
labgrid Documentation

Release 0.1.1.dev300+g6896265

Jan Luebbe, Rouven Czerwinski

Mar 20, 2018

1	Getting Started	3
1.1	Installation	3
1.2	Running Your First Test	4
1.3	Setting Up the Distributed Infrastructure	5
1.4	udev Matching	7
1.5	Using a Strategy	8
2	Overview	9
2.1	Architecture	9
2.2	Remote Resources and Places	12
3	Usage	15
3.1	Library	15
3.2	pytest Plugin	17
3.3	Command-Line	20
3.4	USB stick emulation	20
3.5	hawkBit management API	21
4	Manual Pages	23
4.1	labgrid-client	23
4.2	labgrid-device-config	26
4.3	labgrid-exporter	27
5	Configuration	31
5.1	Resources	31
5.2	Drivers	40
5.3	Strategies	50
5.4	Reporters	51
5.5	Environment Configuration	52
5.6	Exporter Configuration	52
6	Development	55
6.1	Installation	55
6.2	Writing a Driver	56
6.3	Writing a Resource	57
6.4	Writing a Strategy	58
6.5	Contributing	59

6.6	Ideas	60
7	Design Decisions	63
7.1	Out of Scope	63
7.2	In Scope	64
7.3	Further Goals	64
8	Changes	65
8.1	Release 0.2.0 (unreleased)	65
8.2	Release 0.1.0 (released May 11, 2017)	67
9	Modules	69
9.1	labgrid package	69
10	Indices and Tables	133
	Python Module Index	135

Labgrid is an embedded board control python library with a focus on testing, development and general automation. It includes a remote control layer to control boards connected to other hosts.

The idea behind labgrid is to create an abstraction of the hardware control layer needed for testing of embedded systems, automatic software installation and automation during development. Labgrid itself is *not* a testing framework, but is intended to be combined with [pytest](#) (and additional pytest plugins). Please see [Design Decisions](#) for more background information.

It currently supports:

- pytest plugin to write tests for embedded systems connecting serial console or SSH
- remote client-exporter-coordinator infrastructure to make boards available from different computers on a network
- power/reset management via drivers for power switches or onewire PIOs
- upload of binaries via USB: imxusbloader/mxsusbloader (bootloader) or fastboot (kernel)
- functions to control external services such as emulated USB-Sticks and the [hawkBit](#) deployment service

While labgrid is currently used for daily development on embedded boards and for automated testing, several planned features are not yet implemented and the APIs may be changed as more use-cases appear. We appreciate code contributions and feedback on using labgrid on other environments (see [Contributing](#) for details). Please consider contacting us (via a GitHub issue) before starting larger changes, so we can discuss design trade-offs early and avoid redundant work. You can also look at [Ideas](#) for enhancements which are not yet implemented.

This section of the manual contains introductory tutorials for installing labgrid, running your first test and setting up the distributed infrastructure.

1.1 Installation

Depending on your distribution you need some dependencies. On Debian stretch these usually are:

```
$ apt-get install python3 python3-virtualenv python3-pip
```

In many cases, the easiest way is to install labgrid into a virtualenv:

```
$ virtualenv -p python3 labgrid-venv
$ source labgrid-venv/bin/activate
```

Start installing labgrid by cloning the repository and installing the requirements from the *requirements.txt* file:

```
$ git clone https://github.com/labgrid-project/labgrid
$ cd labgrid && pip install -r requirements.txt
$ python3 setup.py install
```

Note: Previous documentation recommended the installation as via pip (*pip3 install labgrid*). This lead to broken installations due to unexpected incompatibilities with new releases of the dependencies. Consequently we now recommend using pinned versions from the *requirements.txt* file for most use cases.

Labgrid also supports the installation as a library via pip, but we only test against library versions specified in the requirements.txt file. Thus when installing directly from pip you have to test compatibility yourself.

Test your installation by running:

```
$ labgrid-client --help
usage: labgrid-client [-h] [-x URL] [-c CONFIG] [-p PLACE] [-d] COMMAND ...
...
```

If the help for `labgrid-client` does not show up, open an [Issue](#). If everything was successful so far, proceed to the next section:

1.1.1 Optional Requirements

Labgrid provides optional features which are not included in the default `requirements.txt`. The tested library version for each feature is included in a separate requirements file. An example for snmp support is:

```
$ pip install -r snmp-requirements.txt
```

Onewire

Onewire support requires the `libow` library with headers, installable on debian via the `libow-dev` package. Use the `onewire-requirements.txt` file to install the correct onewire library version in addition to the normal installation.

SNMP

SNMP support requires to additional packages, `pysnmp` and `pysnmpmibs`. They are included in the `snmp-requirements.txt` file.

Modbus

Modbus support requires an additional package `pyModbusTCP`. It is included in the `modbus-requirements.txt` file.

1.2 Running Your First Test

Start by copying the initial example:

```
$ mkdir ../first_test/
$ cp examples/shell/* ../first_test/
$ cd ../first_test/
```

Connect your embedded board (raspberry pi, riotboard, ...) to your computer and adjust the `port` parameter of the `RawSerialPort` resource and `username` and `password` of the `ShellDriver` driver in `local.yaml`:

```
targets:
  main:
    resources:
      RawSerialPort:
        port: "/dev/ttyUSB0"
    drivers:
      ManualPowerDriver:
        name: "example"
      SerialDriver: {}
      ShellDriver:
        prompt: 'root@\w+: [^ ]+ '
```



```
login_prompt: ' login: '  
username: 'root'
```

You can check which device name gets assigned to your USB-Serial converter by unplugging the converter, running `dmesg -w` and plugging it back in. Boot up your board (manually) and run your first test:

```
$ pytest --lg-env local.yaml test_shell.py
```

It should return successfully, in case it does not, open an [Issue](#).

If you want to build documentation you need some more dependencies:

```
$ pip3 install -r doc-requirements.txt
```

The documentation is inside `doc/`. HTML-Documentation is build using:

```
$ cd doc/  
$ make html
```

The HTML-documentation is written to `doc/.build/html/`.

1.3 Setting Up the Distributed Infrastructure

The labgrid distributed infrastructure consists of three components:

1. Coordinator
2. Exporter
3. Client

The system needs at least one coordinator and exporter, these can run on the same machine. The client is used to access functionality provided by an exporter. Over the course of this tutorial we will set up a coordinator and exporter, and learn how to access the exporter via the client.

1.3.1 Coordinator

To start the coordinator, we will download the labgrid repository, create an extra virtualenv and install the dependencies via the requirements file.

```
$ git clone https://github.com/labgrid-project/labgrid  
$ cd labgrid && virtualenv -p python3 crossbar_venv  
$ source crossbar_venv/bin/activate  
$ pip install -r crossbar-requirements.txt  
$ python setup.py install
```

All necessary dependencies should be installed now, we can start the coordinator by running `crossbar start` inside of the repository.

Note: This is possible because the labgrid repository contains the crossbar configuration the coordinator in the `.crossbar` folder.

1.3.2 Exporter

The exporter needs a configuration file written in YAML syntax, listing the resources to be exported from the local machine. The config file contains one or more named resource groups. Each group contains one or more resource declarations and optionally a location string (see the configuration reference for details).

For example, to export a `RawSerialPort` with the group name *example-port* and the location *example-location*:

```
example-group:
  location: example-location
  RawSerialPort:
    port: /dev/ttyUSB0
```

The exporter can now be started by running:

```
$ labgrid-exporter configuration.yaml
```

Additional groups and resources can be added:

```
example-group:
  location: example-location
  RawSerialPort:
    port: /dev/ttyUSB0
  NetworkPowerPort:
    model: netio
    host: netio1
    index: 3
example-group-2:
  RawSerialPort:
    port: /dev/ttyUSB1
```

Restart the exporter to activate the new configuration.

1.3.3 Client

Finally we can test the client functionality, run:

```
$ labgrid-client resources
kiwi/example-group/NetworkPowerPort
kiwi/example-group/RawSerialPort
kiwi/example-group-2/RawSerialPort
```

You can see the available resources listed by the coordinator. The groups *example-group* and *example-group-2* should be available there.

To show more details on the exported resources, use `-v` (or `-vv`):

```
$ labgrid-client -v resources
Exporter 'kiwi':
  Group 'example-group' (kiwi/example-group/*):
    Resource 'NetworkPowerPort' (kiwi/example-group/NetworkPowerPort[/
↪NetworkPowerPort]):
      {'acquired': None,
       'avail': True,
       'cls': 'NetworkPowerPort',
       'params': {'host': 'netio1', 'index': 3, 'model': 'netio'}}
  ...
```

You can now add a place with:

```
$ labgrid-client --place example-place create
```

And add resources to this place (-p is short for --place):

```
$ labgrid-client -p example-place add-match */example-port/*
```

Which adds the previously defined resource from the exporter to the place. To interact with this place, it needs to be acquired first, this is done by

```
$ labgrid-client -p example-place acquire
```

Now we can connect to the serial console:

```
$ labgrid-client -p example-place console
```

For a complete reference have a look at the `labgrid-client(1)` man page.

1.4 udev Matching

Labgrid allows the exporter (or the client-side environment) to match resources via udev rules. The udev resources become available to the test/exporter as soon as they are plugged into the computer, e.g. allowing an exporter to export all USB ports on a specific hub and making a `NetworkSerialPort` available as soon as it is plugged into one of the hub's ports. The information udev has on a device can be viewed by executing:

```
$ udevadm info /dev/ttyUSB0
...
E: ID_MODEL_FROM_DATABASE=CP210x UART Bridge / myAVR mySmartUSB light
E: ID_MODEL_ID=ea60
E: ID_PATH=pci-0000:00:14.0-usb-0:5:1.0
E: ID_PATH_TAG=pci-0000_00_14_0-usb-0_5_1_0
E: ID_REVISION=0100
E: ID_SERIAL=Silicon_Labs_CP2102_USB_to_UART_Bridge_Controller_P-00-00682
E: ID_SERIAL_SHORT=P-00-00682
E: ID_TYPE=generic
...
```

In this case the device has an `ID_SERIAL_SHORT` key with a unique ID embedded in the USB-serial converter. The resource match configuration for this USB serial converter is:

```
USBSerialPort:
  match:
    'ID_SERIAL_SHORT': 'P-00-00682'
```

This section can now be added under the resource key in an environment configuration or under its own entry in an exporter configuration file.

As the USB bus number can change depending on the kernel driver initialization order, it is better to use the `@ID_PATH` instead of `@sys_name` for USB devices. In the default udev configuration, the path is not available for all USB devices, but that can be changed by creating a udev rules file:

```
SUBSYSTEMS=="usb", IMPORT{builtin}="path_id"
```

1.5 Using a Strategy

Strategies allow the labgrid library to automatically bring the board into a defined state, e.g. boot through the bootloader into the Linux kernel and log in to a shell. They have a few requirements:

- A driver implementing the `PowerProtocol`, if no controllable infrastructure is available a `ManualPowerDriver` can be used.
- A driver implementing the `LinuxBootProtocol`, usually a specific driver for the board's bootloader
- A driver implementing the `CommandProtocol`, usually a `ShellDriver` with a `SerialDriver` below it.

Labgrid ships with two builtin strategies, `BareboxStrategy` and `UBootStrategy`. These can be used as a reference example for simple strategies, more complex tests usually require the implementation of your own strategies.

To use a strategy, add it and its dependencies to your configuration YAML, retrieve it in your test and call the `transition(status)` function.

```
>>> strategy = target.get_driver(strategy)
>>> strategy.transition("barebox")
```

An example using the pytest plugin is provided under *examples/strategy*.

2.1 Architecture

Labgrid can be used in several ways:

- on the command line to control individual embedded systems during development (“board farm”)
- via a pytest plugin to automate testing of embedded systems
- as a python library in other programs

In the labgrid library, a controllable embedded system is represented as a *Target*. *Targets* normally have several *Resource* and *Driver* objects, which are used to store the board-specific information and to implement actions on different abstraction levels. For cases where a board needs to be transitioned to specific states (such as *off*, *in bootloader*, *in Linux shell*), a *Strategy* (a special kind of *Driver*) can be added to the *Target*.

While labgrid comes with implementations for some resources, drivers and strategies, custom implementations for these can be registered at runtime. It is expected that for complex use-cases, the user would implement and register a custom *Strategy* and possibly some higher-level *Drivers*.

2.1.1 Resources

Resources are passive and only store the information to access the corresponding part of the *Target*. Typical examples of resources are *RawSerialPort*, *NetworkPowerPort* and *AndroidFastboot*.

An important type of *Resources* are *ManagedResources*. While normal *Resources* are always considered available for use and have fixed properties (such as the `/dev/ttyUSB0` device name for a *RawSerialPort*), the *ManagedResources* are used to represent interfaces which are discoverable in some way. They can appear/disappear at runtime and have different properties each time they are discovered. The most common examples of *ManagedResources* are the various USB resources discovered using udev, such as *USBSerialPort*, *IMXUSBLoader* or *AndroidFastboot*.

2.1.2 Drivers and Protocols

A labgrid *Driver* uses one (or more) *Resources* and/or other, lower-level *Drivers* to perform a set of actions on a *Target*. For example, the *NetworkPowerDriver* uses a *NetworkPowerPort* resource to control the *Target*'s power supply. In this case, the actions are “on”, “off”, “cycle” and “get”.

As another example, the *ShellDriver* uses any driver implementing the *ConsoleProtocol* (such as a *SerialDriver*, see below). The *ConsoleProtocol* allows the *ShellDriver* to work with any specific method of accessing the board's console (locally via USB, over the network using a console server or even an external program). At the *ConsoleProtocol* level, characters are sent to and received from the target, but they are not yet interpreted as specific commands or their output.

The *ShellDriver* implements the higher-level *CommandProtocol*, providing actions such as “run” or “run_check”. Internally, it interacts with the Linux shell on the target board. For example, it:

- waits for the login prompt
- enters user name and password
- runs the requested shell command (delimited by marker strings)
- parses the output
- retrieves the exit status

Other drivers, such as the *SSHDriver*, also implement the *CommandProtocol*. This way, higher-level code (such as a test suite), can be independent of the concrete control method on a given board.

2.1.3 Binding and Activation

When a *Target* is configured, each driver is “bound” to the resources (or other drivers) required by it. Each *Driver* class has a “bindings” attribute, which declares which *Resources* or *Protocols* it needs and under which name they should be available to the *Driver* instance. The binding resolution is handled by the *Target* during the initial configuration and results in a directed, acyclic graph of resources and drivers. During the lifetime of a *Target*, the bindings are considered static.

In most non-trivial target configurations, some drivers are mutually exclusive. For example, a *Target* may have both a *ShellDriver* and a *BareboxDriver*. Both bind to a driver implementing the *ConsoleProtocol* and provide the *CommandProtocol*. Obviously, the board cannot be in the bootloader and in Linux at the same time, which is represented in labgrid via the *BindingState* (*boundactive*). If, during activation of a driver, any other driver in its bindings is not active, they will be activated as well.

Activating and deactivating *Drivers* is also used to handle *ManagedResources* becoming available/unavailable at runtime. If some resources bound to by the activating drivers are currently unavailable, the *Target* will wait for them to appear (with a per resource timeout). A realistic sequence of activation might look like this:

- enable power (*PowerProtocol.on*)
- activate the *IMXUSBDriver* driver on the target (this will wait for the *IMXUSBLoader* resource to be available)
- load the bootloader (*BootstrapProtocol.load*)
- activate the *AndroidFastbootDriver* driver on the target (this will wait for the *AndroidFastboot* resource to be available)
- boot the kernel (*AndroidFastbootDriver.boot*)
- activate the *ShellDriver* driver on the target (this will wait for the *USBSerialPort* resource to be available and log in)

Any *ManagedResources* which become unavailable at runtime will automatically deactivate the dependent drivers.

