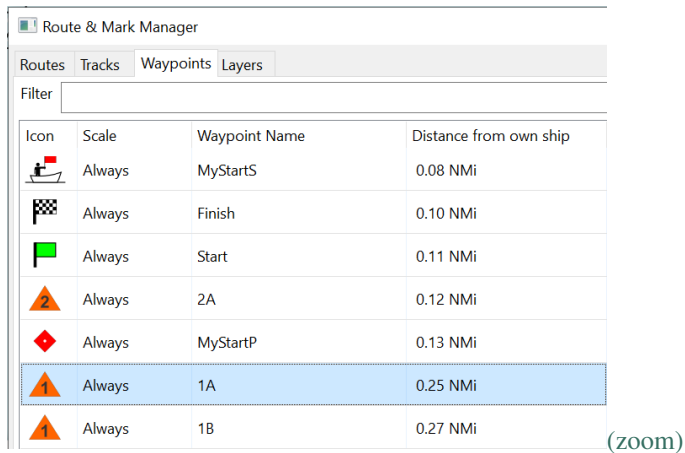
We continue this example from where we left with *DashT Race Start*, i.e. we have just passed the start line and we have a windward leg ahead of us, then a spreader mark which we can get to on reaching starboard tack. The leeward leg will bring us back down below the start line, which we have to reach in the final windward leg. We could make a second tour but for the needs of this example the finish line is on the start line.



To make the race course depicted above there is no need for anything else but the *OpenCPN* Route Manager.

NOTE: Please note that we do not close the route in a loop even if the race is for more than one round. There is no such a notion in the *OpenCPN* Route Manager. For this reason we leave a good distance between the Start and Finish marks - it gives us time to re-activate the route for the second round. The Route Manager automatically deactivates the route once you have passed the Finish mark.

Supposing that the race documentation gives you the names and exact locations of the marks you can enter them in advance into the waypoint list of the *OpenCPN* Route Manager. It is worthwhile to make them containing ordinal numbers if you already know in which order they have to be passed. This can be done by selecting their icons from the *OpenCPN* icon library which contains enumerated marker symbols:



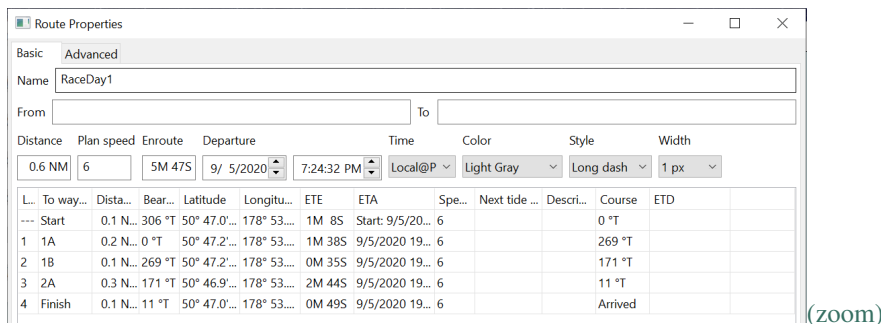
(zoom)

NOTE: It is very important to give **unique names** for each waypoint. Normally this would not be necessary, but since there is a [bug](#) in the way *OpenCPN* (v5.2) communicates us the next waypoint's unique identifier we cannot use that method for now. Therefore *DashT Race Mark* must rely on you to define a unique name for each of your waypoints representing a race mark.

TIP: While you are here in this dialog, it is a good idea to **reduce the default arrival radius** set by the *OpenCPN* Route Manager for each of your race course waypoints by going to *Properties* -> *Extended* -> *Arrival Radius*. Suggested value is 0.025 nautical miles which reduces the chance for a misinterpretation of your intentions by the Route Manager. There is a way to change the *Waypoint Arrival Circle Radius* permanently for each new route point in *OpenCPN's Options - Ships - Routes/Points*. (This is racing, not cruising!)

Simply create a route between each of the waypoints representing the race course using *OpenCPN* Route Manager. The name of the route is free, *DashT Race Mark* will be searching for an **active** route, and there can be only one active route at a time.

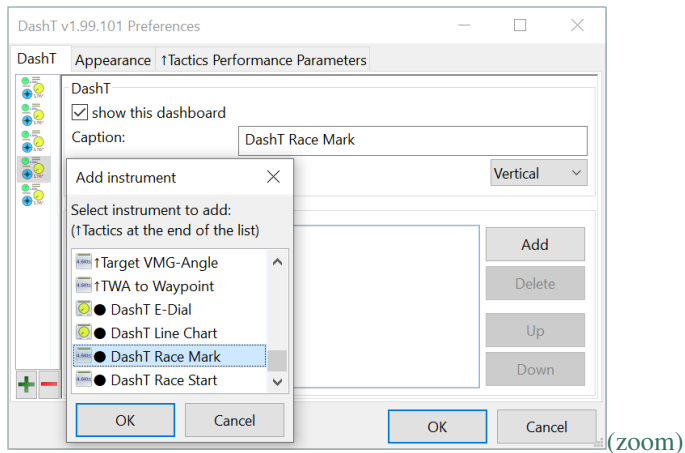
In a sailing race like this you will be sure not to follow the route line. Set the line as discrete as possible, by narrow width, dotted line and gray color:



(zoom)

TIP: Maybe you will find that the organizer's announced positions of markers are outright wrong when you make your first race. Supposing that there are other races coming up, you can quickly drop a memorandum mark for the actual position of each mark when you pass them. Between the two races you will have time to snag the actual race marks over your memorandum marks and this way increase the accuracy of your sailing in the next race!

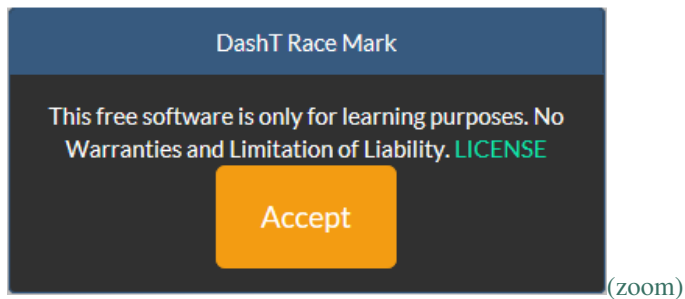
13.2.1 Open application



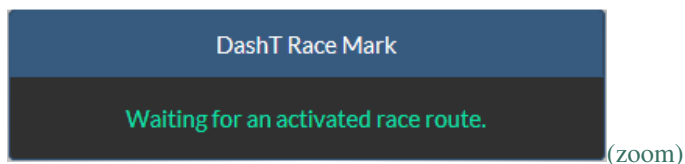
NOTE: Albeit there is no need for any database services or such, please see the corresponding chapter, *InfluxDB / Docker* anyway, for the easiest possible way to get all network based services of *DashT* available with a simple push-button start (Windows) or by a simple command (Linux). Of course, you can have your own way to set the required service, something like how it is done in *EngineDJG script*. But if nothing is set to retrieve the *DashT Race Mark* over the network, local or other, nothing explained below will take place!

As usual with complex applications with big tables and buttons, requiring a large real-estate area, it is better to keep them alone in a dedicated *DashT* instrument window.

For the first time initialization, you are requested to acknowledge and accept the free software license and limitations of liability. By continuing you give your acceptance. If a new application of the same type is created the acceptance is asked again. For this one, this screen will not reappear unless the ID of the instrument is changed in the *ini/conf-file*.

