
PlatformPilot GTK GUI Documentation

Release 1.10.0

Neobotix GmbH

Jan 28, 2026

CONTENTS:

1	Installation	1
1.1	Ubuntu	1
1.2	Windows	1
2	Connecting	3
2.1	Load Maps	3
2.2	Save Maps	4
2.3	Download Maps	4
2.4	Upload Maps	4
2.5	Disconnect	5
3	Grid Mapping	7
3.1	Remote Control	7
3.2	Map Update	8
3.3	Map Editing	8
4	RoadMap Creation	11
4.1	Select Tool	11
4.2	Move Tool	11
4.3	Node Tool	11
4.4	Station Tool	12
4.5	Area Tool	12
4.6	Rotate Tool	12
4.7	Undo / Redo	12
5	Navigation	13
5.1	Set Goal Pose	13
5.2	Set Goal Station	13
5.3	Cancel Goal	13
5.4	Pose Estimate Tool	14
5.5	Set Pose Tool	14
6	Simulation	15
6.1	Types	15
6.2	First Steps	16
6.3	Stopping	16
7	Replay	17
8	PlatformPilot	19
8.1	User Manual	19

8.2	Programmer's Manual	53
8.3	API Reference	79
Index		151

INSTALLATION

1.1 Ubuntu

- Obtain the correct `neobotix-pilot-gtkgui` – debian package for your architecture and Ubuntu version. For example, the file name for the *x86_64 Ubuntu 20.04* package ends with `-x86_64-ubuntu-20.04.deb`.
- Install the package as follows:

```
sudo dpkg -i neobotix-pilot-gtkgui-*.deb
```

- Install missing dependencies via:

```
sudo apt -f install
```

- The contents of the package should now be installed under `/opt/neobotix/pilot-gtkgui/`.
- The default workspace folder is `~/pilot/`. You can change this setting via the environment variable `PILOT_HOME`:

```
export PILOT_HOME=/your/custom/path
```

It is highly recommended to add this line to your `~/.bashrc` to apply it automatically.

- To run the *GTK GUI* execute the following:

```
/opt/neobotix/pilot-gtkgui/scripts/pilot_gtkgui.sh
```

This command should also be available as an entry in the applications menu.

- Alternatively you can install a desktop launcher as follows:

```
xdg-desktop-icon install /usr/share/applications/Neobotix-PlatformPilot-GUI.  
↳ desktop
```

You may need to right click on the icon first and select “Allow Launching” to activate it.

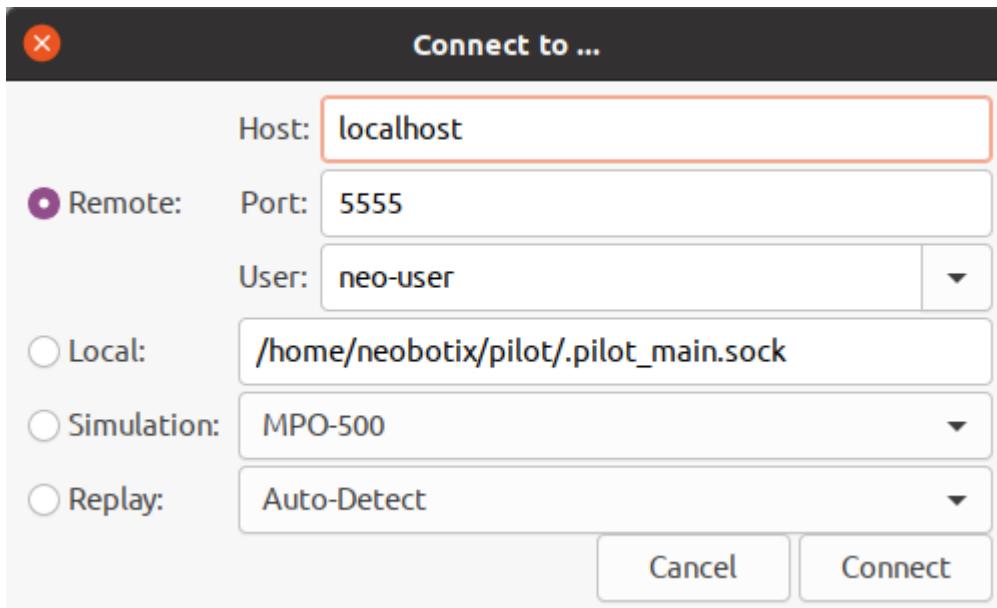
1.2 Windows

- Execute the `neobotix-pilot-gtkgui-?-windows10.exe` installer.
- The package will be installed to your Profile Folder in `C:\Users\?\PlatformPilotGUI\`.
- Execute `PlatformPilotGUI\pilot\pilot_gtkgui.bat` or double click on the Desktop Icon.

CONNECTING



Will open a connect dialog:



When connecting remotely the IP address or hostname and TCP port (default 5555) need to be specified. Optionally a login can be performed with the specified user, which is needed for most management functionality.

When connecting locally via UNIX socket only the socket file is needed (Linux only).

To run a simulation select *Simulation* and the platform type. See [Simulation](#).

To replay a data recording select *Replay*. See [Replay](#).

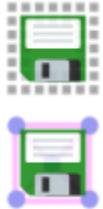
2.1 Load Maps



Will open a file dialog to choose a *.dat map file.

Note: Loading a map does not automatically upload it to the platform (or simulation). See *Upload Maps*.

2.2 Save Maps



Will save the current *Grid Map* or *Road Map* to a file.

Alternatively the `File` menu can be used which offers more options for saving.

2.3 Download Maps



Depending on the current edit mode, either all maps are downloaded (*View Only* mode), only the *Grid Map* is downloaded (*Edit Grid Map* mode) or only the *Road Map* is downloaded (*Edit Road Map* mode).

Upon connecting to a *PlatformPilot* instance both *Grid Map* and *Road Map* are downloaded automatically.

If local changes were overridden by accident it is always possible to revert back.

Alternatively the `File` menu can be used which offers more options for downloading.

2.4 Upload Maps





Depending on the current edit mode, either all maps are uploaded (*View Only* mode), only the *Grid Map* is uploaded (*Edit Grid Map* mode) or only the *Road Map* is uploaded (*Edit Road Map* mode).

Alternatively the *File* menu can be used which offers more options for uploading.

2.5 Disconnect



Disconnects from a running *PlatformPilot* instance.

GRID MAPPING

To create a new *Grid Map* switch to the mapping mode using the *Pilot Mode > Mapping* menu. Now you can move the platform around using the hardware joystick or by using remote control, see below.

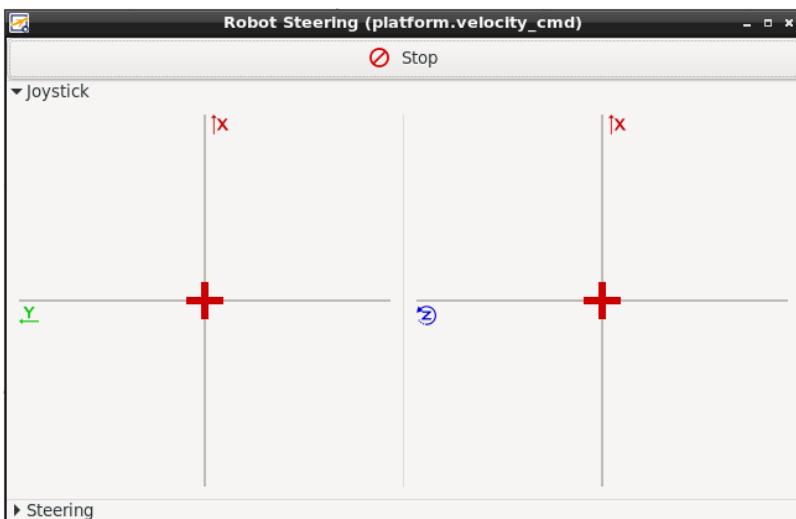
The new *Grid Map* will be created and updated while driving around. When the mapping process is finished you can save the new map and / or upload it to the platform, see *Connecting*.

Note: Requires permissions of *neo-installer* or *neo-admin* when connecting remotely.

3.1 Remote Control



A modal dialog will appear. You can choose between a joystick representation and velocity sliders:





3.2 Map Update

To update the current *Grid Map* switch to the map update mode using the *Pilot Mode > Map Update* menu. The platform needs to be localized before entering this mode.

Note: Requires permissions of *neo-installer* or *neo-admin* when connecting remotely.

3.3 Map Editing

Usually a new map needs to be edited, to do that switch to the *Grid Map* edit mode:



Most edit tools below can be scaled via the vector zoom buttons on the left toolbar:



Alternatively **Ctrl + Scroll Up/Down** can be used.

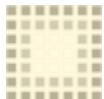
3.3.1 Erase Tool

Obstacles which are not static should be erased with the erase tool:



Holding the left mouse button will erase the area, ie. make it free. Holding the right mouse button will mark the area as unknown.

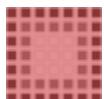
3.3.2 Make Dynamic Tool



Holding the left mouse button will mark the area as dynamic. Holding the right mouse button will erase only dynamic areas.

Dynamic areas are ignored when updating the map.

3.3.3 Make Prohibited Tool



Holding the left mouse button will mark the area as prohibited. Holding the right mouse button will erase only prohibited areas.

Prohibited areas are not entered by a platform. They are also ignored when updating the map.

3.3.4 Draw Line Tool



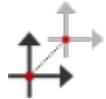
Allows to draw artificial walls. **Left Click** will start and finish a line. Hold **Shift** to draw lines continuously. Hit **ESC** to abort a line.

3.3.5 Crop Tool



Crops the grid map to a rectangular area. **Left Click** will start and finish the rectangle. The area can go beyond the original grid map area, in which case the additional space will be added as unknown.

3.3.6 Set Origin Tool



Allows to change the origin point of the grid map. **Left Click** will set the new origin point at the position of the mouse cursor.

3.3.7 Rotate Grid Map Tool



Allows to change the rotation of the grid map. **Left Click and hold** to rotate with the mouse cursor, release to set. Rotation will happen around the natural origin, i.e. the bottom left corner of the grid map.

3.3.8 Undo / Redo



ROADMAP CREATION

To create or edit the *Road Map* switch to the edit mode:



4.1 Select Tool



Left Click will select an element. Hold **Ctrl** to select multiple elements.

To select a whole stretch of road (shortest path), select a starting node then hold **Shift** and select an end node. Hold **Ctrl + Shift** to append another stretch of road.

4.2 Move Tool



Left Click will select and move an element. Right Click will select an element. Hold Left Mouse Button to move currently selected elements.

4.3 Node Tool



Left Click will add a new node. Right Click will select a node. Hold **Shift** to connect new node with previous selection. Hold **Shift** and Right Click a node to connect with current selection.

4.4 Station Tool



Left Click will add a new station. Hold Left Mouse Button to assign orientation when adding a station. Right Click will select a node. Hold Shift to connect new station with previous selection. Hold Shift and Right Click a node to connect with current selection.

4.5 Area Tool



Use Left Click s to add the vertices of the area, use Shift and Left Click to add the closing vertex or use Right Click to close the area using the previous vertex as the closing vertex. Completed areas as well as single vertices can be selected and moved by means of the Select Tool and the Move Tool. When a single vertex is selected, use Shift and Right Click to add a vertex to the area.

4.6 Rotate Tool



Left Click a station to re-define its orientation, finish via Left Click again.

4.7 Undo / Redo



NAVIGATION

By default, the robot should be in Navigation mode. To switch into navigation mode, use the *Pilot Mode > Navigation* menu.

For autonomous motion, the robot must also be in Automatic motion mode:



If needed initialize the localization first via the *Pose Estimate Tool*, see below.

5.1 Set Goal Pose



Left Click and hold to define a new goal pose. Right Click and hold to define a new goal pose without using the *RoadMap*.

5.2 Set Goal Station



Left Click on a map station to set a new goal station. Right Click on a map station to set a new goal station without using the *RoadMap*.

5.3 Cancel Goal



Will abort the current goal and stop immediately.

5.4 Pose Estimate Tool



Left Click and hold to define a pose estimate (initialize localization). Right Click and hold to shift the pose estimate in XY direction (initialize localization).

5.5 Set Pose Tool

In simulation mode it is possible to “teleport” the platform via:



Left Click and hold to define a pose.

SIMULATION

The PlatformPilot GUI contains a feature-rich simulation of PlatformPilot. Therefore, the GUI is often used as a demo and showcase for the capabilities of a full PlatformPilot installation.

6.1 Types

From the installation, two types of robot simulation are available.

6.1.1 Built-in

This simulation is easy to launch but does not provide a way to configure robot properties and behaviour. It is meant to quickly set up one of the standard robot models and get an overview of the user interface in live action. For many scenarios this is completely sufficient.

The built-in simulation is launched via the *Connecting* dialog. In the dialog window, select *Simulation* and the platform type and confirm with *Connect*.

6.1.2 Standalone

This simulation runs in its own process outside the GUI. From user perspective, it behaves like a regular PlatformPilot installation on a real robot.

Tip: In Linux, make sure that prior to running the standalone simulation, the GUI has been executed at least once. This is necessary to set up your Pilot home folder.

Open your Pilot home folder (usually `~/pilot/` in Linux and `pilot/` in the installation directory in Windows) and change into the subfolder `config/local/`. All configuration for your simulation will be made here. For a basic setup, just make sure that the content of the parent file points to the configs of the robot model you want to simulate.

Run the simulation by executing the `run_pilot_simulation.sh` (Linux) or the `run_pilot_simulation.bat` (Windows) file in the Pilot home folder. The installation also brings launchers for this. In Windows, it should already have been created on your Desktop. In Linux, it can be created with `xdg-desktop-icon install /usr/share/applications/Neobotix-PlatformPilot-GUI-Simulation.desktop`.

Also consult the general instructions of running PlatformPilot at *Running*.

6.2 First Steps

If you are using the standalone simulation, you have to explicitly connect to it in the GUI, usually via the `localhost` address. See [Connecting](#).

In any case, the simulation needs a grid map to create localization and navigation data. If you don't have a map of your own (yet), you can use the provided example map which is available through *File > Stored Grid Maps*.

6.3 Stopping

The built-in simulation can be stopped with the *Disconnect* button.

The standalone simulation runs in its own process that has to be terminated. In its terminal window, either hit `Ctrl+C` or press `Enter` and use the `quit` command. On Linux, the process will also terminate upon receiving the `SIGTERM` signal.

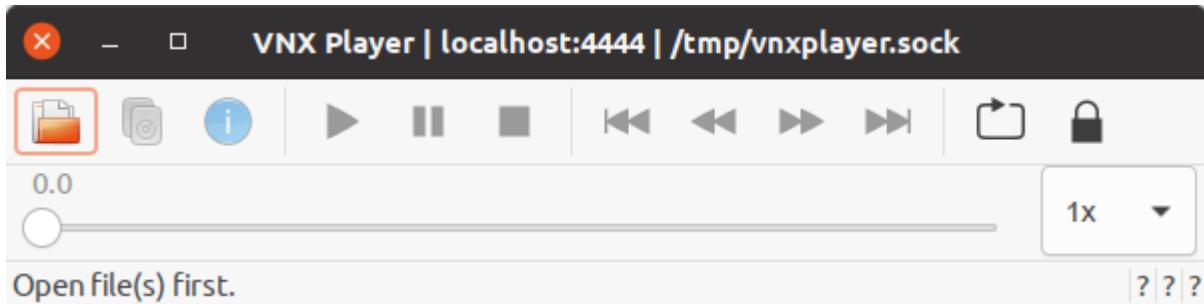
CHAPTER
SEVEN

REPLAY

To replay a data recording open the connect dialog and select *Replay* via:



A *VNX Player* window will open:



It can be operated like a regular audio player.

PLATFORMPILOT

8.1 User Manual

This section describes the necessary steps to install and run PlatformPilot, to manage configurations and to control the behaviour while running. Note that, typically, the setup steps were already executed before delivery of the robot.

8.1.1 Installation

Note: In most cases, PlatformPilot will come preinstalled on your robot. End users will rarely need these instructions.

Ubuntu

- Obtain the correct `neobotix-pilot-core-` debian package for your architecture and Ubuntu version. For example, the file name for the *x86_64 Ubuntu 20.04* package ends with `-x86_64-ubuntu-20.04.deb`.
- Install the package as follows:

```
sudo dpkg -i neobotix-pilot-core-*.deb
```

- Install missing dependencies via:

```
sudo apt -f install
```

- The contents of the package should now be installed under `/opt/neobotix/pilot-core/`.
- Optionally, but highly recommended, install the necessary drivers for hardware acceleration (OpenCL or CUDA) depending on your PC. For example, for OpenCL on an intel chipset you need `intel-opencl-icd`.
- If you received a license security dongle and you already attached it, you may need to unplug it and attach it again in order for the installed udev rule to take effect.
- The default workspace folder is `~/pilot/`. You can change this setting via the environment variable `PILOT_HOME`:

```
export PILOT_HOME=/your/custom/path
```

It is highly recommended to add this line to your `~/.bashrc` to apply it automatically.

- To setup the workspace execute the following script:

```
sh /opt/neobotix/pilot-core/scripts/pilot_setup.sh /opt/neobotix/pilot-core
```

This will setup the pilot workspace folder, as selected in the previous step, with default settings.

- **Change into the pilot workspace folder.** The paths in the following steps will be relative to this location.
- Change the default configuration to match your platform type:

```
echo /opt/neobotix/pilot-core/config/default/mpo-700/ > config/local/parent
```

where you replace mpo-700 with your platform type.

- If you received a license security dongle, edit your config/local/LicenseCheck.json file to replace the place holder with your actual license key which you should find printed on the dongle.
- Set a *name* and *serial* for your platform in config/local/platform.json.
- Optionally set additional custom configuration options in config/local, see [Configuration](#).
- Optionally setup users and passwords, see [User Management](#).
- To enable autostart:

```
cp /usr/share/applications/Neobotix-PlatformPilot.desktop ~/.config/autostart/
```

- In addition you can install a desktop launcher as follows:

```
xdg-desktop-icon install /usr/share/applications/Neobotix-PlatformPilot.desktop
```

You may need to right click on the icon first and select “Allow Launching” to activate it.

- See [Running](#) on how to run *PlatformPilot*.

Windows

- Execute the neobotix-pilot-core-?.exe installer.
- The package will be installed to your Profile Folder in C:\Users\?\PlatformPilot\.
- Change into the folder PlatformPilot\pilot\config\local. Edit the file parent with a line like this:

```
../../../../config/default/mpo-700/
```

where you substitute mpo-700 with your platform type.

- If you received a license security dongle, edit your pilot\config\local\LicenseCheck.json file to replace the place holder with your actual license key which you should find printed on the dongle.
- Set a *name* and *serial* for your platform in pilot\config\local\platform.json.
- Optionally set additional custom configuration options in pilot\config\local, see [Configuration](#).
- Optionally setup users and passwords, see [User Management](#).
- See [Running](#) on how to run *PlatformPilot*.

8.1.2 Running

Note: If you received a license security dongle, make sure to have it attached.

You can always execute the supplied binaries directly, but for convenience we provide little helper scripts that take care of some required command line options.

Pilot

In Windows execute the `pilot/run_pilot.bat` file either in a terminal or by double clicking on it.

In Ubuntu:

```
cd ~/pilot
./run_pilot.sh
```

Additional options can be provided to the scripts as such:

```
./run_pilot.sh --LocalPlanner.vel_limits.max_trans_vel 0.5
run_pilot.bat --LocalPlanner.vel_limits.max_trans_vel 0.5
```

Permanent options can be set by writing config files in `config/local`, see [Configuration](#). For example a file `config/local/LocalPlanner.json`:

```
{
    "vel_limits": {
        "max_trans_vel": 0.5
    }
}
```

will set the default maximum velocity of the platform to 0.5 m/s.

Simulation

The simulation binary will simulate the kinematics as well as sensor data without accessing any hardware. Otherwise there is no difference to `run_pilot.sh`.

In Windows execute the `pilot/run_pilot_simulation.bat` file either in a terminal or by double clicking on it.

In Ubuntu:

```
cd ~/pilot
./run_pilot_simulation.sh
```

Replay

To run *PlatformPilot* in replay mode use the provided script:

```
cd ~/pilot
./run_pilot_replay.sh
```

It will try to connect to a running VNX player on `/tmp/vnxplayer.sock`. Replay mode is used to view what happened during a recorded drive. Similar to simulation mode there is no hardware access.

Data Recording

To make a data recording use the provided script:

```
cd ~/pilot
./record_data.sh
```

The resulting file will be in `user/data/`.

Parameters

It is possible to override the default configuration by either writing local config files in `config/local/` or by specifying parameters on the command line.

The following configuration options can be set for most executables above:

`self_test` If to perform a self test at the beginning. Will exit if it fails. (default = `true`)

`auto_shutdown_host` If to automatically shutdown the host machine in case platform is being turned off via the key switch. (default = `true`)

`with_hardware` If hardware modules should be started, ie. hardware is present on this machine and not on a remote. (default = `true`)

`enable_joystickmng` If to enable joystick input. (default = `true`)

`enable_http_server` If to enable HTTP server on port 8888. Needed for HTTP API, WebGUI, etc. (default = `true`)

`enable_opcua_server` If to enable the `vnx.opc_ua.Server` module. (default = `false`)

`opcua_proxy_map` A map of `vnx.opc_ua.Proxy` modules to start, `[module name => server address]`.

`vnx_proxy_map` A map of `vnx.Proxy` modules to start, `[module name => server address]`.

`vnx_server_map` A map of `vnx.Server` modules to start, `[module name => server endpoint]`.

`vnx_jrpc_server_map` A map of `vnx.JRPC_Server` modules to start, `[module name => server endpoint]`.

`footprint` Footprint of the platform as a JSON object, see `pilot.Footprint`.

`platform` Static platform information as a JSON object, see `pilot.PlatformInfo`.

By default the following servers are started:

`vnx.Server on .pilot_main.sock` Supports connections from the local machine via UNIX domain sockets, using the native VNX binary protocol. Almost all permissions are granted to clients connecting to this server, ie. no logins required. A `vnx.Proxy` is needed to connect. On Linux systems only.

`vnx.Server on 0.0.0.0:5555` Supports connections from anywhere via TCP/IP sockets, using the native VNX binary protocol. Login is required to gain anything more than read-only permissions, see [User Management](#). A `vnx.Proxy` is needed to connect.

In general, most applications accept the following command line arguments:

- The `-h` switch displays a small help message and exits.
- The `-d` switch increases the debug level of the terminal output
- The `-c` switch accepts a directory with configuration files. You can specify several directories by repeating the switch or by providing multiple directories at once. The order of directories matters, they are read from left to right, potentially overriding previous values.
- Finally, you can set specific config values directly on the command line. See [Configuration](#) for details.

Maps

The active *Grid Map* and *Road Map* are stored in the `~/pilot/` folder as `current_grid_map.grid` and `current_road_map.road`.

They can either be copied there by hand or uploaded via the *GTK GUI*.

8.1.3 Docker

You might run into a situation where you want to run multiple simulation instances of PlatformPilot, for example to test out a fleet of robots or to work simultaneously on multiple tasks. However, if you just run multiple instances out of the box, they will all try to access the same network ports and write to the same log files, which is not going to work well. You can set up individual configurations to assign each instance its own resources without collisions, but that requires some work.

To simplify this task we provide a *Docker* configuration file which will help you to set up a testing environment. If you are not familiar with Docker please refer to the official documentation: <https://docs.docker.com/>

Building the Container Image

Every `neobotix-pilot-core-debian` package contains the corresponding *Dockerfile* which can be easily extracted from the package.

- Obtain the correct `neobotix-pilot-core-debian` package for your architecture and Ubuntu version.
- Obtain the *Dockerfile* either by

1. installing the debian package. Then you can find it in:

```
/opt/neobotix/pilot-core/scripts/Dockerfile
```

2. or open the debian package with an archiver program of your choice. Inside you will find another archive called `data.tar.*`. Open it and navigate to:

```
./opt/neobotix/pilot-core/scripts/Dockerfile
```

- Put the *Dockerfile* in the same directory where the debian package is located and build the image like this:

```
docker build -t <image_name> .
```

- After successful build, verify the image by running:

```
docker images
```

Running the Simulation in the Container

You can run multiple instances of a Docker container from one single Docker image. To run multiple PlatformPilot instances you need to redirect ports in order to access them. In this example four instances are started and the ports for the *GTK GUI* and the *WebGUI* are forwarded:

```
docker run --rm --name pilot-core-1 -p 5556:5555 -p 81:8888 <image_name>
docker run --rm --name pilot-core-2 -p 5557:5555 -p 82:8888 <image_name>
docker run --rm --name pilot-core-3 -p 5558:5555 -p 83:8888 <image_name>
docker run --rm --name pilot-core-4 -p 5559:5555 -p 84:8888 <image_name>
```

To stop running containers use:

```
docker stop pilot-core-1
docker stop pilot-core-2
docker stop pilot-core-3
docker stop pilot-core-4
```

8.1.4 Terminal Interface

While *running one of the pilot binaries* you can, at any time, hit the `<Enter>` key which interrupts the log output and gives you a terminal prompt. You can now execute one of several commands. Type it in and press `<Enter>` to execute.

Commands that have an output keep the log paused after execution, to give you time to look at it. The log continues when you press `<Enter>` again.

The following commands are currently available:

- `quit` exits the application
- `debug [level]` changes the level of the output. The bigger the number, the more log messages you get.
- `errors` outputs the last error messages
- `topic [expr]` searches all topic names for `expr`. If there is no exact match, all topics that `expr` is a prefix of are shown. If there are none, all matches are shown.
- `module [expr]` lists all modules that `expr` matches on. If there is an exact match, it displays more detailed information for that module.
- `grep <expr>` restricts terminal output to messages that contain `expr`.
- `journal [expr]` does the same as `grep` but first shows all past log messages that contain `expr`.
- `spy [expr]` does the same as `dump` but without the actual message content.
- `dump [expr]` outputs the messages on all topics that match `expr`.
- `htop [field]` shows the modules with the highest CPU load in a table that refreshes regularly. `field` can be `avg` to change the sorting.
- `exec <module>[.method [args...]]` and `call <module>[.method [args...]]` invoke the given method of the given module with the given arguments. `exec` waits for the command to finish and displays the result while `call` performs an asynchronous call. If no method is given, a list of method signatures for the module is shown.

While typing, the terminal will provide you with suggestions for the command and possible arguments. Press the `<Tab>` key to complete, press it twice to get a list of suggestions. You can also use the up and down arrows to scroll through the history of past commands.

8.1.5 Configuration

When you start a command line tool, you can supply it with one or more configuration destinations. The configuration is written in [JSON](#).

Reading

Hierarchy

The configuration follows a hierarchy that is separated by dots (.) in the names of the keys. The hierarchy is inferred from the file system and the structure of the JSON in the files.

For example, assume you supply the config destination `config/local/` and there is a file `config/local/some/path/Test.json` and it contains the following content:

```
{
  "some_key": {
    "other_key": {
      "the_answer": 42,
      "the_question": ["life", "universe", "everything"]
    }
  }
}
```

In this case, the config key `some.path.Test.some_key.other_key.the_answer` will be read and set to 42 and the key `some.path.Test.some_key.other_key.the_question` will contain a list of strings as shown.

A config destination can contain the special file called `parent`. It should consist of a single line giving a path (absolute or relative to the current destination) to include. This means the contents of the given destination will be treated as if they were present in the current destination. Additionally, the parent destination will be read first which is important for cascading (see below).

Cascading

The structure of the configuration makes it possible (and actually good and common practice) to split a config object across multiple locations. For example your config destination may have a file `Test.json` that contains

```
{ "field_1": 42 }
```

but also reference a parent directory that also has a file `Test.json` that contains

```
{ "field_2": 23 }
```

In effect, both `Test.field_1` and `Test.field_2` are correctly read. By default, we use this technique to separate

- configuration specific to an installation (in `config/local/` in the pilot home folder), which includes
- configuration specific to a robot model (in `config/default/XXX/` in the installation folder), which includes
- generic configuration (in `config/default/generic/` in the installation folder).

This makes the configuration easy to maintain.

Given the rules for the hierarchy and for parent destinations, it is very much possible to specify the same key multiple times. In that case **the key that is read last overwrites any previous one of the same name**. In particular, a destination always overrides its parent.

Appending

It is also possible to append values to an array, by adding a `+` to the variable name as such:

```
{ "array": [1, 2, 3] }
{ "array+": [4, 5, 6] }
```

The resulting array config value will be [1, 2, 3, 4, 5, 6].

Command line

You can also supply configuration on the command line. For example, to set `MyModule.max_velocity` to 5, append `--MyModule.max_velocity 5` to the command line.

Values are read as JSON objects. Make sure to use the correct quoting according to the shell you use. When setting a boolean value to `true`, you can omit the value and just give the key.

Arrays and objects can be provided on the command line as follows:

```
command --array [1, 2, 3, 4] --object {"field": 1234, "array": ["a", "b", "c"]}
```

Config keys given at the command line are read as the ultimate last ones, so they override any previous keys with the same names. This is useful for testing out certain values without constantly having to edit files.

Assignment

While some config keys are read explicitly by the application, most of the assignment happens automagically.

Modules

A module that is started with the name `MyModule` will automatically be assigned all configuration below the `MyModule.` key. From then on it behaves like a class (see below).

Classes

If a configuration key is applied to an instance of a VNX class, the members of the class are assigned the subkeys where the names match.

For example if you have an object of the class `Test` below

```
class Test2 {
    string what;
    int over;
}

class Test {
    string name;
    int value;
    vector<int> numbers;
    Test2 other_one;
}
```

and you assign it the config

```
{
  "name": "Picard",
  "value": 1234,
  "numbers": [1, 2, 3, 4],
  "other_one": {
    "what": "ever",
    "over": 9000
  }
}
```

then all the fields get assigned the config key with the matching name. Notice that `other_one` is another object which will get the same treatment recursively with the embedded config object.

Fields without a matching config key are default initialized and spare keys without a matching field are silently ignored.

8.1.6 User Management

The authentication model is based on *users* who may or may not have permission to perform certain actions. Without authentication, by default, only a limited number of actions are available.

Every user has a name and a password which are used for authentication.

Login

Command line tools usually provide the `-u <username>` switch to provide a user name. Please refer to the documentation *of the respective tool*.

Graphical tools usually provide a graphical way of authentication.

Permissions

Most API functions require a certain permission to be executed. If the permission is not held by the user an error is returned or thrown.

Every user has a set of *access roles* and every access role has a set of *permissions*. The permissions of a user consist of the permissions of their access roles. Mindful assignment to access roles allows for a fine-grained access control in a multi-user environment.

See `vnx.access_role_e` for the set of default access roles and their permissions.

Configuration

Available access roles and their sets of permissions can be configured via the config key `vnx.authentication.permissions` which is an object with access roles (as strings) as keys and a list of permissions (as strings) as values.

Users and their access roles can be configured with the key `vnx.authentication.users` which is a list of *user objects*. Note that the passwords for the config map `vnx.authentication.passwd` are usually kept separately (in a subdirectory) so that they can be protected from reading and writing.

Here is the default configuration `config/default/generic/vnx/authentication.json`:

```
{  
    "users": [  
        {  
            "name": "neo-user",  
            "access_roles": ["USER"]  
        },  
        {  
            "name": "neo-installer",  
            "access_roles": ["INSTALLER"]  
        },  
        {  
            "name": "neo-admin",  
            "access_roles": ["INSTALLER", "ADMIN"]  
        }  
    ],  
    "permissions": {  
        "OBSERVER": [  
            "READ_CONFIG",  
        ],  
        "USER": [  
            "pilot.permission_e.MOVE",  
            "pilot.permission_e.CHARGE",  
            "pilot.permission_e.INITIALIZE",  
            "pilot.permission_e.RECORD_DATA",  
            "pilot.permission_e.EXECUTE_SCRIPT",  
            "vnx.addons.permission_e.READ_DIRECTORY",  
            "vnx.addons.permission_e.FILE_DOWNLOAD",  
        ],  
        "TASK": [  
            "VIEW",  
            "CONST_REQUEST",  
            "pilot.permission_e.MOVE",  
            "pilot.permission_e.CHARGE",  
            "pilot.permission_e.RECORD_DATA",  
            "pilot.permission_e.RELAY_CONTROL",  
            "pilot.permission_e.DISPLAY_CONTROL",  
        ],  
        "INSTALLER": [  
            "pilot.permission_e.MOVE",  
            "pilot.permission_e.CHARGE",  
            "pilot.permission_e.INITIALIZE",  
            "pilot.permission_e.RECORD_DATA",  
            "pilot.permission_e.REMOTE_CONTROL",  
            "pilot.permission_e.RELAY_CONTROL",  
            "pilot.permission_e.DISPLAY_CONTROL",  
            "pilot.permission_e.CHANGE_GRIDMAP",  
            "pilot.permission_e.CHANGE_ROADMAP",  
            "pilot.permission_e.UPLOAD_SCRIPT",  
            "pilot.permission_e.EXECUTE_SCRIPT",  
            "pilot.permission_e.INTERVENE_SCRIPT",  
            "vnx.addons.permission_e.READ_DIRECTORY",  
            "vnx.addons.permission_e.FILE_DOWNLOAD",  
            "vnx.addons.permission_e.FILE_UPLOAD",  
            "vnx.addons.permission_e.FILE_DELETE",  
        ]  
    }  
}
```

As can be seen, built-in permissions (of type `vnx.permission_e`) such as `VIEW` and `READ_CONFIG` can be specified without the full namespace. A permission can also be removed by adding a `!` in front of the name: `!VIEW`.

For more information regarding configuration files see [Configuration](#).

Adding Users

To add new users create a config file `config/local/vnx/authentication.json`:

```
{
  "users+": [
    {
      "name": "foo",
      "access_roles": ["USER", ...]
    },
    ...
  ]
}
```

By removing the `+` in `users+` you can discard the default users which were set in `config/default/generic/vnx/authentication.json`.