2.1.4 Multiple Drivers and Names

Each driver and resource can have an optional name. This parameter is required for all manual creations of drivers and resources. To manually bind to a specific driver set a binding mapping before creating the driver:

```
>>> t = Target("Test")
>>> SerialPort(t, "First")
SerialPort(target=Target(name='Test', env=None), name='First', state=<BindingState.
↳bound: 1>, avail=True, port=None, speed=115200)
>>> SerialPort(t, "Second")
SerialPort(target=Target(name='Test', env=None), name='Second', state=<BindingState.
↳bound: 1>, avail=True, port=None, speed=115200)
>>> t.set_binding_map({"port": "Second"})
>>> sd = SerialDriver(t, "Driver")
>>> sd
SerialDriver(target=Target(name='Test', env=None), name='Driver', state=<BindingState.
↳bound: 1>, txdelay=0.0)
>>> sd.port
SerialPort(target=Target(name='Test', env=None), name='Second', state=<BindingState.
↳bound: 1>, avail=True, port=None, speed=115200)
```

2.1.5 Priorities

Each driver supports a priorities class variable. This allows drivers which implement the same protocol to add a priority option to each of their protocols. This way a *NetworkPowerDriver* can implement the *ResetProtocol*, but if another *ResetProtocol* driver with a higher protocol is available, it will be selected instead.

Note: Priority resolution only takes place if you have multiple drivers which implement the same protocol and you are not fetching them by name.

The target resolves the driver priority via the Method Resolution Order (MRO) of the driver's base classes. If a base class has a *priorities* dictionary which contains the requested Protocol as a key, that priority is used. Otherwise, 0 is returned as the default priority.

To set the priority of a protocol for a driver, add a class variable with the name *priorities*, e.g.

```
@attr.s
class NetworkPowerDriver(Driver, PowerProtocol, ResetProtocol):
    priorities: {PowerProtocol: -10}
```

2.1.6 Strategies

Especially when using labgrid from pytest, explicitly controlling the board's boot process can distract from the individual test case. Each *Strategy* implements the board- or project-specific actions necessary to transition from one state to another. Labgrid includes the *BareboxStrategy* and the *UBootStrategy*, which can be used as-is for simple cases or serve as an example for implementing a custom strategy.

Strategies themselves are not activated/deactivated. Instead, they control the states of the other drivers explicitly and execute actions to bring the target into the requested state.

See the strategy example (`examples/strategy`) and the included strategies in `labgrid/strategy` for some more information.

For more information on the reasons behind labgrid's architecture, see *Design Decisions*.

2.2 Remote Resources and Places

Labgrid contains components for accessing resources which are not directly accessible on the local machine. The main parts of this are:

labgrid-coordinator (crossbar component) Clients and exporters connect to the coordinator to publish resources, manage place configuration and handle mutual exclusion.

labgrid-exporter (CLI) Exports explicitly configured local resources to the coordinator and monitors these for changes in availability or parameters.

labgrid-client (CLI) Configures places (consisting of exported resources) and allows command line access to some actions (such as power control, bootstrap, fastboot and the console).

RemotePlace (managed resource) When used in a *Target*, the RemotePlace expands to the resources configured for the named places.

These components communicate over the [WAMP](#) implementation [Autobahn](#) and the [Crossbar](#) WAMP router.

2.2.1 Coordinator

The *Coordinator* is implemented as a Crossbar component and is started by the router. It provides separate RPC methods for the exporters and clients.

The coordinator keeps a list of all resources for clients and notifies them of changes as they occur. The resource access from clients does not pass through the coordinator, but is instead done directly from client to exporter, avoiding the need to specify new interfaces for each resource type.

The coordinator also manages the registry of “places”. These are used to configure which resources belong together from the user’s point of view. A *place* can be a generic rack location, where different boards are connected to a static set of interfaces (resources such as power, network, serial console, ...).

Alternatively, a *place* can also be created for a specific board, for example when special interfaces such as GPIO buttons need to be controlled and they are not available in the generic locations.

Each place can have aliases to simplify accessing a specific board (which might be moved between generic places). It also has a comment, which is used to store a short description of the connected board.

Finally, a place is configured with one or more *resource matches*. A resource match pattern has the format `<exporter>/<group>/<class>`, where each component may be replaced with the wildcard `*`.

Some commonly used match patterns are:

`*/1001/*` Matches all resources in groups named 1001 from all exporters.

`*/1001/NetworkPowerPort` Matches only the NetworkPowerPort resource in groups named 1001 from all exporters. This is useful to exclude a NetworkSerialPort in group 1001 in cases where the serial console is connected somewhere else (such as via USB on a different exporter).

`exporter1/hub1-port1/*` Matches all resources exported from exporter1 in the group hub1-port1. This is an easy way to match several USB resources related to the same board (such as a USB ROM-Loader interface, Android fastboot and a USB serial gadget in Linux).

To avoid conflicting access to the same resources, a place must be *acquired* before it is used and the coordinator also keeps track of which user on which client host has currently acquired the place. The resource matches are only evaluated while a place is being acquired and cannot be changed until it is *released* again.

2.2.2 Exporter

An exporters registers all its configured resources when it connects to the router and updates the resource parameters when they change (such as (dis-)connection of USB devices). Internally, the exporter uses the normal *Resource* (and *ManagedResource*) classes as the rest of labgrid. By using *ManagedResources*, availability and parameters for resources such as USB serial ports are tracked and sent to the coordinator.

For some specific resources (such as *USBSerialPorts*), the exporter uses external tools to allow access by clients (*ser2net* in the serial port case).

Resources which do not need explicit support in the exporter, are just published as declared in the configuration file. This is useful to register externally configured resources such as network power switches or serial port servers with a labgrid coordinator.

2.2.3 Client

The client requests the current lists of resources and places from the coordinator when it connects to it and then registers for change events. Most of its functionality is exposed via the *labgrid-client* CLI tool. It is also used by the *RemotePlace* resource (see below).

Besides viewing the list of *resources*, the client is used to configure and access *places* on the coordinator. For more information on using the CLI, see the manual page for *labgrid-client*.

2.2.4 RemotePlace

To use the resources configured for a *place* to control the corresponding board (whether in pytest or directly with the labgrid library), the *RemotePlace* resource should be used. When a *RemotePlace* is configured for a *Target*, it will create a client connection to the coordinator, create additional resource objects for those configured for that place and keep them updated at runtime.

The additional resource objects can be bound to by drivers as normal and the drivers do not need to be aware that they were provided by the coordinator. For resource types which do not have an existing, network-transparent protocol (such as USB ROM loaders or JTAG interfaces), the driver needs to be aware of the mapping done by the exporter.

For generic USB resources, the exporter for example maps a *AndroidFastboot* resource to a *NetworkAndroidFastboot* resource and adds a *hostname* property which needs to be used by the client to connect to the exporter. To avoid the need for additional remote access protocols and authentication, labgrid currently expects that the hosts are accessible via SSH and that any file names refer to a shared filesystem (such as NFS or SMB).

Note: Using SSH's session sharing (`ControlMaster auto, ControlPersist, ...`) makes *RemotePlaces* easy to use even for exporters with require passwords or more complex login procedures.

For exporters which are not directly accessible via SSH, add the host to your `.ssh/config` file, with a `ProxyCommand` when need.

3.1 Library

Labgrid can be used directly as a Python library, without the infrastructure provided by the pytest plugin.

3.1.1 Creating and Configuring Targets

The labgrid library provides two ways to configure targets with resources and drivers: either create the *Target* directly or use *Environment* to load a configuration file.

Targets

At the lower level, a *Target* can be created directly:

```
>>> from labgrid import Target
>>> t = Target('example')
```

Next, the required *Resources* can be created:

```
>>> from labgrid.resource import RawSerialPort
>>> rsp = RawSerialPort(t, name=None, port='/dev/ttyUSB0')
```

Note: Since we support multiple drivers of the same type, resources and drivers have a required name attribute. If you don't require support for this functionality set the name to *None*.

Then, a *Driver* needs to be created on the *Target*:

```
>>> from labgrid.driver import SerialDriver
>>> sd = SerialDriver(t, name=None)
```

As the *SerialDriver* declares a binding to a *SerialPort*, the target binds it to the resource created above:

```
>>> sd.port
RawSerialPort(target=Target(name='example', env=None), name=None, state=<BindingState.
↳bound: 1>, avail=True, port='/dev/ttyUSB0', speed=115200)
>>> sd.port is rsp
True
```

Before the driver can be used, it needs to be activated:

```
>>> t.activate(sd)
>>> sd.write(b'test')
```

Active drivers can be accessed by class (any *Driver* or *Protocol*) using some syntactic sugar:

```
>>> target = Target('main')
>>> console = FakeConsoleDriver(target, 'console')
>>> target.activate(console)
>>> target[FakeConsoleDriver]
FakeConsoleDriver(target=Target(name='main', ...), name='console', ...)
>>> target[FakeConsoleDriver, 'console']
FakeConsoleDriver(target=Target(name='main', ...), name='console', ...)
```

Environments

In practice, it is often useful to separate the *Target* configuration from the code which needs to control the board (such as a test case or installation script). For this use-case, labgrid can construct targets from a configuration file in YAML format:

```
targets:
  example:
    resources:
      RawSerialPort:
        port: '/dev/ttyUSB0'
    drivers:
      SerialDriver: {}
```

To parse this configuration file, use the *Environment* class:

```
>>> from labgrid import Environment
>>> env = Environment('example-env.yaml')
```

Using *Environment.get_target*, the configured *Targets* can be retrieved by name. Without an argument, *get_target* would default to 'main':

```
>>> t = env.get_target('example')
```

To access the target's console, the correct driver object can be found by using *Target.get_driver*:

```
>>> from labgrid.protocol import ConsoleProtocol
>>> cp = t.get_driver(ConsoleProtocol)
>>> cp
SerialDriver(target=Target(name='example', env=Environment(config_file='example.yaml
↳')), name=None, state=<BindingState.active: 2>, txdelay=0.0)
>>> cp.write(b'test')
```

When using the `get_driver` method, the driver is automatically activated. The driver activation will also wait for unavailable resources when needed.

3.2 pytest Plugin

Labgrid includes a `pytest` plugin to simplify writing tests which involve embedded boards. The plugin is configured by providing an environment config file (via the `-lg-env` `pytest` option) and automatically creates the targets described in the environment.

Two `pytest` fixtures are provided:

env (session scope) Used to access the `Environment` object created from the configuration file. This is mostly used for defining custom fixtures at the test suite level.

target (session scope) Used to access the 'main' `Target` defined in the configuration file.

The `pytest` plugin also supports the verbosity argument of `pytest`:

- `-vv`: activates the step reporting feature, showing function parameters and/or results
- `-vvv`: activates debug logging

This allows debugging during the writing of tests and inspection during test runs.

3.2.1 Simple Example

As a minimal example, we have a target connected via a USB serial converter (`/dev/ttyUSB0`) and booted to the Linux shell. The following environment config file (`shell-example.yaml`) describes how to access this board:

```
targets:
  main:
    resources:
      RawSerialPort:
        port: '/dev/ttyUSB0'
    drivers:
      SerialDriver: {}
      ShellDriver:
        prompt: 'root@\w+: [^ ]+ '
        login_prompt: ' login: '
        username: 'root'
```

We then add the following test in a file called `test_example.py`:

```
from labgrid.protocol import CommandProtocol

def test_echo(target):
    command = target.get_driver(CommandProtocol)
    result = command.run_check('echo OK')
    assert 'OK' in result
```

To run this test, we simply execute `pytest` in the same directory with the environment config:

```
$ pytest --lg-env shell-example.yaml --verbose
===== test session starts =====
platform linux -- Python 3.5.3, pytest-3.0.6, py-1.4.32, pluggy-0.4.0
...
collected 1 items
```

```
test_example.py::test_echo PASSED
===== 1 passed in 0.51 seconds =====
```

pytest has automatically found the test case and executed it on the target.

3.2.2 Custom Fixture Example

When writing many test cases which use the same driver, we can get rid of some common code by wrapping the *CommandProtocol* in a fixture. As pytest always executes the `confest.py` file in the test suite directory, we can define additional fixtures there:

```
import pytest

from labgrid.protocol import CommandProtocol

@pytest.fixture(scope='session')
def command(target):
    return target.get_driver(CommandProtocol)
```

With this fixture, we can simplify the `test_example.py` file to:

```
def test_echo(command):
    result = command.run_check('echo OK')
    assert 'OK' in result
```

3.2.3 Strategy Fixture Example

When using a *Strategy* to transition the target between states, it is useful to define a function scope fixture per state in `confest.py`:

```
import pytest

from labgrid.protocol import CommandProtocol
from labgrid.strategy import BareboxStrategy

@pytest.fixture(scope='session')
def strategy(target):
    try:
        return target.get_driver(BareboxStrategy)
    except NoDriverFoundError:
        pytest.skip("strategy not found")

@pytest.fixture(scope='function')
def bootloader_command(target, strategy, capsys):
    with capsys.disabled():
        strategy.transition('barebox')
    return target.get_active_driver(CommandProtocol)

@pytest.fixture(scope='function')
def shell_command(target, strategy, capsys):
    with capsys.disabled():
        strategy.transition('shell')
    return target.get_active_driver(CommandProtocol)
```

Note: The `capsys.disabled()` context manager is only needed when using the `ManualPowerDriver`, as it will not be able to access the console otherwise. See the corresponding `pytest` documentation for details.

With the fixtures defined above, switching between bootloader and Linux shells is easy:

```
def test_barebox_initial(bootloader_command):
    stdout = bootloader_command.run_check('version')
    assert 'barebox' in '\n'.join(stdout)

def test_shell(shell_command):
    stdout = shell_command.run_check('cat /proc/version')
    assert 'Linux' in stdout[0]

def test_barebox_after_reboot(bootloader_command):
    bootloader_command.run_check('true')
```

Note: The `bootloader_command` and `shell_command` fixtures use `Target.get_active_driver` to get the currently active `CommandProtocol` driver (either `BareboxDriver` or `ShellDriver`). Activation and deactivation of drivers is handled by the `BareboxStrategy` in this example.

The `Strategy` needs additional drivers to control the target. Adapt the following environment config file (`strategy-example.yaml`) to your setup:

```
targets:
  main:
    resources:
      RawSerialPort:
        port: '/dev/ttyUSB0'
    drivers:
      ManualPowerDriver:
        name: 'example-board'
      SerialDriver: {}
      BareboxDriver:
        prompt: 'barebox@[^:]+:[^ ]+ '
      ShellDriver:
        prompt: 'root@\w+:[^ ]+ '
        login_prompt: ' login: '
        username: 'root'
      BareboxStrategy: {}
```

For this example, you should get a report similar to this:

```
$ pytest --lg-env strategy-example.yaml -v
===== test session starts =====
platform linux -- Python 3.5.3, pytest-3.0.6, py-1.4.32, pluggy-0.4.0
...
collected 3 items

test_strategy.py::test_barebox_initial
main: CYCLE the target example-board and press enter
PASSED
test_strategy.py::test_shell PASSED
test_strategy.py::test_barebox_after_reboot
main: CYCLE the target example-board and press enter
PASSED
```

```
===== 3 passed in 29.77 seconds =====
```

3.2.4 Test Reports

pytest-html

With the `pytest-html` plugin, the test results can be converted directly to a single-page HTML report:

```
$ pip install pytest-html
$ pytest --lg-env shell-example.yaml --html=report.html
```

JUnit XML

JUnit XML reports can be generated directly by pytest and are especially useful for use in CI systems such as Jenkins with the `JUnit Plugin`.