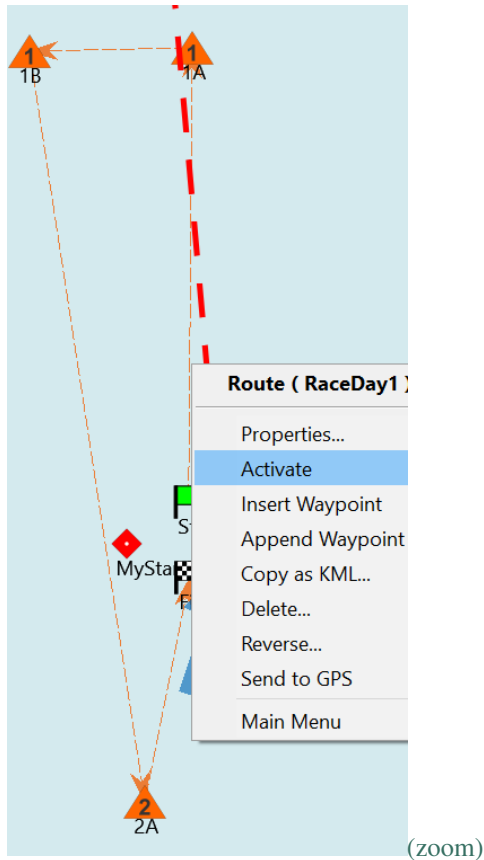


The *DashT Race Mark* will enter the standby state, leaving you time to verify with the other *DashT* instruments that it can receive all data it needs. If you have successfully used *DashT Race Start* you already have got it all. Otherwise a warning message will pop up for missing data.



13.3 Start The Course

If you can (make your best that you can) **activate** the race route *before* you pass the Start mark of the race course. Simply right click on the route line.

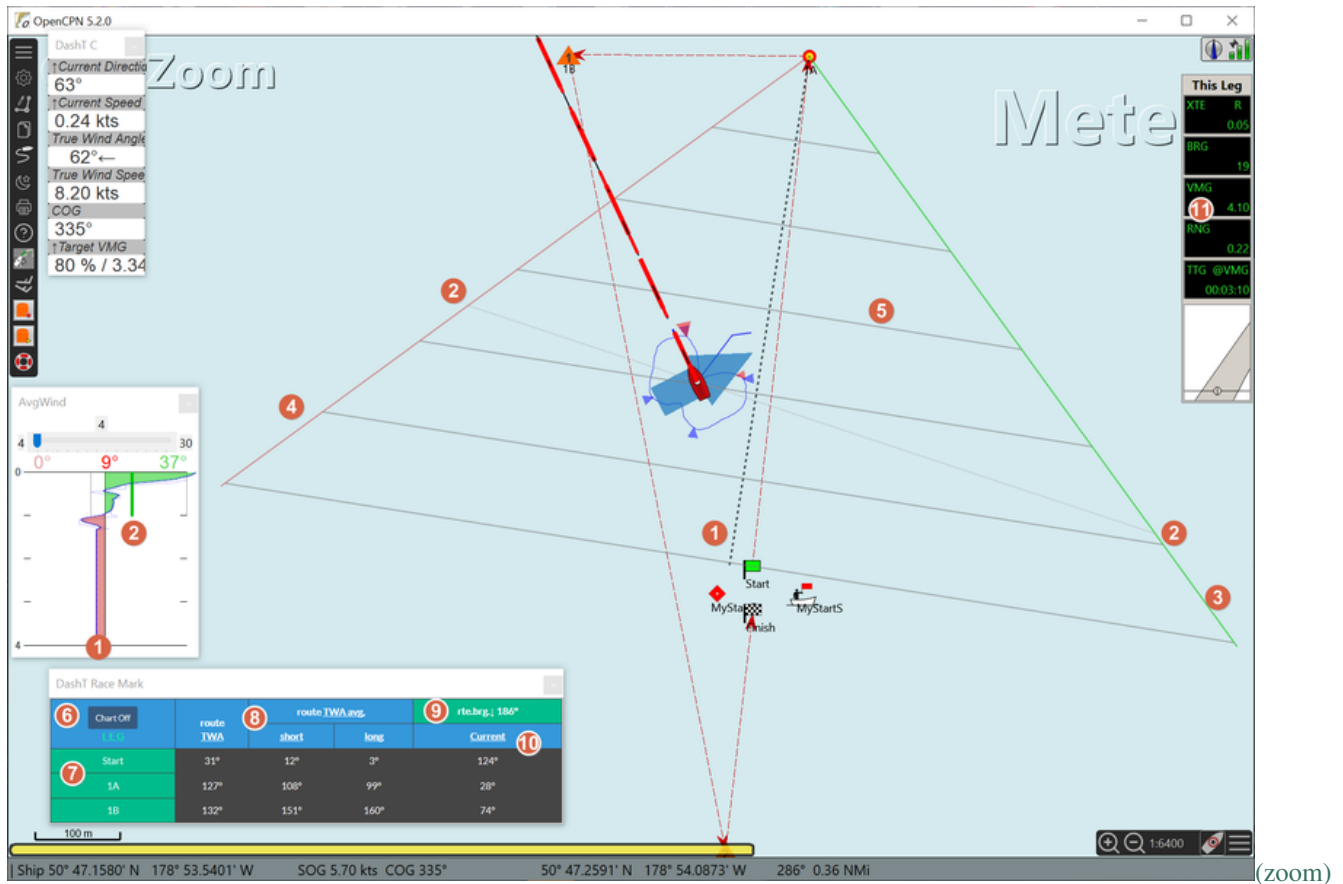


If and when you pass within a reasonable distance to the Start mark, *OpenCPN* Route Manager will put on the first leg, which is a windward leg.

NOTE: If the route paths are very close to each other and if you do *not* pass next to the Start mark's arrival radius, it may well happen that *OpenCPN* route manager puts on a wrong leg because you are closer to it than to the first leg. In this case it is very difficult to recover from that situation by other means than by deactivating the route and then waiting until you cross the next time the first leg's route path, and only then activating again the race course route. Well, you have certainly other things to do, so plan your route carefully so that this cannot happen easily.

13.4 Windward leg

Again, like with *DashT Race Start* a huge quantity of information will be presented both on the chart canvas as an overlay and on the *DashT Race Mark* numerical display dashboard. They are best explained in the logical order, referring to the numbers on the below screenshot of a windward leg:



1. **The Median Wind** direction, provided by the Tactics background process and displayed using the *Average Wind instrument* is depicted with a dotted line which starts from the next waypoint
2. **The short term average wind** direction is shown in a unique ladder rung line, always passing through the boat. It can be used to determine if you are on a *lift* or on a *header*. In the case you suspect the latter it helps you to take the decision should you continue without tacking, nevertheless - it can be also a short drop on your speed due to wind force change, not only the direction: with this auxiliary line you can estimate if you continue to still climb on the ladder towards the waypoint, or not and this way perhaps you can avoid an unnecessary tack.
3. **Starboard layline** and...
4. **Port layline** are based on the Median Wind. Their purpose is to warn you about getting into a dead end situation which may occur if there is a sudden but permanent wind change. It is better to stay closer to the Median Wind center line.
5. **Ladder Rungs** are equally based on the Median Wind. Their interval can be adjusted in the *ini/conf-file*. They allow you to stay on the effective side of climbing towards the windward mark. Controlling your opponents by tacking with them but always higher on the ladder is also helped by these visual aids.
6. **Chart Off** button allows to toggle the chart overlay if you find it to be too much. Please note that the *ini/conf-file* allows you to select the elements which are shown on the chart overlay
7. **Peek data on next legs** - *DashT Race Mark* continuously calculates some key values for the upcoming legs based on the actual, measured data in order to help in the selection of the sails, such as gennakers:
8. **TWA on next legs** is shown for actual, measured value from your boat's system and for the time integrated values, both for the Median Wind and for the short term average wind.
9. **Route bearing back** is a useful help when one is on conditions where the tide or other current is strong. On a leeward leg, one can keep an eye on not getting continuously away from the theoretical route by watching the

previous mark's actual bearing.