Passwords

Passwords can be set or changed via the `vnxpasswd` command line tool, see [Tools](#).

8.1.7 Tools

There is a collection of command line tools to communicate with a running program instance.

Common command line parameters are:

- n <node>** supply a way to connect to the process. It can be a UNIX socket or a network address in the form `host:port`.
- u <user>** authenticate with the given user name. You will be asked to enter the password before the connection is made.

On Linux the tools are found in `/opt/neobotix/pilot-core/bin/` or `/opt/neobotix/pilot-gtkgui/bin/`. On Windows the tools are found in the `bin` folder.

pilot_execute

To execute a custom *Lua Script* on the platform:

```
pilot_execute -n localhost:5555 -u neo-admin -f my_script.lua
```

Above command will run until the script exits. To return immediately add the `-a` option.

Requires permission `UPLOAD_SCRIPT`.

pilot_battery_watchdog

To watch the battery level and optionally shut down the platform when below a certain level.

```
pilot_battery_watchdog -n localhost:5555 -l 10
pilot_battery_watchdog -n localhost:5555 -s 0.42 -u neo-admin
```

Shutting down the platform (-s switch) requires permission HOST_SHUTDOWN. For simply watching the battery level, no login is needed.

vnxpasswd

To change / set a password:

```
vnxpasswd -c config/local/ -u neo-admin
```

vnxservice

To call a method on a module:

```
vnxservice -n localhost:5555 -u neo-user -x HybridPlanner move_to_station Station1 {
    ↪"max_velocity": 0.5}
```

To get an overview of available methods:

```
vnxservice -n localhost:5555 -i HybridPlanner
```

vnxread

To convert a *.dat, *.grid or *.road file to JSON:

```
vnxread -f my_road_map.road
```

vnxgraph

If you have *graphviz* installed, you can visualize the running modules and their topics.

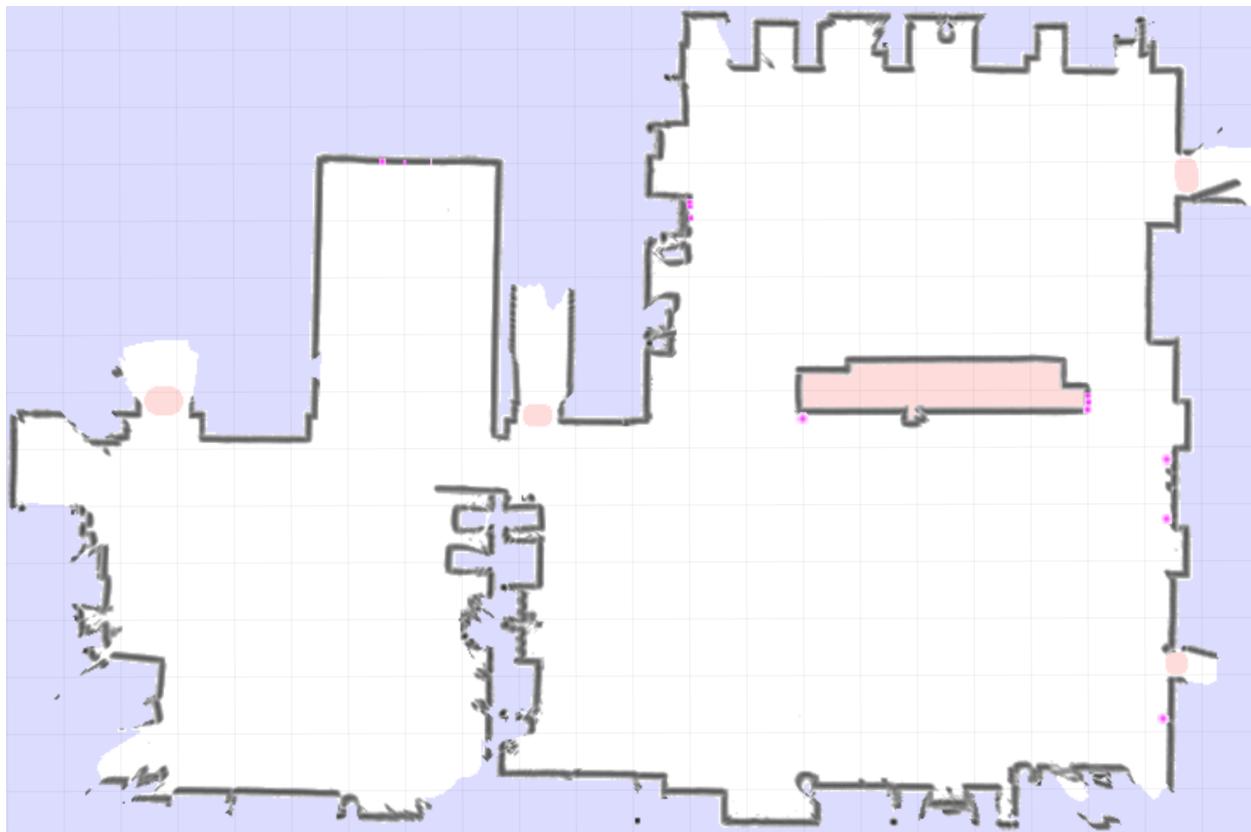
To create a graph of a running VNX process:

```
vnxgraph -n localhost:5555 > graph.dot
dot -Tsvg graph.dot > graph.svg
```

8.1.8 Grid Map

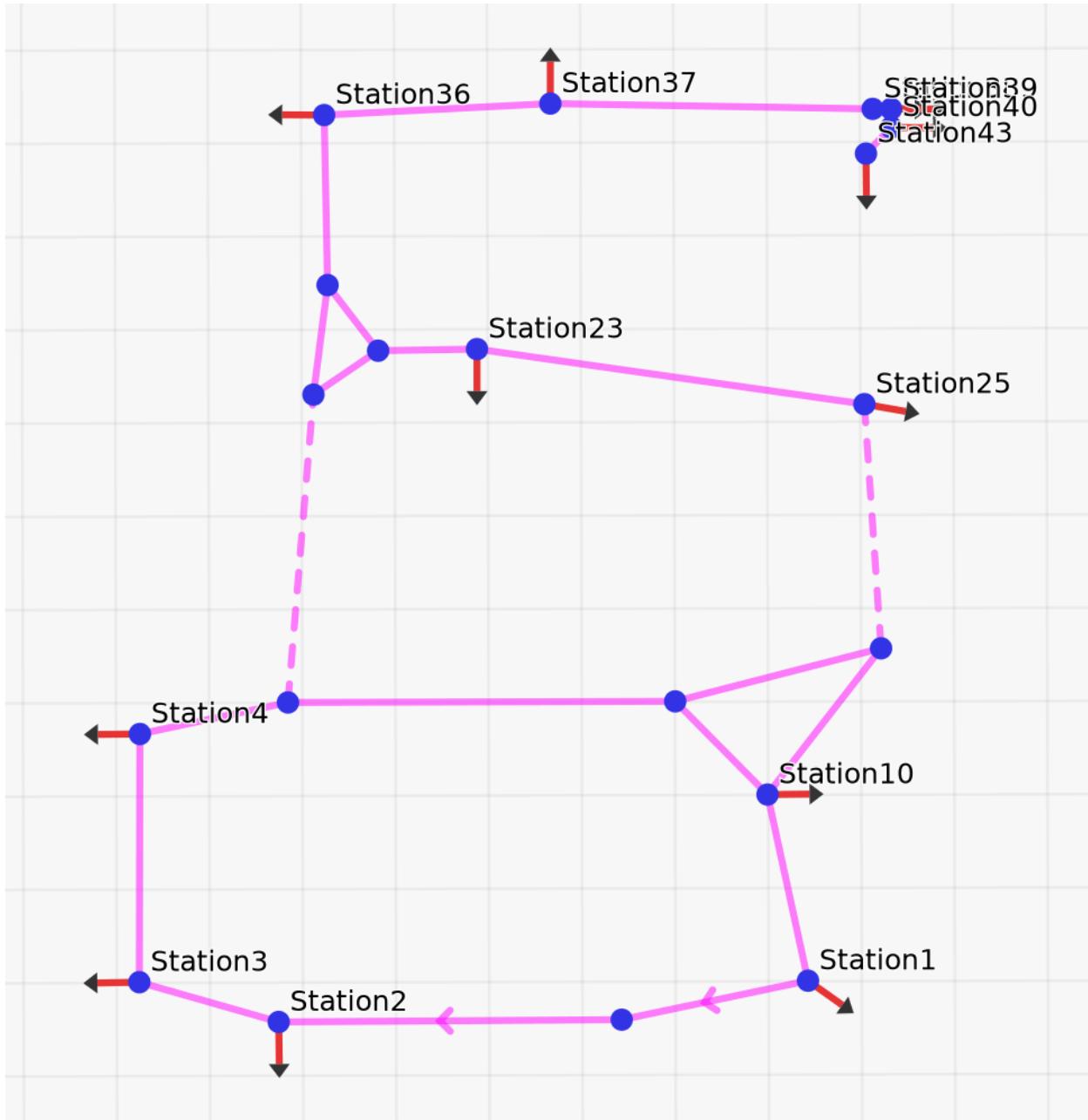
The *Grid Map* is an image representing an occupancy grid map, created via a SLAM mapping algorithm. It has two layers: normal occupancy and reflector occupancy, both consisting of 8-bit data per pixel / cell.

See also: *pilot.GridMapData*, *pilot.OccupancyMapData*



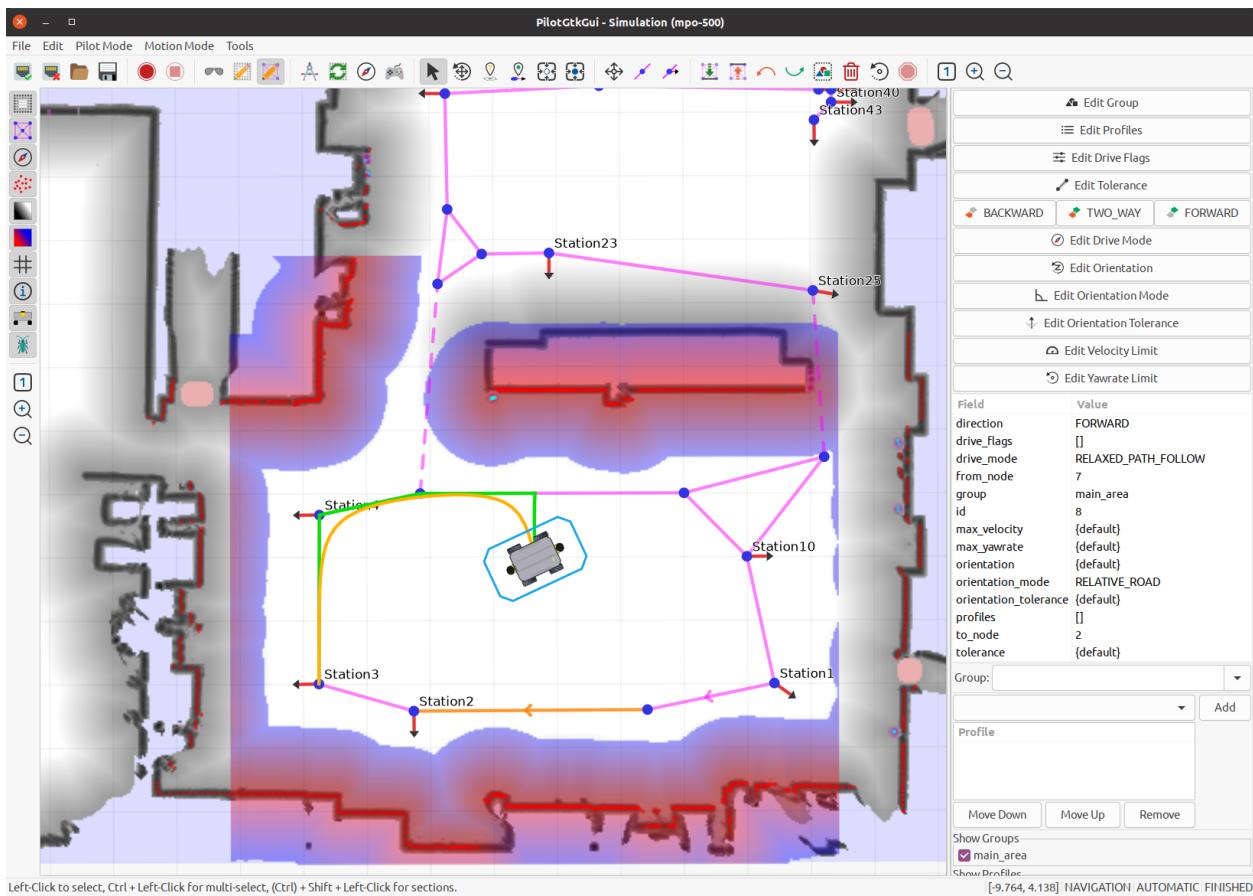
8.1.9 Road Map

The *Road Map* is a navigation map created by hand using the *GTK GUI*. It contains road segments where a platform should drive on, including a lot of additional parameters that can be set.



8.1.10 GTK GUI

The *PlatformPilot GTK GUI* provides a means to create maps, upload / download data to / from a platform, as well as visualize various data and control the platform.



8.1.11 WebGUI

You can interact with the platform using the web interface. You can access it using a web browser of your choice (Firefox and Chrome/Chromium are preferred though) by navigating to `<robot_ip>:8888`.

Toolbar

Toolbar consists of three parts:

App Menu Navigate through the app.

Page title Display information about current page.

Action buttons On every page you will see at least the login / logout button. Other buttons will be explained in the corresponding section.

Map

Changing Pilot Modes

Navigation

To switch into navigation mode:



Navigation mode allows for autonomous operation.

If needed initialize the localization first via the *Pose Estimate Tool*, see below.

Mapping

To create a new *Grid Map* switch to the mapping mode:



Now you can move the platform around using the hardware joystick.

The new *Grid Map* will be created and updated while driving around. When the mapping process is finished you can upload the new map to the platform by clicking the *Save* button.

Note: Requires permissions of *neo-installer* or *neo-admin* when connecting remotely.

Map Update

To update the current *Grid Map* switch to the map update mode:



The platform needs to be localized before entering this mode.

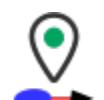
Note: Requires permissions of *neo-installer* or *neo-admin* when connecting remotely.

Set Goal Pose



Left Click / Tap and move to define a new goal pose. Red dot on the icon will indicate this mode is enabled.

Set Goal Station



Left Click / Tap on a map station to set a new goal station. Red dot on the icon will indicate this mode is enabled.

Cancel Goal



Will abort the current goal and stop immediately.

Pose Estimate Tool



Left Click / Tap and move to define a pose estimate (initialize localization). Red dot on the icon will indicate this mode is enabled.

Set Pose Tool

In simulation mode it is possible to “teleport” the platform via:



Left Click / Tap and move to define a pose. Red dot on the icon will indicate this mode is enabled.

Data Recording

To start a recording:



Note: The resulting file will be in `user/data/`.

To stop a recording:



Note: This button is only enabled when a recording is active.

Adjust View



This mode activates overlayed controls for zooming and rotating the view. Also the view can be moved around by Left Click / Tap and move.

Settings



In this context menu you can toggle visibility of various map layers.

Note: The same context menu can be accessed by Right Click / Tap & Hold.

Tip: Displaying Lidar Points and Local Cost Map is CPU intensive. Disable these layers to reduce CPU / GPU usage.

Logs

On this page the last 100 PlatformPilot log messages can be viewed. It is also possible to stop / resume message polling and filter messages by log level using action buttons.

TaskEditor

Introduction

The TaskEditor offers a visual way to create and manage tasks for the PlatformPilot's TaskHandler module. It allows users to generate Lua programs using graphical blocks by dragging and linking them. It also offers easy access to the PlatformPilot's API.

After a little training, the creation of programs is done intuitively. So that even complex sequences of tasks can be implemented very fast.

The TaskEditor is built on top of the *Blockly* library. Further information:

- <https://developers.google.com/blockly>
- <https://github.com/google/blockly/wiki>
- <https://blockly-demo.appspot.com/static/demos/code/index.html>
- <https://blockly.games/>
- <https://blockly-demo.appspot.com/static/tests/playground.html>

Toolbox

The toolbox is the side menu from whence the user may drag and drop blocks into the workspace.

There are two types of blocks with and without return value. Blocks without return value can be used directly to build a program workflow. Blocks with a return value are used as input for other blocks (i.e. variables or parameters).

PlatformPilot

The following blocks offer direct access to PlatformPilots API. See *Lua Script*.

Hardware

read_analog_input		
Parameters	channel	[number]
Returns	[number]	
Corresponding Lua function	read_analog_input ()	

read_digital_input		
Parameters	channel	[number]
Returns	[boolean]	
Corresponding Lua function	read_digital_input ()	

set_relay		
Parameters	channel	[number]
	state	[boolean]
Returns	[void]	
Corresponding Lua function	set_relay ()	

set_digital_output		
Parameters	channel	[number]
	state	[boolean]
Returns	[void]	
Corresponding Lua function	set_digital_output ()	

set_display_text		
Parameters	text	[string]
Returns	[void]	
Corresponding Lua function	set_display_text ()	

charge		
Returns	[void]	
Corresponding Lua function	charge ()	

start_charging		
Returns	[void]	
Corresponding Lua function	start_charging()	

stop_charging		
Returns	[void]	
Corresponding Lua function	stop_charging()	

reset_motors		
Returns	[void]	
Corresponding Lua function	reset_motors()	

Information Requests

get_time_sec		
Returns	[number]	
Corresponding Lua function	get_time_sec()	

get_time_millis		
Returns	[number]	
Corresponding Lua function	get_time_millis()	

get_time_macros		
Returns	[number]	
Corresponding Lua function	get_time_macros()	

get_position		
Returns	[Pose2D]	
Corresponding Lua function	get_position()	

find_closest_station ¹		
Parameters	max_distance	[number]
	position	[Pose2D]
Returns	[string]	
Corresponding Lua function	find_closest_station_name()	

Note: `find_closest_station` in Blockly returns the name of the station while in Lua it returns the `MapStation` object.

get_battery_remaining		
Returns	[number]	
Corresponding Lua function	get_battery_remaining()	

¹ For this blocks to work, you have to insert `require('neobotix')` to your program. (See [require\(\)](#))

is_charging		
Parameters		[boolean]
Corresponding Lua function	is_charging ()	

Log

log_info		
Parameters	message	[string]
Returns	[void]	
Corresponding Lua function	log_info ()	

log_warn		
Parameters	message	[string]
Returns	[void]	
Corresponding Lua function	log_warn ()	

log_error		
Parameters	message	[string]
Returns	[void]	
Corresponding Lua function	log_error ()	

Modules

require (*module_name*)

Parameters **module_name** (*String*) – Load module defined by *module_name*. *module_name* can be either a file name or a folder name. In the latter case all files in the folder will be included.

Returns void

call (*function_name*, *{args}*)

Parameters

- **function_name** (*string*) – Call function named *function_name*.
- **args** (*Array*) – *args* is an array of arguments passed to the function. Use the *create list* with *block* to wrap parameters or use a variable containing either a single parameter or a list of parameters.

Returns Return value depends on what the function to be called returns.

Movement

move_to_station		
Parameters	name	[string]
	options	[pilot.goal_options_t]
Returns	[void] / [boolean]	
Corresponding Lua function	move_to_station ()	

move_to_position		
Parameters	position	[Pose2D]
	options	[<i>pilot.goal_options_t</i>]
Returns	[void] / [boolean]	
Corresponding Lua function	move_to_position()	

move		
Parameters	dx	[number]
	dy	[number]
	dr	[number]
	options	[<i>pilot.goal_options_t</i>]
Returns	[void] / [boolean]	
Corresponding Lua function	move()	

OPC-UA

opc_ua_call		
Parameters	proxy	[string]
	object	{[number],[string]}
	method	[string]
	args	[array]
Returns	[variant]	
Corresponding Lua function	opc_ua_call()	

opc_ua_call_global		
Parameters	proxy	[string]
	method	[string]
	args	[array]
Returns	[variant]	
Corresponding Lua function	opc_ua_call()	

Note: `opc_ua_call_global` internally calls `opc_ua_call` and passes `nil` as the `object` parameter.

opc_ua_read		
Parameters	proxy	[string]
	object	{[number],[string]}
	variable	[string]
Returns	[variant]	
Corresponding Lua function	opc_ua_read()	

opc_ua_read_global		
Parameters	proxy	[string]
	variable	[string]
	Returns	[variant]
Corresponding Lua function	opc_ua_read_global()	

opc_ua_write		
Parameters	proxy	[string]
	object	{[number],[string]}
	variable	[string]
	value	[variant]
Returns	[boolean]	
Corresponding Lua function	opc_ua_write()	

opc_ua_write_global		
Parameters	proxy	[string]
	variable	[string]
	value	[variant]
Returns	[variant]	
Corresponding Lua function	opc_ua_write_global()	

node_id		
Parameters	nsi	[number]
	id	[number]
Returns	{[number],[number]}	
Corresponding Lua function		

node_id		
Parameters	nsi	[number]
	name	[string]
Returns	{[number],[string]}	
Corresponding Lua function		

User Input

wait_for_joystick		
Returns	[number]	
Corresponding Lua function	wait_for_joystick()	

wait_for_joystick_button		
Parameters	button	[number]
Returns	[void]	
Corresponding Lua function	wait_for_joystick_button()	

wait_for_digital_input		
Parameters	channel	[number]
	state	[boolean]
Returns	[void]	
Corresponding Lua function	wait_for_digital_input()	

wait_ms		
Parameters	period	[number]
Returns	[void]	
Corresponding Lua function	wait_ms()	

wait_sec		
Parameters	period	[number]
Returns	[void]	
Corresponding Lua function	wait_sec()	

wait_min		
Parameters	period	[number]
Returns	[void]	
Corresponding Lua function	wait_min()	

wait_hours		
Parameters	period	[number]
Returns	[void]	
Corresponding Lua function	wait_hours()	

Core

The following blocks offer access to basic code concepts like variables, logical expressions and loops.

If you are not familiar with programming concepts, please consult the official blockly documentation, which you can find at

- <https://developers.google.com/blockly>
- <https://github.com/google/blockly/wiki>

However, in the following section you will find a description of some important blocks.

Add-ons

Note: This section contains custom blocks provided by Neobotix GmbH.

`return()`

The `return` block ends current function and returns a value.

`to_string()`

The `to_string` block converts a value of any type to a string.

Variables

Variables can be created in several different ways:

- Some blocks such as `count with` and `for each` use a variable and defines its values. A traditional computer science term for these are loop variables.

- User-defined functions (also known as “procedures”) can define inputs, which creates variables that can be used only within the function. These are traditionally called “parameters” or “arguments”.
- Users may create variables at any time through the “set” block. These are traditionally called “global variables”. Blockly does not support local variables.

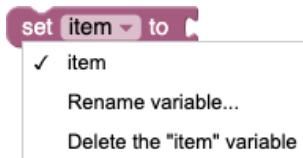
Important blocks

set The set block assigns a value to a variable, creating the variable if it doesn’t already exist.

get The get block provides the value stored in a variable, without changing it.

Dropdown menu

Clicking on a variable’s dropdown symbol (triangle) gives the following menu:



The menu provides the following options.

- the names of all existing variables defined in the program.
- `Rename variable...` changes the name of this variable wherever it appears in the program. Selecting this option opens a prompt for the new name.
- `Delete the ... variable` deletes all blocks that reference this variable wherever it appears in the program.

8.1.12 Fleet Management

Applications with more than one robot in operation at the same time in a shared environment often need some means of fleet management to ensure no crashes or deadlocks occur. PlatformPilot comes with several tools for fleet monitoring and management.

Note: Fleet management is a complex task, not exactly defined and often dependent on the exact application. These tools have been tested in a limited number of scenarios and should be considered experimental. Setting them up should be done by trained experts.

Third-party fleet managers can always use one of PlatformPilot’s command interfaces described at [Programmer’s Manual](#) which allow fine-grained monitoring and control of the platform.

Topology

The fleet manager runs on a dedicated PC that can be reached over the network by all robots. The robots then connect to the fleet manager which keeps track of the connected robots. The robots periodically send status updates ([pilot.Beacon](#)) which the fleet manager collects and aggregates. The server sends back a list of all registered robots and their basic information ([pilot.RadarScan](#)). User interfaces connected to one of the robots will use this information to show the other robots as well.

Fleet managers

There are three types of fleet managers suited for different tasks and differing on the level of control they exert.

AirControl is the most basic form of fleet management. It only manages the data exchange as described above.

Supervisor is a passive fleet manager. The robots plan and move on their own when receiving orders. The Supervisor keeps track of planned trajectories and sends back notifications about conflicts. The robots can take the information into account when planning and also pause movement when a conflict arises.

Tower is an active fleet manager. Goals are sent to the Tower which then performs scheduling, conflict resolution and path planning, before sending the complete path to the robot. It also orders the robots to stop when necessary.

All flavours are contained in the *pilot-core* installation package. See [Installation](#) for installing.

Setup

Control PC

After installation, navigate into the Pilot home folder (usually `~/pilot/`) and create a directory `config/air`. Any custom configuration should be placed in this folder. After that, start the process with one of the following commands:

```
./run_aircontrol.sh  
./run_supervisor.sh  
./run_tower.sh
```

depending on which variant you want to use.

Robot

In the Pilot home folder (usually `~/pilot/`) create a file `config/local/pilot_proxy_map+` with one of the following contents:

```
[  
    ["AirControlProxy", "127.0.0.1:6766"]  
]
```

```
[  
    ["SupervisorProxy", "127.0.0.1:6767"]  
]
```

```
[  
    ["TowerProxy", "127.0.0.1:6768"]  
]
```

where you substitute the actual IP address. PlatformPilot will connect to the remote PC at the next start.

Usage

A few things to note:

- Path planning in a fleet management setting is always done based on a *Road Map*.
- Only robots with the same grid map and the same road map are considered to be in the same environment. Conflicts between robots in different environments will not be detected.
- Robots are identified by their ID which can be found in the robot's `pilot.PilotState` message on topic `platform.pilot_state`. The ID of a robot is bound to the software process and changes on a Pilot restart.

AirControl

Only basic information exchange as described above.

Topics

- `network.robot_info_map` [*pilot.RobotInfoMap*]
- `fleet.radar_scan` [*pilot.RadarScan*]

Supervisor

When using the Supervisor, goals are sent to the robots which then plan their own path, just like in single robot mode. The robots send their position and path to the Supervisor (as part of the Beacon) which keeps track of roadmap elements that are blocked (robot is located there) and/or reserved (robot wants to use it in its path). When the Supervisor detects a conflict, i.e. a robot that tries to reserve a roadmap element that is already reserved or blocked, it sends a negative Clearance message until reservation of the complete path is possible. The robot pauses movement until it receives a positive Clearance.

The Supervisor also regularly sends an overview of blocked and reserved roadmap elements to the robots which then can use it to plan differently in order to prevent conflicts. Whether and how this happens can be configured on the robot with the config key `fleet_mode` of the *RoadMapPlanner* module.

Topics

- `network.robot_info_map` [*pilot.RobotInfoMap*]
- `fleet.radar_scan` [*pilot.RadarScan*]
- `fleet.roadmap_occupancy` [*pilot.RoadMapOccupancy*]
- `fleet.clearance` [*pilot.Clearance*]

Tower

When using the Tower, goals are sent to the Tower directly which does the path planning and only sends the finished path to the robots. The robots don't do path planning on their own.

When a goal is set for a robot, the Tower plans a path from the robot position and then tries to reserve every roadmap element on the path for this robot, taking existing reservations into account. The path is only sent to the robot, once all conflicts have been resolved and all obstacles removed. Otherwise, the Tower will periodically retry to schedule the goal.

Reservations can be non-conflicting, for example driving a road in the same direction, but most combinations are in conflict. Every discovered conflict is analyzed for resolvability. A conflict can be resolved by

- pausing the other robot before it enters the required section
- waiting until the other robot has left the required section, if it was moving anyway, or
- sending the other robot away (to a randomly selected non-conflicting station) if it is standing there.

A robot can only pause or send away another robot if it has a higher priority than the other robot. By default, all robots have priority 0. Priorities can be assigned to robots by the `set_priority` method (see below) and temporarily increased or decreased for one goal via the `pilot.goal_options_t` parameter in the `set_goal_station` method.

The total priority is the sum of robot priority and the goal priority. Robots that are sent away temporarily gain the same priority as the robot that sends them away, so that they can recursively send away robots that are in their way.

Note: A robot is only considered an obstacle if it was moved to its current location by the Tower. This means that after the initial connection, a robot has to be given at least one goal in order to properly participate in traffic management.

Topics

- `network.robot_info_map` [[pilot.RobotInfoMap](#)]
- `fleet.radar_scan` [[pilot.RadarScan](#)]
- `fleet.reservations` [[pilot.RoadmapReservation](#)]

Methods

These methods of the Tower module can be called through one of the interfaces described at [Programmer's Manual](#). The preconfigured user `neo-erp` has the permissions needed to execute them.

`void set_goal_station(Hash64 robot_id, string name, goal_options_t options, Hash64 job)`

Sets the given station name as the goal of the robot with this ID. The path will be sent to the robot once it was planned free of conflicts.

`void stop(Hash64 robot_id, goal_options_t options)`

Cancels the current goal of the robot and sends it to stop at the next map node on its path.

`void set_priority(Hash64 robot_id, int priority)`

Sets the priority of this robot as an arbitrary integer.

8.1.13 License

PlatformPilot is individually licensed under the terms agreed-upon with Neobotix GmbH.

Third-party software

PlatformPilot makes use of the following third-party software under their respective licenses. Unless stated otherwise, the software packages have not been modified by us and can be retrieved from their project homepages and/or repositories. If you are unsure, please feel free to contact us, we will be happy to assist you.

- zlib ([zlib license](#))
- libpng ([libpng license](#))
- libjpeg ([libjpeg license](#))

This software is based in part on the work of the Independent JPEG Group.

- CImg ([CeCILL-C license](#))
- LUA ([MIT License](#))

Copyright © 1994–2021 [Lua.org](#), PUC–Rio.

See below for the license text.

- url-cpp ([MIT License](#))

Copyright (c) 2016–2017 SEOMoz, Inc.

See below for the license text.

- llhttp ([MIT License](#))

Copyright Fedor Indutny, 2018.

See below for the license text.

- Open62541 ([Mozilla Public License Version 2.0](#))

- Eclipse Paho ([Eclipse Public License 2.0](#))

- ifm3d ([Apache License 2.0](#))

- A [custom fork](#) of SOEM ([GPLv2 with exceptions](#))

- Eigen3 ([Mozilla Public License 2.0](#))

- OpenCL headers / OpenCL ICD loader ([Apache License](#))

Copyright (c) 2008–2020 The Khronos Group Inc.

- Several packages by [Automy Inc.](#) ([MIT License](#))

Copyright (c) 2021 Automy Inc.

See below for the license text.

- LibreICONS ([MIT License](#))

Copyright (c) 2018 Diemen Design.

See below for the license text.

License Texts

Some of the licenses of included third-party software require explicit reproduction of the license text.

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

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.