They can also be converted to other formats, such as HTML with `junit2html` tool:

```
$ pip install junit2html
$ pytest --lg-env shell-example.yaml --junit-xml=report.xml
$ junit2html report.xml
```

Labgrid adds additional xml properties to a test run, these are:

- `ENV_CONFIG`: Name of the configuration file
- `TARGETS`: List of target names
- `TARGET_{NAME}_REMOTE`: optional, if the target uses a RemotePlace resource, its name is recorded here
- `PATH_{NAME}`: optional, labgrid records the name and path
- `PATH_{NAME}_GIT_COMMIT`: optional, labgrid tries to record git sha1 values for every path
- `IMAGE_{NAME}`: optional, labgrid records the name and path to the image
- `IMAGE_{NAME}_GIT_COMMIT`: optional, labgrid tries to record git sha1 values for every image

3.3 Command-Line

Labgrid contains some command line tools which are used for remote access to resources. See *labgrid-client*, *labgrid-device-config* and *labgrid-exporter* for more information.

3.4 USB stick emulation

Labgrid makes it possible to use a target as an emulated USB stick, allowing upload, modification, plug and unplug events. To use a target as an emulated USB stick, several requirements have to be met:

- OTG support on one of the device USB ports
- `losetup` from `util-linux`
- `mount` from `util-linux`

- A kernel build with `CONFIG_USB_GADGETFS=m`
- A network connection to the target to use the `SSHDriver` for file uploads

To use USB stick emulation, import `USBStick` from `labgrid.external` and bind it to the desired target:

```
from labgrid.external import USBStick

stick = USBStick(target, '/home/')
```

The above code block creates the stick and uses the `/home` directory to store the device images. `USBStick` images can now be uploaded using the `upload_image` method. Once an image is selected, files can be uploaded and retrieved using the `put_file` and `get_file` methods. The `plug_in` and `plug_out` functions plug the emulated USB stick in and out.

3.5 hawkBit management API

Labgrid provides an interface to the hawkbit management API. This allows a labgrid test to create targets, rollouts and manage deployments.

```
from labgrid.external import HawkbitTestClient

client = HawkbitTestClient('local', '8080', 'admin', 'admin')
```

The above code connects to a running hawkbit instance on the local computer and uses the default credentials to log in. The `HawkbitTestClient` provides various helper functions to add targets, define distribution sets and assign targets.

4.1 labgrid-client

4.1.1 labgrid-client interface to control boards

Author Rouven Czerwinski <r.czerwinski@pengutronix.de>

organization Labgrid-Project

Date 2017-04-15

Copyright Copyright (C) 2016-2017 Pengutronix. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

Version 0.0.1

Manual section 1

Manual group embedded testing

SYNOPSIS

```
labgrid-client --help
labgrid-client -p <place> <command>
labgrid-client places | resources
```

DESCRIPTION

Labgrid is a scalable infrastructure and test architecture for embedded (linux) systems. This is the client to control a boards status and interface with it on remote machines.

OPTIONS

- h, --help** display command line help
- p PLACE, --place PLACE** specify the place to operate on
- x, --crossbar-url** the crossbar url of the coordinator
- c CONFIG, --config CONFIG** set the configuration file
- s STATE, --state STATE** set an initial state before executing a command, requires a configuration file and strategy
- d, --debug** enable debugging

CONFIGURATION FILE

The configuration file follows the description in `labgrid-device-config(1)`.

ENVIRONMENT VARIABLES

Various labgrid-client commands use the following environment variable:

PLACE

This variable can be used to specify a place without using the `-p` option, the `-p` option overrides it.

STATE

This variable can be used to specify a state which the device transitions into before executing a command. Requires a configuration file and a Strategy specified for the device.

LG_CROSSBAR

This variable can be used to set the default crossbar URL (instead of using the `-x` option).

LG_CROSSBAR_REALM

This variable can be used to set the default crossbar realm to use instead of `realm1`.

MATCHES

Match patterns are used to assign a resource to a specific place. The format is: `exporter/group/cls/name`, `exporter` is the name of the exporting machine, `group` is a name defined within the exporter, `cls` is the class of the exported resource and `name` is its name. Wild cards in match patterns are explicitly allowed, `*` matches anything.

LABGRID-CLIENT COMMANDS

monitor Monitor events from the coordinator
resources (r) List available resources
places (p) List available places
show Show a place and related resources
create Add a new place (name supplied by -p parameter)
delete Delete an existing place
add-alias Add an alias to a place
del-alias Delete an alias from a place
set-comment Update or set the place comment
add-match match Add a match pattern to a place, see MATCHES
del-match match Delete a match pattern from a place, see MATCHES
acquire (lock) Acquire a place
release (unlock) Release a place
env Generate a labgrid environment file for a place
power (pw) action Change (or get) a place's power status, where action is one of get, on, off, status
console (con) Connect to the console
fastboot Run fastboot
bootstrap Start a bootloader
io Interact with Onewire devices

EXAMPLES

To retrieve a list of places run:

```
$ labgrid-client places
```

To access a place, it needs to be acquired first, this can be done by running the `acquire` command and passing the placename as a -p parameter:

```
$ labgrid-client -p <placename> acquire
```

Open a console to the acquired place:

```
$ labgrid-client -p <placename> console
```

Add all resources with the group “example-group” to the place example-place:

```
$ labgrid-client -p example-place add-match */example-group/**
```

SEE ALSO

labgrid-exporter(1)

4.2 labgrid-device-config

4.2.1 labgrid test configuration files

Author Rouven Czerwinski <r.czerwinski@pengutronix.de>

organization Labgrid-Project

Date 2017-04-15

Copyright Copyright (C) 2016-2017 Pengutronix. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

Version 0.0.1

Manual section 1

Manual group embedded testing

SYNOPSIS

*.yaml

DESCRIPTION

To integrate a device into a labgrid test, labgrid needs to have a description of the device and how to access it.

This manual page is divided into section, each describing one top-level yaml key.

TARGETS

The `targets:` top key configures a target, it's `drivers` and `resources`.

The top level key is the name of the target, it needs both a `resources` and `drivers` subkey. The order of instantiated `resources` and `drivers` is important, since they are parsed as an ordered dictionary and may depend on a previous driver.

For a list of available resources and drivers refer to <https://labgrid.readthedocs.io/en/latest/configuration.html>.

OPTIONS

The `options:` top key configures various options such as the `crossbar_url`.

KEYS

`crossbar_url` takes as parameter the URL of the crossbar (coordinator) to connect to. Defaults to `'ws://127.0.0.1:20408'`.

`crossbar_realm` takes as parameter the realm of the crossbar (coordinator) to connect to. Defaults to `'realm1'`.

IMAGES

The `images:` top key provides paths to access preconfigured images to flash onto the board.

KEYS

The subkeys consist of image names as keys and their paths as values. The corresponding name can then be used with the appropriate tool found under TOOLS.

TOOLS

The `tools:` top key provides paths to binaries such as fastboot.

KEYS

fastboot Path to the fastboot binary

mxs-usb-loader Path to the mxs-usb-loader binary

imx-usb-loader Path to the imx-usb-loader binary

IMPORTS

The `imports` key is a list of files which are imported by the environment after loading the configuration. Relative paths to the configuration file are also supported.

EXAMPLES

A sample configuration with one *main* target, accessible via SerialPort */dev/ttyUSB0*, allowing usage of the ShellDriver:

```
targets:
  main:
    resources:
      RawSerialPort:
        port: "/dev/ttyUSB0"
    drivers:
      SerialDriver: {}
      ShellDriver:
        prompt: 'root@\w+: [^ ]+ '
        login_prompt: ' login: '
        username: 'root'
```

SEE ALSO

labgrid-client(1), labgrid-exporter(1)

4.3 labgrid-exporter

4.3.1 labgrid-exporter interface to control boards

Author Rouven Czerwinski <r.czerwinski@pengutronix.de>

organization Labgrid-Project

Date 2017-04-15

Copyright Copyright (C) 2016-2017 Pengutronix. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

Version 0.0.1

Manual section 1

Manual group embedded testing

SYNOPSIS

```
labgrid-exporter --help
```

```
labgrid-exporter *.yaml
```

DESCRIPTION

Labgrid is a scalable infrastructure and test architecture for embedded (linux) systems.

This is the man page for the exporter, supporting the export of serial ports, USB devices and various other controllers.

OPTIONS

- | | |
|---------------------------|--|
| -h, --help | display command line help |
| -x, --crossbar-url | the crossbar url of the coordinator |
| -n, --name | the public name of the exporter |
| --hostname | hostname (or IP) published for accessing resources |

-n / --name

This option is used to configure the exporter name under which resources are registered with the coordinator, which is useful when running multiple exporters on the same host. It defaults to the system hostname.

--hostname

For resources like USBSerialPort, USBGenericExport or USBSigrokExport, the exporter needs to provide a host name to set the exported value of the “host” key. If the system hostname is not resolvable via DNS, this option can be used to override this default with another name (or an IP address).

CONFIGURATION

The exporter uses a YAML configuration file which defines groups of related resources. Furthermore the exporter can start helper binaries such as `ser2net` to export local serial ports over the network.

ENVIRONMENT VARIABLES

The following environment variable can be used to configure labgrid-exporter.

LG_CROSSBAR

This variable can be used to set the default crossbar URL (instead of using the `-x` option).

LG_CROSSBAR_REALM

This variable can be used to set the default crossbar realm to use instead of `realm1`.

EXAMPLES

Start the exporter with the configuration file *my-config.yaml*:

```
$ labgrid-exporter my-config.yaml
```

Same as above, but with name `myname`:

```
$ labgrid-exporter -n myname my-config.yaml
```

SEE ALSO

`labgrid-client(1)`, `labgrid-device-config(1)`

This chapter describes the individual drivers and resources used in a device configuration. Drivers can depend on resources or other drivers, whereas resources have no dependencies.

Here the resource *RawSerialPort* provides the information for the *SerialDriver*, which in turn is needed by the *ShellDriver*. Driver dependency resolution is done by searching for the driver which implements the dependent protocol, all drivers implement one or more protocols.

5.1 Resources

5.1.1 Serial Ports

RawSerialPort

A *RawSerialPort* is a serial port which is identified via the device path on the local computer. Take note that re-plugging USB serial converters can result in a different enumeration order.

```
RawSerialPort:
  port: /dev/ttyUSB0
  speed: 115200
```

The example would access the serial port `/dev/ttyUSB0` on the local computer with a baud rate of 115200.

- port (str): path to the serial device
- speed (int): desired baud rate

Used by:

- *SerialDriver*

NetworkSerialPort

A NetworkSerialPort describes a serial port which is exported over the network, usually using RFC2217 or raw tcp.

```
NetworkSerialPort:
  host: remote.example.computer
  port: 53867
  speed: 115200
```

The example would access the serial port on computer `remote.example.computer` via port `53867` and use a baud rate of `115200` with the RFC2217 protocol.

- `host` (str): hostname of the remote host
- `port` (str): TCP port on the remote host to connect to
- `speed` (int): baud rate of the serial port
- `protocol` (str): optional, protocol used for connection: `raw` or `rfc2217`

Used by:

- *SerialDriver*

USBSerialPort

A USBSerialPort describes a serial port which is connected via USB and is identified by matching udev properties. This allows identification through hot-plugging or rebooting.

```
USBSerialPort:
  match:
    'ID_SERIAL_SHORT': 'P-00-00682'
  speed: 115200
```

The example would search for a USB serial converter with the key `ID_SERIAL_SHORT` and the value `P-00-00682` and use it with a baud rate of `115200`.

- `match` (str): key and value for a udev match, see *udev Matching*
- `speed` (int): baud rate of the serial port

Used by:

- *SerialDriver*

5.1.2 Power Ports

NetworkPowerPort

A NetworkPowerPort describes a remotely switchable power port.

```
NetworkPowerPort:
  model: gude
  host: powerswitch.example.computer
  index: 0
```

The example describes port `0` on the remote power switch `powerswitch.example.computer`, which is a `gude` model.

- `model` (str): model of the power switch

- host (str): hostname of the power switch
- index (int): number of the port to switch

Used by:

- *NetworkPowerDriver*

YKUSHPowerPort

A YKUSHPowerPort describes a YEPKIT YKUSH USB (HID) switchable USB hub.

```
YKUSHPowerPort:
  serial: YK12345
  index: 1
```

The example describes port 1 on the YKUSH USB hub with the serial “YK12345”. (use “pykush -l” to get your serial...)

- serial (str): serial number of the YKUSH hub
- index (int): number of the port to switch

Used by:

- *YKUSHPowerDriver*

USBPowerPort

A USBPowerPort describes a generic switchable USB hub as supported by `uhubctl`.

```
USBPowerPort:
  match:
    ID_PATH: pci-0000:00:14.0-usb-0:2:1.0
  index: 1
```

The example describes port 1 on the hub with the ID_PATH “pci-0000:00:14.0-usb-0:2:1.0”. (use “udevadmin info /sys/bus/usb/devices/...” to find the ID_PATH value)

- index (int): number of the port to switch

Used by:

- *USBPowerDriver*

5.1.3 ModbusTCPCoil

A ModbusTCPCoil describes a coil accessible via ModbusTCP.

```
ModbusTCPCoil:
  host: "192.168.23.42"
  coil: 1
```

The example describes the coil with the address 1 on the ModbusTCP device `192.168.23.42`.

- host (str): hostname of the Modbus TCP server e.g. “192.168.23.42:502”
- coil (int): index of the coil e.g. 3
- invert (bool): optional, whether the logic level is be inverted (active-low)

Used by:

- *ModbusCoilDriver*

5.1.4 NetworkService

A NetworkService describes a remote SSH connection.

```
NetworkService:
  address: example.computer
  username: root
```

The example describes a remote SSH connection to the computer *example.computer* with the username *root*. Set the optional password `password` property to make SSH login with a password instead of the key file (needs `sshpass` to be installed)

- `address` (str): hostname of the remote system
- `username` (str): username used by SSH
- `password` (str): password used by SSH

Used by:

- *SSHDriver*

5.1.5 OneWirePIO

A OneWirePIO describes a onewire programmable I/O pin.

```
OneWirePIO:
  host: example.computer
  path: /29.7D6913000000/PIO.0
  invert: false
```

The example describes a *PIO.0* at device address *29.7D6913000000* via the onewire server on *example.computer*.

- `host` (str): hostname of the remote system running the onewire server
- `path` (str): path on the server to the programmable I/O pin
- `invert` (bool): optional, whether the logic level is be inverted (active-low)

Used by:

- *OneWirePIODriver*

5.1.6 USBMassStorage

A USBMassStorage resource describes a USB memory stick or similar device.

```
USBMassStorage:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0-scsi-0:0:0:3'
```

- `match` (str): key and value for a udev match, see *udev Matching*

Used by:

- *USBStorageDriver*
- *NetworkUSBStorageDriver*

5.1.7 NetworkUSBMassStorage

A NetworkUSBMassStorage resource describes a USB memory stick or similar device available on a remote computer.