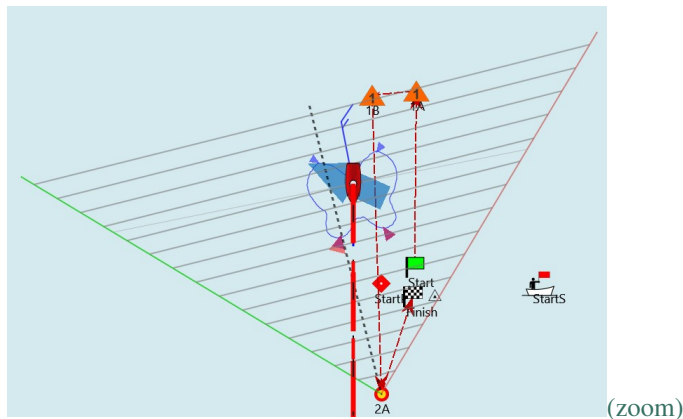
10. **Current “true angle”** is not repeating the current direction which is already shown by the *Tactics* instruments - the value is the angle the current would hit the boat if it would follow the theoretical route of the next leg. The values lesser than 90 degrees and around it would slow down the boat and would perhaps increase the leeway. Values closer to 180 degrees would push the boat and thus give it a speed advantage. Of course, if the tide is turning and the race course is long, the measured/calculated value has not that much of interest compared to some static data tables with time corrections but it would work well enough on a short race course.
11. **Route Manager** dashboard - is something you cannot turn off which is annoying. In particular, do not get fooled with the “VMG”, it is for the route not for the wind! Better not look at it...

13.5 Reaching leg

You will not need help on this. *DashT Race Mark* will stay automatically largely silent on the chart canvas.

13.6 Leeward leg

On the leeward leg the display remains similar than on the windward leg, only that you have to remember that the logic has changed: this time a lift would push you away from the shortest distance toward the leeward mark and you need to use the short term average “ladder rung” crossing your boat to determine if it necessary to gybe. Maybe it is not worthwhile, at least not immediately if you continue your descent on the ladders inclining on the side of the Median Wind; you could just profit from the boost to the speed.



13.7 Final leg

Maybe you do not have time anymore to watch any ladder rungs, but if you will in our use case it is similar to the first leg which was also a windward leg.

Once you have reached or passed the Finish mark of your race course route, the *OpenCPN* Route Manager will automatically deactivate the route and the *DashT Race Mark* will put itself in a standby state.

Victory and glory!

13.8 Troubleshooting

The most common reason for *DashT Race* applications not showing anything is that one or more of the input parameters (input data from your boat's instruments or calculated by Tactics algorithm) is missing.

13.8.1 Missing data

There is a warning message if any of the input parameters is missing but it is shown only once when the application is started. It is easy to forget the warning during the start procedures. When you enter the race course, nothing will be shown. It will be too late to start fixing things at that moment!

Instead, make sure in advance that your boat and Tactics is producing all the necessary data. Please zoom into the above [screenshot](#) which depicts the numerical instruments and the graphical Average Wind instrument. When they are all showing some values and not "--", then you can be assured that *DashT Race Mark* gets also all the data it needs to calculate the graphical overlay and leg predictions.

13.8.2 Route messed up

The second source of trouble is the interaction with the OpenCPN Route Manager. The most common cause is the usage of the default arrival radius of 0.050 nautical miles. This is too big. It can cause, for example that when you activate the route, it gets deactivated immediately. This is because you are too close to the Finish (last point in the route). The remedy is to leave enough space between Finish and Start and to set all routepoints with 0.025 nautical miles arrival radius (not the same as the waypoint "rings").

13.8.3 Application No Show

The web based (script) application is fetched from a server, which is by default set the same as in [InfluxDB / Docker](#) - the nginx http-server on port 8088. If the Docker helper script explained in that chapter fails to set up the InfluxDB or others because of the issues of the application availability (like on arch64/aarch64 - Raspberry - the packages are not necessarily available as on amd64 and x86_64 based systems). The http-server may fail as well and even these instrument's code will not load.

The remedy is simply to switch to another server. For example, as it is explained in and by [EngineDGJ engine/energy helper script](#). It is set to launch the http-service on port 8080. In the ini/conf-file of the OpenCPN you would change all occurrences of 8088 to 8080 to get them from the same server:

```
instrujsURL=http://127.0.0.1:8080/enginedjg/  
[PlugIns/DashT/WebView/RaceStart]  
instrujsURL=http://127.0.0.1:8088/racedashstart/  
[PlugIns/DashT/WebView/RaceMark]  
instrujsURL=http://127.0.0.1:8088/racedashmark/
```

TWEAKS

Most of the defaults settings are enough to get started with *DashT* plug-in and many of the key parameters can be changed using the *Preferences* dialog. However, under the hood there are many more parameters which one could not possibly put in the *Preferences* dialog without making it totally confusing and overstuffed. Maybe one day there will be an effort made to create a way to set these “advanced parameters” with some Graphical User Interface (GUI) solution.

But until that day comes, you can get many things shaped to behave to your liking if you are able to use a text editor: When *OpenCPN* is not running, open its ini/conf-file and follow the instructions in this chapter to make your modifications.

The *OpenCPN* configuration file is located in:

Windows \ProgramData\opencpn\opencpn.ini

Linux ~/.opencpn/opencpn.conf

Since we do not want to attempt to replace *OpenCPN* documentation, please open the *OpenCPN* user’s guide on your computer platform if you cannot find the configuration file from above locations.

NOTE: It goes without saying - **make a backup copy of the file** before start tweaking with it!

TIP: In the same directory you have the *OpenCPN* log file. If something goes wrong, it is good to take a look at the end of that file where the most recent start-up logs are located. You can search lines having string “*dashboard_tactics_pi*” which is used by *DashT* for its log messages (there are usually only few, but if you do not see any, it means quite likely that *DashT* never gets loaded by *OpenCPN*).

In below sections we explain by owner and by key the meaning and eventual usage of each key.

14.1 OpenCPN

14.1.1 Plugins

```
[PlugIns]
PluginOrder=ChartDownloader;GRIB;VDR;DashT
LatestCatalogDownloaded=Master
```

If you do not see here *DashT* in the list there can be an issue with the *OpenCPN* plug-in manager.

Windows:

```
[PlugIns/dashboard_tactics_pi.dll]
bEnabled=1
```

Linux:

```
[PlugIns/libdashboard_tactics_pi.so]
bEnabled=1
```

dashboard_tactis_pi is the name of *DashT* in the *OpenCPN* plug-in catalog. If *bEnabled=0* then *DashT* remains disabled: use *OpenCPN* plug-in manager instead of attempting to tweak this setting yourself.

14.1.2 AUI

```
[AUI]
AUIPerspective=layout2|name=ChartCanvas;caption=;state=768;dir=5;layer=0;row=0;
pos=0;prop=100000;bestw=5;besth=5;minw=298;minh=915;maxw=-1;maxh=-1;floatx=-1;
floaty=-1;floatw=-1;floath=-1|name=DASHT-57881c38-fd95-4000-83bf-3804e5d7179;
caption=AvgWind;state=2098121;dir=4;layer=0;row=0;pos=0;prop=100000;bestw=325;
besth=425;minw=325;minh=425;maxw=-1;maxh=-1;floatx=8;floaty=614;floatw=313;
floath=431|....
```

If you are wondering where *OpenCPN*'s *wxWidgets* stores the *DashT* many window positions (in this example, window pane named *AvgWind*), this is the place. There is nothing you can do by editing but if you have issues, you can delete all the entries before the restart and all windows (not only those of *DashT*) will be piled up above each other so that you can rearrange them. If the instrument cluster window pane's ID changes for some reason (by your tweak?), its position will be lost and it will be reopened in the default position.