8.1.14 Changelog

Release 1.10

Version 1.10.0 (January 2026)

- VDA-5050 fixes and improvements
- Wait signal for hardware
- Active state in joystick mode
- Improvements for Diffsteer kinematics
- open62541 v1.3.15
- Tower fleet manager
- Reshaped coordinate system handling
- ODS auto camera switch
- ABSOLUTE_ROAD orientation option
- Joystick force feedback on activation and error
- Support for Startracker
- Linear kinematics (e.g. XY-table)

Release 1.9

Version 1.9.0 (March 2025)

- Fixed map update on rotated maps
- ScanFilter downsampling
- TaskHandler dynamic autostart
- Argodrive quickstop
- VDA-5050 connector
- OpenCL cache to prevent compiler installation problems
- Joystick deadman axes
- Alternative wheel limit handling in differential kinematics
- WorldLocalization based on external input
- Support for Totalstation
- Windows version info in binaries
- Goals relative to local and target frame
- open62541 v1.3.12
- New motor types
- Fixed possible 180° degree difference to desired orientation
- Allow setting driving orientation for differential platforms

- Changed home folder structure on Windows
- Support for Ubuntu 24.04
- Ifm3d support by default
- EtherCAT support
- Diffsteer kinematics
- Custom road segment cost

Release 1.8

Version 1.8.0 (March 2024)

- Refined support for ROX robots
- Added import / convert tool for grid maps
- Added MQTT client communication
- Fixed OpenCL initialization bug on some platforms
- Fixed append_goal functionality
- Support for IFM ODS
- Support for Pilz Psenscan

Release 1.7

Version 1.7.0 (October 2023)

- Support for ROX robots
- Added SafetyInterface and SafetyState
- Software quick stop
- Changed color of dynamic area
- Added battery watchdog tool
- Added charger coordinate frame
- Support for SICK Visionary cameras
- Support for RelayBoardV3
- Application level support for FlexiSoft
- Fixes for Elmo Gold motor controllers
- Fixed ValueRank of OPCUA variables
- Improvements in CANopen implementation
- Footprint can be changed without reinitialization
- Updated CImg library to 3.2.6
- LocalPlanner target pose rotation fix

Release 1.6

Version 1.6.0 (May 2023)

- New corporate design
- Support for ArgoDrives
- Extended BatteryState data structure
- Documentation available through internal web server
- OPCUA: writable topic variables
- OPCUA: fixed authentication configuration
- OPCUA: certificate issuer and revocation lists
- Updated open62541 to v1.3.6
- Added PCAN error messages (Windows)
- Integrated key pad into TaskHandler
- New Lua command `move_towards`
- Docking based on shape matching
- Shape matching
- Fixed Microscan module error on Nanoscan
- Bicycle kinematics
- Support for Ubuntu 22.04

Release 1.5

Version 1.5.0 (August 2022)

- Added example maps
- OPCUA:
 - open62541 update to v1.3.2
 - `OPC_UA_Server` custom host name and variables
 - open62541 as shared library
 - added subscriptions to `OPC_UA_Proxy`
 - more settings for opcua server and proxy
 - `OPC_UA_Proxy` proper disconnect on error and client renewal
 - Server authentication
- Added Dockerfile creation
- `GlobalCostMap` takes additional cost points
- LUA functions and blocks for map switching
- Implemented map storage in `MapServer`

- Added some useful polygon methods
- Added areas to map match
- Included radar scan into local costmap
- Added basic aircontrol binary
- Added `get_pilot_version` method to `PilotServer`
- Added roadmap adjacency hash
- Added radar scan to virtual scan
- Added default configuration for MP-500
- Setting `ulimit` automatically
- Node for RelayboardV3
- `HttpServer` blocking fix
- Added shortcut to access the documentation
- Added remote joystick tool `pilot_joystick`
- `TaskHandler` fixed publishing volatile execution state
- Omnidrive kinematics with arbitrary number of drives
- Splitted `PowerState` and `RelayBoardData` from `SystemState`
- Full support for ARM64 (e.g. Jetson Nano)
- Dedicated driver module for ELMO motors
- Fleet manager `Tower` implemented
- GlobalPlanner using local cost map overlay by default now
- HybridPlanner can skip road map with `goal_options_t::planner_mode == FREE_ROAMING`
- Terminal output on Windows can use ANSI codes or Win API
- RelayBoardNode retry charging in case no charger detected
- added RelayBoardNode `enable_charge_cycling` option
- LocalPlanner no more goal shortcuts

Release 1.4

Version 1.4.0 (October 2021)

- `LocalPlanner pause()` fix for differential to keep on path
- `TaskHandler execute_*` () functions now have *blocking* option
- added `TaskHandler set_timer_ms` functionality
- `CanNode` automatic re-init in case of persistent motor timeouts
- `TaskHandler` generates random `jobid` if not specified
- `HttpProxy` fix for missing `LogMsg`, `ModuleInfo` and `Event` topics
- `CanNode` fix for timeouts due to blocking publishes

- *CanNode* fix for timeout feedback loop in homeing procedure
- *SerialPort* fix for blocking publish
- added *PlatformInterface* `read_analog_input()` + `read_digital_input()` functions
- added *TaskHandler* `read_analog_input()` + `read_digital_input()` functions
- added *TaskEditor* `read_analog_input` + `read_digital_input` blocks
- added *TaskEditor* map / object support
- fixed *TaskEditor* display regression when switching views
- Added *RoadMapPlanner* `find_closest_station()`
- Added *LUA* `find_closest_station_name()`
- Added *TaskEditor* `find_closest_station` block (which returns station name)

Release 1.3

Version 1.3.0 (August 2021)

- Added OPC-UA Client/Server certificate support
- Added *LUA* functions and corresponding *TaskEditor* blocks:
 - `is_charging()`
 - `get_battery_remaining()`
 - `reset_motors()`
- Added `Event::level` to allow sorting by low/high level events.
- `wait_for_joystick()` waits for activation now
- `LaserCalibration` now also calibrates sensor X/Y position
- Added *RealSense2* support
- LFP battery charge auto cycling between 90 and 100 % (see `RelayBoardNode::battery_type` config)
- Added `SystemState::is_charging`

Release 1.2

Version 1.2.0 (June 2021)

- New `HttpServer` implementation, no longer using *libmicrohttpd*.
 - Supports *Server-Sent-Events (SSE)*, via `/api/stream/...`
 - *Deflate* response compression (multi-threaded)
 - Asynchronous chunked transfers
- `localization.status` topic is also being recorded now.
- New `LocalPlanner` functions: `await_goal()`, `await_goal_ex(...)`, `cancel_goal_await()`
- `LocalPlanner::pause()` now has an optional `bool em_stop` parameter to enable emergency stopping.

- OPC-UA write variable support via `Proxy::write_variable()` and `Proxy::write_object_variable()`.
- TaskHandler now supports script parameters via `execute_file()` and `execute_program()` which are passed on to the `main(...)` function in *LUA*.
- Sending a new goal while driving works as expected now, planners wait for platform to stop before planning new path.
- Kinematics_CanNode fix for CAN bus initialization, now attempts to re-initialize until successful.
- New TaskHandler functions: `get_time_sec()`, `get_time_millis()`, `get_time_micros()`
- HttpSession now contains `session_timeout` as well
- Additional incident reporting for *RelayBoard* and *MPO-700* homing.
- SickMicroscan3 fix for reflector detection
- OPC-UA Proxy authentication support via `username` and `password` configs.
- TaskHandler scripts have REQUEST permission now
- USER (neo-user) has `INTERVENE_SCRIPT` permissions now

Release 1.1

Version 1.1.0 (May 2021)

8.2 Programmer's Manual

This section describes the software interfaces provided by a PlatformPilot instance. Use one or more of them to connect a custom application and exchange data and/or execute commands. Which interface is most suitable depends on the needs of the application.

8.2.1 VNX Interface

The native VNX binary protocol allows to communicate with the *PlatformPilot* in a C++ application, either via TCP/IP, UNIX socket or a direct intra-process connection. Please note that using this protocol requires your C++ application to link against the Pilot and VNX libraries supplied in an installation (see below).

The communication is provided via the `vnx.Server` and `vnx.Proxy` modules.

Server

To enable a `vnx.Server` create a config file `config/local/vnx_server_map`:

```
[  
  ["TcpServer_1", "0.0.0.0:1234"],  
  ["UnixServer_1", "/tmp/mysocket.sock"]  
]
```

This will start two VNX servers listening on TCP/IP address `0.0.0.0:1234` and UNIX socket `/tmp/mysocket.sock` respectively. By default a VNX server is listening on TCP/IP address `0.0.0.0:5555`, which requires a login for most functionality.

To enable user authentication, which is recommended for TCP/IP servers, create a config file config/local/TcpServer_1.json:

```
{  
    "use_authentication": true,  
    "default_access": "OBSERVER"  
}
```

This will require a login to gain more than the default_access permissions. If use_authentication is set to false any user has full permissions and no login is necessary.

Proxy

To run a *vnx.Proxy* which connects to another VNX server create a config file vnx_proxy_map which is used by your process:

```
[  
    ["Proxy_1", "127.0.0.1:5555"],  
    ["Proxy_2", "/tmp/mysocket.sock"]  
]
```

The above proxies will be available under the module names Proxy_1 and Proxy_2. To import / export topics and forward services create a config file for each proxy. For example Proxy_1.json:

```
{  
    "import_list": [  
        "input",  
        "sensors.raw_data",  
        "platform.system_state",  
        "platform.battery_state",  
        "platform.emergency_state",  
        "kinematics.drive_state"  
    ],  
    "export_list": [  
        "platform.drive_cmd",  
    ],  
    "forward_list": [  
        "PlatformInterface"  
    ],  
    "time_sync": true  
}
```

To gain required permissions a login needs to be performed at runtime:

```
#include <vnx/vnx.h>  
#include <vnx/ProxyClient.hxx>  
  
std::string user;  
if(vnx::read_config("user", user)) {  
    vnx::ProxyClient proxy("Proxy");  
    proxy.login(user, vnx::input_password("Password: "));  
}
```

Clients

A *Client* is a way to communicate with a module via (remote) procedure calls.

Synchronization

There are synchronous clients and asynchronous clients. A method call on a synchronous client blocks until the result arrives. A method call on an asynchronous client returns immediately and gives you the possibility to supply callback functions that notify you about the result.

Specific Clients

Every module has a synchronous and an asynchronous client. If the module class is called `MyModule`, the clients are called `MyModuleClient` and `MyModuleAsyncClient`. In order to instantiate them, you also need to know the runtime name of the module (not the name of the class).

Here is an example on how to use the clients for `MyModule` that is running under the name `MyModule_name` to call its method `void do_something(int, string)`.

```
#include <package/MyModuleClient.hxx>
#include <package/MyModuleAsyncClient.hxx>

using namespace package;

MyModuleClient sync_client("MyModule_name");
sync_client.do_something(42, "hello");

std::shared_ptr<MyModuleAsyncClient> async_client =
    std::make_shared<MyModuleAsyncClient>("MyModule_name");
add_async_client(async_client);
async_client->do_something(42, "hello",
    [] (void) {
        std::cout << "OK" << std::endl;
    },
    [] (const vnx::exception& ex) {
        std::cout << "FAIL: " << ex.what() << std::endl;
    }
);
```

The callback functions for asynchronous clients are of the forms `void(const T&)` (where `T` is the return type, or `void`) and `void(const vnx::exception&)` respectively. Supplying either of them is optional and defaults to a null function.

Generic Clients

With a generic client, you can communicate with any module. The handling is similar to the specific case, only this time you call methods by their name and supply the parameters as a `vnx.Object`. The names of the method and the parameters have to match the interface definition.

```
#include <vnx/vnx.h>

vnx::Object args;
args["number"] = 42;
args["message"] = "hello");

vnx::GenericClient sync_client("MyModule_name");
std::shared_ptr<const vnx::Value> ret = sync_client.call("do_something", args);
std::cout << "OK: " << ret->get_field_by_index(0) << std::endl;
```

(continues on next page)

(continued from previous page)

```
std::shared_ptr<vnx::GenericAsyncClient> async_client =
    std::make_shared<vnx::GenericAsyncClient>("MyModule_name");
add_async_client(async_client);
async_client->call("do_something", args,
    [] (std::shared_ptr<const vnx::Value> ret) {
        std::cout << "OK: " << ret->get_field_by_index(0) << std::endl;
    },
    [] (const vnx::exception& ex) {
        std::cout << "FAIL: " << ex.what() << std::endl;
    }
);
```

The callback functions are of the form `void(std::shared_ptr<const vnx::Value>)` and `void(const vnx::exception&)` respectively. Supplying either of them is optional and defaults to a null function.

Compiling

Example on how to compile your C++ application:

```
g++ -std=c++11 application.cpp -I /opt/neobotix/pilot-core/include \
    -L /opt/neobotix/pilot-core/lib -lpilot_core -lvnx_base
```

`/opt/neobotix/pilot-core` can also be replaced by `/opt/neobotix/pilot-gtkgui`, if you have the *GTK GUI* installed.

Example

Connecting to the *PlatformPilot* and sending goals to *HybridPlanner*.

config/default/vnx_proxy_map:

```
[["Proxy", ""]]
```

We do not specify an address in the config above, since we want to give a custom address on the command line later.

config/default/Proxy.json:

```
{"forward_list": ["HybridPlanner"]}
```

move_to_stations.cpp:

```
#include <vnx/vnx.h>
#include <vnx/ProxyClient.hxx>
#include <pilot/HybridPlannerClient.hxx>
using namespace pilot;

int main(int argc, char** argv)
{
    std::map<std::string, std::string> options;
    options["u"] = "user";
    vnx::init("move_to_stations", argc, argv, options);

    std::string user;
    if(vnx::read_config("user", user)) {
```

(continues on next page)

(continued from previous page)

```

    vnx::ProxyClient proxy("Proxy");
    proxy.login(user, vnx::input_password("Password: "));
}

HybridPlannerClient planner("HybridPlanner");

goal_options_t goal_options;
goal_options.max_velocity = 0.5;
goal_options.drive_mode = drive_mode_e::STRICT_PATH_FOLLOW;

while(vnx::do_run()) {
    try {
        planner.move_to_station("Station1");
        vnx::log_info() << "Station1 has been reached!";

        planner.move_to_station("Station2", goal_options);
        vnx::log_info() << "Station2 has been reached!";

        const vnx::Hash64 job_id = vnx::Hash64::rand();
        planner.move_to_station("Station3", {}, job_id);
        vnx::log_info() << "Station3 has been reached! (job " << job_id << ")";
    }
    catch(const std::exception& ex) {
        vnx::log_error() << "move_to_station() failed with: " << ex.what();
        break;
    }
}
vnx::close();
}

```

Compiling:

```

g++ -std=c++11 -o move_to_stations move_to_stations.cpp \
    -I /opt/neobotix/pilot-core/include \
    -L /opt/neobotix/pilot-core/lib -lpilot_core -lvnx_base

```

Running:

```

$ ./move_to_stations -c config/default/ --Proxy.address ~/pilot/.pilot_main.sock

Proxy.address = "/home/neobotix/pilot/.pilot_main.sock"
Proxy.forward_list = ["HybridPlanner"]
config = ["config/default/"]
vnx_proxy_map = [{"Proxy", ""}]
[Proxy] INFO: enable_forward('HybridPlanner', 100, 1000)
[Proxy] INFO: Connected to /home/neobotix/pilot/.pilot_main.sock
[move_to_stations] INFO: Station1 has been reached!
[move_to_stations] INFO: Station2 has been reached!
[move_to_stations] INFO: Station3 has been reached! (job 13373535902967275021)
...

```

Connecting to `.pilot_main.sock` avoids having to login with a username and password to be able to move the platform.

8.2.2 JSON-RPC Interface

This interface uses the [JSON-RPC protocol](#) in version 2.0.

The corresponding proxy module `vnx.JRPC_Proxy` is similar to the `vnx.Proxy` and mostly follows the same protocol, only it wraps its messages into appropriate JSON objects.

Message format

A message in our setting of the JSON RPC protocol may look like this:

```
{  
    "jsonrpc": "2.0",  
    "id": "123some_id456",  
    "method": "MyModule.some_method",  
    "params": {  
        "first": "Hello",  
        "second": 4321  
    }  
}
```

Some remarks:

- The `id` is optional. If you provide it, you will get a response message, otherwise you will get nothing. This includes error messages!
- `method` consists of a module name and its method name, separated by a dot. If you omit the module name (but keep the dot), the request is directed at the proxy module itself.

The proxy module has a method `select_service` to assign a module name for the rest of the connection:

```
{  
    "jsonrpc": "2.0",  
    "id": "123some_id456",  
    "method": ".select_service",  
    "params": {  
        "service_name": "MyModule"  
    }  
}
```

Now you can omit the module name (and the dot) in future requests.

- `params` is an object indexed by the names of the parameters as defined in the interface of the module. You can also supply a list instead, although that is not encouraged.

Server

To enable a `vnx.JRPC_Server` create a config file `config/local/vnx/jrpc_server_map`:

```
[  
    ["JRPC_Server", "0.0.0.0:5556"],  
    ...  
]
```

This will start a JSON RPC server that listens on the TCP/IP address `0.0.0.0:5556`.

Connecting

Open a TCP connection to the port of the server. You should be greeted with a welcome message (method `.on_remote_connect`) that you can safely ignore.

To authenticate use the `on_login` method of the proxy:

```
{
    "jsonrpc": "2.0",
    "method": ".on_login",
    "params": {
        "name": "username",
        "password": "23f/*...*/995"
    }
}
```

where `password` is a SHA-256 hash of the actual password. On success you should receive a call to `.on_remote_login` providing some information on your established session.

Return Values

Note: You will only receive return values if you supply the `id` field in your request.

Return values are wrapped inside a success message with the `result` field holding the actual return value. A successful call to `RoadMapPlanner.find_station` may look like this:

```
{
    "__type": "vnx.JRPC_Success",
    "jsonrpc": "2.0",
    "id": 5566,
    "result": {
        "__type": "pilot.MapStation",
        "drive_flags": [],
        "goal_tolerance": {"__type": "pilot.vector_3f_param_t", "type": "DEFAULT", "x": {"__type": "pilot.float_param_t", "type": "DEFAULT", "value": 0}, "y": {"__type": "pilot.float_param_t", "type": "DEFAULT", "value": 0}, "z": {"__type": "pilot.float_param_t", "type": "DEFAULT", "value": 0}},
        "goal_tune_time": {"__type": "pilot.float_param_t", "type": "DEFAULT", "value": 0},
        "group": "",
        "id": 120,
        "name": "Station120",
        "orientation": -3.13343,
        "position": [-0.611983, 2.81928],
        "profiles": [],
        "tolerance": {"__type": "pilot.float_param_t", "type": "DEFAULT", "value": 0}
    }
}
```

The result of calling a `void` method has a null return value:

```
{
    "__type": "vnx.JRPC_Success",
    "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```

    "id": 43,
    "result": null
}

```

Errors

Note: You will only receive errors if you supply the `id` field in your request.

If your request failed, you will get an error response. It contains an `error` field with an object giving an error code, a short message and the actual error object.

When the method you called throws an exception, the error will look like this:

```

{
    "__type": "vnx.JRPC_Failure",
    "jsonrpc": "2.0",
    "id": 43,
    "error": {
        "__type": "vnx.JRPC_Error",
        "code": 500,
        "message": "roadmap station does not exist",
        "data": {
            "__type": "vnx.InternalError",
            "what": "roadmap station does not exist"
        }
    }
}

```

This shows an error that you get if you lack a permission to do something:

```

{
    "__type": "vnx.JRPC_Failure",
    "jsonrpc": "2.0",
    "id": 42,
    "error": {
        "__type": "vnx.JRPC_Error",
        "code": 403,
        "message": "permission denied (pilot.permission_e.MOVE)",
        "data": {
            "__type": "vnx.PermissionDenied",
            "what": "permission denied (pilot.permission_e.MOVE)",
            "dst_mac": 0,
            "method": "pilot.HybridPlanner.set_goal_station",
            "permission": "pilot.permission_e.MOVE"
        }
    }
}

```

See [vnx.JRPC_Error](#) for a list of error codes.

Topics

As with the VNX protocol, subscribing to a topic is done with the `enable_export` method of the proxy:

```
{
    "jsonrpc": "2.0",
    "method": ".enable_export",
    "params": {
        "topic_name": "some.topic"
    }
}
```

Received samples will have the following form:

```
{
    "jsonrpc": "2.0",
    "method": "!some.topic",
    "params": {
        "seq_num": 42,
        "value": {"__type": "some.Datatype", "time": 1600000023, "some_more": null}
    }
}
```

Notice that we are relaxing the notion of a “method” here: The initial ! makes clear that it actually refers to a topic and the parameters really just hold the value.

Using the same format in the other direction, you can also publish samples on the named topic (given sufficient permissions).

Example

The following shows a manual interaction with the JSON-RPC server, using the `ncat` command line utility. For readability, messages to the server are marked with [-->], answers are marked with [<--].

In this session, the user connects, tries to set a goal with insufficient permissions, logs in, successfully sets a goal station and finally tries to set a goal station that does not exist.

```
$ ncat localhost 5556

[ <-- ] {"__type": "vnx.JRPC_Notification", "jsonrpc": "2.0", "method": ".on_remote_connect", "params": {"__type": "vnx.ProxyInterface.on_remote_connect", "process_id": 11854018157562744543} }

[ --> ] {"jsonrpc": "2.0", "id": "Sogehtsnicht", "method": "HybridPlanner.set_goal_station", "params": {"name": "Station2"}}

[ <-- ] {"__type": "vnx.JRPC_Failure", "jsonrpc": "2.0", "id": "Sogehtsnicht", "error": {"__type": "vnx.JRPC_Error", "code": 403, "message": "permission denied (pilot.permission_e.MOVE)"}, "data": {"__type": "vnx.PermissionDenied", "what": "permission_denied (pilot.permission_e.MOVE)"}, "dst_mac": 18102426972873016303, "method": "pilot.HybridPlanner.set_goal_station", "permission": "pilot.permission_e.MOVE"}}

[ --> ] {"jsonrpc": "2.0", "id": "YOLO", "method": ".on_login", "params": {"name": "neo-user", "password": "460900ad5480f4904557f26d1567881d5755008540b40cf8bf3d22229f3ed562"}}

[ <-- ] {"__type": "vnx.JRPC_Notification", "jsonrpc": "2.0", "method": ".on_remote_login", "params": {"__type": "vnx.ProxyInterface.on_remote_login", "remote_session": {"__type": "vnx.Session", "id": 3553960942762058184, "login_time": 1625461616530749, "permissions": ["pilot.permission_e.CHARGE", "pilot.permission_e.EXECUTE_SCRIPT", "pilot.permission_e.INITIALIZE", "pilot.permission_e.INTERVENE_SCRIPT", "pilot.permission_e.MOVE", "pilot.permission_e.RECORD_DATA", "vnx.addons.permission_e.FILE_DOWNLOAD", "vnx.addons.permission_e.READ_DIRECTORY", "vnx.permission_e.CONST_REQUEST", "vnx.permission_e.READ_CONFIG", "vnx.permission_e.TIME_SYNC", "vnx.permission_e.VIEW"]}, "user": "neo-user"}}

(continues on next page)
```

(continued from previous page)

```
[ <-- ] { "__type": "vnx.JRPC_Success", "jsonrpc": "2.0", "id": "YOLO", "result": null}

[ --> ] { "jsonrpc": "2.0", "id": 42, "method": "HybridPlanner.set_goal_station",
  ↵"params": { "name": "Station2" } }
[ <-- ] { "__type": "vnx.JRPC_Success", "jsonrpc": "2.0", "id": 42, "result": null}

[ --> ] { "jsonrpc": "2.0", "id": 43, "method": "HybridPlanner.set_goal_station",
  ↵"params": { "name": "Gibtsdochgarnicht" } }
[ <-- ] { "__type": "vnx.JRPC_Failure", "jsonrpc": "2.0", "id": 43, "error": { "__type": "vnx.JRPC_Error", "code": 500, "message": "roadmap station does not exist", "data": { "__type": "vnx.Exception", "what": "roadmap station does not exist" } } }

^C
```

8.2.3 HTTP Interface

The HTTP REST API provides access to almost any functionality of the *PlatformPilot* via the HTTP protocol.

The HTTP server is enabled by default on port 8888, however it can be disabled by setting `enable_http_server` to `false`.

The REST API is available on the path `http://localhost:8888/api/`. (replace `localhost` with your target machine)

See also [HttpProxy](#).

Login

Some functionality requires special permissions, see [User Management](#).

To gain necessary permissions you need to login to the HTTP server as follows:

```
curl -I "http://localhost:8888/server/login?user=neo-user&passwd_plain=neobotix"
HTTP/1.1 200 OK
...
Set-Cookie: hsid=7ac1b14c66b6f323-0000d026d3eff249-2d4c9774cd3accb8; Path=/; Max-
  ↵Age=86400; SameSite=Strict;
```

The response will contain a session cookie which can be used as follows:

```
curl -H "Cookie: hsid=7ac1b14c66b6f323-0000d026d3eff249-2d4c9774cd3accb8" http://
  ↵localhost:8888/api/request/...
```

Services

All modules of the *PlatformPilot* are available via the path `/api/request/`.

To get an overview of the available modules:

```
curl http://localhost:8888/api/request/
[ "GlobalCostMap/", "GlobalPlanner/", "GridLocalization/", "HybridPlanner/", ... ]
```

The available methods of a module can be queried as follows:

```
curl http://localhost:8888/api/request/HybridPlanner/
["append_goal", "append_goal_position", "append_goal_positions", "append_goal_station
 ↵", ...]
```

A method can be called as follows:

```
curl -X POST http://localhost:8888/api/request/PilotServer/get_state
{"__type": "pilot.PilotState", ...}

curl -H "Cookie: ..." -X POST -d '{"name": "Station1"}' \
    http://localhost:8888/api/request/HybridPlanner/set_goal_station
```

The parameters of a function are supplied as a JSON object via POST data.

You may need to be logged in to access certain functions, see above.

Topics

Almost all topics of the *PlatformPilot* are available via the path `/api/topic/`.

To get an overview of the available topics:

```
curl http://localhost:8888/api/topic/
["input/", "local_planner/", "navigation/", "platform/", "sensors/", "task_handler/",
 ↵"tf/", "tfd/", "vnx/"]

curl http://localhost:8888/api/topic/platform/
["info", "odometry", "pilot_state", "system_state"]
```

To get the latest sample data of a topic:

```
curl http://localhost:8888/api/topic/platform/odometry
{"__type": "pilot.Odometry", "time": 1613656049067588, ...}
```

To get a tree of the lastest sample data of a domain:

```
curl http://localhost:8888/api/topic/platform
{"info": {"__type": "pilot.PlatformInfo", ...}, ...}
```

To publish a data sample on a topic:

```
curl -H "Cookie: ..." -X POST -d '{"topic": "test.topic", "sample": {"__type": "pilot.
 ↵Pose2D", ...}}' \
    http://localhost:8888/api/request/HttpProxy/publish
```

Note that a `__type` field needs to be specified containing the type name of the sample, such as `pilot.Pose2D` for example.

Configuration

The current configuration tree can be viewed via the path `/api/config/`.

To access protected configuration values, a special permission `PROTECTED_CONFIG` is required, see `vnx.permission_e`.

To get an overview of the available config options:

```
curl http://localhost:8888/api/config/
["GlobalCostMap/", "GlobalPlanner/", "GridLocalization/", "GridMapping/", ...]

curl http://localhost:8888/api/config/GridLocalization/
["broadcast_tf", "confidence_gain", "constrain_threshold", "constrain_threshold_yaw",
 ↴ "gain_factor", ...]
```

To query a specific config value:

```
curl http://localhost:8888/api/config/GridLocalization/gain_factor
0.01
```

To query a sub-tree of the configuration:

```
curl http://localhost:8888/api/config/GridLocalization
{"broadcast_tf": true, "confidence_gain": 0.01, "constrain_threshold": 0.1,
 ↴ "constrain_threshold_yaw": 0.2, "gain_factor": 0.01, ...}
```

To query the entire configuration tree:

```
curl http://localhost:8888/api/config
{...}
```

Log

The terminal output log messages are available via the path `/api/log/`.

Each message is of type `vnx.LogMsg`.

To get all errors which occurred since startup:

```
curl http://localhost:8888/api/log/errors
[...]
```

To get all recent messages:

```
curl http://localhost:8888/api/log/recent
{"1": [...], "2": [...], "3": [...], "4": [...]}
```

To get recent warnings only:

```
curl http://localhost:8888/api/log/recent/2
[...]
```

The log levels are as follows:

- 1 = ERROR
- 2 = WARN
- 3 = INFO
- 4 = DEBUG

Log messages are also written to disk and can be accessed via `http://example:8888/user/data/logs/`.

Events

A history of events is available via the path `/api/events/`.

Each entry is of type `pilot.Event` or any of it's derived types such as `pilot.Incident`.

To get a list of recent events:

```
curl http://localhost:8888/api/events/recent
[...]
```

To get recent error events only:

```
curl http://localhost:8888/api/events/errors
[...]
```

The latest events are at the end of the respective list.

Views

For certain data types there are special views available via the path `/api/view/`.

To get a cost map (see `pilot.CostMapData`) as a PNG image:

```
curl "http://localhost:8888/api/view/cost_map?topic=navigation.local_cost_map&
˓→color=true&alpha=128" > local_cost_map.png
```

To get a grid map (see `pilot.OccupancyMapData`) as a PNG image:

```
curl "http://localhost:8888/api/view/occupancy_map?topic=navigation.grid_map&alpha=255
˓→" > grid_map.png
```

8.2.4 OPC-UA Interface

The OPC-UA interface is provided by the `vnx.opc_ua.Server` and `vnx.opc_ua.Proxy` modules.

They allow to access internal modules via OPC-UA method calls, as well as call methods on another OPC-UA server via a `Lua Script` for example.

Server

To enable the `vnx.opc_ua.Server` set the following config option:

```
cd ~/pilot
echo true > config/local/enable_opcua_server
```

The server will listen on the address `opc.tcp://0.0.0.0:4840`. The port 4840 is the default port for OPC-UA but you can also configure a different port.

The server module is called `OPC_UA_Server`, so it can be configured *as usual* by a config file `config/local/OPC_UA_Server.json`.

Exports

- Pilot modules and their methods are advertised by the server as OPC-UA services. The methods can be called according to the OPC-UA specification. The list of exported services is given by the config key `export_services`.
- Pilot topics are advertised as OPC-UA variables and continually updated whenever a new value is published. The topic variables are writable. An external write operation to a variable results in a publish on the corresponding topic, if the user has sufficient privileges. The list of exported topics is given by the config key `export_topics`.

To expand any of these lists, put a snippet like this in the configuration file:

```
{  
    "export_services+": [  
        "AnotherModule",  
        ...  
    ],  
    "export_topics+": [  
        "another.topic",  
        ...  
    ]  
}
```

In order to *replace* any of the lists, omit the respective + character.

Security

The OPC-UA server supports encrypted communication. To enable it, you have to create or obtain a certificate and put the path in the configuration. You should also consider revisiting the key `security_policies` and adjust it to your needs. For accepting clients, the server uses trust list verification. Client certificates that should be accepted must be given by the the `trust_list` parameter.

```
{  
    "certificate_file": "/some/path/to/server_cert.der",  
    "private_key_file": "/some/path/to/server_key.der",  
    "security_policies": [  
        "NONE",  
        "BASIC_256_SHA_256",  
        "AES_128_SHA_256_RSA_OAEP"  
    ],  
    "trust_list": [  
        "/some/path/to/client_cert.der"  
    ]  
}
```

Proxy

To run a `vnx.opc_ua.Proxy` which connects to another OPC-UA server create the following config file `config/local/opcua_proxy_map`:

```
[  
    ["OPC_UA_Proxy_1", "opc.tcp://127.0.0.1:4840"],  
    ...  
]
```

The above proxy will be available under the module name `OPC_UA_Proxy_1`, see [Lua Script](#) for examples on how to use it.

As [usual](#), the module can be configured by a config file `config/local/OPC_UA_Proxy_1.json`.

Security

You can configure encrypted communication of the Proxy module. First obtain or generate a certificate and put the path in the configuration file. You should also consider choosing a reasonable value for the key `security_mode`. For connection to a server, the Proxy uses trust list verification. Server certificates that should be accepted must be given by the `trust_list` parameter.

```
{
    "certificate_file": "/some/path/to/client_cert.der",
    "private_key_file": "/some/path/to/client_key.der",
    "security_mode": "SIGN_AND_ENCRYPT",
    "trust_list": [
        "/some/path/to/server_cert.der"
    ]
}
```

Certificates

To create self-signed certificates for the server and the proxy side, use the script provided by the open62541 project [here](#) as follows:

```
python create_self-signed.py -k 4096 -c server -u urn:open62541.server.application
python create_self-signed.py -k 4096 -c client -u urn:open62541.client.application
```

The `-u` switch sets the application URI that must match the one configured in the module. The values given here are the defaults.

Data Types

Primitive data types are directly mapped to their OPC-UA counter parts, such as `int` to `INT32`, `float` to `FLOAT`, etc. `string` is directly mapped to a OPC-UA `STRING`. Arrays of said types are directly mapped to OPC-UA arrays.

Anything else will be converted to JSON and transported via a `LocalizedText` object, with the `locale` set to `JSON`.

8.2.5 MQTT Interface

[MQTT](#) is a publish-subscribe protocol to transport arbitrary data between clients. MQTT clients connect to a server (called MQTT broker) and publish and/or subscribe to topics, identified by an (almost) arbitrary name. Topics are often structured in a hierarchy using slashes, for example `neobotix/platform/mpo-700-1/navigation/map_pose`.

PlatformPilot can be set up to act as an MQTT client to an MQTT broker, translating messages from and to internal topics. All sent messages will be JSON objects, except possibly the *Last-Will* message. All messages that should be received also have to be JSON objects.

Client

In your *configuration* folder add or append a file named `mqtt_proxy_list` to contain a list of the MQTT proxies that should be started. Each one will be one MQTT client (usually you only need one per broker).

```
["MQTT_Proxy_1", "MQTT_Proxy_2", ...]
```

The individual options can be set in the corresponding configuration file, e.g. in `MQTT_Proxy_1.json`. The only necessary setting is the broker address, but you probably also want to specify the client ID and the exported topics. Here is an example configuration:

```
{
    "address": "tcp://10.1.30.42:1883",
    "client_id": "nbx-pp-1",
    "export_map": [
        ["platform.system_state", "neobotix/<robot-name>/platform/system_state"],
        ["platform.emergency_state", "neobotix/<robot-name>/platform/emergency_state"]
    ],
    "export_map_ex": [
        ["platform.info", {
            "topic": "neobotix/<robot-name>/platform/info",
            "retained": true
        }]
    ],
    "last_will": {
        "topic": "neobotix/<robot-name>/disconnect",
        "message": "Apparently, an unexpected disconnect happened..."
    }
}
```

Find all configuration options at `vnx.mqtt.Proxy`.

8.2.6 VDA 5050

VDA 5050 is an open standard and protocol for communication between AGVs and a central fleet management service. The fleet management server sends requests and commands to the AGV while the AGV sends status updates and vehicle information back. Message exchange is done via MQTT. PlatformPilot supports VDA 5050 in version 2.0.0.

Enable

There are two things to enable: The *Connector* implementing the VDA 5050 command protocol and an *MQTT proxy* to send and receive MQTT messages.

VDA 5050 Connector

In your *configuration* folder add a file called `enable_vda5050_connector` with the content `true`.

MQTT Proxy

For the MQTT communication, an MQTT proxy module must be started to translate between VDA 5050 messages and internal Pilot communication. See also [MQTT Interface](#).

In your *configuration* folder add or append a file named `mqtt_proxy_list` with content

```
[ "MQTT_Proxy_VDA_5050" ]
```

This will start the module. Configuration of the module is done in the file `MQTT_Proxy_VDA_5050.json`. Most of the configuration is inherited from the default but there are a few fields that you will probably have to change. It is recommended to start with the following file.

```
{
    "address": "10.1.30.42:1883",
    "client_id": "pilot-vda5050",
    "topic_prefix": "uagv/v2/Neobotix/123456789/",
    "last_will": {
        "message": "{\"headerId\": 0, \"timestamp\": 0, \"version\": \"2.0.0\",
        \"manufacturer\": \"Neobotix\", \"serialNumber\": \"123456789\", \
        \"connectionState\": \"CONNECTIONBROKEN\"}"
    }
}
```

- Most important, change the `address` to the actual address of the MQTT broker used by this VDA 5050 server.
- The `client_id` is the name of the MQTT client and should be different for each platform. For example, you can append the (unique) name of the platform which has been configured in the `platform.json` file and gets displayed in the graphical user interface.
- The `topic_prefix` is a prefix for all imported and exported MQTT topics. The one given here follows the naming scheme proposed in the standard but you should at least change the `123456789` to the actual serial number of your platform which has been configured in the `platform.json` file and gets displayed in the graphical user interface.
- Change the serial number in the `last_will` message as well.