Used by:

- *NetworkUSBStorageDriver*

The NetworkUSBMassStorage can be used in test cases by calling the *write_image()*, and *get_size()* functions.

5.1.8 SigrokDevice

A SigrokDevice resource describes a sigrok device. To select a specific device from all connected supported devices use the *SigrokUSBDevice*.

```
SigrokUSBDevice:
  driver: fx2lafw
  channel: "D0=CLK,D1=DATA"
```

- driver (str): name of the sigrok driver to use
- channel (str): channel mapping as described in the sigrok-cli man page

Used by:

- *SigrokDriver*

5.1.9 IMXUSBLoader

An IMXUSBLoader resource describes a USB device in the imx loader state.

```
IMXUSBLoader:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- match (str): key and value for a udev match, see *udev Matching*

Used by:

- *IMXUSBDriver*

5.1.10 MXSUSBLoader

An MXSUSBLoader resource describes a USB device in the mxs loader state.

```
MXSUSBLoader:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- match (str): key and value for a udev match, see *udev Matching*

Used by:

- *MXSUSBDriver*

5.1.11 NetworkMXSUSBLoader

A NetworkMXSUSBLoader describes an *MXSUSBLoader* available on a remote computer.

5.1.12 NetworkIMXUSBLoader

A NetworkIMXUSBLoader describes an *IMXUSBLoader* available on a remote computer.

5.1.13 AndroidFastboot

An AndroidFastboot resource describes a USB device in the fastboot state.

```
AndroidFastboot:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- match (str): key and value for a udev match, see *udev Matching*

Used by:

- *AndroidFastbootDriver*

5.1.14 USBEthernetInterface

A USBEthernetInterface resource describes a USB device Ethernet adapter.

```
USBEthernetInterface:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- match (str): key and value for a udev match, see *udev Matching*

5.1.15 AlteraUSBBlaster

An AlteraUSBBlaster resource describes an Altera USB blaster.

```
AlteraUSBBlaster:
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- match (dict): key and value for a udev match, see *udev Matching*

Used by:

- *OpenOCDDriver*
- *QuartusHPSDriver*

5.1.16 SNMPEthernetPort

A SNMPEthernetPort resource describes a port on an Ethernet switch, which is accessible via SNMP.


```
SNMPEthernetPort:
  switch: "switch-012"
  interface: "17"
```

- switch (str): host name of the Ethernet switch
- interface (str): interface name

5.1.17 SigrokUSBDevice

A `SigrokUSBDevice` resource describes a sigrok USB device.

```
SigrokUSBDevice:
  driver: fx2lafw
  channel: "D0=CLK,D1=DATA"
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
```

- driver (str): name of the sigrok driver to use
- channel (str): channel mapping as described in the sigrok-cli man page
- match (str): key and value for a udev match, see *udev Matching*

Used by:

- *SigrokDriver*

5.1.18 USBSDMuxDevice

A `USBSDMuxDevice` resource describes a Pengutronix `USB-SD-Mux` device.

```
USBSDMuxDevice:
  match:
    '@ID_PATH': 'pci-0000:00:14.0-usb-0:1.2'
```

- match (str): key and value for a udev match, see *udev Matching*

Used by:

- *USBSDMUXDriver*

5.1.19 NetworkUSBSDMuxDevice

A `NetworkUSBSDMuxDevice` resource describes a `USBSDMuxDevice` available on a remote computer.

5.1.20 RemotePlace

A `RemotePlace` describes a set of resources attached to a labgrid remote place.

```
RemotePlace:
  name: example-place
```

The example describes the remote place *example-place*. It will connect to the labgrid remote coordinator, wait until the resources become available and expose them to the internal environment.

- name (str): name or pattern of the remote place

Used by:

- potentially all drivers

5.1.21 udev Matching

udev matching allows labgrid to identify resources via their udev properties. Any udev property key and value can be used, path matching USB devices is allowed as well. This allows exporting a specific USB hub port or the correct identification of a USB serial converter across computers.

The initial matching and monitoring for udev events is handled by the *UdevManager* class. This manager is automatically created when a resource derived from *USBResource* (such as *USBSerialPort*, *IMXUSBLoader* or *AndroidFastboot*) is instantiated.

To identify the kernel device which corresponds to a configured *USBResource*, each existing (and subsequently added) kernel device is matched against the configured resources. This is based on a list of *match entries* which must all be tested successfully against the potential kernel device. Match entries starting with an @ are checked against the device's parents instead of itself; here one matching parent causes the check to be successful.

A given *USBResource* class has builtin match entries that are checked first, for example that the `SUBSYSTEM` is `tty` as in the case of the *USBSerialPort*. Only if these succeed, match entries provided by the user for the resource instance are considered.

In addition to the properties reported by `udevadm monitor --udev --property`, elements of the `ATTR(S){}` dictionary (as shown by `udevadm info <device> -a`) are useable as match keys. Finally `sys_name` allows matching against the name of the directory in `sysfs`. All match entries must succeed for the device to be accepted.

The following examples show how to use the udev matches for some common use-cases.

Matching a USB Serial Converter on a Hub Port

This will match any USB serial converter connected below the hub port 1.2.5.5 on bus 1. The `sys_name` value corresponds to the hierarchy of buses and ports as shown with `lsusb -t` and is also usually displayed in the kernel log messages when new devices are detected.

```
USBSerialPort:
  match:
    '@sys_name': '1-1.2.5.5'
```

Note the @ in the `@sys_name` match, which applies this match to the device's parents instead of directly to itself. This is necessary for the *USBSerialPort* because we actually want to find the `ttyUSB?` device below the USB serial converter device.

Matching an Android Fastboot Device

In this case, we want to match the USB device on that port directly, so we don't use a parent match.

```
AndroidFastboot:
  match:
    'sys_name': '1-1.2.3'
```

Matching a Specific UART in a Dual-Port Adapter

On this board, the serial console is connected to the second port of an on-board dual-port USB-UART. The board itself is connected to the bus 3 and port path 10.2.2.2. The correct value can be shown by running `udevadm info /dev/ttyUSB9` in our case:

```
$ udevadm info /dev/ttyUSB9
P: /devices/pci0000:00/0000:00:14.0/usb3/3-10/3-10.2/3-10.2.2/3-10.2.2.2/3-10.2.2.2:1.
↳1/ttyUSB9/tty/ttyUSB9
N: ttyUSB9
S: serial/by-id/usb-FTDI_Dual_RS232-HS-if01-port0
S: serial/by-path/pci-0000:00:14.0-usb-0:10.2.2.2:1.1-port0
E: DEVLINKS=/dev/serial/by-id/usb-FTDI_Dual_RS232-HS-if01-port0 /dev/serial/by-path/
↳pci-0000:00:14.0-usb-0:10.2.2.2:1.1-port0
E: DEVNAME=/dev/ttyUSB9
E: DEVPATH=/devices/pci0000:00/0000:00:14.0/usb3/3-10/3-10.2/3-10.2.2/3-10.2.2.2/3-10.
↳2.2.2:1.1/ttyUSB9/tty/ttyUSB9
E: ID_BUS=usb
E: ID_MODEL=Dual_RS232-HS
E: ID_MODEL_ENC=Dual\x20RS232-HS
E: ID_MODEL_FROM_DATABASE=FT2232C Dual USB-UART/FIFO IC
E: ID_MODEL_ID=6010
E: ID_PATH=pci-0000:00:14.0-usb-0:10.2.2.2:1.1
E: ID_PATH_TAG=pci-0000_00_14_0-usb-0_10_2_2_2_1_1
E: ID_REVISION=0700
E: ID_SERIAL=FTDI_Dual_RS232-HS
E: ID_TYPE=generic
E: ID_USB_DRIVER=ftdi_sio
E: ID_USB_INTERFACES=:fffff:
E: ID_USB_INTERFACE_NUM=01
E: ID_VENDOR=FTDI
E: ID_VENDOR_ENC=FTDI
E: ID_VENDOR_FROM_DATABASE=Future Technology Devices International, Ltd
E: ID_VENDOR_ID=0403
E: MAJOR=188
E: MINOR=9
E: SUBSYSTEM=tty
E: TAGS=:systemd:
E: USEC_INITIALIZED=9129609697
```

We use the `ID_USB_INTERFACE_NUM` to distinguish between the two ports:

```
USBSerialPort:
match:
  '@sys_name': '3-10.2.2.2'
  'ID_USB_INTERFACE_NUM': '01'
```

Matching a USB UART by Serial Number

Most of the USB serial converters in our lab have been programmed with unique serial numbers. This makes it easy to always match the same one even if the USB topology changes or a board has been moved between host systems.

```
USBSerialPort:
match:
  'ID_SERIAL_SHORT': 'P-00-00679'
```

To check if your device has a serial number, you can use `udevadm info`:

```
$ udevadm info /dev/ttyUSB5 | grep SERIAL_SHORT
E: ID_SERIAL_SHORT=P-00-00679
```

5.2 Drivers

5.2.1 SerialDriver

A SerialDriver connects to a serial port. It requires one of the serial port resources.

Binds to:

- *NetworkSerialPort*
- *RawSerialPort*
- *USBSerialPort*

```
SerialDriver:
  txdelay: 0.05
```

Implements:

- *ConsoleProtocol*

Arguments:

- txdelay (float): time in seconds to wait before sending each byte

5.2.2 ShellDriver

A ShellDriver binds on top of a *ConsoleProtocol* and is designed to interact with a login prompt and a Linux shell.

Binds to:

- *ConsoleProtocol* (see *SerialDriver*)

Implements:

- *CommandProtocol*

```
ShellDriver:
  prompt: 'root@\w+: [^ ]+ '
  login_prompt: ' login: '
  username: 'root'
```

Arguments:

- prompt (regex): shell prompt to match after logging in
- login_prompt (regex): match for the login prompt
- username (str): username to use during login
- password (str): password to use during login
- keyfile (str): optional keyfile to upload after login, making the *SSHDriver* usable
- login_timeout (int): optional, timeout for login prompt detection in seconds (default 60)

- `await_login_timeout` (int): optional, time in seconds of silence that needs to pass before sending a newline to device.
- `console_ready` (regex): optional, pattern used by the kernel to inform the user that a console can be activated by pressing enter.

5.2.3 SSHDriver

A `SSHDriver` requires a `NetworkService` resource and allows the execution of commands and file upload via network.

Binds to:

- `NetworkService`

Implements:

- `CommandProtocol`
- `FileTransferProtocol`

```
SSHDriver:
  keyfile: example.key
```

Arguments:

- `keyfile` (str): filename of private key to login into the remote system (only used if password is not set)

5.2.4 InfoDriver

An `InfoDriver` provides an interface to retrieve system settings and state. It requires a `CommandProtocol`.

Binds to:

- `CommandProtocol` (see `ShellDriver`)

Implements:

- `InfoProtocol`

```
InfoDriver: {}
```

Arguments:

- None

5.2.5 UBootDriver

A `UBootDriver` interfaces with a u-boot boot loader via a `ConsoleProtocol`.

Binds to:

- `ConsoleProtocol` (see `SerialDriver`)

Implements:

- `CommandProtocol`

```
UBootDriver:
  prompt: 'Uboot> '
```

Arguments:

- prompt (regex): u-boot prompt to match
- password (str): optional, u-boot unlock password
- interrupt (str, default="\n"): string to interrupt autoboot (use "\x03" for CTRL-C)
- init_commands (tuple): tuple of commands to execute after matching the prompt
- password_prompt (str): optional, regex to match the uboot password prompt, defaults to "enter Password:"
- boot_expression (str): optional, regex to match the uboot start string defaults to "U-Boot 20d+"
- bootstring (str): optional, regex to match on Linux Kernel boot

5.2.6 SmallUBootDriver

A SmallUBootDriver interfaces with stripped-down UBoot variants that are sometimes used in cheap consumer electronics.

SmallUBootDriver is meant as a driver for UBoot with only little functionality compared to standard a standard UBoot. Especially it copes with the following limitations:

- The UBoot does not have a real password-prompt but can be activated by entering a "secret" after a message was displayed.
- The command line does not have a build-in echo command. Thus this driver uses 'Unknown Command' messages as marker before and after the output of a command.
- Since there is no echo we can not return the exit code of the command. Commands will always return 0 unless the command was not found.

This driver needs the following features activated in UBoot to work:

- The UBoot must not have real password prompt. Instead it must be keyword activated. For example it should be activated by a dialog like the following:
 - UBoot: "Autobooting in 1s..."
 - Labgrid: "secret"
 - UBoot: <switching to console>
- The UBoot must be able to parse multiple commands in a single line separated by ";".
- The UBoot must support the "bootm" command to boot from a memory location.

Binds to:

- *ConsoleProtocol* (see *SerialDriver*)

Implements:

- *CommandProtocol*

```
SmallUBootDriver:
  prompt: 'ap143-2\.0> '
  boot_expression: 'Autobooting in 1 seconds'
  boot_secret: "tpl"
```

Arguments:

- prompt (regex): u-boot prompt to match

- `init_commands` (tuple): tuple of commands to execute after matching the prompt
- `boot_expression` (str): optional, regex to match the uboot start string defaults to “U-Boot 20d+”

5.2.7 BareboxDriver

A `BareboxDriver` interfaces with a barebox bootloader via a `ConsoleProtocol`.

Binds to:

- `ConsoleProtocol` (see `SerialDriver`)

Implements:

- `CommandProtocol`

```
BareboxDriver:
  prompt: 'barebox@[^:]+:[^ ]+ '
```

Arguments:

- `prompt` (regex): barebox prompt to match
- `autoboot` (regex, default=”stop autoboot”): autoboot message to match
- `interrupt` (str, default=”\n”): string to interrupt autoboot (use “\x03” for CTRL-C)
- `bootstring` (regex, default=”Linux version d”): succesfully jumped into the kernel

5.2.8 ExternalConsoleDriver

An `ExternalConsoleDriver` implements the `ConsoleProtocol` on top of a command executed on the local computer.

Implements:

- `ConsoleProtocol`

```
ExternalConsoleDriver:
  cmd: 'microcom /dev/ttyUSB2'
  txdelay: 0.05
```

Arguments:

- `cmd` (str): command to execute and then bind to.
- `txdelay` (float): time in seconds to wait before sending each byte

5.2.9 AndroidFastbootDriver

An `AndroidFastbootDriver` allows the upload of images to a device in the USB fastboot state.

Binds to:

- `AndroidFastboot`

Implements:

- None (yet)

```
AndroidFastbootDriver:
  image: mylocal.image
```

Arguments:

- image (str): filename of the image to upload to the device

5.2.10 OpenOCDDriver

An OpenOCDDriver controls OpenOCD to bootstrap a target with a bootloader.

Binds to:

- *AlteraUSBBlaster*

Implements:

- *BootstrapProtocol*

Arguments:

- config (str): OpenOCD configuration file
- search (str): include search path for scripts
- image (str): filename of image to bootstrap onto the device

5.2.11 QuartusHPSDriver

A QuartusHPSDriver controls the “Quartus Prime Programmer and Tools” to flash a target’s QSPI.

Binds to:

- *AlteraUSBBlaster*

Implements:

- None

Arguments:

- image (str): filename of image to flash QSPI

The driver can be used in test cases by calling the *flash* function. An example strategy is included in Labgrid.

5.2.12 ManualPowerDriver

A ManualPowerDriver requires the user to control the target power states. This is required if a strategy is used with the target, but no automatic power control is available.

Implements:

- *PowerProtocol*