14.2 DashT

14.2.1 Dashboard

```
Version=2
SpeedometerMax=12
COGDamp=0
SpeedUnit=0
SOGDamp=0
DepthUnit=3
DepthOffset=0
DistanceUnit=0
WindSpeedUnit=0
TemperatureUnit=0
UTCOffset=0
```

It is recommended to use *Configuration Dialog* which provides full *wxWidgets* platform specific interface to set these values.

Key / Parameter	Description
Version	Leave it as 2 - this is the compatibility flag with the OpenCPN Dashboard sometimes used in the code, you do not want version 1 features (backward compatibility has not been tested)
SpeedometerMax	Speedometer maximum numerical value
COG-Damp	0 = turns damping off. Value 1 gives $1/2 = 0.5$ damping factor = no damping, >0.5 = damping - see below note
SpeedUnit	-1 = use OpenCPN setting, 0 = [kts], 1 = [mph], 2 = [km/h], 3 = [m/s]
SOG-Damp	0 = turns damping off. Value 1 gives $1/2 = 0.5$ damping factor = no damping, >0.5 = damping - see below note
DepthUnit	3 = [m], 4 = [feet], 5 = [Fathoms], 6 = [Inches], 7 = [Centimeters]
DepthOffset	0 = no offset, otherwise offset in meters (albeit the dialog asks it in centimeters), can be negative or positive floating point value like 0.5; multiplied internally to match the corresponding <i>DepthUnit</i> selection
DistanceUnit	-1 = use OpenCPN setting, 0 = [NMi], 1 = [statute miles], 2 = [km], 3 = [m]
Wind-SpeedUnit	0 = [kts], 1 = [mph], 2 = [km/h], 3 = [m/s]
TemperatureUnit	0 = Celsius, 1 = Fahrenheit
UTCOffset	Divided in 48 steps, from -24 to 24, presenting half an hour offsets from -12:00 to 12:00. Eg. -2 is equivalent of -01:00

NOTE: About COG/SOG filter value: With all the respect, the built-in Dashboard documentation must be mistaken telling that “1” is presenting no filtering; it is *activating* the filtering code but the filtering is, presumably not altering the signal: `dashboard_pi.cpp: mSOGFilter.setFC(g_iDashSOGDamp ? 1.0 / (2.0*g_iDashSOGDamp) : 0.0);`. If you look at the code of `iirfilter.cpp` the value 0.0 turns the filtering completely off. Here is an excerpt of the description of the `iirfilter`: The parameter is a setting to a single order *infinite-impulse-response filter*: The value defines filter’s cutoff frequency. A value of 0.5 is basically no filtering and smaller values decrease the cutoff frequency. If you think of the filter as being “fast” or “slow” then 0.5 is fastest and smaller values are “slower”. The *OpenCPN* Dashboard built-in documentation tells us “A typical filter value of 10 seems to work pretty well.”. That ends up as $1/(2 * 10) = 0.05$ for the *iirfilter*. If we mark the requested *iirfilter* cut-of value with v , and the ini/conf file setting with a , then $a = 1/(2v)$.

14.2.2 Fonts

Fonts are quite difficult to tweak but you can try. They get their default values by *wxWidgets* library’s adaptation to the operating system platform on which it is running:

Windows:

```
FontTitle=1;10;-20;0;0;0;400;1;0;0;1;0;0;2;32;Arial
FontData=1;14;-28;0;0;0;400;0;0;0;1;0;0;2;32;Arial
FontLabel=1;8;-16;0;0;0;400;0;0;0;1;0;0;2;32;Arial
FontSmall=1;8;-16;0;0;0;400;0;0;0;1;0;0;2;32;Arial
```

Linux:

```
FontTitle=Sans Italic 10
FontData=Monospace Bold 14
FontLabel=Monospace 8
FontSmall=Monospace 8
```

However, there is perhaps no need to modify anything here, since the *Configuration Dialog* provides full *wxWidgets* platform specific interface to set these highly platform specific values.

14.2.3 Instrument fonts

There is one tweak possibility provided by *DashT* compared to the *OpenCPN* Dashboard plug-in or its forks: we can set the *default* values and fonts in advance before these are set, in more generic terms. This allows to experiment with what values *wxWidgets* sets for this operating system. Set the values below to your liking, remove the above fields and start *OpenCPN* with *DashT* plug-in enabled:

```
[PlugIns/DashT/Fonts]
TitleFontSize=10
TitleFontFamily=SWISS
TitleFontStyle=ITALIC
TitleFontWeight=wxNORMAL
DataFontSize=14
DataFontFamily=TELETYPE
DataFontStyle=NORMAL
DataFontWeight=BOLD
LabelFontSize=8
LabelFontFamily=TELETYPE
LabelFontStyle=NORMAL
LabelFontWeight=NORMAL
SmallFontSize=8
SmallFontFamily=TELETYPE
SmallFontStyle=NORMAL
SmallFontWeight=NORMAL
```

Please see for the available values in [wxFont documentation](#) which is not going to be repeated here (yes, we use the short names, they will not go away).

NOTE: the above settings are actually designed to be able to pass the user choice from the Dashboard preferences dialog to the JavaScript-based instruments stylesheets - the instrument styles are CSS/SASS controlled. However, nothing has been implemented yet, so if you change something here, the JavaScript instrument's font will not change, sorry, remains in to-do list.

TIP: Meanwhile, please remember that the JavaScript instruments are executed on a kind of a browser window. Depending of the operating system on which you are running you can try to zoom in / zoom out. Typically this is Ctrl-(plus-key) and Ctrl-(minus-key), and Ctrl-(zero-key) to get back to 100% zoom.

14.2.4 Colors

This tweak allows you to change and experiment with the look of the instruments - other than the baro-, wind- and performance history displays which remains on the white background due to their “hair-line” printing by making one pixel-size dots around the canvas (it is not impossible, but really challenging color world...).

```
[PlugIns/DashT/Colors]
BackgroundColor=DASHB
ForegroundColor=DASHF
LabelColor=DASHL
RedColor=DASHR
GreenColor=DASHG
IllustrationsColor1=DASH1
IllustrationsColor2=DASH2
NeedleColor=DASHN
SecondNeedleColor=BLUE3
CentralCircleColor=UBLCK
CompassBackgroundColor=COMP1
EmbossedNeedle=1
EmbosNeedleContourColor=UBLCK
ShowRedGreenFace=1
LowDegRedGreenFace=20
HighDegRedGreenFace=50
ForeroundColor=DASHF
```

You guessed it right - the colors are coming from *OpenCPN* and the idea of *OpenCPN* is to provide - not perhaps the most elegant way - color themes which allows to circulate around the “day”, “dusk” and “night” themes. You can find the currently (*OpenCPN* v5.2) available colors [here](#).

It is better to stick to *OpenCPN* defined names for colors so that you respect (at least to some extent) the the “day”, “dusk” and “night” themes. An example of such usage is available [here](#).