8.2.7 Lua Script

A Lua script that can be executed by the `TaskHandler` module must at least have a main function. The default name is `main`, so the minimal script is:

```
function main()
    -- do something
end
```

Command Reference

The following commands are available to Lua scripts through the programming interface.

Commands denoted `void` return `true` or `false` depending on whether the command succeeded. Commands with a return type either return a value of that type or `nil` if the command failed.

See [https://en.wikipedia.org/wiki/Lua_\(programming_language\)](https://en.wikipedia.org/wiki/Lua_(programming_language)) for more information regarding the Lua script language itself.

Movement

The commands in this section take an optional parameter of type `pilot.goal_options_t`. If not given, the default values are used. Some movements can also be given a job id as a 64 bit `vnx.Hash64`. If not given, a random job id is generated.

```
void move_to_station (string name, goal_options_t options, Hash64 job)
    Moves the platform to the station named name.

void move_to_position (Pose2D position, goal_options_t options, Hash64 job)
    Moves the platform to the given position in the map. See pilot.Pose2D.

void move_to (MapStation station, goal_options_t options, Hash64 job)
    Moves the platform to the place described by station. It may or may not be a station of the Road Map. See pilot.MapStation.

void move (double dx, double dy, double dr, goal_options_t options, string frame = "base_link")
    Moves the platform in the given direction, as seen from the frame's point of view (defaults to base_link, i.e. the platform). See Coordinate Systems. dx and dy are given in meters, dr in rads. The new goal must be within reach, with a maximum distance depending on configuration, but usually around 1 m.

void move_towards (double dx, double dy, double dr, goal_options_t options, string frame = "base_link")
    Same as move (...) but the goal does not have to be in reach but is treated the same as a goal point given by an absolute position. In turn, the calculated path is not necessarily a straight line, depending on obstacles and roadmap constraints on the way.

void cancel_goal ()
    Cancels the current goal (if any) which causes the platform to stop moving and the corresponding move command to fail. This command can be useful in an event handler.
```

Maps

Grid maps and road maps are uniquely identified by a [pilot.map_info_t](#) object.

```
void switch_grid_map (map_info_t map_info)
    Changes the current grid map to the one described by map_info. The map needs to have been uploaded to the platform.

void switch_road_map (map_info_t map_info)
    Changes the current road map to the one described by map_info. The map needs to have been uploaded to the platform.
```

User Input

```
int wait_for_joystick ()
    Waits until any button on the active joystick is pressed and returns the ID of the button. See pilot.JoyData for the available buttons.

void wait_for_joystick_button (int button)
    Waits until the button with ID button is pressed on the active joystick. See pilot.JoyData for the available buttons.

void wait_for_digital_input (int channel, bool state)
    Waits until the digital input channel reaches state state. The digital inputs are enumerated from 0 to 15.

int wait_for_keypad ()
    Waits until any button on the keypad is pressed and returns an ID for the button (see below).

void wait_for_keypad_button (int button)
    Waits until the specified button on the keypad is pressed. button can be any of the following:
```

0	Info
1	Home
2	Start
3	Stop

Timers

```
void wait_ms (int period)
    Pauses for period milliseconds.

void wait_sec (int period)
    Pauses for period seconds.

void wait_min (int period)
    Pauses for period minutes.

void wait_hours (int period)
    Pauses for period hours.

void set_timer_ms (int period, string callback)
    Sets up a periodic timer that repeatedly calls the lua function named callback at the given period.
```

Hardware

```
float read_analog_input (int channel)
    Reads the voltage of an analog input in [V].

bool read_digital_input (int channel)
    Reads the state of a digital input.

void set_relay (int channel, bool state)
    Sets the relay with id channel to state state.

void set_digital_output (int channel, bool state)
    Sets the digital output with id channel to state state. The digital outputs are enumerated from 0 to 15.

void set_display_text (string text)
    Prints text on the first line of the LCD display. There is enough space for 20 characters.

void charge ()
    Starts the charging process. Fails immediately if no charger is detected. Otherwise blocks until the charging is either finished or aborted.

void start_charging ()
    Starts the charging process, i.e. activates the corresponding relay.

void stop_charging ()
    Stops the charging process, i.e. deactivates the corresponding relay.

void set_safety_mode (safety_mode_e mode, uchar station_id = 0)
    Switches the safety mode. See pilot.safety\_mode\_e. This command only works when there is a safety interface available.

void select_safety_field (uint field_id)
    Requests to set the safety field with the given field ID. This command only works when there is a safety interface available.
```

Information Requests

`int get_time_sec()`

Returns the current time stamp in seconds.

`int get_time_millis()`

Returns the current time stamp in milliseconds.

`int get_time_micros()`

Returns the current time stamp in microseconds.

`Pose2D get_position()`

Returns the current position on the map. See [pilot.Pose2D](#).

`MapNode find_station(string name)`

Finds and returns the node / station called name. See [pilot.MapStation](#) and See [pilot.MapNode](#).

`MapStation find_closest_station(double max_distance, Pose2D position)`

Returns the map station closest to position in a radius of at most max_distance meters. position defaults to the current position if known.

`string find_closest_station_name(double max_distance, Pose2D position)`

Same as `find_closest_station(...)` but returns the station name or `nil`.

`float get_battery_remaining()`

Returns the remaining battery charge as a percentage from 0 to 1.

`float get_battery_voltage()`

Returns the battery voltage in V.

`bool is_charging()`

Returns `true` if charging is in progress.

Log

`void log_info(string message)`

Generates a log message with priority *INFO*.

`void log_warn(string message)`

Generates a log message with priority *WARN*.

`void log_error(string message)`

Generates a log message with priority *ERROR*.

Control Flow

`void block()`

Pauses execution of the main thread.

Warning: Only use in event handlers!

`void unblock()`

Resumes execution of the main thread.

Warning: Only use in event handlers!

Builtin Functions

bool auto_charge (string *pre_stage1*, string *pre_stage2*, string *charge_station*, float *undock_distance*, float *max_velocity*)

Automatically docks at a specified charging station, charges the batteries until full and then undocks from the station.

pre_stage1 is a map station somewhere close (less than 1 m) to the charging station with an orientation close to the final docking pose. There should still be enough space to rotate fully without hitting the charging station.

pre_stage2 is a map station from where to start the docking process without having to rotate anymore. There should be about 10 to 20 cm of space between the contacts at this position.

charge_station is a map station where the platform makes contact with the charging station. It should be specified very precisely, within a few mm of accuracy.

undock_distance is the amount of distance to move backwards after finishing the charging process. The default is 0.25 m. Make sure the robot is allowed to move this much backwards, in case of a differential platform without a second laser scanner. *max_velocity* is the maximum velocity in [m/s] with which to dock and undock. The default is 0.05 m/s.

Returns true if successful, false otherwise.

A `require 'neobotix'` is needed to access this function in Lua script. Permissions MOVE and CHARGE are required, see [*pilot.permission_e*](#).

void reset_motors ()

Attempts to re-activate motors (clears any latched errors). Same action as when releasing EM stop.

A `require 'neobotix'` is needed to access this function in Lua script. Permission MOVE is required, see [*pilot.permission_e*](#).

OPC-UA Functions

UA node ids are specified in Lua via an array of two values. Numeric and string node ids are supported as follows: `{0, 1337}` or `{1, "MyObject"}`.

A `require 'neobotix'` is needed to access these functions in Lua script.

Variant **opc_ua_call** (string *proxy*, pair<ushort, Variant> *object*, string *method*, vector<Variant> *args*)

Performs an OPC-UA call via the specified *proxy* and returns the result of it.

proxy is the name of a running `vnx.opc_ua.Proxy` module, see [*opcua_proxy_map*](#).

object is an optional UA node id of the object for which to call the method. Can be set to `nil` in order to call a global method.

method is the method name (OPC-UA browse name).

args is an array of function parameters.

In case of failure `nil` or `false` is returned, depending on if the method has a return value (`nil`) or not (`false`). In case of multiple return values an array is returned.

Variant **opc_ua_read** (string *proxy*, pair<ushort, Variant> *object*, string *variable*)

Reads an OPC-UA variable via the specified *proxy*.

proxy is the name of a running `vnx.opc_ua.Proxy` module, see `opcua_proxy_map`.

object is a UA node id of the object containing the variable. Setting it to `nil` is equivalent to calling `opc_ua_read_global`.

variable is the variable name (OPC-UA browse name).

Returns the value read, or `nil` in case of failure.

Variant **opc_ua_read_global** (string *proxy*, pair<ushort, Variant> *variable*)

Reads a global OPC-UA variable via the specified proxy.

proxy is the name of a running `vnx.opc_ua.Proxy` module, see `opcua_proxy_map`.

variable is a UA node id of a global variable.

Returns the value read, or `nil` in case of failure.

bool **opc_ua_write** (string *proxy*, pair<ushort, Variant> *object*, string *variable*, Variant *value*)

Writes a value to an OPC-UA variable via the specified proxy.

proxy is the name of a running `vnx.opc_ua.Proxy` module, see `opcua_proxy_map`.

object is a UA node id of the object containing the variable. Setting it to `nil` is equivalent to calling `opc_ua_write_global`.

variable is the variable name (OPC-UA browse name).

value is the value to be written.

Returns `true` on success, `false` in case of failure.

bool **opc_ua_write_global** (string *proxy*, pair<ushort, Variant> *variable*, Variant *value*)

Writes a value to a global OPC-UA variable via the specified proxy.

proxy is the name of a running `vnx.opc_ua.Proxy` module, see `opcua_proxy_map`.

variable is a UA node id of a global variable.

value is the value to be written.

Returns `true` on success, `false` in case of failure.

Advanced

Variant **execute** (string *module*, string *method*, Object *params*)

Executes a function *method* of the module *module*. The parameters are specified as key-value pairs in *params*. If the method does not take parameters, *params* can be omitted.

If *method* does not have a return type, the command returns `true` or `false`, depending on whether it succeeded. If *method* does have a return type, either a value of that type is returned or `nil` on failure.

See `vnx.Variant` and `vnx.Object`.

Warning: This command allows almost unlimited access to the functionality of *PlatformPilot*. However, it also allows you to leave the boundaries of safe operation and must therefore be considered dangerous.

Event Handlers

The following functions, if defined in your script, will be called upon specific events. Events are queued and the next event handler will only be called after the current one finished.

Be aware that the calls happen asynchronously to the main thread. The two executions run in parallel while abiding to Lua's [cooperative multithreading](#) model. You can use the `block()` and `unblock()` functions to avoid multithreading issues.

`void on_em_stop()`

Is executed when the emergency stop button is pushed.

`void on_scanner_stop()`

Is executed when the scanner emergency stop is triggered.

`void on_em_reset()`

Is executed when a previous emergency situation (button and/or scanner) is resolved and the platform can move again.

`void on_joystick_button_pressed(int button)`

Is executed when the joystick button with ID `button` changes its state from not pressed to pressed.

`void on_joystick_button_released(int button)`

Is executed when the joystick button with ID `button` changes its state from pressed to not pressed.

`void on_keypad_button_pressed(int button)`

Is executed when the specified keypad button changes its state from not pressed to pressed. For the available button IDs see above.

`void on_keypad_button_released(int button)`

Is executed when the specified keypad button changes its state from pressed to not pressed. For the available button IDs see above.

`void on_digital_input_on(int channel)`

Is executed when the digital input `channel` changes its state from off to on.

`void on_digital_input_off(int channel)`

Is executed when the digital input `channel` changes its state from on to off.

`void on_battery_low()`

Is executed when the battery drops below a low level.

`void on_battery_critical()`

Is executed when the battery drops below a critical level.

Examples

Move to stations in the Road Map in a loop:

```
function main()
    while true do
        move_to_station("Station4");
        move_to_station("Station9");
        move_to_station("Station11");
        wait_ms(1000);
    end
end
```

Move randomly to any of the specified stations in a loop, abort in case of failure, stop when the Y button on the joystick is pressed:

```

function odyssey(stations)
    repeat
        wait_ms(1000);
        index = math.random(1, #stations)
    until not move_to_station(stations[index])
end

function on_joystick_button_pressed(button)
    if button == 3 then
        cancel_goal()
    end
end

function main()
    odyssey({"Station20", "Station10", "Station12", "Station14", "Station1"})
end

```

Dock to a charging station using special parameters:

```

move_to_station("ChargeStation", {
    max_velocity = 0.1,
    drive_flags = {"IGNORE_FOOTPRINT", "DISABLE_ROTATION"}
})

```

Undock from a charging station using a relative move command with special parameters:

```

move(-0.25, 0, 0, {
    max_velocity = 0.1,
    drive_flags = {"IGNORE_FOOTPRINT", "DISABLE_ROTATION"}
})

```

Using the built-in auto_charge(...) function to dock, charge and undock:

```

require 'neobotix'

function main()
    auto_charge("PreStage1", "PreStage2", "ChargeStation", 0.25, 0.05);
end

```

Calling an OPC-UA method:

```

require 'neobotix'

function main()
    ret = opc_ua_call("OPC_UA_Proxy", {1, "vnx.process"}, "get_name")
    log_warn(ret) --> "pilot_main"
    ret = opc_ua_call("OPC_UA_Proxy", {1, "HybridPlanner"}, "set_goal_stations", {
        {"Station4", "Station3", "Station2"}
    })
end

```

With `opcua_proxy_map` set to `[["OPC_UA_Proxy", "opc.tcp://127.0.0.1:4840"]]` and `enable_opcua_server` set to true.

Reading an OPC-UA variable:

```

require 'neobotix'

function main()
    ret = opc_ua_read("OPC_UA_Proxy", {0, 2253}, "ServerArray")
    log_warn(ret) --> ["urn:open62541.server.application"]
end

```

With `opcua_proxy_map` set to `[["OPC_UA_Proxy", "opc.tcp://127.0.0.1:4840"]]` and `enable_opcua_server` set to true.

8.2.8 ROS Bridge

The ROS Bridge allows to integrate PlatformPilot into a ROS / ROS 2 environment, such that it is possible to control the platform via ROS as well as visualize all data in RViz.

Installation

It is assumed that a ROS workspace has already been setup on the platform's PC, for example `~/ros_workspace/`.

To install the ROS Bridge:

Note: Currently ROS and ROS 2 are being maintained in different branches on Github. The default branch is set to ROS 2.

```

cd ~/ros_workspace/src/
git clone https://github.com/neobotix/pilot-ros-bridge.git
cd ~/ros_workspace/
# For ROS 2
colcon build --symlink-install
# For ROS
catkin_make

```

On ROS 2 side, the package depends on `neo_srvs2` and `neo_msgs2`. Similarly, `neo_msgs` and `neo_srvs` for ROS.

On Pilot side, the package depends on an already installed `neobotix-pilot-core` or `neobotix-pilot-gtkgui` package.

Running

To run the ROS bridge on the platform:

```

source ~/ros_workspace/devel/setup.bash
# For ROS 2
ros2 launch pilot_ros_bridge mpo_700.launch
# For ROS
roslaunch pilot_ros_bridge mpo_700.launch

```

Replace `mpo_700.launch` with `mpo_500.launch` or `mp_400.launch`, depending on your platform.

It is also possible to run the ROS bridge on another PC, by adapting the `pilot_node` param in the launch file:

```
<param name="pilot_node" type="str" value="192.168.0.50:5555"/>
```

Replace 192.168.0.50 with the actual IP address of the platform.

In addition you can disable user authentication on the TCP server to allow for all functionality via the ROS Bridge. To do this create a config file `config/local/TcpServer.json`:

```
{  
    "use_authentication": false  
}
```

This is only necessary when running the ROS Bridge on another PC.

Topics

The following topics are enabled on the ROS bridge by default. You can add more by adapting the config file `config/default/generic/Pilot_ROS_Bridge.json` in `pilot-ros-bridge`.

Export

```
"export_map": [  
    ["platform.odometry", "/odom"],  
    ["sensors.laser_scan.lidar_1", "/lidar_1/scan"],  
    ["sensors.laser_scan.lidar_2", "/lidar_2/scan"],  
    ["sensors.filtered_scan.lidar_1", "/lidar_1/scan_filtered"],  
    ["sensors.filtered_scan.lidar_2", "/lidar_2/scan_filtered"],  
    ["kinematics.drive_state", "/drives/joint_states"],  
    ["mapping.grid_map", "/mapping/map"],  
    ["mapping.grid_map_tile", "/mapping/map_tile"],  
    ["mapping.grid_map_tile_ref", "/mapping/map_tile_ref"],  
    ["navigation.grid_map", "/map"],  
    ["navigation.grid_map_tile", "/map_tile"],  
    ["navigation.map_pose", "/map_pose"],  
    ["navigation.map_particles", "/particlecloud"],  
    ["navigation.road_map", "/road_map"],  
    ["navigation.global_path", "/global_path"],  
    ["navigation.local_path", "/local_path"],  
    ["navigation.local_cost_map", "/local_cost_map"],  
    ["navigation.local_cost_map_overlay", "/local_cost_map_overlay"],  
    ["navigation.global_cost_map", "/global_cost_map"],  
    ["navigation.global_cost_map_overlay", "/global_cost_map_overlay"],  
    ["local_planner.target_pose", "/local_planner/target_pose"],  
    ["local_planner.predicted_pose", "/local_planner/predicted_pose"]  
]
```

Import

```
"import_map": [  
    ["/cmd_vel", "geometry_msgs/Twist", "platform.velocity_cmd"],  
    ["/initialpose", "geometry_msgs/PoseWithCovarianceStamped", "navigation.  
    initial_pose"],  
    ["/goal", "geometry_msgs/PoseStamped", "navigation.new_goal_pose"]  
]
```

Note: In ROS 2, 2D Goal Pose tool from RViz2 is utilized for sending the goal to the `/goal` topic. Similarly user can use the same topic from the application side.

8.3 API Reference

This section describes the data types, modules and methods available in PlatformPilot.

8.3.1 Common Datatypes

```
void empty space
bool 8-bit boolean (default = false)
char 8-bit signed integer (default = 0)
uchar 8-bit unsigned integer (default = 0)
short 16-bit signed integer (default = 0)
ushort 16-bit unsigned integer (default = 0)
int 32-bit signed integer (default = 0)
uint 32-bit unsigned integer (default = 0)
long 64-bit signed integer (default = 0)
ulong 64-bit unsigned integer (default = 0)
float 32-bit floating point number (default = 0)
double 64-bit floating point number (default = 0)
string UTF-8 string (default = "")
vector<T> A list of values of type T. (default = [])
optional<T> An optional value of type T. (default = null)
set<T> An ordered set of values of type T. (no duplicates)
map<K, V> An ordered map with key type K and value type V. (no duplicate keys)
pair<K, V> A pair of values of type K and V respectively.
T* A value of type T or any derived type, or null. (default = null)
```

8.3.2 Coordinate Systems

Below are the most common coordinate systems used. In addition to those each sensor usually has its own coordinate system too.

Coordinate positions are always measured in SI units, i.e. meters for distances and rads for angles. Velocities are measured in m/s and rad/s.

Base Link (base_link)

The platform's own coordinate system, usually the center of mass. X points forward, Y points to the left and Z points upwards. The internal name is `base_link`.

Odometry (odom)

This is an arbitrary coordinate system with its origin at the starting position (at the time of power on) of the platform. X and Y form the plane of movement, while Z is pointing upwards and aligned with gravity. Usually there is no motion in Z direction, except when a roll and pitch sensor is installed. The internal name is `odom`.

Map (map)

This is the coordinate system of the currently used *Grid Map* and *Road Map*. Usually the *Grid Map* is created first and its origin is the starting position of where the map was created initially. X and Y form the plane of movement, while Z is pointing upwards and aligned with gravity. The internal name is `map`.

8.3.3 Classes

basic.Transform3D

Class

`Transform3D` represents a 2D/3D transformation from a specified coordinate system to another. It can be described by a 4x4 matrix that transforms a 3D vector from the source coordinate system to the target coordinate system.

Fields

long `time`

POSIX timestamp in [usec].

string `frame`

Source coordinate system name.

string `parent`

Target coordinate system name.

Matrix4d `matrix`

The transformation matrix, such that left multiplying a vector transforms it from the source coordinate system `frame` to the target coordinate system `parent`. For example: `target = matrix * [x, y, z, 1]^T`. See [math.Matrix4d](#).

math.Matrix4d

A 4x4 double (64-bit float) matrix, usually a 3D transformation matrix.

math.Vector2d

A 2D double (64-bit float) vector, usually a (x, y) position. Mathematically a column vector, in JSON an array of 2 numbers.

math.Vector3d

A 3D double (64-bit float) vector, usually (x, y, z) for a 3D position, or (x, y, yaw) for a 2D pose. Mathematically a column vector, in JSON an array of 3 numbers.

math.Vector3f

A 3D float vector, usually (x, y, z) for a 3D position, or (x, y, yaw) for a 2D pose. Mathematically a column vector, in JSON an array of 3 numbers.

pilot.ActiveIncidents

Class

Contains the list of currently active incidents, which is periodically published on topic `platform.active_incidents`.

Fields

`long time`
POSIX timestamp [usec]

`vector<event_t> events`
List of active incidents, see [`pilot.event_t`](#).

pilot.BatteryState

Class

`BatteryState` contains information regarding the platform's batteries.

Fields

`long time`
POSIX timestamp in [usec].

`float remaining`
Percentage of charge remaining, from 0 to 1.

`float voltage`
Battery voltage in [V].

`optional<float> current`
Battery current if available, positive = charging, in [A].

`optional<float> charge`
Battery charge if available, in Coulomb (Ampere seconds).

`optional<float> capacity`
Last full capacity if available, in Coulomb (Ampere seconds).

`optional<float> design_capacity`
Battery capacity as manufactured if available, in Coulomb (Ampere seconds).

float **temperature**

Battery temperatue in [C].

battery_type_e **type**

Battery type, see *pilot.battery_type_e*.

int **module_count**

Number of battery modules, -1 if unknown.

pilot.Beacon

Class

Status updates to be published by robots.

Inherits from *pilot.RobotInfo*.

Fields

long **time**

POSIX timestamp [usec]

pilot.CostMapData

Class

A map representing the navigation cost based on the proximity to walls and other obstacles.

Inherits from *pilot.GridMapData*.

Fields

static const uchar PROHIBITED = 254

static const uchar UNKNOWN = 255

float **cost_scale**

obstacle distance scale [m]

distance = (1 - (cost / 200)) * cost_scale

Image8 **cost**

- 0 to 200 = cost
- 200 = wall
- > 200 = same as occupancy map

Methods

ImageF32 **to_float()** **const**

convert to float format (0 to 1)

```
Image8 to_rgba_image (int alpha) const
    for visualization (with color)

Image8 to_rgba_mono_image (int alpha) const
    for visualization (just grayscale)

static Vector4uc to_rgba (int value, int alpha)
    convert occupancy to RGBA

static Vector4uc to_rgba_mono (int value, int alpha)
    convert occupancy to RGBA mono
```

pilot.EmergencyState

Class

EmergencyState contains information regarding the platform's emergency systems.

Fields

```
long time
    POSIX timestamp in [usec].

safety_code_e code
    The type of the current emergency, see pilot.safety_code_e.

em_stop_state_e state
    Current EM Stop state, see pilot.em_stop_state_e.
```

pilot.Event

Class

Generic event type, as published on topic `platform.events` for example.

Fields

```
long time
    POSIX timestamp [usec]

event_t event
    Event data, see pilot.event_t.

Object info
    Additional information regarding the event, see vnx.Object.
```

pilot.ExecutionError

Class

ExecutionError contains information about execution errors of the *TaskHandler*.

Fields

long **time**
POSIX timestamp in [usec].

Hash64 **jobid**
Current program execution id. See [vnx.Hash64](#).

string **program_name**
Name of the running program.

string **error_message**
The original error message from the language interpreter.

vector<StackFrame> **stack**
Execution stack, most recent first. See [pilot.StackFrame](#).

pilot.ExecutionHistory

Class

A history of recent *execution states* of the *TaskHandler*.

Fields

list<ExecutionState *> **history**

pilot.ExecutionState

Class

ExecutionState contains information about the current state of the *TaskHandler*.

Fields

long **time**
POSIX timestamp in [usec].

execution_state_e **status**
Current execution state, see [pilot.execution_state_e](#).

Hash64 **jobid**
Current program execution id. See [vnx.Hash64](#).

string **program_name**
Name of the running program.

Task ***task**
Current task being executed, if any. See [pilot.Task](#).

vector<StackFrame> **stack**
Execution stack, most recent first. See [pilot.StackFrame](#).

long **time_started**
POSIX time stamp when program was started in [usec].

long time-ended

POSIX time stamp when program has finished in [usec], zero otherwise (ie. still running).

bool is_minor

True if current task is nothing major, for example log message output, etc.

pilot.Footprint

Class

Footprint describes the space needed by a platform, including the safety scanner fields with some additional padding.

Fields

vector<Vector2d> pointsList of 2D (x, y) [m, m] points in `base_link` coordinates that form a polygon representing the footprint. The points should be specified either in clockwise or anti-clockwise direction.See [Coordinate Systems](#).

pilot.GridMapData

Class

Base class of pixel based world maps.

Inherits from `pilot.Sample`.

Fields

string name**long last_modified**
timestamp [usec]**float scale**
size of a pixel in [m]**Vector2d origin**
grid offset (map position of lower left pixel, world to map) [m]**double orientation**
grid rotation (world to map) [rad]

Methods

void transform (Transform3D *sample)**Matrix4d get_grid_to_frame () const**
computes grid to 'frame' 2.5D transformation matrix**Matrix4d get_frame_to_grid () const**
computes 'frame' to grid 2.5D transformation matrix

```
bool same_as (GridMapData *other) const
    returns true if both have same name and last_modified time
map_info_t get_info () const
```

pilot.Incident

Class

Generic incident type, as published on `platform.events` for example.

External modules can publish their own incidents on topic `platform.incidents`. They will be handled by [Pilot-Server](#) and re-published on `platform.events` as required.

Inherits from [pilot.Event](#).

Fields

```
bool is_active
    If this incident is currently active, ie. not a one time event.
bool is_cleared
    If this incident was active before and is now cleared, ie. no longer active.
int timeout_ms = 3000
    Timeout in case of an active incident in [ms]. timeout_ms <= 0 disables the feature. If the incident is not re-published within the specified timeout it will be automatically cleared by PilotServer.
```

pilot.JoyData

Class

`JoyData` contains data from a connected joystick.

Fields

```
long time
    POSIX timestamp in [usec].
float deadzone
    For axes values in the interval (-deadzone, deadzone), zero should be assumed.
vector<float> axes
    List of joystick axis and their position from -1 to 1. The values are listed in the following order: [JOYAXIS_LEFT_X, JOYAXIS_LEFT_Y, JOYAXIS_LT, JOYAXIS_RIGHT_X, JOYAXIS_RIGHT_Y, JOYAXIS_RT]
vector<bool> buttons
    List of joystick buttons and their state. The values are listed in the following order: [JOYBUTTON_A, JOYBUTTON_B, JOYBUTTON_X, JOYBUTTON_Y, JOYBUTTON_LB, JOYBUTTON_RB, JOYBUTTON_BACK, JOYBUTTON_START]
```

pilot.LaserPointCloud

Class

LaserPointCloud contains additional information for laser point clouds.

Inherits from *pilot.PointCloud2D*.

Fields

vector<uchar> **intensity**

Measured intensities for each point in *points*. An intensity ≥ 100 is a reflector, 200 is a perfect reflector.

pilot.LocalPlannerState

Class

LocalPlannerState represents the current state of the *LocalPlanner*.

Fields

long **time**

POSIX timestamp in [usec].

Hash64 **job**

Unique id of the current job / goal.

long **path_time**

POSIX timestamp of the current path in [usec].

double **path_length**

Total path length in [meters].

PathPoint2D ***point**

Current path point if any (ie. closest to current position).

PathPoint2D ***goal**

Current goal point, ie. last point in current path.

goal_options_t **goal_options**

Current goal options.

Vector2d **pos_error**

Current (x, y) position error in [meters].

double **yaw_error**

Current yaw orientation error in [radians].

float **update_rate**

Current control update rate in [1/s].

local_planner_state_e **state**

Current local planner state, see *pilot.local_planner_state_e*.

wait_reason_e **wait_reason**

Current reason for waiting, see *pilot.wait_reason_e*.

limit_reason_e `velocity_reason`

Current reason for velocity limitation, see [*pilot.limit_reason_e*](#).

limit_reason_e `yawrate_reason`

Current reason for yawrate limitation, see [*pilot.limit_reason_e*](#).

float `time_stuck`

How long the platform has already been stuck in [seconds], in case `state == STUCK`.

float `progress`

Relative progress, from 0 to 1. Ratio of traversed path distance vs. total path length.

bool `is_backwards`

If platform is currently driving backwards.

bool `is_restricted`

If platform is currently restricted from turning around.

vector<int> `path_history`

Sliding window history of traversed path element ids, newest at the end. See [*pilot.MapElement*](#). Usually contains at least the last 10 elements.

pilot.LocalizationStatus

Class

LocalizationStatus contains information about the current state of localization.

Fields

long `time`

POSIX timestamp in [usec].

localization_mode_e `mode`

Current localization mode, see [*pilot.localization_mode_e*](#).

set<string> `sensors`

Current list of sensors used for localization.

float `update_rate`

Current update rate in [1/s]

int `num_points`

Number of valid sensor points, excluding reflectors.

int `num_points_total`

Number of valid sensor points, including reflector points.

Vector3f `std_dev`

Current particle spread standard deviation (x, y, yaw) [m, m, rad]. See [*math.Vector3f*](#).

float `score`

Current scan matching score, ie. *average likelihood*. More is better, maximum is 1.0 (ie. all scan points are matching exactly to the Grid Map).

pilot.MapArea

Class

Describes a special area within a map.

Fields

```
string name  
string type  
string description  
polygon_t outline  
vector<area_property_e> flags
```

pilot.MapElement

Class

MapElement is the base class for an entity in the Road Map.

Fields

```
int id  
    Unique id for the element, starting at zero.  
string group  
    Logical group name, mostly for visualization purposes.  
vector<string> profiles  
    List of profiles assinged to this element, parameters are copied / over-written in the order of profiles listed.
```

pilot.MapMatch

Class

MapMatch describes the current position in terms of the *Road Map*, when possible.

Fields

```
long time  
    POSIX timestamp in [usec].  
bool is_valid  
    If a match was found or not.  
float distance  
    Shortest distance from platform center to matched node or segment in [meters].
```

MapNode *node

The matched *pilot.MapNode* if near a node.

RoadSegment *segment

The matched *pilot.RoadSegment* if on a segment, which is normally the case.

vector<MapArea *> areas

Areas the platform is inside of.

pilot.MapNode

MapNode describes a node in a Road Map. It has a position as well as an optional name, in addition to some parameters.

Inherits from *pilot.MapElement*.

Fields

int parent = -1

Parent node id, if any. If a *pilot.MapNode* or *pilot.MapStation* has a parent its position (and orientation) is defined relative to it.

string name

Optional node name, usually a station name.

Vector2d position

2D (x, y) map position, see *math.Vector2d*.

Vector3d offset

3D (x, y, yaw) pose offset relative to parent, see *math.Vector3d*. If parent is a *pilot.MapStation* the offset is relative to its orientation.

set<drive_flags_e> drive_flags

A set of drive flags for this node, see *pilot.drive_flags_e*.

pilot.MapProfile

Class

MapProfile represents a set of parameters which can be applied to multiple map elements at once.

Fields

string name

Profile name, should be unique.

string description

Profile description text.

Object node

A set of *pilot.MapNode* parameters. See *vnx.Object*.

Object segment

A set of *pilot.RoadSegment* parameters. See *vnx.Object*.

Object `station`

A set of `pilot.MapStation` parameters. See `vnx.Object`.

pilot.MapStation**Class**

`MapStation` describes a station in a Road Map, something which can be set as a goal. It has a position as well as an orientation, in addition to some parameters.

Inherits from `pilot.MapNode`.

Fields**float `orientation`**

Absolute map goal orientation in [radians].

float_param_t `goal_tune_time`

Additional time for fine tuning position in [seconds]. See `pilot.float_param_t`.

vector_3f_param_t `goal_tolerance`

Maximum goal position error (x, y, yaw) [m, m, rad], see `pilot.vector_3f_param_t`.

pilot.NodeReservation**Class**

Reservation status of a roadmap node or station.

Fields**node_reservation_type_e `status`**

Type of the reservation. Either `STOP` for stopping on this node or `DRIVE` for driving through.

set<Hash64> `has`

IDs of robot(s) that currently hold this reservation with the given type.

map<Hash64, node_reservation_type_e> `wants`

IDs of robots that want to reserve this node, together with the type they want.

pilot.OccupancyMapData**Class**

Map with environment information used for navigation.

Inherits from `pilot.GridMapData`.

Fields

```
static const uchar FREE = 0
static const uchar DYNAMIC = 253
static const uchar PROHIBITED = 254
static const uchar UNKNOWN = 255
```

Image8 **occupancy**

Represents the pixel grid.

- 0 to 100 = wall
- 101 to 200 = reflector
- 254 = prohibited
- 255 = unknown

Methods

Image8 **to_mono_image () const**

convert to 8-bit format (0 to 255)

Image8 **to_rgba_image (int alpha) const**

for visualization

ImageF32 **to_float (float special, bool combined) const**

convert to float format (0 to 1)

OccupancyMapData ***to_reflector_map () const**

convert to reflector occupancy (shift 200 to 100, erase 0 to 100)

OccupancyMapData ***to_combined_map () const**

convert reflectors to normal occupancy (map 200 to 100)

static Vector4uc **to_rgba (int value, int alpha)**

convert occupancy to RGBA

pilot.Odometry

Class

Odometry contains information about the platform's odometry, as well as velocities. See also [Coordinate Systems](#).