```
ManualPowerDriver:  
  name: 'example-board'
```

Arguments:

- name (str): name of the driver (will be displayed during interaction)

5.2.13 ExternalPowerDriver

An ExternalPowerDriver is used to control a target power state via an external command.

Implements:

- *PowerProtocol*

```
ExternalPowerDriver:
  cmd_on: example_command on
  cmd_off: example_command off
  cmd_cycle: example_command cycle
```

Arguments:

- cmd_on (str): command to turn power to the board on
- cmd_off (str): command to turn power to the board off
- cycle (str): optional command to switch the board off and on
- delay (float): configurable delay in seconds between off and on if cycle is not set

5.2.14 NetworkPowerDriver

A NetworkPowerDriver controls a *NetworkPowerPort*, allowing control of the target power state without user interaction.

Binds to:

- *NetworkPowerPort*

Implements:

- *PowerProtocol*

```
NetworkPowerDriver:
  delay: 5.0
```

Arguments:

- delay (float): optional delay in seconds between off and on

5.2.15 YKUSHPowerDriver

A YKUSHPowerDriver controls a *YKUSHPowerPort*, allowing control of the target power state without user interaction.

Binds to:

- *YKUSHPowerPort*

Implements:

- *PowerProtocol*

```
YKUSHPowerDriver:
  delay: 5.0
```

Arguments:

- delay (float): optional delay in seconds between off and on

5.2.16 DigitalOutputPowerDriver

A DigitalOutputPowerDriver can be used to control the power of a Device using a DigitalOutputDriver.

Using this driver you probably want an external relay to switch the power of your DUT.

Binds to:

- *DigitalOutputProtocol*

```
DigitalOutputPowerDriver:  
  delay: Delay for a power cycle
```

Arguments:

- delay (float): configurable delay in seconds between off and on

5.2.17 USBPowerDriver

A USBPowerDriver controls a *USBPowerPort*, allowing control of the target power state without user interaction.

Binds to:

- *USBPowerPort*

Implements:

- *PowerProtocol*

```
USBPowerPort:  
  delay: 5.0
```

Arguments:

- delay (float): optional delay in seconds between off and on

5.2.18 SerialPortDigitalOutputDriver

The SerialPortDigitalOutputDriver makes it possible to use a UART as a 1-Bit general-purpose digital output.

This driver sits on top of a SerialDriver and uses the it's pyserial- port to control the flow control lines.

Implements:

- *DigitalOutputProtocol*

```
SerialPortDigitalOutputDriver:  
  signal: "DTR"  
  bindings: { serial : "nameOfSerial" }
```

Arguments:

- signal (str): control signal to use: DTR or RTS
- bindings (dict): A named ressource of the type SerialDriver to bind against. This is only needed if you have multiple SerialDriver in your environment (what is likely to be the case if you are using this driver).

5.2.19 ModbusCoilDriver

A ModbusCoilDriver controls a *ModbusTCP* resource. It can set and get the current state of the resource.

Binds to:

- *ModbusTCP*

Implements:

- *DigitalOutputProtocol*

```
ModbusCoilDriver: {}
```

Arguments:

- None

5.2.20 MXSUSBDriver

A MXSUSBDriver is used to upload an image into a device in the mxs USB loader state. This is useful to bootstrap a bootloader onto a device.

Binds to:

- *MXSUSBLoader*
- *NetworkMXSUSBLoader*

Implements:

- *BootstrapProtocol*

```
MXSUSBDriver:
  image: mybootloader.img
```

Arguments:

- image (str): The image to bootstrap onto the target

5.2.21 IMXUSBDriver

A IMXUSBDriver is used to upload an image into a device in the imx USB loader state. This is useful to bootstrap a bootloader onto a device.

Binds to:

- *IMXUSBLoader*
- *NetworkIMXUSBLoader*

Implements:

- *BootstrapProtocol*

```
IMXUSBDriver:
  image: mybootloader.img
```

Arguments:

- image (str): The image to bootstrap onto the target

5.2.22 USBStorageDriver

A USBStorageDriver allows access to a USB stick or similar device via the *USBMassStorage* resource.

Binds to:

- *USBMassStorage*

Implements:

- None (yet)

```
USBStorageDriver: {}
```

Arguments:

- None

5.2.23 NetworkUSBStorageDriver

A NetworkUSBStorageDriver allows access to a USB stick or similar local or remote device.

Binds to:

- *USBMassStorage*
- *NetworkUSBMassStorage*

Implements:

- None (yet)

```
NetworkUSBStorageDriver: {}
```

Arguments:

- None

5.2.24 OneWirePIODriver

A OneWirePIODriver controls a *OneWirePIO* resource. It can set and get the current state of the resource.

Binds to:

- *OneWirePIO*

Implements:

- *DigitalOutputProtocol*

```
OneWirePIODriver: {}
```

Arguments:

- None

5.2.25 QEMUDriver

The QEMUDriver allows the usage of a qemu instance as a target. It requires several arguments, listed below. The kernel, flash, rootfs and dtb arguments refer to images and paths declared in the environment configuration.

Binds to:

- None

```
QEMUDriver:
  qemu_bin: qemu_arm
  machine: vexpress-a9
  cpu: cortex-a9
  memory: 512M
  boot_args: "root=/dev/root console=ttyAMA0,115200"
  extra_args: ""
  kernel: kernel
  rootfs: rootfs
  dtb: dtb
```

```
tools:
  qemu_arm: /bin/qemu-system-arm
paths:
  rootfs: ../images/root
images:
  dtb: ../images/mydtb.dtb
  kernel: ../images/vmlinuz
```

Implements:

- *ConsoleProtocol*
- *PowerProtocol*

Arguments:

- `qemu_bin` (str): reference to the tools key for the QEMU binary
- `machine` (str): QEMU machine type
- `cpu` (str): QEMU cpu type
- `memory` (str): QEMU memory size (ends with M or G)
- `extra_args` (str): extra QEMU arguments, they are passed directly to the QEMU binary
- `boot_args` (str): optional, additional kernel boot argument
- `kernel` (str): optional, reference to the images key for the kernel
- `disk` (str): optional, reference to the images key for the disk image
- `flash` (str): optional, reference to the images key for the flash image
- `rootfs` (str): optional, reference to the paths key for use as the virtio-9p filesystem
- `dtb` (str): optional, reference to the image key for the device tree

The qemudriver also requires the specification of:

- a tool key, this contains the path to the qemu binary
- an image key, the path to the kernel image and optionally the dtb key to specify the build device tree
- a path key, this is the path to the rootfs

5.2.26 SigrokDriver

The SigrokDriver uses a SigrokDriver Resource to record samples and provides them during test runs.

Implements:

- None yet

The driver can be used in test cases by calling the *capture*, *stop* and *analyze* functions.

5.2.27 USBSDMuxDriver

The USBSDMuxDriver uses a USBSDMuxDevice resource to control a USB-SD-Mux device via `usbndmux` tool.

Implements:

- None yet

The driver can be used in test cases by calling the *set_mode()* function with argument being *dut*, *host*, *off*, or *client*.

5.3 Strategies

Strategies are used to ensure that the device is in a certain state during a test. Such a state could be the boot loader or a booted Linux kernel with shell.

5.3.1 BareboxStrategy

A BareboxStrategy has three states:

- unknown
- barebox
- shell

to transition to the shell state:

```
t = get_target("main")
s = BareboxStrategy(t)
s.transition("shell")
```

this command would transition from the boot loader into a Linux shell and activate the shelldriver.

5.3.2 ShellStrategy

A ShellStrategy has three states:

- unknown
- off
- shell

to transition to the shell state:

```
t = get_target("main")
s = ShellStrategy(t)
s.transition("shell")
```

this command would transition directly into a Linux shell and activate the shelldriver.

5.3.3 UBootStrategy

A UBootStrategy has three states:

- unknown
- uboot
- shell

to transition to the shell state:

```
t = get_target("main")
s = UBootStrategy(t)
s.transition("shell")
```

this command would transition from the boot loader into a Linux shell and activate the shelldriver.

5.4 Reporters

5.4.1 StepReporter

The StepReporter outputs individual labgrid steps to *STDOUT*.

```
from labgrid.stepreporter import StepReporter
StepReporter.start()
```

The Reporter can be stopped with a call to the stop function:

```
from labgrid.stepreporter import StepReporter
StepReporter.stop()
```

Stopping the StepReporter if it has not been started will raise an AssertionError, as will starting an already started StepReporter.

5.4.2 ConsoleLoggingReporter

The ConsoleLoggingReporter outputs read calls from the console transports into files. It takes the path as a parameter.

```
from labgrid.consoleloggingreporter import ConsoleLoggingReporter
ConsoleLoggingReporter.start(". ")
```

The Reporter can be stopped with a call to the stop function:

```
from labgrid.consoleloggingreporter import ConsoleLoggingReporter

ConsoleLoggingReporter.stop()
```

Stopping the `ConsoleLoggingReporter` if it has not been started will raise an `AssertionError`, as will starting an already started `StepReporter`.

5.5 Environment Configuration

The environment configuration for a test environment consists of a YAML file which contains targets, drivers and resources. The invocation order of objects is important here since drivers may depend on other drivers or resources.

The skeleton for an environment consists of:

```
targets:
  <target-1>:
    resources:
      <resource-1>:
        <resource-1 parameters>
      <resource-2>:
        <resource-2 parameters>
    drivers:
      <driver-1>:
        <driver-1 parameters>
      <driver-2>: {} # no parameters for driver-2
  <target-2>:
    resources:
      <resources>
    drivers:
      <drivers>
  <more targets>
options:
  <option-1 name>: <value for option-1>
  <more options>
images:
  <image-1 name>: <absolute or relative path for image-1>
  <more images>
tools:
  <tool-1 name>: <absolute or relative path for tool-1>
  <more tools>
imports:
  - <import.py>
```

If you have a single target in your environment, name it “main”, as the `get_target` function defaults to “main”.

All the resources and drivers in this chapter have a YAML example snippet which can simply be added (at the correct indentation level, one level deeper) to the environment configuration.

5.6 Exporter Configuration

The exporter is configured by using a YAML file (with a syntax similar to the environment configs used for `pytest`) or by instantiating the `Environment` object. To configure the exporter, you need to define one or more *resource groups*, each containing one or more *resources*. This allows the exporter to group resources for various usage scenarios, e.g.

all resources of a specific place or for a specific test setup. For information on how the exporter fits into the rest of labgrid, see *Remote Resources and Places*.

The basic structure of an exporter configuration file is:

```
<group-1>:
  <resources>
<group-2>:
  <resources>
```

The simplest case is with one group called “group1” containing a single *USBSerialPort*:

```
group1:
  USBSerialPort:
    match:
      '@sys_name': '3-1.3'
```

To reduce the amount of repeated declarations when many similar resources need to be exported, the *Jinja2 template engine* is used as a preprocessor for the configuration file:

```
## Iterate from group 1001 to 1016
# for idx in range(1, 17)
{{ 1000 + idx }}:
  NetworkSerialPort:
    {host: r11, port: {{ 4000 + idx }}}
  NetworkPowerPort:
    # if 1 <= idx <= 8
    {model: apc, host: apc1, index: {{ idx }}}
    # elif 9 <= idx <= 12
    {model: netio, host: netio4, index: {{ idx - 8 }}}
    # elif 13 <= idx <= 16
    {model: netio, host: netio5, index: {{ idx - 12 }}}
    # endif
# endfor
```

Use # for line statements (like the for loops in the example) and ## for line comments. Statements like {{ 4000 + idx }} are expanded based on variables in the Jinja2 template.

The first step is to install labgrid into a local virtualenv.

6.1 Installation

Clone the git repository:

```
git clone https://github.com/labgrid-project/labgrid && cd labgrid
```

Create and activate a virtualenv for labgrid:

```
virtualenv -p python3 venv  
source venv/bin/activate
```

Install required dependencies:

```
sudo apt install libow-dev
```

Install the development requirements:

```
pip install -r dev-requirements.txt
```

Install labgrid into the virtualenv in editable mode:

```
pip install -e .
```

Tests can now be run via:

```
python -m pytest --lg-env <config>
```

6.2 Writing a Driver

To develop a new driver for labgrid, you need to decide which protocol to implement, or implement your own protocol. If you are unsure about a new protocol's API, just use the driver directly from the client code, as deciding on a good API will be much easier when another similar driver is added.

Labgrid uses the `attrs` library for internal classes. First of all import `attr`, the protocol and the common driver class into your new driver file.

```
import attr

from labgrid.driver.common import Driver
from labgrid.protocol import ConsoleProtocol
```

Next, define your new class and list the protocols as subclasses of the new driver class. Try to avoid subclassing existing other drivers, as this limits the flexibility provided by connecting drivers and resources on a given target at runtime.

```
import attr

from labgrid.driver.common import Driver
from labgrid.protocol import ConsoleProtocol

@attr.s(cmp=False)
class ExampleDriver(Driver, ConsoleProtocol):
    pass
```

The `ConsoleExpectMixin` is a mixin class to add expect functionality to any class supporting the `ConsoleProtocol` and has to be the first item in the subclass list. Using the mixin class allows sharing common code, which would otherwise need to be added into multiple drivers.

```
import attr

from labgrid.driver.common import Driver
from labgrid.driver.consoleexpectmixin import ConsoleExpectMixin
from labgrid.protocol import ConsoleProtocol

@attr.s(cmp=False)
class ExampleDriver(ConsoleExpectMixin, Driver, ConsoleProtocol):
    pass
```

Additionally the driver needs to be registered with the `target_factory` and provide a bindings dictionary, so that the `Target` can resolve dependencies on other drivers or resources.

```
import attr

from labgrid.factory import target_factory
from labgrid.driver.common import Driver
from labgrid.driver.consoleexpectmixin import ConsoleExpectMixin
from labgrid.protocol import ConsoleProtocol

@target_factory.reg_driver
@attr.s(cmp=False)
class ExampleDriver(ConsoleExpectMixin, Driver, ConsoleProtocol):
    bindings = { "port": SerialPort }
    pass
```

The listed resource `SerialPort` will be bound to `self.port`, making it usable in the class. Checks are performed that the target which the driver binds to has a `SerialPort`, otherwise an error will be raised.

If your driver can support alternative resources, you can use a set of classes instead of a single class:

```
bindings = { "port": {SerialPort, NetworkSerialPort}}
```

Optional bindings can be declared by including `None` in the set:

```
bindings = { "port": {SerialPort, NetworkSerialPort, None}}
```

If you need to do something during instantiation, you need to add a `__attrs_post_init__` method (instead of the usual `__init__` used for non-attr-classes). The minimum requirement is a call to `super().__attrs_post_init__()`.

```
import attr

from labgrid.factory import target_factory
from labgrid.driver.common import Driver
from labgrid.driver.consoleexpectmixin import ConsoleExpectMixin
from labgrid.protocol import ConsoleProtocol

@target_factory.reg_driver
@attr.s(cmp=False)
class ExampleDriver(ConsoleExpectMixin, Driver, ConsoleProtocol):
    bindings = { "port": SerialPort }

    def __attrs_post_init__(self):
        super().__attrs_post_init__()
```

All that's left now is to implement the functionality described by the used protocol, by using the API of the bound drivers and resources.

6.3 Writing a Resource

To add a new resource to labgrid, we import `attr` into our new resource file. Additionally we need the `target_factory` and the common `Resource` class.