14.2.5 Extra dial tweaks

Key / Parameter	Description
EmbossedNeedle	When 1 creates an illusion of an embossed needle, set to 0 if you want the classical, flat one
ShowRedGreen-Face	Red and green circles, present in some dials can be turned off by setting 0 here
LowDegRedGreen-Face	If the above remains in 1, then one can set the angle in which the informative red / green circle segments ends

14.2.6 Dashboard index

DashboardCount=5 - one thing you need to understand with the *OpenCPN* and wxWidgets dashboard management using the ini/conf-file is that it does not delete the old dashboard block from the ini/conf-file, it just decreases the counters if you delete a dashboard (a window pane collecting instruments) or an instrument in it. At startup we loop on this counter and search for the following block structures - let’s check the anatomy of one of those:

```
[PlugIns/DashT/Dashboard3]
Name=DASHT-57881c38-fd95-4000-83bf-43804e5d7179
```

(continues on next page)

(continued from previous page)

```

Caption=AvgWind
Orientation=V
Persistence=1
InstrumentCount=1
Instrument1=73
InstrumentID1=
Instrument2=91
InstrumentID2=71ced54e-9110-4dd7-a778-ec133c11496f
Instrument3=27
InstrumentID3=

```

Key / Pa- ram- eter	Description
Name	DASHT-[UID] to make difference from Dashboard instrument panes, prefix is different and UID algorithm more advanced
Caption	You can see this in the window pane title and in the Preferences
Ori- enta- tion	V = vertical; H = horizontal
Per- sis- tence	Badly chosen name since it is more like ‘visible by default’; make sure that you have at least one dashboard (window pane) visible, otherwise you need to go to plug-in manager to get hold of the Preferences dialog
In- stru- ment- Count	As mentioned this is used to control the actual instruments, not the entries below which can be more numerous than the instrument count - they are never deleted by wxWidgets - maybe there is a way to do it but <i>OpenCPN</i> Dashboard has not considered it necessary (and it certainly is not the end of the world)
In- stru- ment1	The number is the index on which the <i>OpenCPN</i> Dashboard will loop to create instruments on a window pane called “dashboard” - the type is an internal code for instrument type , in this case a Tactics Average Wind instrument
In- stru- men- tID1	This instrument is of an old <i>OpenCPN</i> Dashboard type and it does not need a unique identifier
In- stru- ment2	The instrument in this position was a <i>DashT Race Start</i> but it is gone now
In- stru- men- tID2	<i>DashT</i> adds an unique ID for all instruments. This is used to store the parameters of a web based instrument, for example by the system’s web browser library. TIP: to make such an instrument to ‘forget’ it setting, just change any number in the string to something else!

There can be even more of ‘forgotten’ instruments. If you are really bugged by these, or even ‘forgotten’ instrument panes you can very well remove them, there is no harm.

14.3 DashT/Tactics

NOTE *DashT* attempts to keep full compatibility with *Tactics* instruments and algorithms; there is no way to guarantee that all parameters are available in both plug-ins. Therefore, if using *DashT* use this documentation and if you are using *Tactics* plug-in, use its own documentation.

```
CurrentDampingFactor=0.003
LaylineDampingFactor=0.2
LaylineLengthonChart=5
MinLaylineWidth=2
MaxLaylineWidth=30
LaylineWidthDampingFactor=0.2
ShowLaylinesOnChart=1
ShowCurrentOnChart=1
CMGSynonym=CMG
VMGSynonym=VMG
DataExportSeparator=;
DataExportUTC-ISO8601=0
DataExportClockticks=0
TacticsImportChecked=0
```

Key / Parameter	Description
Current-Damping-Factor	The layline damping factor determines how fast the laylines react on your course changes, i.e. your COG changes. Low values mean high damping. <i>DashT Tactics</i> recommends values from 0.002 to 0.008 to start your experimenting with.
Layline-LenghtorChart	The width of the boat laylines is based on the yawing of the boat (vertical axis), i.e. your COG changes. The idea is to display the COG range where you're sailing to.
Layline-LenghtorChart	Length of the boat laylines in [nm]
ShowLaylinesOnChart	Toggles laylines visibility as an overlay on the chart canvas. 0 turns them off and 1 back on.
ShowCurrentOnChart	Toggles current arrow visibility as an overlay on the chart canvas. 0 turns it off and 1 back on.
CMGSynonym	This setting allows you to whatever abbreviation you want for the <i>self-explanatory vector Tactics algorithms is calculating</i> . Finishing the endless discussions in the forums about the meaning of life, too bad!
VMGSynonym	Same reason for the existence of this abbreviation settin. I guess that the author of the mentioned, albeit simple vector chart just got enough of it = do what you want!
Data-Export-Separator	This is used by all CSV-export functions - such as baro-, wind- and performance history - as separator. Space as separator (not recommended) requires “ “ quotes around it. TAB as separator presents as “ <code>\t</code> ”. All printable characters do not require quotes around them. Maybe you want to stick to semicolon (default) and comma (as file name implies), after all.
DataExportISO8601	You want to set this to 1 if you plan to import the CSV-file later on some application like a time series database, or perhaps some Python script. The dull second as time accuracy is not enough for them. But for your eyes and for your wristwatch only, you do not need this
Data-Export-Clock-ticks	Same, the aforementioned application types like timestamps. This is the best of it all, milliseconds since EPOCH! (a trivia question: were you born before or after the Unix Epoch?)
Tactic-sImportCheck	This is used only if you have both the excellent and light-weight <i>Tactics</i> plug-in installed together with <i>DashT</i> - you are suggested to import <i>Tactics</i> settings into <i>DashT</i> and you can continue to use <i>Tactics</i> as before, the two will not get mixed

14.3.1 Performance

```
PolarFile=C:\\Program Files (x86)\\OpenCPN\\plugins\\  
weather_routing_pi\\data\\polars\\yourboat100.pol  
BoatLeewayFactor=10  
fixedLeeway=30  
UseHeelSensor=0  
UseFixedLeeway=0  
UseManHeelInput=1  
CorrectSTWwithLeeway=0  
CorrectAWwithHeel=0  
ForceTrueWindCalculation=0  
UseSOGforTWCalc=0  
ShowWindbarbOnChart=1  
ShowPolarOnChart=1  
PersistentChartPolarAnimation=1  
Heel_5kn_45Degree=5  
Heel_5kn_90Degree=8  
Heel_5kn_135Degree=5  
Heel_10kn_45Degree=8  
Heel_10kn_90Degree=10  
Heel_10kn_135Degree=11  
Heel_15kn_45Degree=25  
Heel_15kn_90Degree=20  
Heel_15kn_135Degree=13  
Heel_20kn_45Degree=20  
Heel_20kn_90Degree=16  
Heel_20kn_135Degree=15  
Heel_25kn_45Degree=25  
Heel_25kn_90Degree=20  
Heel_25kn_135Degree=20  
ExpPolarSpeed=0  
ExpCourseOtherTack=0  
ExpTargetVMG=0  
ExpVMG_CMG_Diff_Gain=0  
ExpCurrent=0  
NKE_TrueWindTableBug=0  
TwaMarkUseShortAvgWind=1
```

Key / Parameter	Description
PolarFile	(a single line , BTW) Your boat's polar, or a polar close to your boat - you get plenty of those with the <i>weather_routing_pi</i> plug-in! See Tactics Polar file discussion - it is not a bad idea to prepare several of those for your boat, say 90%, 95%, 100% and 105%(!).
PolarLookable-Output-File	Same as the above but use a .csv extension for a comma separated value output file which is built from the above PolarFile according the algorithm explained in Tactics Polar section. Once you have recovered your file and inspected it, you may want to remove this line from your configuration file - there is no need to create at every start!
BoatLeeway-Factor	<i>Leeway</i> is the 'drift' of the boat due to heel/wind influence. Low values mean high performance of hull. If we mark <i>Leeway</i> as L , <i>BoatLeewayFactor</i> as c , <i>Heel</i> angle as a and Speed Through Water <i>STW</i> as v then $L(v) = c * a / v^2$ - one word, keep you paddlewheel clean!
fixedLeeway	This parameter has dual purpose to limit the <i>Leeway</i> in calculations, depending of the <i>UseFixedLeeway</i> : if 0 (use a heel sensor or manual "heel polar") then this value limits <i>Leeway</i> to this maximum value; if 1 (do not use manual heel 'polar' nor the heel sensor): <i>Leeway</i> gets this value.
UseHeelSensor	Use the internal heel sensor if available. Important for the correct calculation of the surface current.
UseFixedLeeway	Select the behavior of <i>fixedLeeway</i> parameter
UseManual-Heel-Input	If no heel sensor is available, you can create a manual "heel polar" below. Just read/enter the data from a mechanical heel sensor (e.g. available on compass). Use True Wind Speed & Angle only ! Take care: motoring w/o sails and a heel will show wrong current data!
CorrectSTWwithLeeway	Correct STW with Leeway: apply a correction to your log speed throughout the plug-in based on the calculated Leeway and Current. Does make only sense with a real heel sensor. Make sure your instruments do not already apply this correction!
CorrectAWS/AWAwithHeel	Correct AWS/AWA with Heel: Use with care, this is normally done by the instruments themselves as soon as you have an integrated, original equipment heel sensor. But if it your own heel sensor, this can make a use of it!
ForceTrueWindCalculation	Force True Wind Calculation: Internally calculates True Wind data (TWS,TWA,TWD) and uses it within the whole plug-in even if there is True Wind data available via NMEA or Signal K.
UseSOGforTW	Use SOG instead of STW for True Wind Calculations: Recommended (as the True Wind blows over the earth surface, we should calculate it with Speed Over Ground. This eliminates the influence of currents). > > > However, some sailors say that are interested in how the wind blows over the water, so you can make your choice here! > > > BTW, some other, innovative sailors are even using the great pair of software <i>DashT</i> or <i>Tactics</i> only for this feature while their paddlewheel has got stuck - well, it is like going to plough a potato field with a Ferrari! Better not think too much about this usage for not to lose motivation...
ShowWindBarb	Toggles wind barb visibility as an overlay on the chart canvas. 0 turns it off and 1 back on.
ShowPolarOnChart	Toggles the polar visibility as an overlay on the chart canvas. 0 turns it off and 1 back on.