Inherits from [basic.Transform3D](#).

Fields

Vector3d **position**

Current 3D (x, y, z) [m] odom position.

Vector3d **orientation**

Current 3D (roll, pitch, yaw) [rad] odom orientation.

Vector3f **linear_velocity**

Current 3D (x, y, z) [m/s] odom velocity.

Vector3f **angular_velocity**
 Current 3D (roll, pitch, yaw) [rad/s] odom angular velocity.

pilot.Path2D

Class

Path2D represents a 2D path made of individual *pilot.PathPoint2D* points, in a given coordinate system.
 Inherits from *pilot.Sample*.

Fields

Hash64 **job**
 Unique job id for this path / goal.

vector<PathPoint2D *> **points**
 List of path points, see *pilot.PathPoint2D*.

pilot.PathPoint2D

Class

PathPoint2D represents a point on a path to navigate. They are usually generated from a Road Map and then optimized while driving. It can also be a final goal position.

Inherits from *pilot.Pose2D*.

Fields

int **map_id** = -1
 Id of the map element in the Road Map, if any.

drive_mode_e **drive_mode** = DEFAULT
 Specifies how the robot should navigate, see *pilot.drive_mode_e*.

set<drive_flags_e> **drive_flags**
 A set of drive flags for this path point, see *pilot.drive_flags_e*.

float_param_t **orientation**
 Yaw angle offset relative to *pose.z()* (holonomic only) in [radians]. See *pilot.float_param_t*.

float_param_t **max_velocity**
 Maximum velocity in [m/s]. See *pilot.float_param_t*.

float_param_t **max_yawrate**
 Maximum yawrate in [rad/s]. See *pilot.float_param_t*.

float_param_t **goal_tune_time**
 Additional time for fine tuning position in [seconds]. See *pilot.float_param_t*.

vector_3f_param_t **tolerance**
 Maximum position error (x, y, yaw) [m, m, rad], see *pilot.vector_3f_param_t*.

bool **is_restricted**
True if a 360 degree rotation is not possible at this location.

pilot.PilotState

Class

PilotState contains information about the current state of the robot.

Fields

long **time**
POSIX timestamp in [usec]

Hash64 **robot_id**
Unique robot id.

string **pilot_version**
Version number of the platform software.

pilot_mode_e **pilot_mode**
Current pilot mode, see [pilot.pilot_mode_e](#).

motion_mode_e **motion_mode**
Current motion mode, see [pilot.motion_mode_e](#).

LocalPlannerState ***planner**
Current local planner state, see [pilot.LocalPlannerState](#).

LocalizationStatus ***localization**
Current localization status, see [pilot.LocalizationStatus](#).

ExecutionState ***execution**
Current task execution status, see [pilot.ExecutionState](#).

ActiveIncidents ***incidents**
Currently active incidents, see [pilot.ActiveIncidents](#).

bool **is_recording**
If data recording is currently active, see topic `vnx.recorder_status`.

pilot.PlatformInfo

Class

PlatformInfo contains static information about the robot.

Fields

platform_type_e **type**
Type of the platform, see [pilot.platform_type_e](#).

string **name**
Name of the robot.

string `serial`

Serial number of the robot.

long `date_of_manufacture`

POSIX timestamp regarding the date of manufacture of the robot in [seconds].

vector<string> `features`

List of special features that the platform has.

pilot.PointCloud2D

Class

PointCloud2D contains a cloud of 2D points, usually laser points. See also [*pilot.LaserPointCloud*](#).

Inherits from [*pilot.Sample*](#).

Fields

string `sensor`

Original sensor coordinate frame. (also name of sensor)

sensor_2d_range_t `field`

Field of view of the sensor, relative to its own coordinate system, see [*pilot.sensor_2d_range_t*](#).

Transform3D *`base_to_odom`

Transformation from `base_link` to `odom` frame at the time of this scan.

Transform3D *`sensor_to_base`

Transformation from `sensor` to `base_link` frame.

vector<Vector2d> `points`

List of 2D (x, y) points [meters], in `frame` coordinates, see [*pilot.Sample*](#).

pilot.Pose2D

Class

Pose2D represents a 2D pose (x, y, yaw) in the specified coordinate system.

Inherits from [*basic.Transform3D*](#).

Fields

Vector3d `pose`

2D pose (x, y, yaw) [m, m, rad], see [*math.Vector3d*](#).

Matrix3f `covariance`

Optional pose covariance matrix.

pilot.PoseArray2D

Class

An aggregation of poses.

Inherits from *pilot.Sample*.

Fields

vector<Vector3d> **poses**
[x, y, yaw] (m, m, rad)

Methods

void **transform** (Transform3D **sample*)

pilot.PowerState

Class

PowerState contains information about the platform's power supply.

Fields

long **time**
POSIX timestamp in [usec].

power_system_type_e **power_system_type**
Type of power system, see *pilot.power_system_type_e*.

charging_state_e **charging_state**
Current charging state, see *pilot.charging_state_e*.

bool **is_charging**
If charging is enabled.

double **charging_current**
The charging current, if charging is enabled, in [A].

pilot.RadarScan

Class

Overview of the robots of the fleet, their basic information, dimensions, positions and movement.

Fields

vector<radar_sample_t> **samples**
One radar sample per robot, see *pilot.radar_sample_t*.

pilot.RelayBoardData

Class

RelayBoardData contains the data sent by the RelayBoard.

Fields

float **ambient_temperature**

Ambient temperature inside the platform in [C].

vector<bool> **relay_states**

State of the power relays, usually there is 4 of them.

keypad_state_t **keypad_state**

Keypad button states, if available, see [*pilot.keypad_state_t*](#).

pilot.RelayBoardV3Data

Class

RelayBoardV3Data contains the data sent by the RelayBoardV3.

Fields

long **time**

POSIX timestamp [usec]

string **firmware_version**

long **uptime**

Time since last module restart / reconfig [usec]

float **ambient_temperature**

Ambient temperature in °C

vector<bool> **relay_states**

vector<bool> **digital_input_states**

vector<bool> **keypad_button_states**

map<led_color_e, double> **led_states**

Color => amount [0-1], see [*pilot.led_color_e*](#).

bool **key_switch_off_state**

true for pressed

bool **release_structure_state**

pilot.RoadMapData

Class

RoadMapData contains all the information regarding a *Road Map*.

Fields

string **name**

Name of the map.

long **last_modified**

POSIX timestamp when the map was last modified in [usec].

vector<MapNode *> **nodes**

List of map nodes, see [pilot.MapNode](#).

vector<RoadSegment *> **segments**

List of road segments, see [pilot.RoadSegment](#).

map<string, MapProfile *> **profiles**

Map of map profiles [name => profile], see [pilot.MapProfile](#).

vector<MapArea *> **areas**

List of map areas, see [pilot.MapArea](#).

[pilot.RoadSegment](#)

Class

RoadSegment represents a connection in the *Road Map* from one [pilot.MapNode](#) to another.

Inherits from [pilot.MapElement](#).

Fields

int **from_node**

[pilot.MapNode](#) id from where the segment begins.

int **to_node**

[pilot.MapNode](#) id to where the segment goes.

char **direction** = 0

-1 = one way backwards, 0 = two way, 1 = one way forwards

drive_mode_e **drive_mode** = RELAXED_PATH_FOLLOW

Drive mode to be used on this segment, see [pilot.drive_mode_e](#).

orientation_mode_e **orientation_mode** = RELATIVE_ROAD

Orientation mode to be used on this segment, see [pilot.orientation_mode_e](#).

float_param_t **tolerance**

Lateral tolerance in [meters], how much the path can be modified while driving (to avoid obstacles, etc). See [pilot.float_param_t](#).

float_param_t **orientation**

Orientation to drive at on this segment in [radians]. Actual orientation depends on *orientation_mode*. See [pilot.float_param_t](#).

float_param_t **orientation_tolerance**

Maximum deviation allowed from the target orientation on this segment in [radians]. See [pilot.float_param_t](#).

float_param_t **max_velocity**

Maximum velocity allowed on this segment in [m/s]. See [pilot.float_param_t](#).

float_param_t **max_yawrate**

Maximum yawrate allowed on this segment in [rad/s]. See [*pilot.float_param_t*](#).

set<drive_flags_e> **drive_flags**

A set of drive flags for this segment, see [*pilot.drive_flags_e*](#).

pilot.RoadmapReservation

Class

Overview over currently reserved roadmap elements. Roadmap elements not listed are currently free.

Fields

map_info_t **map_info**

The roadmap, see [*pilot.map_info_t*](#).

map<int, NodeReservation> **nodes**

Reservations of nodes and stations, identified by their ID. See [*pilot.NodeReservation*](#).

map<int, SegmentReservation> **segments**

Reservations of segments, identified by their ID. See [*pilot.SegmentReservation*](#).

pilot.RobotInfo

Class

Information about the overall status of a robot.

Fields

Hash64 **id**

unique robot id

optional<map_info_t> **grid_map**

used grid map, see [*pilot.map_info_t*](#)

optional<map_info_t> **road_map**

used road map, see [*pilot.map_info_t*](#)

optional<pose_2d_t> **map_pose**

optional<Vector3f> **velocity**

relative to base_link (x, y, yaw) [m/s, m/s, rad/s]

PilotState ***state**

see [*pilot.PilotState*](#)

Path2D ***map_path**

current global path in the map, see [*pilot.Path2D*](#)

MapMatch ***map_match**

see [*pilot.MapMatch*](#)

Footprint ***footprint**

see *pilot.Footprint*

PlatformInfo ***platform**

see *pilot.PlatformInfo*

SystemState ***system_status**

see *pilot.SystemState*

BatteryState ***battery_state**

see *pilot.BatteryState*

EmergencyState ***emergency_state**

see *pilot.EmergencyState*

vector<PointCloud2D *> **laser_scans**

in map coordinates

pilot.RobotInfoMap

Class

Aggregated list of *pilot.RobotInfo* objects.

Fields

map<Hash64, RobotInfo *> **info_map**

Maps a robot's ID to its RobotInfo object.

pilot.SafetyState

Class

SafetyState contains information about the platform's safety state, if available.

Fields

uint **current_safety_field**

Currently active safety field.

set<uchar> **triggered_cutoff_paths**

A list of the currently triggered cutoff paths, as configured e.g. in safety laser scanners.

pilot.Sample

Class

Sample is a base class for any type requiring a time stamp as well as a coordinate system.

Fields

long **time**

POSIX timestamp in [usec]

string **frame**

Coordinate system name, usually map, odom or base_link, see [Coordinate Systems](#).

pilot.SegmentReservation

Class

Reservation status of a roadmap segment.

Fields

segment_reservation_type_e **status**

Type of the reservation. Either FORWARD for driving from start node to end node or BACKWARD for the opposite direction.

set<Hash64> **has**

IDs of robot(s) that currently hold this reservation with the given type.

map<Hash64, segment_reservation_type_e> **wants**

IDs of robots that want to reserve this segment, together with the type they want.

pilot.StackFrame

Class

StackFrame contains information about the context of execution of the [TaskHandler](#).

Fields

string **method**

Method name of the function on the call stack.

int **line_number**

Line number in the source code.

pilot.SystemState

Class

SystemState contains information about the platform's system state.

Fields

vector<system_error_e> **system_errors**

List of system errors currently active, see [pilot.system_error_e](#).

bool **is_shutdown**

If platform has been asked to shutdown now, via the key switch.

bool **is_initialized**

If platform hardware is initialized.

pilot.Task

Class

Task contains information about the current task being executed by the [TaskHandler](#).

Fields

Hash64 **id**

Unique task id, randomly generated. See [vnx.Hash64](#).

string **module**

Module name which is handling the task, empty if handled internally.

string **method**

Method name of the task being executed.

Object **args**

Parameters for the task being executed. See [vnx.Object](#).

pilot.VelocityCmd

Class

VelocityCmd contains the current commanded platform velocity. Usually only X / Y velocity and yawrate are specified.

Fields

long **time**

POSIX timestamp in [usec].

Vector3f **linear**

3D (x, y, z) translational velocity in [m/s].

Vector3f **angular**

3D (roll, pitch, yaw) angular velocity in [rad/s].

bool **allow_wheel_reset** = true

If to allow automatic wheel resetting of an MPO-700.

bool **reset_wheels** = false

If to reset the omnidrive modules (if available) to their home position, when not moving.

pilot.VelocityLimits

Class

VelocityLimits defines different limits and thresholds regarding platform velocity.

Fields

float **max_vel_x**

Maximum forward velocity (positive) [m/s]

float **min_vel_x**

Maximum backwards velocity (negative) [m/s]

float **max_vel_y**

Maximum lateral velocity (positive) [m/s]

float **max_trans_vel**

Maximum translational velocity in any direction (positive) [m/s]

float **min_trans_vel**

Minimum translational velocity (positive) [m/s]

float **max_rot_vel**

Maximum absolute yawrate (positive) [rad/s]

float **min_rot_vel**

Minimum absolute yawrate (positive) [rad/s]

float **trans_stopped_vel**

Threshold for deciding when platform is stopped based on translational velocity (positive) [m/s]

float **rot_stopped_vel**

Threshold for deciding when platform is stopped based on rotational velocity (positive) [rad/s]

pilot.area_property_e

Enumeration

Describes a property of a *pilot.MapArea*.

enumerator DANGER

enumerator NO_HUMANS

enumerator DARK

enumerator LOUD

enumerator WET_FLOOR

enumerator DIRTY_FLOOR

enumerator DRAFTY

`pilot.battery_code_e`

Enumeration

Describes an error or warning regarding the battery.

`enumerator LOW`

If battery is below ~25% remaining.

`enumerator CRITICAL`

If battery is below ~10% remaining.

`enumerator OVERHEAT`

If battery temperature is above 50C.

`pilot.battery_type_e`

Enumeration

Denotes the battery type.

`enumerator AGM`

Lead battery with absorbent glass mat technology

`enumerator LFP`

Lithium iron phosphate battery

`pilot.charging_state_e`

Enumeration

Displays the charging state of the battery.

`enumerator NOT_CHARGING`

`enumerator IS_CHARGING`

`enumerator NO_CHARGER`

`enumerator BRAKES_OPEN`

`enumerator EM_STOP`

`enumerator ABORTED`

`enumerator FINISHED`

`pilot.drive_flags_e`

Enumeration

Flags that modify robot navigation or behavior.

`enumerator FIXED_ORIENTATION`

Flag to disable orientation optimization / modification (holonomic only). Will keep orientation from Road Map, even when the path itself is changed. Used to keep a specific orientation while moving.

enumerator IGNORE_FOOTPRINT

Flag to disable collision avoidance, use with care. Used to dock with external components where obstacles can be very close to the robot (inside footprint).

enumerator DISABLE_ROTATION

Disable rotation of the robot (forced zero control yawrate). Will prevent robot from rotating while moving, or rotating in place. Used to dock with external components where rotation would lead to damage of the robot or components. If rotation is necessary to reach the goal, a failure will be issued.

enumerator BACKWARD_OVERRIDE

Allows driving backwards even when normally prohibited. Used in special cases where unsafe backwards driving is necessary.

enumerator OMNI_DIRECTIONAL

Allows driving with arbitrary orientation by default (holonomic only). Will permit infinite yaw lookahead, such as to minimize total yaw motion. If not set, a platform will usually align with orientation of the road.

enumerator DISABLE_WHEEL_RESET

Disables automatic wheel resetting of the MPO-700. Used to increase positioning accuracy at certain stations.

enumerator RESET_WHEELS

Will reset the wheels of an MPO-700 at the (goal) station to their home positions.

pilot.drive_mode_e**Enumeration**

Specifies how the robot should navigate.

enumerator STRICT_PATH_FOLLOW

Follows a given path exactly, without trying to optimize it. If there is an obstacle the robot will wait.

enumerator RELAXED_PATH_FOLLOW

The given path is optimized while driving. Obstacles may be avoided, corners are cut.

enumerator FREE_PATH_FOLLOW

Same as RELAXED_PATH_FOLLOW except that in case of a blockage the robot is allowed to generate a new path by itself.

enumerator FREE_ROAMING

Will ignore the geometry of the path segment(s) and use the *GlobalPlanner* to create an actual path to follow.

enumerator DEFAULT

One of the above, depending on robot configuration, usually RELAXED_PATH_FOLLOW.

pilot.em_stop_state_e**Enumeration**

Different states of the Emergency Stop system.

enumerator FREE

Emergency Stop is not engaged, platform is free to move.

enumerator STOPPED

Emergency Stop is engaged, platform is prevented from moving.

enumerator CONFIRMED

Emergency Stop has been released manually, will go to FREE shortly.

pilot.event_code_e

Enumeration

Describes an event.

enumerator NEW_GOAL

enumerator GOAL_REACHED

enumerator GOAL_CANCELED

enumerator NEW_TASK

enumerator TASK_COMPLETED

enumerator TASK_FAILED

enumerator EXECUTION_PAUSED

enumerator EXECUTION_RESUMED

enumerator EXECUTION_CANCELED

pilot.event_t

Struct

event_t describes an event, without an attached timestamp.

The fields `module`, `code_type` and `code` uniquely define a specific event type that can occur.

Fields

event_type_e type

Event type, see [*pilot.event_type_e*](#).

string module

Module name that generated the event.

string code_type

Type name for the `code` field, usually the enum type name. For example: `pilot.safety_code_e`.

string code

Event code, usually an enum value. For example: `EMERGENCY_STOP`.

pilot.event_type_e

Enumeration

Describes the type of a [*pilot.event_t*](#).

enumerator ERROR

enumerator WARNING
enumerator NOTIFICATION

pilot.execution_state_e

Enumeration

Possible states of task execution.

enumerator RUNNING
Currently executing a task.
enumerator PAUSED
Execution has been paused.
enumerator FINISHED
Execution has finished.
enumerator CANCELED
Execution was canceled.

pilot.fleet_mode_e

Enumeration

Effect of received fleet information on path planning.

enumerator IGNORE
Plan like in single robot mode.
enumerator COMPLY
Only use occupied road segments if there is no other way.
enumerator OBEY
Pretend occupied road segments do not exist. This can cause a goal to be not reachable at all and the planning to fail.

pilot.float_param_t

Struct

float_param_t represents a `float` value with additional semantic information.

It can always be specified as just a `float` value, in which case `type` will be set to `CUSTOM`.

Fields

`param_type_e type`
Semantic type information, see *pilot.param_type_e*.

`float value`
The actual value, depending on `type` it may not be used.

pilot.goal_options_t

Struct

A set of options to modify how a goal is to be reached.

Fields

optional<float> max_velocity

Global velocity limit in [m/s], optional.

optional<float> max_time_stuck

How long to wait until aborting goal when stuck in [seconds] (default = infinite)

optional<drive_mode_e> drive_mode

Custom drive mode for the entire path, see [*pilot.drive_mode_e*](#).

optional<drive_mode_e> planner_mode

Custom drive mode for path planning (global planner)

set<drive_flags_e> drive_flags

Set of additional drive flags for final goal position. See [*pilot.drive_flags_e*](#).

int priority

Priority of the goal, can be used by fleet management.

pilot.keypad_state_t

Struct

An object of this type represents the current state of the keypad.

For each button there is a boolean member which is true when the button is pressed and false when it is not.

Fields

bool info_button

bool home_button

bool start_button

bool stop_button

bool break_release_button

bool digital_input[3]

An array entry is true if the corresponding digital input is active.

pilot.led_color_e

Enumeration

enumerator RED

enumerator GREEN

```
enumerator BLUE
enumerator WHITE
```

pilot.limit_reason_e

Enumeration

Possible reasons for limiting velocity or yawrate.

enumerator DEFAULT_MAX

Default maximum for the platform. (global configuration)

enumerator CUSTOM_MAX

Custom limit (at current position). (Road Map parameter, or via. *pilot.goal_options_t*)

enumerator ACCEL_MAX

Limited due to hitting maximum allowed acceleration / deceleration.

enumerator LOCAL_COST

Limited due to obstacles in path or close to path.

enumerator PATH_DEVIATION

Limited due to unexpected deviation from path.

enumerator ORIENTATION_DEVIATION

Limited due to unexpected deviation from target orientation.

enumerator CURVE_LIMIT

Limited due to maximum lateral acceleration.

enumerator WAITING

Limited due to waiting for a certain condition.

enumerator GOAL_MAX

Limited due to approaching final goal position.

enumerator LIMIT_AHEAD

Limited due to approaching a lower limit on the path ahead.

enumerator OBSTACLE

Limited due to obstacle blocking the path.

enumerator STOPPING

Limited due to trying to stop.

enumerator YAW_ERROR

Limited due to orientation deviation.

pilot.local_planner_state_e

Enumeration

Possible states of the *LocalPlanner*.

enumerator IDLE

Waiting for a job / goal.

enumerator WAITING

Waiting to move, due to several possible reasons, for example waiting for localization to initialize.

enumerator TRANSLATING

Moving in X / Y direction, can include rotation also.

enumerator ROTATING

Rotating in place, without X / Y movement.

enumerator ADJUSTING

Rotating in place while making small adjustments in X / Y position. For *differential* kinematics only.

enumerator TURNING

Rotating in place to turn around. For *differential* kinematics only.

enumerator STUCK

Unable to continue for now, due to obstacles in the path.

enumerator FINISHED

Current goal has been reached, waiting for new job / goal.

enumerator CANCELED

Current goal has been canceled, stopping now.

enumerator LOST

Current position could not be matched to the given path, requesting new path.

pilot.localization_mode_e

Enumeration

Possible modes of localization.

enumerator NONE

No localization running.

enumerator NO_MAP

No Grid Map available.

enumerator NO_INPUT

No sensor input available.

enumerator NO_ODOMETRY

No odometry available.

enumerator LOST

Unable to localize, position most likely wrong.

enumerator INITIALIZING

Performing initial localization, waiting for confidence to increase.

enumerator DEAD_RECKONING

Unable to localize at current position, extrapolating based on odometry.

enumerator MODE_1D

Partial localization in one direction, *DEAD_RECKONING* otherwise.

enumerator MODE_1D_YAW

Partial localization in one direction plus orientation, *DEAD_RECKONING* otherwise.

enumerator MODE_2D

Partial localization in X and Y direction, *DEAD_RECKONING* for orientation.

enumerator MODE_2D_YAW

Full localization in X and Y direction as well as orientation.

pilot.map_info_t

Struct

Unique identifier for a *Grid Map* or a *Road Map*.

string name

Map name.

long last_modified

Last modified time as Posix timestamp in microseconds.

string description

Map description. This member is not used for map identification.

bool same_as (map_info_t other) const

Returns true if name and last_modified are the same on both objects.

pilot.motion_mode_e

Enumeration

A set of motion modes of the robot, defining who is controlling the motion.

enumerator NONE

No motion is possible in this mode.

enumerator CUSTOM

Custom motion commands on topic `platform.velocity_cmd` control the robot.

enumerator JOYSTICK

Motion commands from a connected joystick control the robot.

enumerator AUTOMATIC

The *LocalPlanner* is controlling the robot, usually following a path generated by the *HybridPlanner*.

pilot.orientation_mode_e

Enumeration

Different modes affecting the orientation of the platform.

enumerator RELATIVE_ROAD

Specified orientation is relative to the *pilot.RoadSegment* in driving direction.

enumerator ABSOLUTE_ROAD

Same as RELATIVE_ROAD but the orientation is the same in both driving directions.

enumerator ABSOLUTE_MAP

Specified orientation is an absolute map orientation (ie. relative to the map coordinate system).

pilot.param_type_e

Enumeration

Semantic type of a parameter.

enumerator DEFAULT

The default value should be used.

enumerator CUSTOM

The custom value that is provided should be used.

enumerator OPTIMIZED

Similar to *CUSTOM*, means the value was optimized internally instead of specified by a user.

enumerator DISABLED

The feature corresponding to the parameter should be disabled.

enumerator IGNORE

The parameter should be ignored.

pilot.permission_e

Enumeration

Permissions that can be given to certain users. See also *vnx.permission_e*.

enumerator MOVE

Permission to move robot.

enumerator CHARGE

Permission to charge robot.

enumerator INITIALIZE

Permission to initialize localization.

enumerator RECORD_DATA

Permission to start a data recording.

enumerator REMOTE_CONTROL

Permission to control robot remotely.

enumerator RELAY_CONTROL

Permission to switch relays.

enumerator DISPLAY_CONTROL

Permission to display text on the LCD.

enumerator SAFETY_FIELD_CONTROL

Permission to switch the safety field.

enumerator CHANGE_GRIDMAP

Permission to change the active Grid Map.

enumerator CHANGE_ROADMAP

Permission to change the active Road Map.

enumerator UPLOAD_SCRIPT

Permission to upload / overwrite scripts.

enumerator EXECUTE_SCRIPT

Permission to execute scripts.

enumerator INTERVENE_SCRIPT

Permission to pause / resume scripts.

pilot.pilot_mode_e

Enumeration

A set of operating modes of the robot.

enumerator MAPPING

Creation of a new map or updating of an existing map.

enumerator NAVIGATION

Normal navigation mode, using localization with an existing map.

enumerator TELEOP

Basic operation mode, without localization and path planning.

enumerator REPLAY

Special simulation mode to recompute dynamic data based on recorded sensor data, for visualization purposes.

pilot.platform_type_e

Enumeration

Available platform types.

enumerator MP_400

Neobotix MP-400, see <https://www.neobotix-roboter.de/produkte/mobile-roboter/mobiler-roboter-mp-400>.

enumerator MP_500

Neobotix MP-500, see <https://www.neobotix-roboter.de/produkte/mobile-roboter/mobiler-roboter-mp-500>.

enumerator MPO_500

Neobotix MPO-500, see <https://www.neobotix-roboter.de/produkte/mobile-roboter/mobiler-roboter-mpo-500>.

enumerator MPO_700

Neobotix MPO-700, see <https://www.neobotix-roboter.de/produkte/mobile-roboter/mobiler-roboter-mpo-700>.

enumerator ROX_DIFF

enumerator ROX_TRIKE

enumerator ROX_MECA

enumerator ROX_ARGO

Variants of the modular Neobotix ROX, see <https://www.neobotix-roboter.de/produkte/mobile-roboter/mobiler-roboter-rox>.

enumerator CUSTOM

Custom platform.

pilot.polygon_t

Struct

polygon_t describes a geometric polygon in a certain frame of reference.

Fields

string **frame**

Coordinate system name, for example base_link.

vector<Vector2d> **points**

List of points forming the polygon, see *math.Vector2d*. The order of points does not matter, clock-wise or anti-clock-wise is both supported.

pilot.power_system_type_e

Enumeration

Specifies the voltage of the system.

enumerator POWER_24V

The system runs on 24 volts.

enumerator POWER_48V

The system runs on 48 volts.

pilot.radar_sample_t

Class

Basic information, dimensions and location of a robot. Used in *pilot.RadarScan*.

Fields

Hash64 **src_id**

unique id of platform

map_info_t **grid_map**

used grid map, see *pilot.map_info_t*

map_info_t **road_map**

used road map, see *pilot.map_info_t*

Vector3d **map_pose**

pose in the map (x, y, yaw) in m, m, rad

Vector3f **velocity**

relative to base_link (x, y, yaw) in m/s, m/s, rad/s

Footprint ***footprint**

see *pilot.Footprint*

PlatformInfo ***platform**

see *pilot.PlatformInfo*

pilot.safety_code_e

Enumeration

Describes an error of the safety system.

enumerator **NONE**

enumerator **SCANNER_STOP**

One of the safety scanners has stopped the platform.

enumerator **EMERGENCY_STOP**

One of the emergency stop buttons has been pressed.

enumerator **RADIO_EMERGENCY_STOP**

Radio emergency stop has been triggered.

pilot.safety_mode_e

Enumeration

Describes an operation mode which allows special (usually relaxed) safety measurements to be active.

enumerator **NONE**

enumerator **APPROACHING**

enumerator **DEPARTING**

enumerator **WORKING**

enumerator **HANDLING**

pilot.sensor_2d_range_t

Struct

Specifies the range in which a sensor measurement is considered valid.

Fields

float **min_angle**

normalized between -pi to +pi [rad]

float **max_angle**

normalized between -pi to +pi [rad]

float **min_range**

[m]

float **max_range**

[m]

Methods

bool **is_valid**(float *distance*) **const**

bool **is_within**(float *angle*, float *distance*) **const**

```
bool is_within_xy (Vector2f point) const  
bool is_within_point (laser_point_t point) const  
void add_margin (float delta_angle, float delta_range)
```

[pilot.system_error_e](#)

Enumeration

The type of a system error.

```
enumerator CHARGING_RELAY_ERROR  
enumerator BRAKE_RELEASE_BUTTON_ERROR  
    Brake release button is pressed.  
enumerator MOTOR_ERROR  
enumerator SAFETY_RELAY_ERROR  
enumerator POWER_RELAY_ERROR  
enumerator EM_STOP_SYSTEM_ERROR  
    Failure of an emergency stop button or a safety device (e.g. laser scanner).  
enumerator EDM_ERROR  
    Error reported by external device monitoring.  
enumerator MODE_SELECT_ERROR  
enumerator MAXIMUM_VELOCITY  
enumerator STANDSTILL_VIOLATION  
enumerator COMMUNICATION_ERROR  
    Communication error between system and safety monitoring.  
enumerator SCANNER_DIRTY
```

[pilot.vector_3f_param_t](#)

Struct

vector_3f_param_t represents a 3D float vector with additional semantic information.

It can always be specified as just a 3D float vector, in which case the *types* will be set to CUSTOM.

Fields

```
float_param_t x  
    X value, see pilot.float\_param\_t.  
float_param_t y  
    Y value, see pilot.float\_param\_t.  
float_param_t z  
    Z value, see pilot.float\_param\_t.
```

pilot.wait_reason_e

Enumeration

Possible reasons for not starting movement yet.

enumerator PAUSED

Job has been paused.

enumerator CANCELED

Job has been canceled.

enumerator CLEARANCE

Waiting for clearance from a fleet manager.

enumerator ODOMETRY

Waiting for odometry.

enumerator LOCALIZATION

Waiting for sufficient localization.

enumerator LOCAL_COST_MAP

Waiting for local cost map.

enumerator PATH_OPTIMIZER

Waiting for path optimizier.

pilot.base.canopen.emcy_register_e

Enumeration

CANopen emergency register.

enumerator GENERIC

enumerator CURRENT

enumerator VOLTAGE

enumerator TEMPERATURE

enumerator COMMUNICATION

enumerator DEVICE_PROFILE_SPECIFIC

enumerator MANUFACTURER_SPECIFIC

pilot.kinematics.KinematicsState

Class

Corresponding topic `kinematics.state`.

Fields

`long time`
POSIX timestamp [usec]

bool `is_vel_cmd`
Nonzero velocity commanded.

bool `is_moving`
Physical movement.

pilot.motors.elmo.MotorControlData

Class

Motor controller data for Elmo motors, which is published on topic `motors.motor_control_data`.

Fields

long `time`
POSIX timestamp [usec]

string `name`
Motor position

float `motor_current`
Current commanded by motor controller [A]

float `motor_velocity`
Velocity commanded by motor controller, without gearbox [rad/s]

float `motor_position`
Position commanded by motor controller, without gearbox [rad]

pilot.motors.elmo.MotorState

Class

Motor state for Elmo motors, which is published on topic `motors.state`.

Fields

long `time`
POSIX timestamp [usec]

map<string, motor_info_t> `motor_info`
Motor info, see [`pilot.motors.elmo.motor_info_t`](#).

pilot.motors.elmo.error_code_e

Enumeration

Contains Elmo controller MF and SR error codes.

enumerator `NONE`

enumerator `FAILURE`
A fault occurred while the motor was enabled.

enumerator TIMEOUT**enumerator UNDER_VOLTAGE**

Undervoltage: The amplifier is not measuring the minimum required voltage.

enumerator OVER_VOLTAGE

Ovvoltage: The amplifier is measuring a voltage which is higher than the allowed threshold.

enumerator SAFETY

Safety: One or two of the safety inputs are in safety state.

enumerator SHORT_CIRCUIT

Short Protection: The current has exceeded a range which is considered as a phase-to-phase or phase-to-ground short.

enumerator OVER_HEATING

Over-temperature: The drive is sensing a temperature which exceeds the maximum allowed temperature limit.

enumerator FEEDBACK_LOSS

Feedback loss: no match between encoder and Hall location.

enumerator PEAK_CURRENT_EXCEEDED

The peak current has been exceeded. Possible reasons are drive malfunction or bad tuning of the current controller.

enumerator DRIVE_INHIBIT

External Inhibit was triggered (INH/ENB).

enumerator HALL_SENSOR_ERROR

Two digital Hall sensors were changed at the same time. Error occurs because digital Hall sensors must be changed one at a time.

enumerator SPEED_TRACK_ERROR

Speed tracking error.

enumerator POSITION_TRACK_ERROR

Position tracking error.

enumerator INCONSISTENT_DATABASE

Cannot start because of inconsistent database. The type of database inconsistency is reflected in the status SR report.

enumerator HEARTBEAT_FAILURE

Heartbeat failure. Error occurs only if drive is set to abort under heartbeat failure in a CANopen network.

enumerator SERVO_DRIVE_FAULT

Amplifier problem: Indicates the problem that the power section of the drive has encountered.

enumerator SPEED_LIMIT_EXCEEDED

Overspeed indication.

enumerator ELECTRICAL_ZERO_NOT_FOUND

Failed to find the electrical zero of the motor in an attempt to start it with an incremental encoder and no digital Hall sensors. The reason may be that the applied motor current did not suffice for moving the motor from its position.

enumerator POSITION_LIMIT_EXCEEDED

Feedback is out of position limits.

enumerator MOTOR_STUCK

If the motor speed is lower than expected and the measured current is higher than expected, and if this is observed for more than expected msec, the motor is considered to be in the "Motor Stuck" state.

`enumerator UNKNOWN_FAILURE`

`pilot.motors.elmo.motor_info_t`

Class

Contains Elmo controller motor information.