```
import attr

from labgrid.factory import target_factory
from labgrid.driver.common import Resource
```

Next we add our own resource with the `Resource` parent class and register it with the `target_factory`.

```
import attr

from labgrid.factory import target_factory
from labgrid.driver.common import Resource

@target_factory.reg_resource
@attr.s(cmp=False)
class ExampleResource(Resource):
    pass
```

All that is left now is to add attributes via `attr.ib()` member variables.

```
import attr

from labgrid.factory import target_factory
from labgrid.driver.common import Resource

@target_factory.reg_resource
@attr.s(cmp=False)
class ExampleResource(Resource):
    examplevar1 = attr.ib()
    examplevar2 = attr.ib()
```

The `attr.ib()` style of member definition also supports defaults and validators, see the [attrs documentation](#).

6.4 Writing a Strategy

Labgrid only offers two basic strategies, for complex use cases a customized strategy is required. Start by creating a strategy skeleton:

```
import enum

import attr

from labgrid.step import step
from labgrid.driver.common import Strategy

class Status(enum.Enum):
    unknown = 0

class MyStrategy(Strategy):
    bindings = {
    }

    status = attr.ib(default=Status.unknown)

    @step
    def transition(self, status, *, step):
        if not isinstance(status, Status):
            status = Status[status]
        if status == Status.unknown:
            raise StrategyError("can not transition to {}".format(status))
        elif status == self.status:
            step.skip("nothing to do")
            return # nothing to do
        else:
            raise StrategyError(
                "no transition found from {} to {}".format(self.status, status)
            )
        self.status = status
```

The `bindings` variable needs to declare the drivers necessary for the strategy, usually one for power, boot loader and shell. The `Status` class needs to be extended to cover the states of your strategy, then for each state an `elif` entry in the transition function needs to be added.

Lets take a look at the builtin `BareboxStrategy`. The `Status` enum for Barebox:

```
class Status(enum.Enum):
    unknown = 0
    barebox = 1
    shell = 2
```

defines 2 custom states and the *unknown* state as the start point. These two states are handled in the transition function:

```
elif status == Status.barebox:
    # cycle power
    self.target.activate(self.power)
    self.power.cycle()
    # interrupt barebox
    self.target.activate(self.barebox)
elif status == Status.shell:
    # transition to barebox
    self.transition(Status.barebox)
    self.barebox.boot("")
    self.barebox.await_boot()
    self.target.activate(self.shell)
```

Here the *barebox* state simply cycles the board and activates the driver, while the *shell* state uses the barebox state to cycle the board and then boot the linux kernel.

6.5 Contributing

Thank you for thinking about contributing to labgrid! Some different backgrounds and use-cases are essential for making labgrid work well for all users.

The following should help you with submitting your changes, but don't let these guidelines keep you from opening a pull request. If in doubt, we'd prefer to see the code earlier as a work-in-progress PR and help you with the submission process.

6.5.1 Workflow

- Changes should be submitted via a [GitHub pull request](#).
- Try to limit each commit to a single conceptual change.
- Add a signed-off-by line to your commits according to the *Developer's Certificate of Origin* (see below).
- Check that the tests still work before submitting the pull request. Also check the CI's feedback on the pull request after submission.
- When adding new drivers or resources, please also add the corresponding documentation and test code.
- If your change affects backward compatibility, describe the necessary changes in the commit message and update the examples where needed.

6.5.2 Code

- Follow the [PEP 8](#) style.
- Use `attr.ib` attributes for public attributes of your drivers and resources.
- Use `isort` to sort the import statements.

6.5.3 Documentation

- Use semantic linefeeds in .rst files.

6.5.4 Developer's Certificate of Origin

Labgrid uses the [Developer's Certificate of Origin 1.1](#) with the same process as used for the Linux kernel:

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

1. The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
2. The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
3. The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
4. I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

Then you just add a line (using `git commit -s`) saying:

Signed-off-by: Random J Developer <random@developer.example.org>

using your real name (sorry, no pseudonyms or anonymous contributions).

6.6 Ideas

6.6.1 Driver Preemption

To allow better handling of unexpected reboots or crashes, inactive Drivers could register callbacks on their providers (for example the `BareboxDriver` it's `ConsoleProtocol`). These callbacks would look for indications that the Target has changed state unexpectedly (by looking for the bootloader startup messages, in this case). The inactive Driver could then cause a preemption and would be activated. The current caller of the originally active driver would be notified via an exception.

6.6.2 File Transfer to Exporters

Currently, the exporter and client expect to have a shared filesystem (see for example how the `AndroidFastbootDriver` works when accessing a `NetworkAndroidFastboot` resource). To remove this limitation, we should have a common way to make files available to the exporter, possibly by generating a hash locally and rsyncing new files to the exporter.

6.6.3 Remote Target Reservation

For integration with CI systems (like Jenkins), it would help if the CI job could reserve and wait for a specific target. This could be done by managing a list of waiting users in the coordinator and notifying the current user on each invocation of labgrid-client that another user is waiting. The reservation should expire after some time if it is not used to lock the target after it becomes available.

6.6.4 Step Tracing

The Step infrastructure already collects timing and nesting information on executed commands, but is currently only used for in pytest or via the standalone StepReporter. By writing these events to a file (or sqlite database) as a trace, we can collect data over multiple runs for later analysis. This would become more useful by passing recognized events (stack traces, crashes, ...) and benchmark results via the Step infrastructure.

6.6.5 Target Feature Flags

It would be useful to support configuring feature flags in the target YAML definition. Then individual tests could be skipped if a required feature is unavailable on the current target without manually modifying the test suite.

This document outlines the design decisions influencing the development of labgrid.

7.1 Out of Scope

Out of scope for labgrid are:

7.1.1 Integrated Build System

In contrast to some other tools, labgrid explicitly has no support for building target binaries or images.

Our reasons for this are:

- Several full-featured build systems already exist and work well.
- We want to test unmodified images produced by any build system (OE/Yocto, PTXdist, Buildroot, Debian, ...).

7.1.2 Test Infrastructure

Labgrid does not include a test framework.

The main reason is that with `pytest` we already have a test framework which:

- makes it easy to write tests
- reduces boilerplate code with flexible fixtures
- is easy to extend and has many available plugins
- allows using any Python library for creating inputs or processing outputs
- supports test report generation

Furthermore, the hardware control functionality needed for testing is also very useful during development, provisioning and other areas, so we don't want to hide that behind another test framework.

7.2 In Scope

- usable as a library for hardware provisioning
- device control via:
 - serial console
 - SSH
 - file management
 - power and reset
- emulation of external services:
 - USB stick emulation
 - external update services (Hawkbitt)
- bootstrap services:
 - fastboot
 - imxusbloader

7.3 Further Goals

- tests should be equivalent for workstations and servers
- discoverability of available boards
- distributed board access

8.1 Release 0.2.0 (unreleased)

8.1.1 New Features

- The target now saves its attached drivers, resources and protocols in a lookup table, avoiding the need of importing many Drivers and Protocols (see *Syntactic sugar for Targets*)
- The new subcommand `labgrid-client monitor` shows resource or places changes as they happen, which is useful during development or debugging.
- The new *QEMUDriver* runs a system image in QEmu and implements the *ConsoleProtocol* and *PowerProtocol*. This allows using labgrid without any real hardware.
- The bootloader drivers now have a `reset` method.
- The *BareboxDriver*'s boot string is now configurable, which allows it to work with the `quiet` Linux boot parameter.
- The environment yaml file can now list Python files (under the 'imports' key). They are imported before constructing the Targets, which simplifies using custom Resources, Drivers or Strategies.
- The pytest plugin now stores metadata about the environment yaml file in the junit XML output.
- The `labgrid-client` tool now understands a `--state` option to transition to the provided state using a *Strategy*. This requires an environment yaml file with a *RemotePlace* Resources and matching Drivers.
- Resource matches for places configured in the coordinator can now have a name, allowing multiple resources with the same class.
- The new *Target.__getitem__* method makes writing using protocols less verbose.
- The *NetworkPowerDriver* now support the newer NETIO 4 models.
- The *ShellDriver* now supports configuring the login prompt timeout.
- The *SerialDriver* now supports using plain TCP instead of RFC 2217, which is needed from some console servers.

- Experimental: The labgrid-autoinstall tool was added (see below).

8.1.2 Incompatible Changes

- When using the coordinator, it must be upgrade together with the clients because of the newly introduce match names.
- Resources and Drivers now need to be created with an explicit name parameter. It can be `None` to keep the old behaviour. See below for details.
- Classes derived from `Resource` or `Driver` now need to use `@attr.s(cmp=False)` instead of `@attr.s` because of a change in the `attrs` module version 17.1.0.

8.1.3 Syntactic sugar for Targets

Targets are now able to retrieve requested drivers, resources or protocols by name instead of by class. This allows removing many imports, e.g.

```
from labgrid.driver import ShellDriver

shell = target.get_driver(ShellDriver)
```

becomes

```
shell = target.get_driver("ShellDriver")
```

Also take a look at the examples, they have been ported to the new syntax as well.

8.1.4 Multiple Driver Instances

For some Protocols, it is useful to allow multiple instances.

DigitalOutputProtocol: A board may have two jumpers to control the boot mode in addition to a reset GPIO. Previously, it was not possible to use these on a single target.

ConsoleProtocol: Some boards have multiple console interfaces or expose a login prompt via a USB serial gadget.

PowerProtocol: In some cases, multiple power ports need to be controlled for one Target.

To support these use cases, Resources and Drivers must be created with a name parameter. When updating your code to this version, you can either simply set the name to `None` to keep the previous behaviour. Alternatively, pass a string as the name.

Old:

```
>>> t = Target("MyTarget")
>>> SerialPort(t)
SerialPort(target=Target(name='MyTarget', env=None), state=<BindingState.bound: 1>,
↳avail=True, port=None, speed=115200)
>>> SerialDriver(t)
SerialDriver(target=Target(name='MyTarget', env=None), state=<BindingState.bound: 1>,
↳txdelay=0.0)
```

New (with name=None):

```
>>> t = Target("MyTarget")
>>> SerialPort(t, None)
SerialPort(target=Target(name='MyTarget', env=None), name=None, state=<BindingState.
↳bound: 1>, avail=True, port=None, speed=115200)
>>> SerialDriver(t, None)
SerialDriver(target=Target(name='MyTarget', env=None), name=None, state=<BindingState.
↳bound: 1>, txdelay=0.0)
```

New (with real names):

```
>>> t = Target("MyTarget")
>>> SerialPort(t, "MyPort")
SerialPort(target=Target(name='MyTarget', env=None), name='MyPort', state=
↳<BindingState.bound: 1>, avail=True, port=None, speed=115200)
>>> SerialDriver(t, "MyDriver")
SerialDriver(target=Target(name='MyTarget', env=None), name='MyDriver', state=
↳<BindingState.bound: 1>, txdelay=0.0)
```

8.1.5 Auto-Installer Tool

To simplify using labgrid for provisioning several boards in parallel, the `labgrid-autoinstall` tool was added. It reads a YAML file defining several targets and a Python script to be run for each board. Internally, it spawns a child process for each target, which waits until a matching resource becomes available and then executes the script.

For example, this makes it simple to load a bootloader via the [BootstrapProtocol](#), use the [AndroidFastbootDriver](#) to upload a kernel with `initramfs` and then write the target's eMMC over a USB Mass Storage gadget.

Note: `labgrid-autoinstall` is still experimental and no documentation has been written.

8.2 Release 0.1.0 (released May 11, 2017)

This is the initial release of labgrid.

9.1 labgrid package

9.1.1 Subpackages

labgrid.autoinstall package

Submodules

labgrid.autoinstall.main module

The autoinstall.main module runs an installation script automatically on multiple targets.

```
class labgrid.autoinstall.main.Handler (env, args, name)  
    Bases: multiprocessing.context.Process
```

```
    __init__ (env, args, name)
```

```
    __module__ = 'labgrid.autoinstall.main'
```

```
    run ()
```

```
    run_once ()
```

```
class labgrid.autoinstall.main.Manager (env, args)
```

```
    Bases: object
```

```
    __dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'Manager' objects>, 'co
```

```
    __init__ (env, args)
```

```
    __module__ = 'labgrid.autoinstall.main'
```

```
    __weakref__
```

```
        list of weak references to the object (if defined)
```

```
    configure ()
```

```
join()
```

```
start()
```

```
labgrid.autoinstall.main.main()
```

labgrid.driver package

Subpackages

labgrid.driver.power package

Submodules

labgrid.driver.power.apc module

```
labgrid.driver.power.apc.set(host, index, value)
```

```
labgrid.driver.power.apc.get(host, index)
```

labgrid.driver.power.digipower module

```
labgrid.driver.power.digipower.set(host, index, value)
```

```
labgrid.driver.power.digipower.get(host, index)
```

labgrid.driver.power.gude module

```
labgrid.driver.power.gude.set(host, index, value)
```

```
labgrid.driver.power.gude.get(host, index)
```

labgrid.driver.power.gude24 module

```
labgrid.driver.power.gude24.set(host, index, value)
```

```
labgrid.driver.power.gude24.get(host, index)
```

labgrid.driver.power.netio module

```
labgrid.driver.power.netio.set(host, index, value)
```

```
labgrid.driver.power.netio.get(host, index)
```

labgrid.driver.power.netio_kshell module

tested with NETIO 4C, should be compatible with all NETIO 4-models

```
labgrid.driver.power.netio_kshell.set(host, index, value)
```

```
labgrid.driver.power.netio_kshell.get(host, index)
```

Submodules

labgrid.driver.bareboxdriver module

```
class labgrid.driver.bareboxdriver.BareboxDriver(target, name, prompt="", auto-
boot='stop autoboot', interrupt='n',
bootstring='Linux version \d')
```

Bases: `labgrid.driver.commandmixin.CommandMixin`, `labgrid.driver.common.Driver`, `labgrid.protocol.commandprotocol.CommandProtocol`, `labgrid.protocol.linuxbootprotocol.LinuxBootProtocol`

BareboxDriver - Driver to control barebox via the console. BareboxDriver binds on top of a ConsoleProtocol.

Parameters `prompt` (*str*) – The default Barebox Prompt

```
bindings = {'console': <class 'labgrid.protocol.consoleprotocol.ConsoleProtocol'>}
```

```
__attrs_post_init__ ()
```

```
on_activate ()
```

Activate the BareboxDriver

This function checks for a prompt and awaits it if not already active

```
on_deactivate ()
```

Deactivate the BareboxDriver

Simply sets the internal status to 0

```
run (cmd: str, *, step, timeout: int = 30)
```

Runs the specified command on the shell and returns the output.

Parameters

- `cmd` (*str*) – command to run on the shell
- `timeout` (*int*) – optional, timeout in seconds

Returns if successful, None otherwise

Return type Tuple[List[str],List[str], int]

```
reset ()
```

Reset the board via a CPU reset

```
get_status ()
```

Retrieve status of the BareboxDriver 0 means inactive, 1 means active.

Returns status of the driver

Return type int

```
await_boot ()
```

Wait for the initial Linux version string to verify we successfully jumped into the kernel.