14.3.2 AverageWind

```
[PlugIns/DashT/Tactics/AverageWind]
ShortAvgTimePercentage=25
AvgTime=240
```

Key / Parameter	Description
ShortAvgTimePercentage	Ratio in percentage of the <i>AvgTime</i> minimum percentage is 10, default is 25, maximum is 50
AvgTime	This is the normal “long” integration time, minimum and default is 240 seconds = 4 minutes, while the maximum is 1800 seconds = 30 minutes

14.3.3 BaroHistory

```
[PlugIns/DashT/Tactics/BaroHistory]
Exportrate=60
BaroHistoryExportfile=C:\\Users\\JoeDoe\\Downloads\\barotest.csv
```

Export rate is in seconds.

14.3.4 PolarPerformance

```
[PlugIns/DashT/Tactics/PolarPerformance]
Exportrate=5
BaroHistoryExportfile=C:\\Users\\JoeDoe\\Downloads\\perftest.csv
```

Export rate is in seconds.

14.3.5 Windhistory

```
[PlugIns/DashT/Tactics/Windhistory]
Exportrate=5
BaroHistoryExportfile=C:\\Users\\JoeDoe\\Downloads\\windtest.csv
```

Export rate is in seconds.

14.3.6 Odograph

```
[PlugIns/DashT/Tactics/Odograph]
GrandTotal=0
DataPointInterval=30
OdographExportfile=C:\\Users\\JoeDoe\\Downloads\\odograph.csv
ShowBoatLog=1
```

Key / Pa- ram- eter	Description
Grand- Total	The persistent storage for the total trip counter
Data- PointIn- terval	Time in seconds between the two GNSS points and the distance calculations between them. It is also the time period of the storage in the CSV file. Default value is 30 seconds, equivalent of 77 meters at 5 knots. Minimum value is 10 seconds (51m @10kn). Possible values are 10, 20, 30, etc. seconds, <i>i.e.</i> 15 seconds would become 10 seconds
Odogra Ex- port- file	A helper so that one does not type the file name if it remains the same
Show- Boat- Log	Put here 0 if you do not want to show the boat's navigation system's log-based trip data - for example, if it is also making the log data calculation based on GNSS data

14.3.7 Streamout

```
[PlugIns/DashT/Tactics/Streamout]
ConfigFile=streamout.json
```

The quite rich (in features) database schema is set in a separate JSON-file. The default configuration file streams into a file which can be imported into a InfluxDB v2 database. However, it is possible to stream directly into a live database using HTTP protocol. In this case you would modify an existing HTTP-configuration file template and put its name here. See [Configuration file management](#) regarding the settings in the JSON-file.

14.3.8 StreaminSk

```
[PlugIns/DashT/Tactics/StreaminSk]
ConfigFile=streamin-sk.json
```

In a similar manner like with the output streaming, the Signal K input streaming is configured with a separate JSON file. The default one being good for the local Signal K server node connection you may to make your own copy, if your server is located, for example on another computer.

14.4 DashT/WebView

```
[PlugIns/DashT/WebView]
[PlugIns/DashT/WebView/EngineDJG]
instrujsURL=http://127.0.0.1:8080/enginedjg/
[PlugIns/DashT/WebView/RaceStart]
instrujsURL=http://127.0.0.1:8088/racedashstart/
[PlugIns/DashT/WebView/RaceMark]
instrujsURL=http://127.0.0.1:8088/racedashmark/
```

(continues on next page)

(continued from previous page)

```
[PlugIns/DashT/WebView/TimesTUI]
instrujsURL=http://127.0.0.1:8088/timestui/
```

This is to explain with a URL the *DashT* from which network port it can expect to find and fetch the corresponding JavaScript instrument code to be integrated with the *DashT* dashboard.

The default values correspond to the helper script set services explained in this document.

Nothing prevents to change these values to point to an external server or internally to different ports if you so like, there is no limitation from where the code is coming from, as long as it is from the same *DashT* version.

NOTE: you may be tempted to use `file://` protocol. Be aware that on all operating systems supported by *DashT* the code fetched with this protocol will not be allowed to save any of its parameters due to obvious security measures the manufacturers have undertaken in recent years. Therefore this protocol support has been definitely dropped from *DashT* as well.

DEBUG: One can use `about:blank` in URLs to debug issues on *wxWidgets WebView* implementation on a particular platform, to eliminate all HTML5 and JavaScript code and their dynamic interactions.

Cache: At every restart or at the orientation change *DashT* is making a *Reload()* request on *wxWidgets WebView* instance to fetch the latest versions of the drivers. But if you change, say *EngineDJG paths* by modifying the *common.js* file and expecting your changes being visible in the instrument it may happen that your changes are not getting loaded. This is certainly depending of the *wxWidgets WebView* implementation on you platform but also of the web server you are using. The tweak in this case is simply to change the name *common.js* to something else both on the folder and in the *index.html* file and the restart the OpenCPN application, or to change the orientation of the window pane on which the instruments are located.

14.4.1 Race

```
[PlugIns/DashT/Race]
[PlugIns/DashT/Race/RaceStart]
LaylineWidth=3
GridSize=1
GridStep=0.026998
GridLineWidth=1
GridBoldInterval=2
ZeroBurnSeconds=60
[PlugIns/DashT/Race/RaceMark]
LaylineWidth=2
RungStep=0.053996
RungLineWidth=2
AvgWindLineWidth=3
ShortAvgWindRungLineWidth=1
```

Key / Parameter	Description
LaylineWidth	Pen width on the canvas in pixels
GridSize	In [NMi] - a square behind the start line
GridStep	In [NMi] - 50 meters is 0.026998 NMi, minimum is 0.00486 NMi which corresponds to about 9 meters (otherwise you can start playing 'Painted Black'...)
Grid-LineWidth	Pen width on the canvas in pixels (for non-bold line)
GridBoldInterval	1=all lines are bold, 2 every 2nd is bold, etc.
ZeroBurnSeconds	Continuously calculate from the polar where the inviting Zero Burn mark should be kept on the COG line and at which point it should disappear if the boat has passed it - or has passed its chance
RungStep	In [NMi] = 100 meters is about 0.053996 NMi
Rung-LineWidth	Pen width on the canvas in pixels
AvgWind-LineWidth	Pen width on the canvas in pixels
ShortAvg-WindRung-LineWidth	Pen width on the canvas in pixels