Fields

`motor_state_e state`

Motor info, see [`pilot.motors.elmo.motor_state_e`](#).

`set<error_code_e> errors`

Set of error codes, see [`pilot.motors.elmo.error_code_e`](#).

`set<pilot.base.canopen.emcy_register_e> emcy_register`

CANopen emergency register, see [`pilot.base.canopen.emcy_register_e`](#).

`bool is_enabled() const`

`pilot.motors.elmo.motor_state_e`

Enumeration

Elmo controller status information.

`enumerator PRE_INITIALIZED`

`enumerator OPERATION_ENABLED`

`enumerator OPERATION_DISABLED`

`enumerator MOTOR_FAILURE`

`enumerator MOTOR_TIMEOUT`

vnx.Hash64

Hash64 is a 64-bit unsigned integer, which usually represents a CRC64 hash of a string, or sometimes simply a random number.

The specific hash function used is CRC-64/XZ, see <https://reveng.sourceforge.io/crc-catalogue/17plus.htm#crc.cat.crc-64-xz>.

vnx.JRPC_Error

Class

The error object that is contained in error messages returned by [*JSON-RPC Interface*](#).

Fields

int **code**

One of the following error codes:

```
static const int PARSE_ERROR = -32700;
static const int INVALID_REQUEST = -32600;
static const int METHOD_NOT_FOUND = -32601;
static const int INVALID_PARAMS = -32602;
static const int INTERNAL_ERROR = -32603;

static const int PERMISSION_DENIED = 403;
static const int EXCEPTION = 500;
```

string **message**

A short message to indicate what went wrong.

Exception ***data**

The actual exception object as thrown on the server side.

vnx.LogMsg

Class

Represents a log message.

Fields

static int **ERROR** = 1

static int **WARN** = 2

static int **INFO** = 3

static int **DEBUG** = 4

long **time**

int **level**

int **display_level** = 3

string **process**

string **module**

string **message**

Methods

string **get_output () const**

Returns a properly formatted line ready to be printed out.

vnx.ModuleInfo**Class**

Information about a module.

Fields

```
long time
    time stamp (virtual time) [usec]

Hash64 id
    unique module id

Hash64 src_mac
    source mac for publishing

string name
    module name

string type
    type name

long time_started
    POSIX timestamp [usec]

long time_idle
    current stats (see vnx_heartbeat_interval_ms) [usec]

long time_running
    current stats (see vnx_heartbeat_interval_ms) [usec]

long time_idle_total
    since start of module [usec]

long time_running_total
    since start of module [usec]

long num_async_pending
    number of pending async requests (waiting for returns)

long num_async_process
    number of async requests being processed right now

vector<string> sub_topics
    topic subscriptions

vector<string> pub_topics
    topic publishers

map<Hash64, Endpoint *> remotes
    map of connected processes (process id => endpoint)

TypeCode type_code
    module type code
```

Methods

```
double get_cpu_load() const
    0 to 1

double get_cpu_load_total() const
    0 to 1 (total average)
```

vnx.Object

Object represents an arbitrary object which is dynamically created and does not have a type. However an optional type name can be specified via a special field called `__type`. Internally it is a map of `string` keys to *vnx.Variant* values.

JSON

In JSON format an object is specified as follows:

```
{
    "__type": "optional.type.name.here",
    "field": "value",
    "some": 1234,
    "array": [1, 2, 3, 4],
    "nested": {
        "example": "value",
        ...
    },
    ...
}
```

Lua Script

In Lua Script an object can be created as follows:

```
{
    field = "value",
    some = 1234,
    array = {1, 2, 3, 4},
    nested = {
        example = "value",
        ...
    },
    ...
}
```

Native C++

In native C++ an object can be created as follows:

```
#include <vnx/vnx.h>

vnx::Object obj;
```

(continues on next page)

(continued from previous page)

```
obj["field"] = "value";
obj["some"] = 1234;
obj["array"] = std::vector<int>{1, 2, 3, 4};

vnx::Object nested;
nested["example"] = "value";
obj["nested"] = nested;

std::cout << obj << std::endl;
```

vnx.User

Class

A User object represents an entity who can authenticate in order to operate with a special set of permissions.

Fields

string **name**
The name of the user as used to log in.

string **hashed_password**
A salted SHA-256 hash of the password.

vector<string> **access_roles**
user access roles

set<string> **permissions**
additional, user specific permissions

vnx.Variant

Variant represents a value of arbitrary type which is assigned dynamically. It can be an integral (ie. `int`, `float`, ...), a `string`, a `vector<T>` of values, a `set<T>` of values, a `map<K, V>` of values or an *vnx.Object*, for example.

JSON

In JSON format a *Variant* could be anything, for example: `null`, `true`, `false`, a number, a `string`, an array of values or an `object`.

Lua Script

In Lua Script a *Variant* is a normal Lua variable and can be anything, for example: `nil`, `true`, `false`, a number, a `string`, an array of values, a `table` or an `object`.

Native C++

In native C++ a *Variant* can be created / assigned as follows:

```
#include <vnx/vnx.h>

vnx::Variant example(1234);

std::cout << example.to<int>() << std::endl;      // 1234

example = "value";

std::cout << example.to<std::string>() << std::endl;      // value

example = std::vector<int>{1, 2, 3, 4};

std::cout << example << std::endl;                  // [1, 2, 3, 4]

vnx::Object obj;
obj["field"] = "value";
example = obj;

std::cout << example << std::endl;                  // {"field": "value"}
```

vnx.access_role_e

Enumeration

A set of default access roles and their default permissions.

See also [vnx.permission_e](#).

enumerator DEFAULT

Default access role, mostly used for anonymous users. Does not provide any permissions by default, but that can be changed via configuration.

enumerator VIEWER

Read-only access role. Provides permissions: *VIEW*, *TIME_SYNC*.

enumerator OBSERVER

Same as *VIEWER*, with additional permission *CONST_REQUEST*.

enumerator USER

Same as *OBSERVER*, with additional permission *READ_CONFIG*.

enumerator INSTALLER

Same as *USER*, with additional permissions: *PUBLISH*, *WRITE_CONFIG*, *START*, *STOP*, *RESTART*, *SHUTDOWN*, *SELF_TEST*.

enumerator ADMIN

Same as *INSTALLER*, with additional permissions: *REQUEST*, *PROTECTED_CONFIG*, *PROXY_IMPORT*, *PROXY_EXPORT*, *PROXY_FORWARD*, *HOST_SHUTDOWN*, *LOCAL*.

vnx.permission_e

Enumeration

Generic permissions that can be given to certain users.

enumerator VIEW

Permission to subscribe to topics (and access process statistics).

enumerator CONST_REQUEST

Permission to execute const methods that do not explicitly set a permission.

enumerator PUBLISH

Permission to publish samples.

enumerator REQUEST

Permission to execute all methods that do not explicitly set a permission.

enumerator READ_CONFIG

Permission to read config values.

enumerator WRITE_CONFIG

Permission to change config values.

enumerator PROTECTED_CONFIG

Permission to read/write protected config values (like user passwords etc).

enumerator START

Permission to start new modules.

enumerator STOP

Permission to stop a running module.

enumerator RESTART

Permission to restart a module.

enumerator SHUTDOWN

Permission to shutdown the process.

enumerator HOST_SHUTDOWN

Permission to shutdown the host machine.

enumerator SELF_TEST

Permission to execute self tests.

vnx.opc_ua.DataChange

Class

DataChange reports data changes for a [vnx.opc_ua.monitored_item_t](#) as part of a [vnx.opc_ua.subscription_t](#) to an OPC-UA server.

Fields

int **subscription_id**

int **monitored_item_id**

Variant **value**

The new value of the item.

vnx.opc_ua.monitored_item_t**Struct**

Describes a monitored item as part of a subscription to an OPC-UA server. See [vnx.opc_ua.subscription_t](#).

Fields

monitoring_mode_e **monitoring_mode** = REPORTING

See [vnx.opc_ua.monitoring_mode_e](#).

double **sampling_interval** = 250

bool **discard_oldest** = true

int **queue_size** = 1

node_id_t **id**

The node to monitor. See [vnx.opc_ua.node_id_t](#).

bool **monitor_changes** = true

TopicPtr **output_data**

An optional topic on which to publish data changes as a [vnx.opc_ua.DataChange](#) object.

vnx.opc_ua.monitoring_mode_e**Enumeration**

Monitoring mode of a [vnx.opc_ua.monitored_item_t](#) inside a subscription to an OPC-UA server.

enumerator DISABLED

enumerator SAMPLING

enumerator REPORTING

vnx.opc_ua.node_id_t**Struct**

Identification of an OPC-UA node.

Examples:

- {"index": 1, "name": "question"}
- {"index": 1, "name": 1236}

Fields

short **index**

Namespace index.

Variant **name**

Node name.

vnx.opc_ua.security_mode_e

Enumeration

The security required by the *vnx.opc_ua.Proxy* from the Server.

enumerator ANY

Accept anything that is also supported by the server.

enumerator NONE

enumerator SIGN

enumerator SIGN_AND_ENCRYPT

vnx.opc_ua.security_policy_e

Enumeration

The security policy of an endpoint provided by the *vnx.opc_ua.Server*.

enumerator NONE

No security, no encryption. Can be used without a private key and even without a certificate

enumerator BASIC_128_RSA_15

(Deprecated!)

enumerator BASIC_256

(Deprecated!)

enumerator BASIC_256_SHA_256

enumerator AES_128_SHA_256_RSA_OAEP

vnx.opc_ua.subscription_t

Struct

Describes a subscription to an OPC-UA server.

Fields

```
double publishing_interval = 500
int lifetime = 10000
int max_keepalive = 10
int max_notifications_per_publish = 0
bool publishing_enabled = true
char priority = 0
vector<monitored_item_t> monitored_items
```

The list of monitored items associated with this subscription. See *vnx.opc_ua.monitored_item_t*.

TopicPtr **output_data**

An optional topic on which to publish data changes of all monitored items as a *vnx.opc_ua.DataChange* object.

`vnx.mqtt.export_t`

Struct

Definition of an MQTT export.

Fields

string **topic**

MQTT topic name.

bool **retained** = false

Tells the broker to buffer this message and deliver it to later subscribers.

qos_e **qos** = AT_LEAST_ONCE

Quality of service, see [`vnx.mqtt.qos_e`](#).

`vnx.mqtt.import_t`

Struct

Definition of an MQTT import.

Fields

TopicPtr **topic**

Internal topic name.

optional<string> **type**

Optional data type name for automatic conversion.

qos_e **qos** = AT_LEAST_ONCE

Quality of service, see [`vnx.mqtt.qos_e`](#).

`vnx.mqtt.last_will_t`

Struct

Definition of the MQTT client's *Last-Will* message.

Extends [`vnx.mqtt.export_t`](#).

Fields

string **message**

`vnx.mqtt.qos_e`

Enumeration

Quality of Service for an MQTT message.

```
enumerator FIRE_AND_FORGET
enumerator AT_LEAST_ONCE
enumerator EXACTLY_ONCE
```

8.3.4 Modules

GlobalPlanner

Module

The *GlobalPlanner* module supports path planning based on a global cost map, which is generated from a Grid Map. The module uses an *A** style path finding algorithm.

Most functions require permission `pilot.permission_e.MOVE`, see [`pilot.permission_e`](#).

Functions

Common Parameters

`goal_options_t options` Optional goal options, see [`pilot.goal_options_t`](#). If not specified will use default values.

`Hash64 job` Optional job id, to identify the new goal. If not specified (or set to zero) will generate a new random id. See [`vnx.Hash64`](#).

Asynchronous Move Functions

The following functions set / append a new goal while returning immediately.

```
void set_goal (PathPoint2D goal, goal_options_t options, Hash64 job)
    Sets a new goal using the provided pose and parameters in goal. Any pending goals are canceled beforehand.
    See pilot.PathPoint2D. Requires permission MOVE.
```

Synchronous Move Functions

The following functions return when the new goal is physically reached, or the goal was canceled.

```
void move_to (PathPoint2D goal, goal_options_t options, Hash64 job)
    Same as set_goal(...) but will block until goal is reached or canceled. See pilot.PathPoint2D. Requires
    permission MOVE.
```

HttpProxy

Module

The *HttpProxy* module provides a HTTP REST API, see *HTTP Interface*.

Most functions require a special permission, see *vnx.permission_e*.

Functions

void `publish` (Object *sample*, string *topic*)

Will publish a given *sample* on the specified *topic*. Requires permission `PUBLISH`. See *vnx.Object*.

Object `multi_request` (map<string, string> *paths*) `const`

Performs multiple /api/ HTTP requests in one. *paths* is a map from paths to output names, for example:
[["/topic/...", "topic1"], ...]

The result will be a *vnx.Object* containing the individual results, for example: {"topic1": { ... }, ...}

HybridPlanner

Module

The *HybridPlanner* module supports a combined path planning by taking into account a Road Map as well as a Grid Map. It automatically chooses to drive on the Road Map when possible, only to fall back to Grid Map based navigation if needed. As such it supports different ways to specify a goal, either by station name, by providing a modified station structure or by specifying an arbitrary pose.

Multiple goals can be specified, in which case the goals are traversed one after the other, without waiting or stopping at intermediate goals.

Most functions require permission `pilot.permission_e.MOVE`, see *pilot.permission_e*.

Functions

Common Parameters

goal_options_t `options` Optional goal options, see *pilot.goal_options_t*. If not specified will use default values.

If *options.planner_mode* is set to `FREE_ROAMING`, the HybridPlanner ignores the roadmap and uses free path planning.

Hash64 `job` Optional job id, to identify the new goal. If not specified (or set to zero) will generate a new random id. See *vnx.Hash64*.

Asynchronous Move Functions

The following functions set / append a new goal while returning immediately. If there is already a goal in progress, the `set_*` functions cancel it before setting the new goal (the call blocks until the robot stops) while the `append_*` functions append the new goal after the end of the path.

```
void set_goal (MapStation goal, goal_options_t options, Hash64 job)
    Sets a new goal using the provided pose and parameters in goal. Any pending goals are canceled beforehand.
    See pilot.MapStation. Requires permission MOVE.

void set_goal_station (string name, goal_options_t options, Hash64 job)
    Sets a new goal using the provided station name. Any pending goals are canceled beforehand. The station must
    exist in the current Road Map. Requires permission MOVE.

void set_goal_position (PathPoint2D goal, goal_options_t options, Hash64 job)
    Sets a new goal using the provided pose and parameters in goal. Similar to set_goal(). Any pending goals
    are canceled beforehand. See pilot.PathPoint2D. Requires permission MOVE.

void append_goal (MapStation goal, goal_options_t options, Hash64 job)
    Same as set_goal(...) but will not cancel active or pending goals. See pilot.MapStation. Requires per-
    mission MOVE.

void append_goal_station (string name, goal_options_t options, Hash64 job)
    Same as set_goal_station(...) but will not cancel active or pending goals. Requires permission MOVE.

void append_goal_position (PathPoint2D goal, goal_options_t options, Hash64 job)
    Same as set_goal_position(...) but will not cancel active or pending goals. See pilot.PathPoint2D.
    Requires permission MOVE.
```

Synchronous Move Functions

The following functions return when the new goal is physically reached, or the goal was canceled.

```
void move_to (MapStation goal, goal_options_t options, Hash64 job)
    Same as set_goal(...) but will block until goal is reached or canceled. See pilot.MapStation. Requires
    permission MOVE.

void move_to_station (string name, goal_options_t options, Hash64 job)
    Same as set_goal_station(...) but will block until goal is reached or canceled. Requires permission
    MOVE.

void move_to_position (PathPoint2D goal, goal_options_t options, Hash64 job)
    Same as set_goal_position(...) but will block until goal is reached or canceled. See pi-
    lot.PathPoint2D. Requires permission MOVE.
```

Asynchronous Move Sequence Functions

The following functions set / append a list of new goals while returning immediately. If there is already a goal in progress, the **set_*** functions cancel it before setting the new goal (the call blocks until the robot stops) while the **append_*** functions append the new goals after the end of the path.

```
void set_goals (vector<MapStation> goals, goal_options_t options, Hash64 job)
    Sets a list of new goals, similar to set_goal(...). See pilot.MapStation. Requires permission MOVE.

void set_goal_stations (vector<string> names, goal_options_t options, Hash64 job)
    Sets a list of new goals, similar to set_goal_station(...). The stations must exist in the current Road
    Map. Requires permission MOVE.

void set_goal_positions (vector<PathPoint2D> goals, goal_options_t options, Hash64 job)
    Sets a list of new goals, similar to set_goal_position(...). See pilot.PathPoint2D. Requires permission
    MOVE.
```

```
void append_goals (vector<MapStation> goals, goal_options_t options, Hash64 job)
    Appends a list of new goals, similar to append_goal(...). See pilot.MapStation. Requires permission MOVE.

void append_goal_stations (vector<string> names, goal_options_t options, Hash64 job)
    Appends a list of new goals, similar to append_goal_station(...). Requires permission MOVE.

void append_goal_positions (vector<PathPoint2D> goals, goal_options_t options, Hash64 job)
    Appends a list of new goals, similar to append_goal_position(...). See pilot.PathPoint2D. Requires permission MOVE.
```

Synchronous Move Sequence Functions

The following functions return when the last goal specified has been physically reached, or the goals were canceled.

```
void move_to_stations (vector<MapStation> goals, goal_options_t options, Hash64 job)
    Same as set_goals(...) but will block until last goal is reached or canceled. See pilot.MapStation. Requires permission MOVE.

void move_to_positions (vector<PathPoint2D> goals, goal_options_t options, Hash64 job)
    Same as set_goal_positions(...) but will block until last goal is reached or canceled. Requires permission MOVE.

void move_to_positions (vector<PathPoint2D> goals, goal_options_t options, Hash64 job)
    Same as set_goal_positions(...) but will block until last goal is reached or canceled. See pilot.PathPoint2D. Requires permission MOVE.
```

LocalPlanner

Module

The *LocalPlanner* module generates velocity commands to keep the robot on a given path or to reach a certain goal which is in reach. Usually the given path is generated by the *HybridPlanner* and depending on configuration further optimized by the *LocalPlanner*.

Most functions require a special permission, see [pilot.permission_e](#).

Functions

Common Parameters

goal_options_t options Optional goal options, see [pilot.goal_options_t](#). If not specified will use default values.

Hash64 job Optional job id, to identify the new goal. If not specified (or set to zero) will generate a new random id. See [vnx.Hash64](#).

General Functions

Path2D *get_path() const

Returns the current original map path being followed, if any. See [pilot.Path2D](#).

Path2D *get_optimized_path() const

Returns the current optimized map path being followed, if any. See [pilot.Path2D](#).

LocalPlannerState **get_state()** **const**
Returns the current planner state. See [pilot.LocalPlannerState](#).

Movement Functions

void **pause** (bool *em_stop*)
Will pause movement, similar to an EM stop, but with a soft deceleration. However, if *em_stop* is set to *true* will perform an emergency stop. *resume()* needs to be called to continue with the current or any new goal.

void **resume** ()
Will resume movement after having been paused, while adhering to acceleration limits.

void **await_goal()** **const**
Waits for the current goal to finish. If currently idle it returns immediately.

void **await_goal_ex** (Hash64 *job*) **const**
Waits for a specific goal to finish. Either the currently active or finished goal, or any in the future. If the specified goal has already been reached in the past and subsequently been superseeded by another, this function will fail, ie. it will wait forever. Ideally this function is called before setting the goal to be waited for.

void **cancel_goal** ()
Will cancel any pending goal and bring the robot to an immediate stop. Requires permission MOVE.

void **cancel_goal_await** ()
Same as *cancel_goal()* and then *await_goal()* in one call.

void **set_path** (Path2D **path*, goal_options_t *options*)
Sets a new path to be followed and returns immediately. Any pending goal is canceled beforehand. The path needs to be in map coordinates. See [pilot.Path2D](#). Requires permission MOVE.

void **update_path** (Path2D **path*, goal_options_t *options*)
Updates the current path to be followed without stopping or canceling the current goal. If the new path does not include the current position (to within some tolerance) a failure will be issued. The path needs to be in map coordinates and it's *job* id needs to match the current job id. See [pilot.Path2D](#). Requires permission MOVE.

void **follow_path** (Path2D **path*, goal_options_t *options*)
Same as *set_path(...)* but will block until the goal is reached or canceled. See [pilot.Path2D](#). Requires permission MOVE.

void **set_goal** (PathPoint2D *goal*, goal_options_t *options*, Hash64 *job*)
Sets a new goal position, either relative to the current position or an absolute map position. The new goal must be within reach, with a maximum distance depending on configuration, but usually around 1 m. See [pilot.PathPoint2D](#). Requires permission MOVE.

void **move_to** (PathPoint2D *goal*, goal_options_t *options*, Hash64 *job*)
Same as *set_goal(...)* but will block until the goal is reached or canceled. See [pilot.PathPoint2D](#). Requires permission MOVE.

MapServer

Module

The *MapServer* module handles all grid maps (see [Grid Map](#)) and road maps (see [Road Map](#)) that are uploaded to the platform. All uploaded maps are stored in a storage and can be retrieved at any time. Additionally, the currently active grid map and road map are remembered across restarts.

Maps are identified using a [pilot.map_info_t](#) object.

Some functions require permissions `pilot.permission_e.CHANGE_GRIDMAP` or `pilot.permission_e.CHANGE_ROADMAP`, see [`pilot.permission_e`](#).

Functions

`OccupancyMapData *get_grid_map() const`

Returns the currently active grid map.

`RoadMapData *get_road_map() const`

Returns the currently active road map.

`void set_grid_map(OccupancyMapData *map)`

Uploads a grid map to the map storage and immediately uses it for navigation.

Requires permission `permission_e.CHANGE_GRIDMAP`.

`void switch_grid_map(map_info_t map_info)`

Switches the current grid map for a different one from the map storage.

Requires permission `permission_e.CHANGE_GRIDMAP`.

`OccupancyMapData *download_grid_map(map_info_t map_info) const`

Returns the grid map identified by the `map_info` from the map storage.

`void delete_grid_map(map_info_t map_info)`

Deletes the given grid map from the map storage. If the map is the currently active map, it stays active.

`vector<map_info_t> get_grid_maps_info() const`

Returns a list of all grid maps present in the map storage.

`void set_road_map(RoadMapData *map)`

Uploads a road map to the map storage and immediately uses it for navigation.

Requires permission `permission_e.CHANGE_ROADMAP`.

`void switch_road_map(map_info_t map_info)`

Switches the current road map for a different one from the map storage.

Requires permission `permission_e.CHANGE_ROADMAP`.

`RoadMapData *download_road_map(map_info_t map_info) const`

Returns the road map identified by the `map_info` from the map storage.

`void delete_road_map(map_info_t map_info)`

Deletes the given road map from the map storage. If the map is the currently active map, it stays active.

`vector<map_info_t> get_road_maps_info() const`

Returns a list of all road maps present in the map storage.

PilotServer

Module

The `PilotServer` module is the control center of the robot, it handles switching between different modes such as mapping and navigation or automatic drive mode and teleoperation. In addition it allows to initialize the localization as well as start / stop a data recording.

Most functions require a special permission, see [`pilot.permission_e`](#).

Functions

PilotState **get_state () const**

Returns the current pilot state, see [pilot.PilotState](#).

PlatformInfo **get_platform_info () const**

Returns platform info, see [pilot.PlatformInfo](#).

void **set_pose_estimate (Vector3d pose, long time)**

Initializes the localization by providing a pose estimate (x, y, yaw) [m, m, rad] in map coordinates. Optionally a time stamp can be provided in [usec]. See [math.Vector3d](#). Requires permission INITIALIZE.

void **switch_pilot_mode (pilot_mode_e new_mode, Object config, bool keep_motion_mode = false)**

Switches the current pilot mode to new_mode. Platform has to be stationary. Optionally a user config can be provided with config, overriding default values. If keep_motion_mode is set to true, the current motion mode will not be changed automatically. See [pilot.pilot_mode_e](#) and [vnx.Object](#). Requires permission REMOTE_CONTROL.

void **switch_motion_mode (motion_mode_e new_mode)**

Switches the current motion mode to new_mode. Platform has to be stationary. See [pilot.motion_mode_e](#). Requires permission REMOTE_CONTROL.

void **switch_footprint (Footprint footprint)**

Sets a new footprint to use. Platform has to be stationary. See [pilot.Footprint](#). Requires permission REMOTE_CONTROL.

void **start_recording (string file_name)**

Starts a new data recording using the optional file name file_name. The recording will end up in the `~/pilot/user/data/` folder. If file_name is empty a default name will be chosen. In any case a timestamp will be appended to the file name to make it unique. Requires permission RECORD_DATA.

void **stop_recording ()**

Stops the current data recording process. Requires permission RECORD_DATA.

PlatformInterface

Interface

The *PlatformInterface* is an interface to the hardware abstraction module running on a platform. It allows access to hardware features such as digital inputs / outputs, analog inputs, power relays and the LCD.

Most functions require a special permission, see [pilot.permission_e](#).

Functions

Charging Functions

void **charge ()**

Attempts to charge the platform's batteries when connected to a charging station. Will return upon completion of charging process. In case of an error (such as no contact to the charging station) an exception will be thrown. Requires permission CHARGE.

void **start_charging ()**

Attempts to start the charging process at a charging station. Returns immediately in case of success, or throws an exception. Requires permission CHARGE.

```
void stop_charging()
```

Stops the charging process at a charging station. Returns immediately in case of success, or throws an exception. Requires permission CHARGE.

Input / Output Functions

```
void set_relay(int channel, bool state)
```

Switch a relay on or off, depending on state. *channel* is an index from 0 to 3 (in most cases) denoting the relay to switch. Requires permission RELAY_CONTROL.

```
void set_digital_output(int channel, bool state)
```

Switches a digital output to the specified state. *channel* is an index from 0 to 15 (in most cases) denoting the output pin. Requires permission RELAY_CONTROL as well as an installed *IOBoard*.

```
void set_display_text(string line)
```

Sets the first line on the LCD to the given text, for the next 30 seconds. Requires permission DISPLAY_CONTROL.

```
float read_analog_input(int channel)
```

Reads the voltage of an analog input in [V].

```
bool read_digital_input(int channel)
```

Reads the state of a digital input.

```
void wait_for_digital_input(int channel, bool state)
```

Waits for a digital input to reach the specified state. If it's already reached will return immediately. *channel* is an index from 0 to 15 (in most cases) denoting the input pin. Requires permission CONST_REQUEST as well as an installed *IOBoard*.

Other Functions

```
void shutdown()
```

Will terminate the process, shutdown the machine and turn off the platform. Cannot be undone except by turning on the platform by hand again. Requires permission SHUTDOWN_HOST, see [vnx.permission_e](#).

RelayBoardV3Node

Module

Handles communication with the RelayBoardV3.

In particular, this module sends the initial configuration to the board.

Options

Object **remote_config**

A complete configuration set for the RelayBoardV3, see [vnx.Object](#). It contains information about which modules to start and their individual configuration. If the sent configuration object differs from the last one received by the RelayBoardV3, it will stop all modules and start from a fresh state.

RoadMapPlanner

Module

The *RoadMapPlanner* module provides navigation support based on a *Road Map*.

Most functions require a special permission, see *pilot.permission_e*.

Options

`fleet_mode_e fleet_mode = COMPLY`

If and how the planner should react to occupancy messages from a fleet manager. See *pilot.fleet_mode_e*.

Functions

Common Parameters

`goal_options_t options` Optional goal options, see *pilot.goal_options_t*. If not specified will use default values.

`Hash64 job` Optional job id, to identify the new goal. If not specified (or set to zero) will generate a new random id. See *vnx.Hash64*.

General Functions

`RoadMapData *get_road_map() const`

Returns the currently used *Road Map*, if any. See *pilot.RoadMapData*.

`MapStation *find_station(string name) const`

Returns the corresponding map station by that name, if any. See *pilot.MapStation*.

`MapStation *find_closest_station(double max_distance, Pose2D *position) const`

Returns the map station closest to position in a radius of at most `max_distance` meters. `position` defaults to the current position if known.

Movement Functions

`void set_goal_station(string name, goal_options_t options, Hash64 job)`

Sets a new goal using the provided station name and returns immediately. Any pending goals are canceled beforehand. The station must exist in the current *Road Map*. Requires permission `MOVE`.

`void move_to_station(string name, goal_options_t options, Hash64 job)`

Same as `set_goal_station(...)` but will block until goal is reached or canceled. Requires permission `MOVE`.

SafetyInterface

Interface

The *SafetyInterface* is available on platforms with a safety control device (e.g. SICK FlexiSoft) managed by the control software. It provides high level access to certain safety features.

Most functions require a special permission, see [pilot.permission_e](#).

Functions

Safety Field Functions

`void set_safety_mode (safety_mode_e mode, uchar station_id = 0)`

Sets the current safety mode, including the station ID if needed for the given mode. See [pilot.safety_mode_e](#).

Requires permission SAFETY_MODE_CONTROL.

`void select_safety_field (uint field_id)`

Switches the currently active safety field to the one with the given ID. Use `field_id = 0` to disable or revert to preset. Requires permission SAFETY_FIELD_CONTROL.

TaskHandler

Module

The *TaskHandler* module allows the execution of a sequence of tasks, specified by a Lua script, which can either be written by hand or generated by using the graphical programming interface [TaskEditor](#).

See [Lua Script](#) for more information on how to write a script by hand.

Most functions require a special permission, see [pilot.permission_e](#).

Options

`string autostart = ""`

File to execute at startup.

`uint autostart_delay_ms = 5000`

how long to wait before autostart. [ms]

Functions

Common Parameters

Hash64 jobid Optional job id, to identify a new program execution. If not specified (or set to zero) will generate a new random id. See [vnx.Hash64](#).

Execute Functions

The following functions will cancel any running program beforehand, start to execute the new program and return immediately.

`void execute_file (string filename, vector<Variant> params, Hash64 jobid, bool blocking)`

Starts to execute the specified Lua script file, i.e. executes its main function with the given parameters. The `filename` is relative to the pilot home folder, e.g. `~/pilot/`. If `blocking` is set to `true`, the call blocks until the program is finished or aborted, otherwise it returns immediately. Requires permission EXECUTE_SCRIPT.

```
void execute_program(string program, vector<Variant> params, Hash64 jobid, bool blocking)
```

Starts to execute the given Lua script code, i.e. executes its main function with the given parameters. If *blocking* is set to `true`, the call blocks until the program is finished or aborted, otherwise it returns immediately. Requires permission `UPLOAD_SCRIPT`.

Intervene Functions

```
void cancel_program()
```

Will cancel a running program at the next opportunity, usually after finishing the current task. Requires permission `INTERVENE_SCRIPT`.

```
void pause_program()
```

Will pause a running program at the next opportunity, usually after finishing the current task. Requires permission `INTERVENE_SCRIPT`.

```
void resume_program()
```

Will resume executing a paused program. Requires permission `INTERVENE_SCRIPT`.

`vnx.JRPC_Proxy`

Module

The `vnx.JRPC_Proxy` module provides a way to connect to another `vnx.JRPC_Server`.

This module and the `vnx.Proxy` module share the same base and are largely identical in handling.

Options

The options are identical to `vnx.Proxy`.

Methods

The module has all methods of `vnx.Proxy` and additionally the following ones.

```
void select_service(string service_name)
```

For the rest of the connection, all method calls by `JSON-RPC Interface` that do not specify a module will be directed to `service_name`.

Supply an empty string or `.` to reset it to the proxy module itself.

`vnx.JRPC_Server`

Module

The `vnx.JRPC_Server` module provides a JSON RPC server to interface with the running process.

On the other end a `vnx.JRPC_Proxy` is needed to connect with the server.

This module and the `vnx.Server` module share the same base and are largely identical in handling.

Options

Identical to [vnx.Server](#).

vnx.Proxy

Module

The [vnx.Proxy](#) module provides a way to connect to another [vnx.Server](#).

Options

string **address**

URL to connect to, for example `localhost:1234` (TCP/IP) or `/tmp/mysocket.sock` for a UNIX socket. If no TCP port is specified (ie. `localhost`) a default port of `4444` is used. If no IP address is specified (ie. `:1234`) a default of `localhost` is used.

vector<string> **import_list**

List of topics to import from the sever.

vector<string> **export_list**

List of topics to export to the server.

vector<string> **forward_list**

List of services to forward to the server. Usually a service is a module's name. Requests to these services are then sent to the server who will forward them to the actual modules processing them. Return messages are automatically routed back to the clients.

map<Hash64, string> **tunnel_map**

Same as `forward_list` but with a different service address locally [*local alias => remote service name*]. Will forward requests on the Hash64 address to the string service on the server. string service names are converted to a Hash64 address internally. See [vnx.Hash64](#).

map<string, string> **import_map**

Same as `import_list` but will map to a different topic name locally. [*remote name -> local name*]

map<string, string> **export_map**

Same as `export_list` but will map to a different topic name on the server. [*local name -> remote name*]

bool **auto_import** = false

If to import all subscribed to topics from the server. Only used in special cases, since most of the time it leads to network loops.

bool **time_sync** = false

If to syncronize local virtual time with the server. Same as importing the `vnx.time_sync` topic. `vnx::get_time_*` () functions will then return a time which is in sync with the server.

bool **block_until_connect** = true

If to block client requests until first sucessful connect.

bool **block_until_reconnect** = false

If to block client requests until next sucessful reconnect, in case connection breaks down.

int **max_queue_ms** = 100

Maximum queue length in [ms] for receiving data internally, 0 = unlimited. If the network is overloaded at most this amount of data (in terms of latency) will be buffered internally before starting to drop messages. This queue is in addition to the internal TCP send buffer.

```
int max_queue_size = 1000
    Maximum queue size in [number of messages] for receiving data internally, 0 = unlimited. Similar to
    max_queue_ms.
int recv_buffer_size = 0
    TCP receive buffer size (0 = default) [bytes]
int send_buffer_size = 131072
    TCP send buffer size, bigger equals more latency in case of network overload. (0 = default) [bytes]
string default_access = "DEFAULT"
    Default access level for anonymous clients, see User Management. Only if use_authentication = true, otherwise
    there are no restrictions, ie. full permissions to any user.
```

vnx.Server

Module

The *vnx.Server* module provides a VNX server to interface with the running process.