```
boot (name: str)
```

Boot the default or a specific boot entry

Parameters `name` (*str*) – name of the entry to boot

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True,
```

```
__init__ (target, name, prompt=", autoboot='stop autoboot', interrupt='\n', bootstring='Linux ver-  
sion \d')
```

```
__module__ = 'labgrid.driver.bareboxdriver'
```

```
__repr__ ()  
Automatically created by attrs.
```

labgrid.driver.commandmixin module

```
class labgrid.driver.commandmixin.CommandMixin
```

Bases: object

CommandMixin implementing common functions for drivers which support the CommandProtocol

```
__attrs_post_init__ ()
```

```
wait_for (cmd, pattern, timeout=30.0, sleepduration=1)
```

```
run_check (cmd: str, timeout=30)
```

External run_check function, only available if the driver is active. Runs the supplied command and returns the stdout, raises an ExecutionError otherwise.

Parameters `cmd` (*str*) – command to run on the shell

Returns stdout of the executed command

Return type List[str]

```
__dict__ = mappingproxy({'wait_for': <function CommandMixin.wait_for>, '__dict__': <
```

```
__module__ = 'labgrid.driver.commandmixin'
```

```
__weakref__
```

list of weak references to the object (if defined)

labgrid.driver.common module

```
class labgrid.driver.common.Driver (target, name)
```

Bases: `labgrid.binding.BindingMixin`

Represents a driver which is used externally or by other drivers. It implements functionality based on directly accessing the Resource or by building on top of other Drivers.

Life cycle: - create - bind (n times) - activate - usage - deactivate

```
__attrs_post_init__ ()
```

```
get_priority (protocol)
```

Retrieve the priority for a given protocol

Arguments: protocol - protocol to search for in the MRO

Returns value of the priority if it is found, 0 otherwise.

Return type Int

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name)
```

```
__module__ = 'labgrid.driver.common'
```

```
__repr__ ()  
    Automatically created by attrs.
```

```
labgrid.driver.common.check_file (filename, *, command_prefix=[])
```

labgrid.driver.consoleexpectmixin module

```
class labgrid.driver.consoleexpectmixin.ConsoleExpectMixin
```

```
    Bases: object
```

Console driver mixin to implement the read, write, expect and sendline methods. It uses the internal `_read` and `_write` methods.

The class using the `ConsoleExpectMixin` must provide a logger and a `txdelay` attribute.

```
__attrs_post_init__ ()
```

```
read (size=1, timeout=0.0)
```

```
write (data)
```

```
sendline (line)
```

```
sendcontrol (char)
```

```
expect (pattern, timeout=-1)
```

```
resolve_conflicts (client)
```

```
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'ConsoleExpectMix
```

```
__module__ = 'labgrid.driver.consoleexpectmixin'
```

```
__weakref__  
    list of weak references to the object (if defined)
```

labgrid.driver.exception module

```
exception labgrid.driver.exception.ExecutionError (msg, stdout=None, stderr=None)
```

```
    Bases: Exception
```

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
```

```
__init__ (msg, stdout=None, stderr=None)
```

```
__module__ = 'labgrid.driver.exception'
```

```
__repr__ ()  
    Automatically created by attrs.
```

```
__weakref__  
    list of weak references to the object (if defined)
```

```
exception labgrid.driver.exception.CleanUpError (msg)
```

```
    Bases: Exception
```

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
```

```
__init__ (msg)
```

```
__module__ = 'labgrid.driver.exception'
```

`__repr__()`
Automatically created by attrs.

`__weakref__`
list of weak references to the object (if defined)

labgrid.driver.externalconsoledriver module

```
class labgrid.driver.externalconsoledriver.ExternalConsoleDriver(target, name,
                                                                cmd, txde-
                                                                lay=0.0)
```

Bases: *labgrid.driver.consoleexpectmixin.ConsoleExpectMixin, labgrid.driver.common.Driver, labgrid.protocol.consoleprotocol.ConsoleProtocol*

Driver implementing the ConsoleProtocol interface using a subprocess

```
__attrs_post_init__()  
open()  
Starts the subprocess, does nothing if it is already closed  
close()  
Stops the subprocess, does nothing if it is already closed  
on_deactivate()  
__abstractmethods__ = frozenset()  
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True  
__init__(target, name, cmd, txdelay=0.0)  
__module__ = 'labgrid.driver.externalconsoledriver'  
__repr__()  
Automatically created by attrs.
```

labgrid.driver.fake module

```
class labgrid.driver.fake.FakeConsoleDriver(target, name, txdelay=0.0)
```

Bases: *labgrid.driver.consoleexpectmixin.ConsoleExpectMixin, labgrid.driver.common.Driver, labgrid.protocol.consoleprotocol.ConsoleProtocol*

```
__attrs_post_init__()  
open()  
close()  
__abstractmethods__ = frozenset()  
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True  
__init__(target, name, txdelay=0.0)  
__module__ = 'labgrid.driver.fake'  
__repr__()  
Automatically created by attrs.
```

```
class labgrid.driver.fake.FakeCommandDriver (target, name)
    Bases: labgrid.driver.commandmixin.CommandMixin, labgrid.driver.common.Driver,
           labgrid.protocol.commandprotocol.CommandProtocol

    run (*args)

    run_check (*args)

    get_status ()

    __abstractmethods__ = frozenset ()

    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name)

    __module__ = 'labgrid.driver.fake'

    __repr__ ()
        Automatically created by attrs.

class labgrid.driver.fake.FakeFileTransferDriver (target, name)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.filetransferprotocol.
           FileTransferProtocol

    get (*args)

    put (*args)

    __abstractmethods__ = frozenset ()

    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name)

    __module__ = 'labgrid.driver.fake'

    __repr__ ()
        Automatically created by attrs.

class labgrid.driver.fake.FakePowerDriver (target, name)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.powerprotocol.
           PowerProtocol

    on (*args)

    off (*args)

    cycle (*args)

    __abstractmethods__ = frozenset ()

    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name)

    __module__ = 'labgrid.driver.fake'

    __repr__ ()
        Automatically created by attrs.
```

labgrid.driver.fastbootdriver module

```
class labgrid.driver.fastbootdriver.AndroidFastbootDriver (target, name, im-
                                                         age=None)
    Bases: labgrid.driver.common.Driver
```

```
bindings = {'fastboot': <class 'labgrid.resource.remote.NetworkAndroidFastboot'>, <c
__attrs_post_init__ ()
on_activate ()
on_deactivate ()
__call__ (*args)
boot (filename)
flash (partition, filename)
continue_boot ()
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name, image=None)
__module__ = 'labgrid.driver.fastbootdriver'
__repr__ ()
    Automatically created by attrs.
```

labgrid.driver.infodriver module

```
class labgrid.driver.infodriver.InfoDriver (target, name)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.infoprotocol.
           InfoProtocol
    InfoDriver implementing the InfoProtocol on top of CommandProtocol drivers
bindings = {'command': <class 'labgrid.protocol.commandprotocol.CommandProtocol'>}
__attrs_post_init__ ()
get_ip (interface='eth0')
    Returns the IP of the supplied interface
get_service_status (service)
    Returns True if service is active, False in all other cases
get_hostname ()
__abstractmethods__ = frozenset ()
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name)
__module__ = 'labgrid.driver.infodriver'
__repr__ ()
    Automatically created by attrs.
```

labgrid.driver.modbusdriver module

```
class labgrid.driver.modbusdriver.ModbusCoilDriver (target, name)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.digitaloutputprotocol.
           DigitalOutputProtocol
bindings = {'coil': <class 'labgrid.resource.modbus.ModbusTCPCoil'>}
```



```

__attrs_post_init__()
set (status)
get ()
__abstractmethods__ = frozenset ()
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name)
__module__ = 'labgrid.driver.modbusdriver'
__repr__ ()
    Automatically created by attrs.

```

labgrid.driver.onewiredriver module

```

class labgrid.driver.onewiredriver.OneWirePIODriver (target, name)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.digitaloutputprotocol.
           DigitalOutputProtocol
    bindings = {'port': <class 'labgrid.resource.onewirereport.OneWirePIO'>}
__attrs_post_init__()
set (status)
get ()
__abstractmethods__ = frozenset ()
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name)
__module__ = 'labgrid.driver.onewiredriver'
__repr__ ()
    Automatically created by attrs.

```

labgrid.driver.openocddriver module

```

class labgrid.driver.openocddriver.OpenOCDDriver (target, name, config, search=None,
                                                    image=None)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.bootstrapprotocol.
           BootstrapProtocol
    bindings = {'interface': {<class 'labgrid.resource.remote.NetworkAlteraUSBBlaster'>},
__attrs_post_init__()
load (filename=None)
__abstractmethods__ = frozenset ()
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name, config, search=None, image=None)
__module__ = 'labgrid.driver.openocddriver'

```

```
__repr__ ()  
    Automatically created by attrs.
```

labgrid.driver.powerdriver module

```
class labgrid.driver.powerdriver.PowerResetMixin  
    Bases: labgrid.protocol.resetprotocol.ResetProtocol  
  
    ResetMixin implements the ResetProtocol for drivers which support the PowerProtocol  
  
    priorities = {<class 'labgrid.protocol.resetprotocol.ResetProtocol'>: -10}  
  
    __attrs_post_init__ ()  
  
    reset ()  
  
    __abstractmethods__ = frozenset ()  
  
    __attrs_attrs__ = ()  
  
    __init__ ()  
  
    __module__ = 'labgrid.driver.powerdriver'  
  
    __repr__ ()  
        Automatically created by attrs.
```

```
class labgrid.driver.powerdriver.ManualPowerDriver (target, name)  
    Bases: labgrid.driver.common.Driver, labgrid.driver.powerdriver.  
           PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol  
  
    ManualPowerDriver - Driver to tell the user to control a target's power  
  
    on ()  
  
    off ()  
  
    cycle ()  
  
    __abstractmethods__ = frozenset ()  
  
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True  
    __init__ (target, name)  
  
    __module__ = 'labgrid.driver.powerdriver'  
  
    __repr__ ()  
        Automatically created by attrs.
```

```
class labgrid.driver.powerdriver.ExternalPowerDriver (target, name, cmd_on, cmd_off,  
                                                       cmd_cycle=None, delay=2.0)  
    Bases: labgrid.driver.common.Driver, labgrid.driver.powerdriver.  
           PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol  
  
    ExternalPowerDriver - Driver using an external command to control a target's power  
  
    on ()  
  
    off ()  
  
    cycle ()  
  
    __abstractmethods__ = frozenset ()  
  
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, cmd_on, cmd_off, cmd_cycle=None, delay=2.0)
```

```
__module__ = 'labgrid.driver.powerdriver'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.driver.powerdriver.NetworkPowerDriver(target, name, delay=2.0)
```

```
Bases: labgrid.driver.common.Driver, labgrid.driver.powerdriver.
        PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol
```

NetworkPowerDriver - Driver using a networked power switch to control a target's power

```
bindings = {'port': <class 'labgrid.resource.power.NetworkPowerPort'>}
```

```
__attrs_post_init__()
```

```
on()
```

```
off()
```

```
cycle()
```

```
get()
```

```
__abstractmethods__ = frozenset()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, delay=2.0)
```

```
__module__ = 'labgrid.driver.powerdriver'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.driver.powerdriver.DigitalOutputPowerDriver(target, name, de-
        lay=1.0)
```

```
Bases: labgrid.driver.common.Driver, labgrid.driver.powerdriver.
        PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol
```

DigitalOutputPowerDriver uses a DigitalOutput to control the power of a DUT.

```
bindings = {'output': <class 'labgrid.protocol.digitaloutputprotocol.DigitalOutputPro
```

```
__attrs_post_init__()
```

```
on()
```

```
off()
```

```
cycle()
```

```
get()
```

```
__abstractmethods__ = frozenset()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, delay=1.0)
```

```
__module__ = 'labgrid.driver.powerdriver'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.driver.powerdriver.YKUSHPowerDriver (target, name, delay=2.0)
  Bases:      labgrid.driver.common.Driver,      labgrid.driver.powerdriver.
             PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol

  YKUSHPowerDriver - Driver using a YEPKIT YKUSH switchable USB hub to control a target's power -
  https://www.yepkit.com/products/ykush

  bindings = {'port': <class 'labgrid.resource.ykushpowerport.YKUSHPowerPort'>}

  __attrs_post_init__ ()

  on ()

  off ()

  cycle ()

  get ()

  __abstractmethods__ = frozenset ()

  __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
  __init__ (target, name, delay=2.0)

  __module__ = 'labgrid.driver.powerdriver'

  __repr__ ()
    Automatically created by attrs.
```

```
class labgrid.driver.powerdriver.USBPowerDriver (target, name, delay=2.0)
  Bases:      labgrid.driver.common.Driver,      labgrid.driver.powerdriver.
             PowerResetMixin, labgrid.protocol.powerprotocol.PowerProtocol

  USBPowerDriver - Driver using a power switchable USB hub and the uhubctl tool (https://github.com/mvp/uhubctl) to control a target's power

  bindings = {'hub': {<class 'labgrid.resource.remote.NetworkUSBPowerPort'>, <class 'labgrid.resource.remote.NetworkUSBPowerPort'>}}

  __attrs_post_init__ ()

  on ()

  off ()

  cycle ()

  get ()

  __abstractmethods__ = frozenset ()

  __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
  __init__ (target, name, delay=2.0)

  __module__ = 'labgrid.driver.powerdriver'

  __repr__ ()
    Automatically created by attrs.
```

labgrid.driver.qemudriver module

The QEMUDriver implements a driver to use a QEMU target

```
class labgrid.driver.qemudriver.QEMUDriver(target, name, qemu_bin, machine, cpu,
                                           memory, extra_args, boot_args=None,
                                           kernel=None, disk=None, rootfs=None,
                                           dtb=None, flash=None)
```

Bases: *labgrid.driver.consoleexpectmixin.ConsoleExpectMixin, labgrid.driver.common.Driver, labgrid.protocol.powerprotocol.PowerProtocol, labgrid.protocol.consoleprotocol.ConsoleProtocol*

The QEMUDriver implements an interface to start targets as qemu instances.

The kernel, flash, rootfs and dtb arguments refer to images and paths declared in the environment configuration.

Parameters

- **qemu_bin** (*str*) – reference to the tools key for the QEMU binary
- **machine** (*str*) – QEMU machine type
- **cpu** (*str*) – QEMU cpu type
- **memory** (*str*) – QEMU memory size (ends with M or G)
- **extra_args** (*str*) – extra QEMU arguments, they are passed directly to the QEMU binary
- **boot_args** (*str*) – optional, additional kernel boot argument
- **kernel** (*str*) – optional, reference to the images key for the kernel
- **disk** (*str*) – optional, reference to the images key for the disk image
- **flash** (*str*) – optional, reference to the images key for the flash image
- **rootfs** (*str*) – optional, reference to the paths key for use as the virtio-9p filesystem
- **dtb** (*str*) – optional, reference to the image key for the device tree

```
__attrs_post_init__ ()
```

```
on_activate ()
```

```
on_deactivate ()
```

```
on ()
```

Start the QEMU subprocess, accept the unix socket connection and afterwards start the emulator using a QMP Command

```
off ()
```

Stop the emulator using a monitor command and await the exitcode

```
cycle ()
```

Cycle the emulator by restarting it

```
monitor_command (command)
```

Execute a monitor_command via the QMP

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True,
```

```
__init__(target, name, qemu_bin, machine, cpu, memory, extra_args, boot_args=None, kernel=None,
          disk=None, rootfs=None, dtb=None, flash=None)
```

```
__module__ = 'labgrid.driver.qemudriver'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.driver.quartushpsdriver module

```
class labgrid.driver.quartushpsdriver.QuartusHPSDriver(target, name, image=None,  
                                                    cable_number=None)
```

Bases: *labgrid.driver.common.Driver*

```
bindings = {'interface': {<class 'labgrid.resource.remote.NetworkAlteraUSBBlaster'>,  
__attrs_post_init__()  
on_deactivate()  
flash (filename=None, address=0)  
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True  
__init__ (target, name, image=None, cable_number=None)  
__module__ = 'labgrid.driver.quartushpsdriver'  
__repr__ ()  
    Automatically created by attrs.
```

labgrid.driver.serialdriver module

```
class labgrid.driver.serialdriver.SerialDriver(target, name, txdelay=0.0)  
Bases: labgrid.driver.consoleexpectmixin.ConsoleExpectMixin, labgrid.driver.  
common.Driver, labgrid.protocol.consoleprotocol.ConsoleProtocol
```

Driver implementing the ConsoleProtocol interface over a SerialPort connection

```
bindings = {'port': {<class 'labgrid.resource.base.SerialPort'>, <class 'labgrid.reso  
__attrs_post_init__()  
on_activate()  
on_deactivate()  
open ()  
    Opens the serialport, does nothing if it is already closed  
__abstractmethods__ = frozenset ()  
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True  
__init__ (target, name, txdelay=0.0)  
__module__ = 'labgrid.driver.serialdriver'  
__repr__ ()  
    Automatically created by attrs.  
close ()  
    Closes the serialport, does nothing if it is already closed
```

labgrid.driver.shelldriver module

The ShellDriver provides the CommandProtocol, ConsoleProtocol and InfoProtocol on top of a SerialPort.