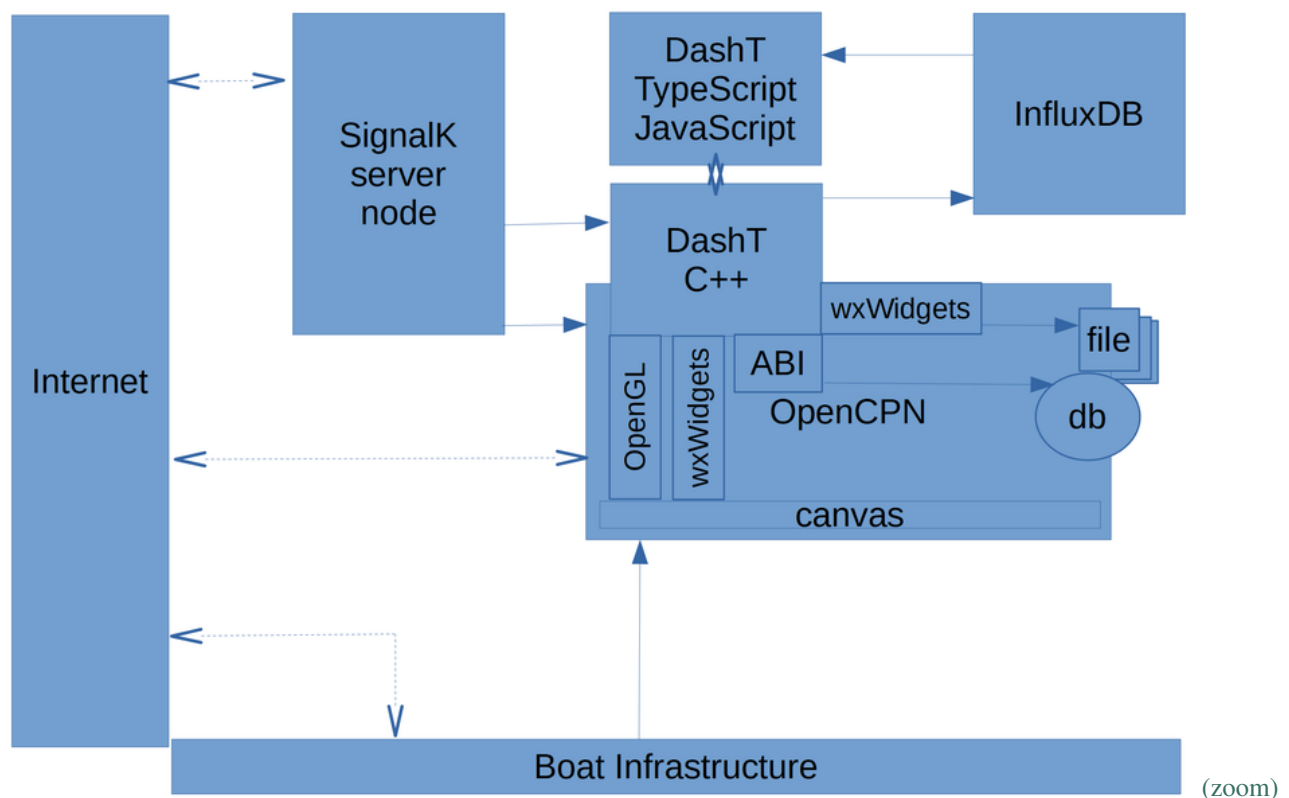
SECURITY

We live in a connected world. This chapter gives an insight to the security aspects of the *DashT* overlay plug-in for the popular *OpenCPN* chart plotter. It presents the software security policy and explains its implementation.

TIP: Even if you are not that much of interested in this type of talk, please quickly take a look at the [data flow diagram](#) and you can spot out is your boat's infrastructure concerned with any potential security issues. If it is, you have certainly already given a thought to the security aspects and can adapt *DashT*, accordingly. See also: [Privacy](#).

15.1 Data flow

The diagram below depicts the data flow in and out from the overlay functions.



DashT C++ (plug-in) part **receives data** from following sources:

1. From OpenCPN

- NMEA-0183 messages ;
- GPS fix and magnetic deviation ;
- Routing and waypoint change information.

2. From Signal K server node

- Signal K delta channel by subscription - nothing if no instruments subscribing.

DashT C++ (plug-in) part **sends data** to following destinations:

1. To OpenCPN

- Route and waypoint creation and modification requests via the ABI ;
- Storage of data from historical instruments into the files under the responsibility of OpenCPN ;
- Storage of all data from all sources into a InfluxDB line data file under the responsibility of OpenCPN.

2. To InfluxDB v2 Time Series database

- All received data or a selection of that data.

DashT TypeScript/JavaScript (web-based instruments) part **receives data** from following sources:

1. From InfluxDB v2 Time Series database

- Read back any selected data from the data written into it by the C++ (plug-in) part.

15.2 Security Policy

The security has an uttermost priority. The user's computer or infrastructure security shall not been compromised or the system integrity threatened - not directly or indirectly - because he or she is using *DashT* plug-in.

The security implementation of *DashT* shall meet the **SMART criteria**, taking into account the surrounding FOSS and commercial ecosystem and putting those in balance with the very limited development and maintenance resources.

The achievability shall be described by an implementation plan.

The implementation plan shall clearly define the interfaces, to demonstrate how those "doors" are kept closed and how the own code base is kept safe.

The implementation plan shall demonstrate constant measurability by describing automated inspection against the currently applicable advisory.

The security threat shall be communicated to the end user via [GitHub project security page](#) by publishing a security advisory.

It shall be under each end user's own responsibility to follow the security advisory.

The security fixes, if applicable are implemented only in the latest development version.

The security fixes or patches are no applied to earlier versions of this software.

User shall be therefore obliged to **upgrade to the latest published version or to an intermediate development version** to get the applied security fix if he or she deem necessary to do so.

There is **no guarantee in time** when the security fix will be applied; or that it will be applied at all.

15.3 Security Implementation

Please see the [dedicated document in the software package's documentation](#) (*the link is pointing to dedicated document in source code repository, see docs/security*).

PRIVACY

This chapter gives the user information about the privacy aspects of the *DashT* overlay plug-in for the popular [OpenCPN](#) chart plotter. The objective is to allow the end user to have a clear view what information is potentially collected and stored (or communicated) when *DashT* overlay plug-in is used.

16.1 Privacy policy

DashT does not collect data about you or about your boat. Period.

Please [report](#) if you find a breach to this security policy or other privacy concern.

16.2 Position

DashT is aware - via *OpenCPN* of your boat's position. This data is stored only if one uses *InfluxDB Out* streamer. The data is stored either in a file on your computer or directly on an *InfluxDB* v2 server of your choice. The server can be anywhere in the network and the control of it is out of scope of *DashT*. You can opt out to store your position - or any other boat's data parameter by modifying a configuration file.

16.3 Personal information

No information related to your account is collected. We could not care less who you are and how often you wash your shirts.

16.4 Local storage (plug-in)

All parameters and settings are stored in a local file in clear text format, as described in [Tweaks](#).

16.5 Local storage (browser)

In browser-based applications and instruments, a priority is given to the Local Storage of the browser platform. This storage is accessible only locally, as the name implies. The data stored is defined and strictly limited to the *DashT* configuration parameters.

16.6 Cookies (browser)

Cookies are accessible from a remote server and locally. In *DashT* they are not used. There is only one exception to this - if you are using a Windows system which has not been updated in 2020 or later and you use `file://` protocol, cookies will be used to store *DashT* configuration parameters - nothing else. Please update your system, use `http://` protocol and *Local storage* will be used.

NOTE: If you use third party web applications outside of *OpenCPN* and *DashT*, such as *InfluxDB* v2 or *Grafana* to visualize the data you have collected with *InfluxDB Out* streamer they will use cookies to store your connection and other parameters. However they are out of the scope for *DashT* privacy policy.