On the other end a *vnx.Proxy* is needed to connect with the server.

Options

```
string address
    URL to bind to, for example 0.0.0.0:1234 (TCP/IP) or /tmp/mysocket.sock for a UNIX socket.
    UNIX sockets need to have a file ending of *.sock. If no TCP port is specified (ie. 0.0.0.0) a default port
    of 4444 is used. If no IP address is specified (ie. :1234) a default of localhost is used.

bool use_authentication = false
    If to require user authentication (login) to gain more permissions than default_access. If set to false
    any user has full permissions and no login is necessary.

vector<string> export_list
    List of topics to automatically export to all clients without them asking for it. Samples published on these topics
    are forwarded to all clients and re-published there.

int max_queue_ms = 100
    Maximum queue length in [ms] for receiving data internally, 0 = unlimited. If the network is overloaded at most
    this amount of data (in terms of latency) will be buffered internally before starting to drop messages. This queue
    is in addition to the internal TCP send buffer.

int max_queue_size = 1000
    Maximum queue size in [number of messages] for receiving data internally, 0 = unlimited. Similar to
    max_queue_ms.

int recv_buffer_size = 0
    TCP receive buffer size (0 = default) [bytes]

int send_buffer_size = 131072
    TCP send buffer size, bigger equals more latency in case of network overload. (0 = default) [bytes]

string default_access = "DEFAULT"
    Default access level for anonymous clients, see User Management. Only if use_authentication = true, otherwise
    there are no restrictions, ie. full permissions to any user.
```

vnx.opc_ua.Proxy

Module

The `vnx.opc_ua.Proxy` module provides a OPC-UA proxy to connect to a OPC-UA server.

It allows to call methods as well as read variables on the server.

Options

string `address`

OPC-UA server url to connect to, for example `opc.tcp://127.0.0.1:4840`.

string `username`

Optional user name for server login.

string `password`

Password for server login, required when `username` is set.

int `connect_interval_ms` = 1000

Interval to check for connection loss and reconnect. 0 to disable.

int `session_timeout_ms` = 86400000

Session timeout in ms.

int `keepalive_interval_ms` = 10000

Connectivity check interval in ms.

int `secure_channel_lifetime_ms` = 600000

Secure channel life time in ms.

string `certificate_file` = ""

Optional path to a client certificate file (DER format).

string `private_key_file` = ""

Optional path to a client private key file (DER format).

string `application_name` = "pilot.opc_ua.proxy"

Optional application name.

string `application_uri` = "urn:open62541:client:application"

Must match the URI in the certificate.

vector<string> `trust_list`

Server certificate trust files (CRL format). This only has an effect if a client certificate is configured.

security_mode_e `security_mode` = ANY

Required security for connections. See `vnx.opc_ua.security_mode_e`.

vector<subscription_t> `subscriptions`

A list of subscriptions to create on connect. See `vnx.opc_ua.subscription_t`.

bool `block_until_connect` = true

If to block client requests until first sucessful connect.

bool `block_until_reconnect` = false

If to block client requests until next sucessful reconnect, in case connection breaks down.

Functions

UA node ids are provided via a `pair<ushort, Variant>`, where the first value is the namespace index and the second value either an integer or a string.

In case a UA method returns more than one value the Variant return value will contain an array of variants, ie. `vector<Variant>`. See also [vnx.Variant](#).

Variant `call` (string *method*, vector<Variant> *args*) `const`

Calls a global *method* with the given arguments and returns the result. The implicit object for this call is `UA_NS0ID_OBJECTSFOLDER`. Requires permission `vnx.opc_ua.permission_e.CALL`.

Variant `object_call` (pair<ushort, Variant> *object*, string *method*, vector<Variant> *args*) `const`

Calls a method on the *object* with the given arguments and returns the result. Requires permission `vnx.opc_ua.permission_e.CALL`.

Variant `read_variable` (pair<ushort, Variant> *node*) `const`

Reads the value of a given variable *node*.

Variant `read_object_variable` (pair<ushort, Variant> *object*, string *variable*) `const`

Reads the value of a given *variable* of the *object*.

void `write_variable` (pair<ushort, Variant> *node*, Variant *value*)

Writes a value to the given global variable *node*. Requires permission `vnx.opc_ua.permission_e.WRITE`.

void `write_object_variable` (pair<ushort, Variant> *object*, string *variable*, Variant *value*)

Writes a value to the given *variable* of the *object*. Requires permission `vnx.opc_ua.permission_e.WRITE`.

void `browse_all` ()

Finds all available objects and variables on the server. Will be done automatically on every connect and re-connect.

vnx.opc_ua.Server

Module

The `vnx.opc_ua.Server` module provides a OPC-UA server to interface with the running process.

Depending on the configuration different modules and topics are offered on the OPC-UA interface.

Options

string `custom_hostname`

An optional custom host name.

uint `port` = 4840

The port number the server binds to.

set<string> `export_services`

List of modules to offer on the OPC-UA interface as objects.

set<string> `export_topics`

List of topics to offer on the OPC-UA interface as variables.

string `certificate_file`

Optional path to a server certificate file (DER format).

```

string private_key_file
    Optional path to a server private key file (DER format).

string application_name = "pilot.opc_ua.server"
    Optional application name.

string application_uri = "urn:open62541.server.application"
    Must match the URI in the certificate.

vector<string> trust_list
    Client certificate trust files (CRL format). This only has an effect if a server certificate is configured.

bool add_insecure_discovery = true
    Allow discovery with security policy NONE even if not configured. This is necessary if you want a secure server (i.e. no NONE policy) that can still be discovered by a client. If security policy NONE is configured, this option has no effect.

vector<security_policy_e> security_policies
    A list of the security/encryption policies to provide. Most policies require a certificate and a private key. See vnx.opc\_ua.security\_policy\_e

vector<pair<node_id_t, Variant>> variables
    A list of writable variables to create. For each variable a vnx.opc\_ua.node\_id\_t and an initial value is expected. The type of the variable is guessed from the initial value.

map<string, node_id_t> error_variables
    Create optional error variables for exported services where error messages from their method calls are published. This can be necessary, since OPC-UA does not support returning error messages.

bool use_authentication
    If to require authentication from clients. If set to false, any connected user has full permissions.

bool allow_anonymous_access
    Allow access without authentication.

string default_access
    Default access level for anonymous clients, see User Management. This only has an effect if use_authentication is set to true.

```

vnx.mqtt.Proxy

Module

The `vnx.mqtt.Proxy` module provides a way to connect to an *MQTT* broker.

Options

```

string address = "tcp://localhost:1883"
    Address of the MQTT broker to connect to.

string client_id = "vnx-mqtt"
    Unique client id.

map<vnx.TopicPtr, string> export_map
    Mapping of internal topics to be mapped to MQTT topics.

map<vnx.TopicPtr, export_t> export_map_ex
    Same as export_map but with more options. See vnx.mqtt.export\_t.

```

```
map<string, vnx.TopicPtr> import_map
    Mapping of MQTT topics to be mapped to internal topics. MQTT-style wildcards are allowed, but only without overlaps.

map<string, import_t> import_map_ex
    Same as import_map but with more options. See vnx.mqtt.import_t.

optional<last_will_t> last_will
    Optionally sets the MQTT Last-Will message. See vnx.mqtt.last_will_t.

string topic_prefix = ""
    Optional prefix that will be prepended to all MQTT topic names.

string username
    Optional user name for authentication.

string password
    Optional password for authentication.

int connect_timeout = 1

int keepalive_interval = 60

bool clean_session = true
    If broker should always create a new session.

bool wait_for_ack = true
    Every message must be acknowledged before the next can be sent

int connect_interval_ms = 1000
    Interval in which to attempt to connect.
```

8.3.5 Topics

Topics are names under which data samples are published. Any number of subscribers can then receive the published data.

Topics are arranged in a tree and it is possible to subscribe to a whole sub-tree. For example a subscription to `input` will receive samples from `input.joystick.data` as well as `input.velocity_cmd`.

Topics do not have an assigned data type, but in most cases only a specific data type is published.

input

```
input.joystick.data [pilot.JoyData] Joystick input samples.  
input.velocity_cmd [pilot.VelocityCmd] Joystick velocity commands.
```

kinematics

```
kinematics.drive_can.errors CAN error frames.  
kinematics.drive_can.receive CAN messages from the drives to the motor module.  
kinematics.drive_can.send CAN messages from motor module to the drives.  
kinematics.drive_cmd Drive command input for the motors module. Dependent on kinematics can contain position and velocity in SI units.
```

kinematics.drive_state Feedback from the motor module. Dependent on kinematics can contain position, velocity and torque in SI units.

kinematics.drive_state_raw Same as `kinematics.drive_state` but using encoder increments as the units.

kinematics.state [`pilot.kinematics.KinematicsState`] Simplified indicator of the current state of the kinematics.

kinematics.velocity_cmd [`pilot.VelocityCmd`] Commanded velocity input for the kinematics module.

local_planner

local_planner.state [`pilot.LocalPlannerState`] *LocalPlanner* state updates.

local_planner.target_pose [`pilot.Pose2D`] *LocalPlanner* target pose updates.

localization

localization.map_tile [`pilot.OccupancyMapData`] Localization map tile, ie. current section of the *Grid Map*.

localization.particles [`pilot.PoseArray2D`] Localization particle swarm in map coordinates.

localization.status [`pilot.LocalizationStatus`] Localization status updates.

mapping

mapping.grid_map [`pilot.OccupancyMapData`] New *Grid Map* created by mapping.

mapping.pose_graph [`pilot.RoadMapData`] Mapping pose graph.

motors

motors.motor_control_data [`pilot.motors.elmo.MotorControlData`] Motor data depending on the controller manufacturer.

motors.state [`pilot.motors.elmo.MotorState`] Motor state depending on the controller manufacturer.

navigation

navigation.footprint [`pilot.Footprint`] Platform footprint.

navigation.global_cost_map [`pilot.CostMapData`] Global cost map.

navigation.global_path [`pilot.Path2D`] Current global path in map coordinates.

navigation.grid_map [`pilot.OccupancyMapData`] Current *Grid Map*.

navigation.initial_pose [`pilot.Pose2D`] Pose estimates to initialize Localization.

navigation.local_cost_map [`pilot.CostMapData`] Local cost map.

navigation.local_cost_map_overlay [`pilot.CostMapData`] Local cost map with global cost map overlaid.

navigation.local_path [`pilot.Path2D`] Current optimized local path in `odom` coordinates.

`navigation.map_match` [*pilot.MapMatch*] Current map match.
`navigation.map_pose` [*pilot.Pose2D*] Current map pose in map coordinates.
`navigation.new_goal_pose` [*pilot.Pose2D*] New goals can be published here.
`navigation.new_grid_map` [*pilot.OccupancyMapData*] New *Grid Map* can be published here.
`navigation.new_road_map` [*pilot.RoadMapData*] New *Road Map* can be published here.
`navigation.odom_pose` [*pilot.Pose2D*] Current local pose in `odom` coordinates.
`navigation.road_map` [*pilot.RoadMapData*] Current *Road Map*.
`navigation.velocity_cmd` [*pilot.VelocityCmd*] Velocity commands from *LocalPlanner*.

network

`network.beacons` [*pilot.Beacon*] Beacons for fleet management.

platform

`platform.battery_state` [*pilot.BatteryState*] Battery state updates.
`platform.emergency_state` [*pilot.EmergencyState*] Emergency state updates.
`platform.events` [*pilot.Event, pilot.Incident*] Generic events.
`platform.incidents` [*pilot.Incident*] External incidents, to be handled by *PilotServer*.
`platform.active_incidents` [*pilot.ActiveIncidents*] List of currently active incidents.
`platform.info` [*pilot.PlatformInfo*] Static platform info.
`platform.odometry` [*pilot.Odometry*] Odometry samples.
`platform.pilot_state` [*pilot.PilotState*] Pilot state updatas.
`platform.system_state` [*pilot.SystemState*] System state update.
`platform.safety_state` [*pilot.SafetyState*] Safety state update.
`platform.power_state` [*pilot.PowerState*] System power state.
`platform.relay_board_data` [*pilot.RelayBoardData, pilot.RelayBoardV3Data*] RelayBoard data.
`platform.velocity_cmd` [*pilot.VelocityCmd*] Custom velocity commands.

sensors

`sensors.point_cloud.*` [*pilot.PointCloud2D, pilot.LaserPointCloud*] Laser point clouds in `odom` coordinates.

task_handler

`task_handler.current_task` [*pilot.ExecutionState*] Current task being executed by *TaskHandler* module.
`task_handler.current_event_task` [*pilot.ExecutionState*] Current event task being executed by *TaskHandler* module.
`task_handler.execution_history` [*pilot.ExecutionHistory*] Execution history of *TaskHandler* module.

task_handler.event_history [*pilot.ExecutionHistory*] Event execution history of *TaskHandler* module.

task_handler.execution_error [*pilot.ExecutionError*] Current execution error, if any.

tf

tf.map.odom [*pilot.Pose2D*] Localization updates.

tf.odom.base_link [*pilot.Odometry*] Odometry samples.

tfd.map

tfd.map.local_planner.target_pose [*pilot.Pose2D*] Same as *local_planner.target_pose* but in map coordinates.

tfd.map.navigation.local_path [*pilot.Pose2D*] Same as *navigation.local_path* but in map coordinates.

tfd.map.sensors.point_cloud.* [*pilot.Pose2D*] Same as *sensors.point_cloud.** but in map coordinates.

vnx

vnx.log_out [*vnx.LogMsg*] Terminal log messages.

vnx.module_info [*vnx.ModuleInfo*] Module info updates.

vnx.recorder_status [*vnx.RecorderStatus*] Status information of the data recorder, when active.

genindex

genindex

INDEX

A

auto_charge (*C++ function*), 73

B

basic::Transform3D::frame (*C++ member*), 80
basic::Transform3D::matrix (*C++ member*),
80

basic::Transform3D::parent (*C++ member*),
80
basic::Transform3D::time (*C++ member*), 80
block (*C++ function*), 72

C

call () (*built-in function*), 39
cancel_goal (*C++ function*), 70
charge (*C++ function*), 71

E

execute (*C++ function*), 74

F

find_closest_station (*C++ function*), 72
find_closest_station_name (*C++ function*), 72
find_station (*C++ function*), 72

G

get_battery_remaining (*C++ function*), 72
get_battery_voltage (*C++ function*), 72
get_position (*C++ function*), 72
get_time_micros (*C++ function*), 72
get_time_millis (*C++ function*), 72
get_time_sec (*C++ function*), 72

I

is_charging (*C++ function*), 72

L

log_error (*C++ function*), 72
log_info (*C++ function*), 72
log_warn (*C++ function*), 72

M

move (*C++ function*), 70
move_to (*C++ function*), 70
move_to_position (*C++ function*), 70
move_to_station (*C++ function*), 70
move_towards (*C++ function*), 70

O

on_battery_critical (*C++ function*), 75
on_battery_low (*C++ function*), 75
on_digital_input_off (*C++ function*), 75
on_digital_input_on (*C++ function*), 75
on_em_reset (*C++ function*), 75
on_em_stop (*C++ function*), 75
on_joystick_button_pressed (*C++ function*),
75
on_joystick_button_released (*C++ function*),
75
on_keypad_button_pressed (*C++ function*), 75
on_keypad_button_released (*C++ function*), 75
on_scanner_stop (*C++ function*), 75
opc_ua_call (*C++ function*), 73
opc_ua_read (*C++ function*), 73
opc_ua_read_global (*C++ function*), 74
opc_ua_write (*C++ function*), 74
opc_ua_write_global (*C++ function*), 74

P

pilot::ActiveIncidents::events (*C++ member*), 81
pilot::ActiveIncidents::time (*C++ member*), 81
pilot::area_property_e::DANGER (*C++ enumerator*), 103
pilot::area_property_e::DARK (*C++ enumerator*), 103
pilot::area_property_e::DIRTY_FLOOR
(*C++ enumerator*), 103
pilot::area_property_e::DRAFTY (*C++ enumerator*), 103
pilot::area_property_e::LOUD (*C++ enumerator*), 103

pilot::area_property_e::NO_HUMANS (C++ enumerator), 103
pilot::area_property_e::WET_FLOOR (C++ enumerator), 103
pilot::base::canopen::emcy_register_e::COMMUNICATOR (C++ enumerator), 104
pilot::base::canopen::emcy_register_e::CURRENT (C++ enumerator), 104
pilot::base::canopen::emcy_register_e::DEVICE_TYPE (C++ enumerator), 104
pilot::base::canopen::emcy_register_e::ENERGY:CostMapData::PROHIBITED (C++ member), 82
pilot::base::canopen::emcy_register_e::MANUFACTURER_SPECIFIC (C++ enumerator), 117
pilot::base::canopen::emcy_register_e::MANUFACTURER_SPECIFIC_to_float (C++ function), 82
pilot::base::canopen::emcy_register_e::TEMPERATURE:TempMapData::to_rgba (C++ function), 83
pilot::base::canopen::emcy_register_e::VOLTAGE:CostMapData::to_rgba_image (C++ function), 82
pilot::battery_code_e::CRITICAL (C++ enumerator), 104
pilot::battery_code_e::LOW (C++ enumerator), 104
pilot::battery_code_e::OVERHEAT (C++ enumerator), 104
pilot::battery_type_e::AGM (C++ enumerator), 104
pilot::battery_type_e::LFP (C++ enumerator), 104
pilot::BatteryState::capacity (C++ member), 81
pilot::BatteryState::charge (C++ member), 81
pilot::BatteryState::current (C++ member), 81
pilot::BatteryState::design_capacity (C++ member), 81
pilot::BatteryState::module_count (C++ member), 82
pilot::BatteryState::remaining (C++ member), 81
pilot::BatteryState::temperature (C++ member), 82
pilot::BatteryState::time (C++ member), 81
pilot::BatteryState::type (C++ member), 82
pilot::BatteryState::voltage (C++ member), 81
pilot::Beacon::time (C++ member), 82
pilot::charging_state_e::ABORTED (C++ enumerator), 104
pilot::charging_state_e::BRAKES_OPEN (C++ enumerator), 104
pilot::charging_state_e::EM_STOP (C++ enumerator), 104
pilot::charging_state_e::FINISHED (C++ enumerator), 104
pilot::charging_state_e::IS_CHARGING (C++ enumerator), 104
pilot::charging_state_e::NO_CHARGER (C++ enumerator), 104
pilot::charging_state_e::NOT_CHARGING (C++ enumerator), 104
pilot::CostMapData::cost (C++ member), 82
pilot::CostMapData::to_rgba (C++ function), 83
pilot::CostMapData::to_rgba_image (C++ function), 82
pilot::CostMapData::to_rgba_mono (C++ function), 83
pilot::CostMapData::to_rgba_mono_image (C++ function), 83
pilot::CostMapData::UNKNOWN (C++ member), 82
pilot::drive_flags_e::BACKWARD_OVERRIDE (C++ enumerator), 105
pilot::drive_flags_e::DISABLE_ROTATION (C++ enumerator), 105
pilot::drive_flags_e::DISABLE_WHEEL_RESET (C++ enumerator), 105
pilot::drive_flags_e::FIXED_ORIENTATION (C++ enumerator), 104
pilot::drive_flags_e::IGNORE_FOOTPRINT (C++ enumerator), 104
pilot::drive_flags_e::OMNI_DIRECTIONAL (C++ enumerator), 105
pilot::drive_flags_e::RESET_WHEELS (C++ enumerator), 105
pilot::drive_mode_e::DEFAULT (C++ enumerator), 105
pilot::drive_mode_e::FREE_PATH_FOLLOW (C++ enumerator), 105
pilot::drive_mode_e::FREE_ROAMING (C++ enumerator), 105
pilot::drive_mode_e::RELAXED_PATH_FOLLOW (C++ enumerator), 105
pilot::drive_mode_e::STRICT_PATH_FOLLOW (C++ enumerator), 105
pilot::em_stop_state_e::CONFIRMED (C++ enumerator), 105
pilot::em_stop_state_e::FREE (C++ enumerator), 105
pilot::em_stop_state_e::STOPPED (C++ enumerator), 105

pilot::EmergencyState::code (*C++ member*), 83
 pilot::EmergencyState::state (*C++ member*), 83
 pilot::EmergencyState::time (*C++ member*), 83
 pilot::Event::event (*C++ member*), 83
 pilot::Event::info (*C++ member*), 83
 pilot::Event::time (*C++ member*), 83
 pilot::event_code_e::EXECUTION_CANCELED (*C++ enumerator*), 106
 pilot::event_code_e::EXECUTION_PAUSED (*C++ enumerator*), 106
 pilot::event_code_e::EXECUTION_RESUMED (*C++ enumerator*), 106
 pilot::event_code_e::GOAL_CANCELED (*C++ enumerator*), 106
 pilot::event_code_e::GOAL_REACHED (*C++ enumerator*), 106
 pilot::event_code_e::NEW_GOAL (*C++ enumerator*), 106
 pilot::event_code_e::NEW_TASK (*C++ enumerator*), 106
 pilot::event_code_e::TASK_COMPLETED (*C++ enumerator*), 106
 pilot::event_code_e::TASK_FAILED (*C++ enumerator*), 106
 pilot::event_t::code (*C++ member*), 106
 pilot::event_t::code_type (*C++ member*), 106
 pilot::event_t::module (*C++ member*), 106
 pilot::event_t::type (*C++ member*), 106
 pilot::event_type_e::ERROR (*C++ enumerator*), 106
 pilot::event_type_e::NOTIFICATION (*C++ enumerator*), 107
 pilot::event_type_e::WARNING (*C++ enumerator*), 106
 pilot::execution_state_e::CANCELED (*C++ enumerator*), 107
 pilot::execution_state_e::FINISHED (*C++ enumerator*), 107
 pilot::execution_state_e::PAUSED (*C++ enumerator*), 107
 pilot::execution_state_e::RUNNING (*C++ enumerator*), 107
 pilot::ExecutionError::error_message (*C++ member*), 84
 pilot::ExecutionError::jobid (*C++ member*), 84
 pilot::ExecutionError::program_name (*C++ member*), 84
 pilot::ExecutionError::stack (*C++ member*), 84
 pilot::ExecutionError::time (*C++ member*), 84
 pilot::ExecutionHistory::history (*C++ member*), 84
 pilot::ExecutionState::is_minor (*C++ member*), 85
 pilot::ExecutionState::jobid (*C++ member*), 84
 pilot::ExecutionState::program_name (*C++ member*), 84
 pilot::ExecutionState::stack (*C++ member*), 84
 pilot::ExecutionState::status (*C++ member*), 84
 pilot::ExecutionState::task (*C++ member*), 84
 pilot::ExecutionState::time (*C++ member*), 84
 pilot::ExecutionState::time_ended (*C++ member*), 85
 pilot::ExecutionState::time_started (*C++ member*), 84
 pilot::fleet_mode_e::COMPLY (*C++ enumerator*), 107
 pilot::fleet_mode_e::IGNORE (*C++ enumerator*), 107
 pilot::fleet_mode_e::OBEY (*C++ enumerator*), 107
 pilot::float_param_t::type (*C++ member*), 107
 pilot::float_param_t::value (*C++ member*), 107
 pilot::Footprint::points (*C++ member*), 85
 pilot::GlobalPlanner::move_to (*C++ function*), 130
 pilot::GlobalPlanner::set_goal (*C++ function*), 130
 pilot::goal_options_t::drive_flags (*C++ member*), 108
 pilot::goal_options_t::drive_mode (*C++ member*), 108
 pilot::goal_options_t::max_time_stuck (*C++ member*), 108
 pilot::goal_options_t::max_velocity (*C++ member*), 108
 pilot::goal_options_t::planner_mode (*C++ member*), 108
 pilot::goal_options_t::priority (*C++ member*), 108
 pilot::GridMapData::get_frame_to_grid (*C++ function*), 85
 pilot::GridMapData::get_grid_to_frame (*C++ function*), 85
 pilot::GridMapData::get_info (*C++ func-*

tion), 86
pilot::GridMapData::last_modified (C++ member), 85
pilot::GridMapData::name (C++ member), 85
pilot::GridMapData::orientation (C++ member), 85
pilot::GridMapData::origin (C++ member), 85
pilot::GridMapData::same_as (C++ function), 85
pilot::GridMapData::scale (C++ member), 85
pilot::GridMapData::transform (C++ function), 85
pilot::HttpProxy::multi_request (C++ function), 131
pilot::HttpProxy::publish (C++ function), 131
pilot::HybridPlanner::append_goal (C++ function), 132
pilot::HybridPlanner::append_goal_position, 132
pilot::HybridPlanner::append_goal_position, 133
pilot::HybridPlanner::append_goal_station, 132
pilot::HybridPlanner::append_goal_station, 133
pilot::HybridPlanner::append_goals (C++ function), 132
pilot::HybridPlanner::move_to (C++ function), 132
pilot::HybridPlanner::move_to_position (C++ function), 132
pilot::HybridPlanner::move_to_positions (C++ function), 133
pilot::HybridPlanner::move_to_station (C++ function), 132
pilot::HybridPlanner::move_to_stations (C++ function), 133
pilot::HybridPlanner::move_to_stations (C++ function), 133
pilot::HybridPlanner::set_goal (C++ function), 131
pilot::HybridPlanner::set_goal_position (C++ function), 132
pilot::HybridPlanner::set_goal_positions, 132
pilot::HybridPlanner::set_goal_station (C++ function), 132
pilot::HybridPlanner::set_goal_stations (C++ function), 132
pilot::HybridPlanner::set_goals (C++ function), 132
pilot::Incident::is_active (C++ member), 86
pilot::Incident::is_cleared (C++ member), 86
pilot::Incident::timeout_ms (C++ member), 86
pilot::JoyData::axes (C++ member), 86
pilot::JoyData::buttons (C++ member), 86
pilot::JoyData::deadzone (C++ member), 86
pilot::JoyData::time (C++ member), 86
pilot::keypad_state_t::break_release_button (C++ member), 108
pilot::keypad_state_t::digital_input (C++ member), 108
pilot::keypad_state_t::home_button (C++ member), 108
pilot::keypad_state_t::info_button (C++ member), 108
pilot::keypad_state_t::start_button (C++ member), 108
pilot::keypad_state_t::stop_button (C++ member), 108
pilot::kinematics::KinematicsState::is_moving (C++ member), 118
pilot::kinematics::KinematicsState::is_vel_cmd (C++ member), 117
pilot::kinematics::KinematicsState::time (C++ member), 117
pilot::LaserPointCloud::intensity (C++ member), 87
pilot::led_color_e::BLUE (C++ enumerator), 109
pilot::led_color_e::GREEN (C++ enumerator), 108
pilot::led_color_e::RED (C++ enumerator), 108
pilot::led_color_e::WHITE (C++ enumerator), 109
pilot::limit_reason_e::ACCEL_MAX (C++ enumerator), 109
pilot::limit_reason_e::CURVE_LIMIT (C++ enumerator), 109
pilot::limit_reason_e::CUSTOM_MAX (C++ enumerator), 109
pilot::limit_reason_e::DEFAULT_MAX (C++ enumerator), 109
pilot::limit_reason_e::GOAL_MAX (C++ enumerator), 109
pilot::limit_reason_e::LIMIT_AHEAD (C++ enumerator), 109
pilot::limit_reason_e::LOCAL_COST (C++ enumerator), 109
pilot::limit_reason_e::OBSTACLE (C++ enumerator), 109
pilot::limit_reason_e::ORIENTATION_DEVIATION

pilot::limit_reason_e::PATH_DEVIATION (C++ enumerator), 109	(C++ member), 88
pilot::limit_reason_e::STOPPING (C++ enumerator), 109	pilot::LocalizationStatus::num_points_total (C++ member), 88
pilot::limit_reason_e::WAITING (C++ enumerator), 109	pilot::LocalizationStatus::score (C++ member), 88
pilot::limit_reason_e::YAW_ERROR (C++ enumerator), 109	pilot::LocalizationStatus::sensors (C++ member), 88
pilot::local_planner_state_e::ADJUSTING (C++ enumerator), 110	pilot::LocalizationStatus::std_dev (C++ member), 88
pilot::local_planner_state_e::CANCELED (C++ enumerator), 110	pilot::LocalizationStatus::time (C++ member), 88
pilot::local_planner_state_e::FINISHED (C++ enumerator), 110	pilot::LocalPlanner::await_goal (C++ function), 134
pilot::local_planner_state_e::IDLE (C++ enumerator), 109	pilot::LocalPlanner::await_goal_ex (C++ function), 134
pilot::local_planner_state_e::LOST (C++ enumerator), 110	pilot::LocalPlanner::cancel_goal (C++ function), 134
pilot::local_planner_state_e::ROTATING (C++ enumerator), 110	pilot::LocalPlanner::cancel_goal_await (C++ function), 134
pilot::local_planner_state_e::STUCK (C++ enumerator), 110	pilot::LocalPlanner::follow_path (C++ function), 134
pilot::local_planner_state_e::TRANSLATING (C++ enumerator), 109	pilot::LocalPlanner::get_optimized_path (C++ function), 133
pilot::local_planner_state_e::TURNING (C++ enumerator), 110	pilot::LocalPlanner::get_path (C++ function), 133
pilot::local_planner_state_e::WAITING (C++ enumerator), 109	pilot::LocalPlanner::get_state (C++ function), 134
pilot::localization_mode_e::DEAD_RECKONING (C++ enumerator), 110	pilot::LocalPlanner::move_to (C++ function), 134
pilot::localization_mode_e::INITIALIZING (C++ enumerator), 110	pilot::LocalPlanner::pause (C++ function), 134
pilot::localization_mode_e::LOST (C++ enumerator), 110	pilot::LocalPlanner::resume (C++ function), 134
pilot::localization_mode_e::MODE_1D (C++ enumerator), 110	pilot::LocalPlanner::set_goal (C++ function), 134
pilot::localization_mode_e::MODE_1D_YAW (C++ enumerator), 110	pilot::LocalPlanner::set_path (C++ function), 134
pilot::localization_mode_e::MODE_2D (C++ enumerator), 110	pilot::LocalPlanner::update_path (C++ function), 134
pilot::localization_mode_e::MODE_2D_YAW (C++ enumerator), 110	pilot::LocalPlannerState::goal (C++ member), 87
pilot::localization_mode_e::NO_INPUT (C++ enumerator), 110	pilot::LocalPlannerState::goal_options (C++ member), 87
pilot::localization_mode_e::NO_MAP (C++ enumerator), 110	pilot::LocalPlannerState::is_backwards (C++ member), 88
pilot::localization_mode_e::NO_ODOMETRY (C++ enumerator), 110	pilot::LocalPlannerState::is_restricted (C++ member), 88
pilot::localization_mode_e::NONE (C++ enumerator), 110	pilot::LocalPlannerState::job (C++ member), 87
pilot::LocalizationStatus::mode (C++ member), 88	pilot::LocalPlannerState::path_history (C++ member), 88
pilot::LocalizationStatus::num_points	pilot::LocalPlannerState::path_length