16.7 Calling back home

DashT as an application does not attempt to call back home.

Instrument *Line Chart* contains two third-party open source libraries which announce that they want to call back home if they are used in a web application. They provide both an opt-out for this, which is done by *DashT* for both of them.

NOTE 1: Albeit all code is open source in the libraries used, it is impossible to guarantee that there is no code in any library which would not, potentially call back home. This has known to happen in the past but usually these guys are detected quickly, reported to the community, removed, shamed and banned forever. We trust the announcement made by those who, potentially want a call back.

NOTE 2: If you use third party web applications outside of *OpenCPN* and *DashT*, such as *InfluxDB* v2 or *Grafana* to visualize the data you have collected with *InfluxDB Out* streamer they are likely to call back home. If you use the helper method suggested by *DashT*, explained in *InfluxDB/Docker container creation*, it sets `--reporting-disabled` command line parameter for *InfluxDB* v2 server it creates. There is no such option for *Grafana*.

INDICES AND TABLES

- `genindex`
- `search`

INDEX

B

BLE, 71

Browser, 86

Internet Explorer, 86

C

Configuration, 138

Dashboards, 143

DashT, 140

OpenCPN, 139

Tweaks, 138

D

Data, 71

interchange, 71

NMEA-0183, 18

Source priority, 19

Docker, 87

Attach console, 102

command line, 98

composer, 98

Dashboard, 98

Details, 98

Grafana, 93

InfluxDB, 87

nginx, 99

Scripts, 90

E

Engine/Energy, 75

Add paths, 82

Change Display, 81

Change Subscription, 82

common.js, 83

Config kills all, 86

Configuration, 78

Customization, 82

Helper scripts, 78

Installation, 78

Introduction, 77

Language, 83

Node.js, 78

Rich infras, 84

Rich Linux, 84

Rich Windows, 85

Search again, 81

Subscription, 79

Troubleshooting, 85

G

Getting Started, 14

GPIO, 71

Grafana, 93

Dashboard, 95

Data Source, 93

Flux, 93

Ports, 93

I

I2C, 71

InfluxDB, 87

Backup, 92

Data backups, 107

Data File, 105

Debugging, 108

Developer, 102

Docker, 87

Grafana, 87

HTTP streamout, 107

Introduction, 89

Line Chart, 109

Line data import, 107

Scripts, 90

Set up, 91

Storage, 92

Streamer, 103

VDR, from, 108

Installation, 1

GKT2, 14

Linux, 9

Other, 13

Platforms, 3

Plug-In Manager, 3

Plug-in Manager, 13

- Requirements, 3
- Windows, 4
- Instruments, 19
 - Actual CMG, 53
 - Actual VMG (*wind*), 52
 - Air temperature, 22
 - Average Wind, 45
 - AWA, 23
 - AWA/AWS Dial, 23
 - AWA/TWA Dial, 22
 - AWS, 24
 - AWS Dial, 23
 - Baro Dial, 24
 - Baro History, 24
 - Barometer, 24
 - Bearing Compass, 55
 - COG, 25
 - CPU clock, 27
 - Current, 44
 - Cursor, 25
 - Dashboard, 19
 - Depth, 25
 - Depth Graph, 25
 - EngineDJG, 75
 - From Ownship, 26
 - GPS Clock, 26
 - GPS Compass, 26
 - GPS loc.time, 27
 - Heel, 27
 - InfluxDB Out, 103
 - Leeway, 44
 - loc.sunrise/set, 28
 - Log, 30
 - Magnetic COG, 28
 - Magnetic HDG, 28
 - Moon Phase, 28
 - Pitch, 29
 - Polar Compass, 57
 - Polar Performance, 55
 - Polar speed, 51
 - Position, 29
 - Rudder, 29
 - Rudder graph, 29
 - Satellite Status, 27
 - Satellites, 26
 - Signal K In, 67
 - SOG, 30
 - SOG Speedo, 30
 - STW, 30
 - Tactics, 34
 - Target CMG, 54
 - Target CMG Angle, 54
 - Target VMG (*wind*), 53
 - Target VMG Angle, 52

- Trip log, 31
- True Compass, 31
- True HDG, 31
- TWA, 32
- TWA to Waypoint, 45
- TWA/TWS Dial, 32
- TWD/TWS Dial, 32
- UTC sunrise/set, 31
- VMG (*route*), 33
- Water temperature, 33
- Wind history, 33
- Introduction, 1
 - History, 1

K

- Keys, 71
 - Signal K, 71

L

- Language, 83
 - Engine/Energy, 83
- Line Chart, 109
 - Introduction, 111
 - Numerical, 117
 - Set up, 114
- Linux, 9
 - Installation, 9
 - Installation troubleshooting, 11
 - Uninstall, 11

N

- nginx, 99
 - HTML/JS, 97
 - Troubleshooting, 102
- NMEA-0183, 18
 - Signal K, 72
- NMEA-2000, 71

P

- Paths, 71
 - Signal K, 71
- Polar file, 48
 - example, 49
 - loading, 48
- Privacy, 155
 - Call backs, 158
 - Cookies, 158
 - Local Storages, 157
 - Policy, 157
 - Position, 157
 - Your information, 157

R

- Race Mark, 129

- Define Course, 131
- Final leg, 137
- Introduction, 131
- Leeward leg, 137
- Reaching leg, 137
- Start Course, 134
- Troubleshooting, 137
- Windward leg, 135
- Race Start, 117
 - Approach Zone, 122
 - Define line, 119
 - Drop Marks, 124
 - Heat is on, 127
 - Introduction, 119
 - Pre-determined markers, 121
 - Race is on, 129
 - Start Zone, 125
 - Wind shift direction, 120
- Routing, 129
 - Race, 129

S

- Security, 152
 - Data flow, 153
 - Implementation, 154
 - Policy, 154
- Signal K, 67
 - BLE, 71
 - Configuration, 73
 - data, 71
 - GPIO, 71
 - HALT state, 74
 - I2C, 71
 - keys, 71
 - NMEA-0183, 70, 72
 - NMEA-2000, 71
 - No connection, 74
 - No data, 74
 - operation, 71
 - Pane orientation, 74
 - paths, 71
 - server node, 69
 - Timestamp issues, 75
 - Troubleshooting, 73
- Start line, 117

T

- Tactics, 34
 - Align Compass, 66
 - Boat laylines, 43
 - Calculate Current, 41
 - Calculate Leeway, 40
 - Current on chart, 42
 - Data Export, 64

- Features, 35
 - Heel corrections, 40
 - Introduction, 35
 - NKE Perf.Records, 61
 - Performance data, 50
 - Polar on chart, 49
 - Prerequisites, 38
 - SOG not STW, 40
 - Temporary WP, 59
 - Terminology, 65
 - True wind data, 39
 - Vector graph, 51
 - Wind barbs, 43
 - With Polar, 48
 - Without Polar, 39
- Troubleshooting, 85
 - Engine/Energy, 85
 - Linux installation, 11
 - nginx, 102
 - Race Mark, 137
 - Use browser, 86
 - Windows installation, 8
- Tweaks, 138
 - COG/SOGDamp, 141
 - DashT, 140
 - DashT Colors, 142
 - DashT Dashboard, 140
 - DashT Dials, 143
 - DashT Fonts, 141
 - DashT Race, 151
 - DashT WebView/blank, 151
 - DashT WebView/cache, 151
 - DashT WebView/file, 151
 - DashT WebView/HTTP, 150
 - Tactics, 144
 - Tactics Average Wind, 149
 - Tactics CSV out, 149
 - Tactics Performance, 146
 - Tactics Stream in, 150
 - Tactics Stream out, 150

U

- UID, 86
 - Forget config., 86

W

- Windows, 4
 - Installation, 4
 - Installation troubleshooting, 8
 - Uninstall, 6