pilot::LocalPlannerState::path_time
(*C++ member*), 87

pilot::LocalPlannerState::point
(*C++ member*), 87

pilot::LocalPlannerState::pos_error
(*C++ member*), 87

pilot::LocalPlannerState::progress
(*C++ member*), 88

pilot::LocalPlannerState::state
(*C++ member*), 87

pilot::LocalPlannerState::time
(*C++ member*), 87

pilot::LocalPlannerState::time_stuck
(*C++ member*), 88

pilot::LocalPlannerState::update_rate
(*C++ member*), 87

pilot::LocalPlannerState::velocity_reason
(*C++ member*), 87

pilot::LocalPlannerState::wait_reason
(*C++ member*), 87

pilot::LocalPlannerState::yaw_error
(*C++ member*), 87

pilot::LocalPlannerState::yawrate_reason
(*C++ member*), 88

pilot::map_info_t::description
(*C++ member*), 111

pilot::map_info_t::last_modified
(*C++ member*), 111

pilot::map_info_t::name
(*C++ member*), 111

pilot::map_info_t::same_as
(*C++ function*), 111

pilot::MapArea::description
(*C++ member*), 89

pilot::MapArea::flags
(*C++ member*), 89

pilot::MapArea::name
(*C++ member*), 89

pilot::MapArea::outline
(*C++ member*), 89

pilot::MapArea::type
(*C++ member*), 89

pilot::MapElement::group
(*C++ member*), 89

pilot::MapElement::id
(*C++ member*), 89

pilot::MapElement::profiles
(*C++ member*), 89

pilot::MapMatch::areas
(*C++ member*), 90

pilot::MapMatch::distance
(*C++ member*), 89

pilot::MapMatch::is_valid
(*C++ member*), 89

pilot::MapMatch::node
(*C++ member*), 89

pilot::MapMatch::segment
(*C++ member*), 90

pilot::MapMatch::time
(*C++ member*), 89

pilot::MapNode::drive_flags
(*C++ member*), 90

pilot::MapNode::name
(*C++ member*), 90

pilot::MapNode::offset
(*C++ member*), 90

pilot::MapNode::parent
(*C++ member*), 90

pilot::MapNode::position
(*C++ member*), 90

pilot::MapProfile::description
(*C++ member*), 90

pilot::MapProfile::name
(*C++ member*), 90

pilot::MapProfile::node
(*C++ member*), 90

pilot::MapProfile::segment
(*C++ member*), 90

pilot::MapProfile::station
(*C++ member*), 90

pilot::MapServer::delete_grid_map
(*C++ function*), 135

pilot::MapServer::delete_road_map
(*C++ function*), 135

pilot::MapServer::download_grid_map
(*C++ function*), 135

pilot::MapServer::download_road_map
(*C++ function*), 135

pilot::MapServer::get_grid_map
(*C++ function*), 135

pilot::MapServer::get_grid_maps_info
(*C++ function*), 135

pilot::MapServer::get_road_map
(*C++ function*), 135

pilot::MapServer::get_road_maps_info
(*C++ function*), 135

pilot::MapServer::set_grid_map
(*C++ function*), 135

pilot::MapServer::set_road_map
(*C++ function*), 135

pilot::MapServer::switch_grid_map
(*C++ function*), 135

pilot::MapServer::switch_road_map
(*C++ function*), 135

pilot::MapStation::goal_tolerance
(*C++ member*), 91

pilot::MapStation::goal_tune_time
(*C++ member*), 91

pilot::MapStation::orientation
(*C++ member*), 91

pilot::motion_mode_e::AUTOMATIC
(*C++ enumerator*), 111

pilot::motion_mode_e::CUSTOM
(*C++ enumerator*), 111

pilot::motion_mode_e::JOYSTICK
(*C++ enumerator*), 111

pilot::motion_mode_e::NONE
(*C++ enumerator*), 111

pilot::motors::elmo::error_code_e::DRIVE_INHIBIT
(*C++ enumerator*), 119

pilot::motors::elmo::error_code_e::ELECTRICAL_ZERO
(*C++ enumerator*), 119

pilot::motors::elmo::error_code_e::FAILURE
(*C++ enumerator*), 118

pilot::motors::elmo::error_code_e::FEEDBACK_LOSS
(*C++ enumerator*), 119

pilot::motors::elmo::error_code_e::HALL_SENSOR::ERRORS::elmo::MotorControlData::motor_current
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::HEARTBEAT_FAILURES::elmo::MotorControlData::motor_position
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::INCONSISTENT_DATABASE::elmo::MotorControlData::motor_velocity
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::MOTOR_PSTUCK::motors::elmo::MotorControlData::name
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::NONE pilot::motors::elmo::MotorControlData::time
 (*C++ enumerator*), 118
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::OVER_HEATING::motors::elmo::MotorState::motor_info
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::OVER_VOLTAGE::motors::elmo::MotorState::time
 (*C++ enumerator*), 119
 (*C++ member*), 118
 pilot::motors::elmo::error_code_e::PEAK_CURRENT::RESERVATION::has (*C++ member*),
 (*C++ enumerator*), 119
 91
 pilot::motors::elmo::error_code_e::POSITION_OUT_OF_LIMIT::RESERVATION::status (*C++ member*),
 (*C++ enumerator*), 119
 91
 pilot::motors::elmo::error_code_e::POSITION_OUT_OF_TRACK::RESERVATION::wants (*C++ member*),
 (*C++ enumerator*), 119
 91
 pilot::motors::elmo::error_code_e::SAFETY::pilot::OccupancyMapData::DYNAMIC (*C++ member*), 92
 (*C++ enumerator*), 119
 pilot::motors::elmo::error_code_e::SERVO_DRIVE::BALANCY::OccupancyMapData::FREE (*C++ member*),
 (*C++ enumerator*), 119
 92
 pilot::motors::elmo::error_code_e::SHORT_CIRCUIT::OccupancyMapData::occupancy (*C++ member*),
 (*C++ enumerator*), 119
 92
 pilot::motors::elmo::error_code_e::SPEED_LIMIT::EXCEEDED::MapData::PROHIBITED
 (*C++ member*), 92
 (*C++ enumerator*), 119
 pilot::motors::elmo::error_code_e::SPEED_TRACK_ERROR::BALANCY::MapData::to_combined_map
 (*C++ enumerator*), 119
 (*C++ function*), 92
 pilot::motors::elmo::error_code_e::TIMEOUT::pilot::OccupancyMapData::to_float (*C++ function*),
 (*C++ enumerator*), 118
 92
 pilot::motors::elmo::error_code_e::UNDER_VOLTAGE::OccupancyMapData::to_mono_image
 (*C++ enumerator*), 119
 (*C++ function*), 92
 pilot::motors::elmo::error_code_e::UNKNOWN::FAILURE::OccupancyMapData::to_reflector_map
 (*C++ enumerator*), 119
 (*C++ function*), 92
 pilot::motors::elmo::motor_info_t::empty::OccupancyMapData::to_rgba (*C++ function*), 92
 (*C++ member*), 120
 pilot::motors::elmo::motor_info_t::error::pilot::OccupancyMapData::to_rgba_image
 (*C++ member*), 120
 (*C++ function*), 92
 pilot::motors::elmo::motor_info_t::is_enabled::OccupancyMapData::UNKNOWN (*C++ member*),
 (*C++ enumerator*), 120
 92
 pilot::motors::elmo::motor_info_t::state::pilot::Odometry::angular_velocity (*C++ member*),
 (*C++ enumerator*), 120
 93
 pilot::motors::elmo::motor_state_e::MOTOR_FAILURE::Odometry::linear_velocity (*C++ member*),
 (*C++ enumerator*), 120
 92
 pilot::motors::elmo::motor_state_e::MOTOR_BITIMEOUT::Odometry::orientation (*C++ member*),
 (*C++ enumerator*), 120
 92
 pilot::motors::elmo::motor_state_e::OPERATION_DODGE::Odometry::position (*C++ member*), 92
 (*C++ enumerator*), 120
 pilot::orientation_mode_e::ABSOLUTE_MAP
 pilot::motors::elmo::motor_state_e::OPERATION_FOOTABLE (*C++ enumerator*), 111
 (*C++ member*), 92
 pilot::orientation_mode_e::ABSOLUTE_ROAD
 pilot::motors::elmo::motor_state_e::PRE_INITIAI (*C++ enumerator*), 111
 (*C++ member*), 92
 pilot::orientation_mode_e::RELATIVE_ROAD

(C++ enumerator), 111
pilot::param_type_e::CUSTOM (C++ enumerator), 112
pilot::param_type_e::DEFAULT (C++ enumerator), 111
pilot::param_type_e::DISABLED (C++ enumerator), 112
pilot::param_type_e::IGNORE (C++ enumerator), 112
pilot::param_type_e::OPTIMIZED (C++ enumerator), 112
pilot::Path2D::job (C++ member), 93
pilot::Path2D::points (C++ member), 93
pilot::PathPoint2D::drive_flags (C++ member), 93
pilot::PathPoint2D::drive_mode (C++ member), 93
pilot::PathPoint2D::goal_tune_time (C++ member), 93
pilot::PathPoint2D::is_restricted (C++ member), 93
pilot::PathPoint2D::map_id (C++ member), 93
pilot::PathPoint2D::max_velocity (C++ member), 93
pilot::PathPoint2D::max_yawrate (C++ member), 93
pilot::PathPoint2D::orientation (C++ member), 93
pilot::PathPoint2D::tolerance (C++ member), 93
pilot::permission_e::CHANGE_GRIDMAP (C++ enumerator), 112
pilot::permission_e::CHANGE_ROADMAP (C++ enumerator), 112
pilot::permission_e::CHARGE (C++ enumerator), 112
pilot::permission_e::DISPLAY_CONTROL (C++ enumerator), 112
pilot::permission_e::EXECUTE_SCRIPT (C++ enumerator), 112
pilot::permission_e::INITIALIZE (C++ enumerator), 112
pilot::permission_e::INTERVENE_SCRIPT (C++ enumerator), 112
pilot::permission_e::MOVE (C++ enumerator), 112
pilot::permission_e::RECORD_DATA (C++ enumerator), 112
pilot::permission_e::RELAY_CONTROL (C++ enumerator), 112
pilot::permission_e::REMOTE_CONTROL (C++ enumerator), 112
pilot::permission_e::SAFETY_FIELD_CONTROL (C++ enumerator), 112
pilot::permission_e::UPLOAD_SCRIPT (C++ enumerator), 112
pilot::pilot_mode_e::MAPPING (C++ enumerator), 113
pilot::pilot_mode_e::NAVIGATION (C++ enumerator), 113
pilot::pilot_mode_e::REPLAY (C++ enumerator), 113
pilot::pilot_mode_e::TELEOP (C++ enumerator), 113
pilot::PilotServer::get_platform_info (C++ function), 136
pilot::PilotServer::get_state (C++ function), 136
pilot::PilotServer::set_pose_estimate (C++ function), 136
pilot::PilotServer::start_recording (C++ function), 136
pilot::PilotServer::stop_recording (C++ function), 136
pilot::PilotServer::switch_footprint (C++ function), 136
pilot::PilotServer::switch_motion_mode (C++ function), 136
pilot::PilotServer::switch_pilot_mode (C++ function), 136
pilot::PilotState::execution (C++ member), 94
pilot::PilotState::incidents (C++ member), 94
pilot::PilotState::is_recording (C++ member), 94
pilot::PilotState::localization (C++ member), 94
pilot::PilotState::motion_mode (C++ member), 94
pilot::PilotState::pilot_mode (C++ member), 94
pilot::PilotState::pilot_version (C++ member), 94
pilot::PilotState::planner (C++ member), 94
pilot::PilotState::robot_id (C++ member), 94
pilot::PilotState::time (C++ member), 94
pilot::platform_type_e::CUSTOM (C++ enumerator), 113
pilot::platform_type_e::MP_400 (C++ enumerator), 113
pilot::platform_type_e::MP_500 (C++ enumerator), 113
pilot::platform_type_e::MPO_500 (C++ enumerator), 113

pilot::platform_type_e::MPO_700 (C++ enumerator), 113
pilot::platform_type_e::ROX_ARGO (C++ enumerator), 113
pilot::platform_type_e::ROX_DIFF (C++ enumerator), 113
pilot::platform_type_e::ROX_MECA (C++ enumerator), 113
pilot::platform_type_e::ROX_TRIKE (C++ enumerator), 113
pilot::PlatformInfo::date_of_manufacture (C++ member), 95
pilot::PlatformInfo::features (C++ member), 95
pilot::PlatformInfo::name (C++ member), 94
pilot::PlatformInfo::serial (C++ member), 94
pilot::PlatformInfo::type (C++ member), 94
pilot::PlatformInterface::charge (C++ function), 136
pilot::PlatformInterface::read_analog_input (C++ function), 137
pilot::PlatformInterface::read_digital_input (C++ function), 137
pilot::PlatformInterface::set_digital_output (C++ function), 137
pilot::PlatformInterface::set_display_text (C++ function), 137
pilot::PlatformInterface::set_relay (C++ function), 137
pilot::PlatformInterface::shutdown (C++ function), 137
pilot::PlatformInterface::start_charging (C++ function), 136
pilot::PlatformInterface::stop_charging (C++ function), 136
pilot::PlatformInterface::wait_for_digital_input (C++ function), 137
pilot::PointCloud2D::base_to_odom (C++ member), 95
pilot::PointCloud2D::field (C++ member), 95
pilot::PointCloud2D::points (C++ member), 95
pilot::PointCloud2D::sensor (C++ member), 95
pilot::PointCloud2D::sensor_to_base (C++ member), 95
pilot::polygon_t::frame (C++ member), 114
pilot::polygon_t::points (C++ member), 114
pilot::Pose2D::covariance (C++ member), 95
pilot::Pose2D::pose (C++ member), 95
pilot::PoseArray2D::poses (C++ member), 96
pilot::PoseArray2D::transform (C++ func- tion), 96
pilot::power_system_type_e::POWER_24V (C++ enumerator), 114
pilot::power_system_type_e::POWER_48V (C++ enumerator), 114
pilot::PowerState::charging_current (C++ member), 96
pilot::PowerState::charging_state (C++ member), 96
pilot::PowerState::is_charging (C++ member), 96
pilot::PowerState::power_system_type (C++ member), 96
pilot::PowerState::time (C++ member), 96
pilot::radar_sample_t::footprint (C++ member), 114
pilot::radar_sample_t::grid_map (C++ member), 114
pilot::radar_sample_t::map_pose (C++ member), 114
pilot::radar_sample_t::platform (C++ member), 114
pilot::radar_sample_t::road_map (C++ member), 114
pilot::radar_sample_t::src_id (C++ member), 114
pilot::radar_sample_t::velocity (C++ member), 114
pilot::RadarScan::samples (C++ member), 96
pilot::RelayBoardData::ambient_temperature (C++ member), 97
pilot::RelayBoardData::keypad_state (C++ member), 97
pilot::RelayBoardData::relay_states (C++ member), 97
pilot::RelayBoardV3Data::ambient_temperature (C++ member), 97
pilot::RelayBoardV3Data::digital_input_states (C++ member), 97
pilot::RelayBoardV3Data::firmware_version (C++ member), 97
pilot::RelayBoardV3Data::key_switch_off_state (C++ member), 97
pilot::RelayBoardV3Data::keypad_button_states (C++ member), 97
pilot::RelayBoardV3Data::led_states (C++ member), 97
pilot::RelayBoardV3Data::relay_states (C++ member), 97
pilot::RelayBoardV3Data::release_structure_state (C++ member), 97
pilot::RelayBoardV3Data::time (C++ member), 97
pilot::RelayBoardV3Data::uptime (C++ member), 97

member), 97
pilot::RoadMapData::areas (C++ member), 98
pilot::RoadMapData::last_modified (C++ member), 98
pilot::RoadMapData::name (C++ member), 98
pilot::RoadMapData::nodes (C++ member), 98
pilot::RoadMapData::profiles (C++ member), 98
pilot::RoadMapData::segments (C++ member), 98
pilot::RoadMapPlanner::find_closest_station (C++ function), 138
pilot::RoadMapPlanner::find_station (C++ function), 138
pilot::RoadMapPlanner::fleet_mode (C++ member), 138
pilot::RoadMapPlanner::get_road_map (C++ function), 138
pilot::RoadMapPlanner::move_to_station (C++ function), 138
pilot::RoadMapPlanner::set_goal_station (C++ function), 138
pilot::RoadmapReservation::map_info (C++ member), 99
pilot::RoadmapReservation::nodes (C++ member), 99
pilot::RoadmapReservation::segments (C++ member), 99
pilot::RoadSegment::direction (C++ member), 98
pilot::RoadSegment::drive_flags (C++ member), 99
pilot::RoadSegment::drive_mode (C++ member), 98
pilot::RoadSegment::from_node (C++ member), 98
pilot::RoadSegment::max_velocity (C++ member), 98
pilot::RoadSegment::max_yawrate (C++ member), 98
pilot::RoadSegment::orientation (C++ member), 98
pilot::RoadSegment::orientation_mode (C++ member), 98
pilot::RoadSegment::orientation_tolerance (C++ member), 98
pilot::RoadSegment::to_node (C++ member), 98
pilot::RoadSegment::tolerance (C++ member), 98
pilot::RobotInfo::battery_state (C++ member), 100
pilot::RobotInfo::emergency_state (C++ member), 100
pilot::RobotInfo::footprint (C++ member), 99
pilot::RobotInfo::grid_map (C++ member), 99
pilot::RobotInfo::id (C++ member), 99
pilot::RobotInfo::laser_scans (C++ member), 100
pilot::RobotInfo::map_match (C++ member), 99
pilot::RobotInfo::map_path (C++ member), 99
pilot::RobotInfo::map_pose (C++ member), 99
pilot::RobotInfo::platform (C++ member), 100
pilot::RobotInfo::road_map (C++ member), 99
pilot::RobotInfo::state (C++ member), 99
pilot::RobotInfo::system_status (C++ member), 100
pilot::RobotInfo::velocity (C++ member), 99
pilot::RobotInfoMap::info_map (C++ member), 100
pilot::safety_code_e::EMERGENCY_STOP (C++ enumerator), 115
pilot::safety_code_e::NONE (C++ enumerator), 115
pilot::safety_code_e::RADIO_EMERGENCY_STOP (C++ enumerator), 115
pilot::safety_code_e::SCANNER_STOP (C++ enumerator), 115
pilot::safety_mode_e::APPROACHING (C++ enumerator), 115
pilot::safety_mode_e::DEPARTING (C++ enumerator), 115
pilot::safety_mode_e::HANDLING (C++ enumerator), 115
pilot::safety_mode_e::NONE (C++ enumerator), 115
pilot::safety_mode_e::WORKING (C++ enumerator), 115
pilot::SafetyInterface::select_safety_field (C++ function), 139
pilot::SafetyInterface::set_safety_mode (C++ function), 139
pilot::SafetyState::current_safety_field (C++ member), 100
pilot::SafetyState::triggered_cutoff_paths (C++ member), 100
pilot::Sample::frame (C++ member), 101
pilot::Sample::time (C++ member), 101
pilot::SegmentReservation::has (C++ member), 101

pilot::SegmentReservation::status (C++ member), 101
 pilot::SegmentReservation::wants (C++ member), 101
 pilot::sensor_2d_range_t::add_margin (C++ function), 116
 pilot::sensor_2d_range_t::is_valid (C++ function), 115
 pilot::sensor_2d_range_t::is_within (C++ function), 115
 pilot::sensor_2d_range_t::is_within_poin (C++ function), 116
 pilot::sensor_2d_range_t::is_within_xy (C++ function), 115
 pilot::sensor_2d_range_t::max_angle (C++ member), 115
 pilot::sensor_2d_range_t::max_range (C++ member), 115
 pilot::sensor_2d_range_t::min_angle (C++ member), 115
 pilot::sensor_2d_range_t::min_range (C++ member), 115
 pilot::StackFrame::line_number (C++ member), 101
 pilot::StackFrame::method (C++ member), 101
 pilot::system_error_e::BRAKE_RELEASE_BUTTON_ERROR (C++ enumerator), 116
 pilot::system_error_e::CHARGING_RELAY_ERROR (C++ enumerator), 116
 pilot::system_error_e::COMMUNICATION_ERROR (C++ enumerator), 116
 pilot::system_error_e::EDM_ERROR (C++ enumerator), 116
 pilot::system_error_e::EM_STOP_SYSTEM_ERROR (C++ enumerator), 116
 pilot::system_error_e::MAXIMUM_VELOCITY (C++ enumerator), 116
 pilot::system_error_e::MODE_SELECT_ERROR (C++ enumerator), 116
 pilot::system_error_e::MOTOR_ERROR (C++ enumerator), 116
 pilot::system_error_e::POWER_RELAY_ERROR (C++ enumerator), 116
 pilot::system_error_e::SAFETY_RELAY_ERROR (C++ enumerator), 116
 pilot::system_error_e::SCANNER_DIRTY (C++ enumerator), 116
 pilot::system_error_e::STANDSTILL_VIOLATION (C++ enumerator), 116
 pilot::SystemState::is_initialized (C++ member), 102
 pilot::SystemState::is_shutdown (C++ member), 102

pilot::Task::args (C++ member), 102
 pilot::Task::id (C++ member), 102
 pilot::Task::method (C++ member), 102
 pilot::Task::module (C++ member), 102
 pilot::TaskHandler::autostart (C++ member), 139
 pilot::TaskHandler::autostart_delay_ms (C++ member), 139
 pilot::TaskHandler::cancel_program (C++ function), 140
 pilot::TaskHandler::execute_file (C++ function), 139
 pilot::TaskHandler::execute_program (C++ function), 139
 pilot::TaskHandler::pause_program (C++ function), 140
 pilot::TaskHandler::resume_program (C++ function), 140
 pilot::vector_3f_param_t::x (C++ member), 116
 pilot::vector_3f_param_t::y (C++ member), 116
 pilot::vector_3f_param_t::z (C++ member), 116
 pilot::VelocityCmd::allow_wheel_reset (C++ member), 102
 pilot::VelocityCmd::angular (C++ member), 102
 pilot::VelocityCmd::linear (C++ member), 102
 pilot::VelocityCmd::reset_wheels (C++ member), 102
 pilot::VelocityCmd::time (C++ member), 102
 pilot::VelocityLimits::max_rot_vel (C++ member), 103
 pilot::VelocityLimits::max_trans_vel (C++ member), 103
 pilot::VelocityLimits::max_vel_x (C++ member), 103
 pilot::VelocityLimits::max_vel_y (C++ member), 103
 pilot::VelocityLimits::min_rot_vel (C++ member), 103
 pilot::VelocityLimits::min_trans_vel (C++ member), 103
 pilot::VelocityLimits::min_vel_x (C++ member), 103
 pilot::VelocityLimits::rot_stopped_vel (C++ member), 103
 pilot::VelocityLimits::trans_stopped_vel (C++ member), 103
 pilot::wait_reason_e::CANCELED (C++ enum)

merator), 117
 pilot::wait_reason_e::CLEARANCE (C++ enumerator), 117
 pilot::wait_reason_e::LOCAL_COST_MAP (C++ enumerator), 117
 pilot::wait_reason_e::LOCALIZATION (C++ enumerator), 117
 pilot::wait_reason_e::ODOMETRY (C++ enumerator), 117
 pilot::wait_reason_e::PATH_OPTIMIZER (C++ enumerator), 117
 pilot::wait_reason_e::PAUSED (C++ enumerator), 117

R

read_analog_input (C++ function), 71
 read_digital_input (C++ function), 71
 remote_config (C++ member), 137
 require () (built-in function), 39
 reset_motors (C++ function), 73
 return () (built-in function), 42

S

select_safety_field (C++ function), 71
 set_digital_output (C++ function), 71
 set_display_text (C++ function), 71
 set_goal_station (C++ function), 46
 set_priority (C++ function), 46
 set_relay (C++ function), 71
 set_safety_mode (C++ function), 71
 set_timer_ms (C++ function), 71
 start_charging (C++ function), 71
 stop (C++ function), 46
 stop_charging (C++ function), 71
 switch_grid_map (C++ function), 70
 switch_road_map (C++ function), 70

T

to_string () (built-in function), 42

U

unblock (C++ function), 72

V

vnx::access_role_e::ADMIN (C++ enumerator), 125
 vnx::access_role_e::DEFAULT (C++ enumerator), 125
 vnx::access_role_e::INSTALLER (C++ enumerator), 125
 vnx::access_role_e::OBSERVER (C++ enumerator), 125
 vnx::access_role_e::USER (C++ enumerator), 125

vnx::access_role_e::VIEWER (C++ enumerator), 125
 vnx::JRPC_Error::code (C++ member), 121
 vnx::JRPC_Error::data (C++ member), 121
 vnx::JRPC_Error::message (C++ member), 121
 vnx::JRPC_Proxy::select_service (C++ function), 140
 vnx::LogMsg::DEBUG (C++ member), 121
 vnx::LogMsg::display_level (C++ member), 121
 vnx::LogMsg::ERROR (C++ member), 121
 vnx::LogMsg::get_output (C++ function), 121
 vnx::LogMsg::INFO (C++ member), 121
 vnx::LogMsg::level (C++ member), 121
 vnx::LogMsg::message (C++ member), 121
 vnx::LogMsg::module (C++ member), 121
 vnx::LogMsg::process (C++ member), 121
 vnx::LogMsg::time (C++ member), 121
 vnx::LogMsg::WARN (C++ member), 121
 vnx::ModuleInfo::get_cpu_load (C++ function), 123
 vnx::ModuleInfo::get_cpu_load_total (C++ function), 123
 vnx::ModuleInfo::id (C++ member), 122
 vnx::ModuleInfo::name (C++ member), 122
 vnx::ModuleInfo::num_async_pending (C++ member), 122
 vnx::ModuleInfo::num_async_process (C++ member), 122
 vnx::ModuleInfo::pub_topics (C++ member), 122
 vnx::ModuleInfo::remotes (C++ member), 122
 vnx::ModuleInfo::src_mac (C++ member), 122
 vnx::ModuleInfo::sub_topics (C++ member), 122
 vnx::ModuleInfo::time (C++ member), 122
 vnx::ModuleInfo::time_idle (C++ member), 122
 vnx::ModuleInfo::time_idle_total (C++ member), 122
 vnx::ModuleInfo::time_running (C++ member), 122
 vnx::ModuleInfo::time_running_total (C++ member), 122
 vnx::ModuleInfo::time_started (C++ member), 122
 vnx::ModuleInfo::type (C++ member), 122
 vnx::ModuleInfo::type_code (C++ member), 122
 vnx::mqtt::export_t::qos (C++ member), 129
 vnx::mqtt::export_t::retained (C++ member), 129
 vnx::mqtt::export_t::topic (C++ member), 129

vnx::mqtt::import_t::qos (*C++ member*), 129 *(C++ member)*, 127
 vnx::mqtt::import_t::topic (*C++ member*), 129 vnx::opc_ua::monitored_item_t::monitoring_mode
(C++ member), 127
 vnx::mqtt::import_t::type (*C++ member*), 129 vnx::opc_ua::monitored_item_t::output_data
(C++ member), 127
 vnx::mqtt::last_will_t::message (*C++ member*), 129 vnx::opc_ua::monitored_item_t::queue_size
(C++ member), 127
 vnx::mqtt::Proxy::address (*C++ member*), 145 vnx::opc_ua::monitored_item_t::sampling_interval
(C++ member), 127
 vnx::mqtt::Proxy::clean_session (*C++ member*), 146 vnx::opc_ua::monitoring_mode_e::DISABLED
(C++ enumerator), 127
 vnx::mqtt::Proxy::client_id (*C++ member*), 145 vnx::opc_ua::monitoring_mode_e::REPORTING
(C++ enumerator), 127
 vnx::mqtt::Proxy::connect_interval_ms (*C++ member*), 146 vnx::opc_ua::monitoring_mode_e::SAMPLING
(C++ enumerator), 127
 vnx::mqtt::Proxy::connect_timeout (*C++ member*), 146 vnx::opc_ua::node_id_t::index (*C++ mem-
ber*), 127
 vnx::mqtt::Proxy::export_map (*C++ mem-
ber*), 145 vnx::opc_ua::node_id_t::name (*C++ mem-
ber*), 127
 vnx::mqtt::Proxy::export_map_ex (*C++ mem-
ber*), 145 vnx::opc_ua::Proxy::address (*C++ member*),
 143
 vnx::mqtt::Proxy::import_map (*C++ mem-
ber*), 145 vnx::opc_ua::Proxy::application_name
(C++ member), 143
 vnx::mqtt::Proxy::import_map_ex (*C++ mem-
ber*), 146 vnx::opc_ua::Proxy::application_uri
(C++ member), 143
 vnx::mqtt::Proxy::keepalive_interval
(C++ member), 146 vnx::opc_ua::Proxy::block_until_connect
(C++ member), 143
 vnx::mqtt::Proxy::last_will (*C++ member*),
 146 vnx::opc_ua::Proxy::block_until_reconnect
(C++ member), 143
 vnx::mqtt::Proxy::password (*C++ member*),
 146 vnx::opc_ua::Proxy::browse_all (*C++ func-
tion*), 144
 vnx::mqtt::Proxy::topic_prefix (*C++ mem-
ber*), 146 vnx::opc_ua::Proxy::call (*C++ function*), 144
 vnx::mqtt::Proxy::username (*C++ member*),
 146 vnx::opc_ua::Proxy::certificate_file
(C++ member), 143
 vnx::mqtt::Proxy::wait_for_ack (*C++ mem-
ber*), 146 vnx::opc_ua::Proxy::connect_interval_ms
(C++ member), 143
 vnx::mqtt::qos_e::AT_LEAST_ONCE (*C++ enum-
erator*), 130 vnx::opc_ua::Proxy::keepalive_interval_ms
(C++ member), 143
 vnx::mqtt::qos_e::EXACTLY_ONCE (*C++ enum-
erator*), 130 vnx::opc_ua::Proxy::object_call
(C++ function), 144
 vnx::mqtt::qos_e::FIRE_AND_FORGET (*C++ enum-
erator*), 130 vnx::opc_ua::Proxy::password (*C++ mem-
ber*), 143
 vnx::opc_ua::DataChange::monitored_item_id
(C++ member), 126 vnx::opc_ua::Proxy::private_key_file
(C++ member), 143
 vnx::opc_ua::DataChange::subscription_id
(C++ member), 126 vnx::opc_ua::Proxy::read_object_variable
(C++ function), 144
 vnx::opc_ua::DataChange::value (*C++ mem-
ber*), 126 vnx::opc_ua::Proxy::read_variable
(C++ function), 144
 vnx::opc_ua::monitored_item_t::discard_oldest
(C++ member), 127 vnx::opc_ua::Proxy::secure_channel_lifetime_ms
(C++ member), 143
 vnx::opc_ua::monitored_item_t::id (*C++ mem-
ber*), 127 vnx::opc_ua::Proxy::security_mode
(C++ member), 143
 vnx::opc_ua::monitored_item_t::monitor_changes
(C++ member), 143 vnx::opc_ua::Proxy::session_timeout_ms

vnx::opc_ua::Proxy::subscriptions (C++ member), 143

vnx::opc_ua::Proxy::trust_list (C++ member), 143

vnx::opc_ua::Proxy::username (C++ member), 143

vnx::opc_ua::Proxy::write_object_variable (C++ function), 144

vnx::opc_ua::Proxy::write_variable (C++ function), 144

vnx::opc_ua::security_mode_e::ANY (C++ enumerator), 128

vnx::opc_ua::security_mode_e::NONE (C++ enumerator), 128

vnx::opc_ua::security_mode_e::SIGN (C++ enumerator), 128

vnx::opc_ua::security_mode_e::SIGN_AND_EMGRYP (C++ enumerator), 128

vnx::opc_ua::security_policy_e::AES_128_SHA256_PCRSA_OAED (C++ enumerator), 128

vnx::opc_ua::security_policy_e::BASIC_128nRSAOp5 (C++ enumerator), 128

vnx::opc_ua::security_policy_e::BASIC_256nSHA256 (C++ enumerator), 128

vnx::opc_ua::security_policy_e::NONE (C++ enumerator), 128

vnx::opc_ua::Server::add_insecure_discovery (C++ member), 145

vnx::opc_ua::Server::allow_anonymous_access (C++ member), 145

vnx::opc_ua::Server::application_name (C++ member), 145

vnx::opc_ua::Server::application_uri (C++ member), 145

vnx::opc_ua::Server::certificate_file (C++ member), 144

vnx::opc_ua::Server::custom_hostname (C++ member), 144

vnx::opc_ua::Server::default_access (C++ member), 145

vnx::opc_ua::Server::error_variables (C++ member), 145

vnx::opc_ua::Server::export_services (C++ member), 144

vnx::opc_ua::Server::export_topics (C++ member), 144

vnx::opc_ua::Server::port (C++ member), 144

vnx::opc_ua::Server::private_key_file (C++ member), 144

vnx::opc_ua::Server::security_policies (C++ member), 145

vnx::opc_ua::Server::trust_list (C++ member), 145

vnx::opc_ua::Server::use_authentication (C++ member), 145

vnx::opc_ua::Server::variables (C++ member), 145

vnx::opc_ua::subscription_t::lifetime (C++ member), 128

vnx::opc_ua::subscription_t::max_keepalive (C++ member), 128

vnx::opc_ua::subscription_t::max_notifications_per (C++ member), 128

vnx::opc_ua::subscription_t::monitored_items (C++ member), 128

vnx::opc_ua::subscription_t::output_data (C++ member), 128

vnx::opc_ua::subscription_t::priority (C++ member), 128

vnx::opc_ua::subscription_t::publishing_enabled (C++ member), 128

vnx::opc_ua::subscription_t::publishing_interval (C++ member), 128

vnx::opc_ua::subscription_t::CONST_REQUEST (C++ enumerator), 126

vnx::opc_ua::subscription_t::HOST_SHUTDOWN (C++ enumerator), 126

vnx::opc_ua::subscription_t::PROTECTED_CONFIG (C++ enumerator), 126

vnx::opc_ua::subscription_t::PUBLISH (C++ enumerator), 126

vnx::opc_ua::subscription_t::READ_CONFIG (C++ enumerator), 126

vnx::opc_ua::subscription_t::REQUEST (C++ enumerator), 126

vnx::opc_ua::subscription_t::RESTART (C++ enumerator), 126

vnx::opc_ua::subscription_t::SELF_TEST (C++ enumerator), 126

vnx::opc_ua::subscription_t::SHUTDOWN (C++ enumerator), 126

vnx::opc_ua::subscription_t::START (C++ enumerator), 126

vnx::opc_ua::subscription_t::STOP (C++ enumerator), 126

vnx::opc_ua::subscription_t::VIEW (C++ enumerator), 125

vnx::opc_ua::subscription_t::WRITE_CONFIG (C++ enumerator), 126

vnx::Proxy::address (C++ member), 141

vnx::Proxy::auto_import (C++ member), 141

vnx::Proxy::block_until_connect (C++ member), 141

vnx::Proxy::block_until_reconnect (C++ member), 141

vnx::Proxy::default_access (*C++ member*),
142
vnx::Proxy::export_list (*C++ member*), 141
vnx::Proxy::export_map (*C++ member*), 141
vnx::Proxy::forward_list (*C++ member*), 141
vnx::Proxy::import_list (*C++ member*), 141
vnx::Proxy::import_map (*C++ member*), 141
vnx::Proxy::max_queue_ms (*C++ member*), 141
vnx::Proxy::max_queue_size (*C++ member*),
141
vnx::Proxy::recv_buffer_size (*C++ member*), 142
vnx::Proxy::send_buffer_size (*C++ member*), 142
vnx::Proxy::time_sync (*C++ member*), 141
vnx::Proxy::tunnel_map (*C++ member*), 141
vnx::Server::address (*C++ member*), 142
vnx::Server::default_access (*C++ member*),
142
vnx::Server::export_list (*C++ member*), 142
vnx::Server::max_queue_ms (*C++ member*),
142
vnx::Server::max_queue_size (*C++ member*),
142
vnx::Server::recv_buffer_size (*C++ member*), 142
vnx::Server::send_buffer_size (*C++ member*), 142
vnx::Server::use_authentication (*C++ member*), 142
vnx::User::access_roles (*C++ member*), 124
vnx::User::hashed_password (*C++ member*),
124
vnx::User::name (*C++ member*), 124
vnx::User::permissions (*C++ member*), 124

W

wait_for_digital_input (*C++ function*), 70
wait_for_joystick (*C++ function*), 70
wait_for_joystick_button (*C++ function*), 70
wait_for_keypad (*C++ function*), 70
wait_for_keypad_button (*C++ function*), 70
wait_hours (*C++ function*), 71
wait_min (*C++ function*), 71
wait_ms (*C++ function*), 71
wait_sec (*C++ function*), 71