```
class labgrid.driver.shelldriver.ShellDriver(target, name, prompt, login_prompt,
                                             username, password="", keyfile="",
                                             login_timeout=60, console_ready="",
                                             await_login_timeout=2)
```

Bases: `labgrid.driver.commandmixin.CommandMixin`, `labgrid.driver.common.Driver`, `labgrid.protocol.commandprotocol.CommandProtocol`, `labgrid.protocol.filetransferprotocol.FileTransferProtocol`

ShellDriver - Driver to execute commands on the shell ShellDriver binds on top of a ConsoleProtocol.

Parameters

- **prompt** (*regex*) – the shell prompt to detect
- **login_prompt** (*regex*) – the login prompt to detect
- **username** (*str*) – username to login with
- **password** (*str*) – password to login with
- **keyfile** (*str*) – keyfile to bind mount over users authorized keys
- **login_timeout** (*int*) – optional, timeout for login prompt detection

```
bindings = {'console': <class 'labgrid.protocol.consoleprotocol.ConsoleProtocol'>}
```

```
__attrs_post_init__()
```

```
on_activate()
```

```
on_deactivate()
```

```
run(cmd, timeout=30.0, codec='utf-8', decodeerrors='strict')
```

```
get_status()
```

Returns the status of the shell-driver. 0 means not connected/found, 1 means shell

```
put_ssh_key(key)
```

```
put_bytes(buf: bytes, remotefile: str)
```

Upload a file to the target. Will silently overwrite the remote file if it already exists.

Parameters

- **buf** (*bytes*) – file contents
- **remotefile** (*str*) – destination filename on the target

Raises

- `IOError` – if the provided localfile could not be found
- `ExecutionError` – if something else went wrong

```
put(localfile: str, remotefile: str)
```

Upload a file to the target. Will silently overwrite the remote file if it already exists.

Parameters

- **localfile** (*str*) – source filename on the local machine
- **remotefile** (*str*) – destination filename on the target

Raises

- `IOError` – if the provided localfile could not be found
- `ExecutionError` – if something else went wrong

get_bytes (*remotefile: str*)

Download a file from the target.

Parameters **remotefile** (*str*) – source filename on the target

Returns (bytes) file contents

Raises

- `IOError` – if localfile could be written
- `ExecutionError` – if something went wrong

get (*remotefile: str, localfile: str*)

Download a file from the target. Will silently overwrite the local file if it already exists.

Parameters

- **remotefile** (*str*) – source filename on the target
- **localfile** (*str*) – destination filename on the local machine (can be relative)

Raises

- `IOError` – if localfile could be written
- `ExecutionError` – if something went wrong

run_script (*data: bytes, timeout: int = 60*)

Upload a script to the target and run it.

Parameters

- **data** (*bytes*) – script data
- **timeout** (*int*) – timeout for the script to finish execution

Returns str, stderr: str, return_value: int)

Return type Tuple of (stdout

Raises

- `IOError` – if the provided localfile could not be found
- `ExecutionError` – if something else went wrong

run_script_file (*scriptfile: str, *args, timeout: int = 60*)

Upload a script file to the target and run it.

Parameters

- **scriptfile** (*str*) – source file on the local file system to upload to the target
- ***args** – (list of str): any arguments for the script as positional arguments
- **timeout** (*int*) – timeout for the script to finish execution

Returns str, stderr: str, return_value: int)

Return type Tuple of (stdout

Raises

- `ExecutionError` – if something went wrong
- `IOError` – if the provided localfile could not be found

__abstractmethods__ = frozenset ()


```

__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__(target, name, prompt, login_prompt, username, password="", keyfile="", login_timeout=60,
           console_ready="", await_login_timeout=2)
__module__ = 'labgrid.driver.shelldriver'
__repr__()
    Automatically created by attrs.

```

labgrid.driver.sigrokdriver module

class labgrid.driver.sigrokdriver.**SigrokDriver**(target, name)

Bases: *labgrid.driver.common.Driver*

The SigrokDriver uses sigrok-cli to record samples and expose them as python dictionaries.

Parameters **bindings** (*dict*) – driver to use with sigrok

```
bindings = {'sigrok': {<class 'labgrid.resource.udev.SigrokUSBDevice'>, <class 'labgrid
```

```
__attrs_post_init__()
```

```
on_activate()
```

```
on_deactivate()
```

```
capture(filename, samplerate='200k')
```

```
stop()
```

```
analyze(args, filename=None)
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name)
```

```
__module__ = 'labgrid.driver.sigrokdriver'
```

```
__repr__()
```

Automatically created by attrs.

labgrid.driver.smallubootdriver module

```

class labgrid.driver.smallubootdriver.SmallUBootDriver(target, name, prompt="",
                                                         password="", interrupt='n',
                                                         init_commands=NOTHING,
                                                         password_prompt='enter
                                                         Password:',
                                                         boot_expression='U-
                                                         Boot 20d+', boot-
                                                         string='Linux version
                                                         \d', boot_secret='a')

```

Bases: *labgrid.driver.ubootdriver.UBootDriver*

SmallUBootDriver is meant as a driver for UBoot with only little functionality compared to standard a standard UBoot. Especially it copes with the following limitations:

- The UBoot does not have a real password-prompt but can be activated by entering a “secret” after a message was displayed.

- The command line does not have a build-in echo command. Thus this driver uses ‘Unknown Command’ messages as marker before and after the output of a command.
- Since there is no echo we can not return the exit code of the command. Commands will always return 0 unless the command was not found.

This driver needs the following features activated in UBoot to work:

- The UBoot must not have real password prompt. Instead it must be keyword activated. For example it should be activated by a dialog like the following: UBoot: “Autobooting in 1s...” Labgrid: “secret” UBoot: <switching to console>
- The UBoot must be able to parse multiple commands in a single line separated by “;”.
- The UBoot must support the “bootm” command to boot from a memory location.

This driver was created especially for the following devices:

- TP-Link WR841 v11

boot (*name*)

Boot the device from the given memory location using ‘bootm’.

Parameters *name* (*str*) – address to boot

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, prompt=", password=", interrupt='\n', init_commands=NOTHING, password_prompt='enter Password:', boot_expression='U-Boot 20\d+', bootstring='Linux version \d', boot_secret='a')
```

```
__module__ = 'labgrid.driver.smallubootdriver'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.driver.sshdriver module

The SSHDriver uses SSH as a transport to implement CommandProtocol and FileTransferProtocol

```
class labgrid.driver.sshdriver.SSHDriver (target, name, keyfile="")
```

Bases: `labgrid.driver.commandmixin.CommandMixin`, `labgrid.driver.common.Driver`, `labgrid.protocol.commandprotocol.CommandProtocol`, `labgrid.protocol.filetransferprotocol.FileTransferProtocol`

SSHDriver - Driver to execute commands via SSH

```
bindings = {'networkservice': <class 'labgrid.resource.networkservice.NetworkService'
```

```
__attrs_post_init__ ()
```

```
on_activate ()
```

```
on_deactivate ()
```

```
run (cmd, codec='utf-8', decodeerrors='strict')
```

Execute *cmd* on the target.

This method runs the specified *cmd* as a command on its target. It uses the ssh shell command to run the command and parses the exitcode. *cmd* - command to be run on the target

returns: (stdout, stderr, returncode)

```
get_status ()
```

The SSHDriver is always connected, return 1

```
put (filename, remotepath=None)
```

```
get (filename, destination='.')
```

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True,
```

```
__init__ (target, name, keyfile="")
```

```
__module__ = 'labgrid.driver.sshdriver'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.driver.ubootdriver module

The U-Boot Module contains the UBootDriver

```
class labgrid.driver.ubootdriver.UBootDriver (target, name, prompt="", password="",
                                             interrupt='n', init_commands=NOTHING,
                                             password_prompt='enter Password:',
                                             boot_expression='U-Boot 20\nd+', boot-
                                             string='Linux version \d')
```

Bases: `labgrid.driver.commandmixin.CommandMixin`, `labgrid.driver.common.Driver`, `labgrid.protocol.commandprotocol.CommandProtocol`, `labgrid.protocol.linuxbootprotocol.LinuxBootProtocol`

UBootDriver - Driver to control uboot via the console. UBootDriver binds on top of a ConsoleProtocol.

Parameters

- **prompt** (*str*) – The default UBoot Prompt
- **password** (*str*) – optional password to unlock UBoot
- **init_commands** (*Tuple[str]*) – a tuple of commands to run after unlock
- **interrupt** (*str*) – interrupt character to use to go to prompt
- **password_prompt** (*str*) – string to detect the password prompt
- **boot_expression** (*str*) – string to search for on UBoot start
- **bootstring** (*str*) – string that indicates that the Kernel is booting

```
bindings = {'console': <class 'labgrid.protocol.consoleprotocol.ConsoleProtocol'>}
```

```
__attrs_post_init__ ()
```

```
on_activate ()
```

Activate the UBootDriver

This function checks for a prompt and awaits it if not already active

```
on_deactivate ()
```

Deactivate the UBootDriver

Simply sets the internal status to 0

```
run (cmd)
```

Runs the specified command on the shell and returns the output.

Parameters `cmd` (*str*) – command to run on the shell

Returns if successful, None otherwise

Return type Tuple[List[str],List[str], int]

get_status ()

Retrieve status of the UBootDriver. 0 means inactive, 1 means active.

Returns status of the driver

Return type int

reset ()

Reset the board via a CPU reset

await_boot ()

Wait for the initial Linux version string to verify we successfully jumped into the kernel.

boot (*name*)

Boot the default or a specific boot entry

Parameters `name` (*str*) – name of the entry to boot

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, prompt=", password=", interrupt='\n', init_commands=NOTHING, pass-  
word_prompt='enter Password:', boot_expression='U-Boot 20\d+', bootstring='Linux  
version \d')
```

```
__module__ = 'labgrid.driver.ubootdriver'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.driver.usbloader module

```
class labgrid.driver.usbloader.MXSUSBDriver (target, name, image=None)
```

```
Bases: labgrid.driver.common.Driver, labgrid.protocol.bootstrapprotocol.  
BootstrapProtocol
```

```
bindings = {'loader': {<class 'labgrid.resource.udev.MXSUSBLoader'>, <class 'labgrid.
```

```
__attrs_post_init__ ()
```

```
on_activate ()
```

```
on_deactivate ()
```

```
load (filename=None)
```

```
__abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, image=None)
```

```
__module__ = 'labgrid.driver.usbloader'
```

```
__repr__ ()
```

Automatically created by attrs.

```
class labgrid.driver.usbloader.IMXUSBDriver (target, name, image=None)
    Bases: labgrid.driver.common.Driver, labgrid.protocol.bootstrapprotocol.
           BootstrapProtocol

    bindings = {'loader': {<class 'labgrid.resource.udev.IMXUSBLoader'>, <class 'labgrid.
    __attrs_post_init__ ()
    on_activate ()
    on_deactivate ()
    load (filename=None)
    __abstractmethods__ = frozenset ()
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name, image=None)
    __module__ = 'labgrid.driver.usbloader'
    __repr__ ()
        Automatically created by attrs.
```

labgrid.driver.usbstorage module

```
class labgrid.driver.usbstorage.USBStorageDriver (target, name)
    Bases: labgrid.driver.common.Driver

    bindings = {'storage': {<class 'labgrid.resource.udev.USBSDMuxDevice'>, <class 'labgr
    __attrs_post_init__ ()
    on_activate ()
    on_deactivate ()
    write_image (filename)
    get_size ()
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name)
    __module__ = 'labgrid.driver.usbstorage'
    __repr__ ()
        Automatically created by attrs.
```

labgrid.external package

Submodules

labgrid.external.hawkbit module

```
class labgrid.external.hawkbit.HawkbitTestClient (host, port, username, password, ver-
                                                    sion=1.0)
    Bases: object
    __attrs_post_init__ ()
```

add_target (*target_id: str, token: str*)
Add a target to the HawkBit Installation

Parameters

- **target_id** (-) – the (unique) device name of the target to add
- **token** (-) – pre-shared key to authenticate the target

delete_target (*target_id: str*)
Delete a target from the HawkBit Installation

Parameters **target_id** (-) – the (unique) device name of the target to delete

add_swmodule (*modulename: str*)

delete_swmodule (*module_id: str*)
Delete a softwaremodule from the HawkBit Installation

Parameters **module_id** (-) – the ID given by hawkBit for the module

add_distributionset (*module_id, name=None*)

delete_distributionset (*distset_id: str*)
Delete a distrubitionset from the HawkBit Installation

Parameters **distset_id** (-) – the ID of the distribution set to delete

add_artifact (*module_id: str, filename: str*)

delete_artifact (*module_id: str, artifact_id: str*)
Delete an artifact from the HawkBit Installation

Parameters **artifact_id** (-) – the ID of the artifact to delete

assign_target (*distribution_id, target_id*)

add_rollout (*name, distribution_id, groups*)

start_rollout (*rollout_id*)

post (*endpoint: str*)

post_json (*endpoint: str, data: dict*)

post_binary (*endpoint: str, filename: str*)

delete (*endpoint: str*)

get_endpoint (*endpoint: str*)

```
__attrs_attrs__ = (Attribute(name='host', default=NOTHING, validator=<instance_of vali
```

```
__dict__ = mappingproxy({'__attrs_attrs__': (Attribute(name='host', default=NOTHING, v
```

```
__init__ (host, port, username, password, version=1.0)
```

```
__module__ = 'labgrid.external.hawkbit'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

exception labgrid.external.hawkbit.**HawkbiError** (*msg*)
Bases: Exception

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=None, repr=True, cr
```

```

__init__(msg)
__module__ = 'labgrid.external.hawkbit'
__repr__()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)

```

labgrid.external.usbstick module

The USBStick module provides support to interactively use a simulated USB device in a test.

```
class labgrid.external.usbstick.USBStatus
```

Bases: `enum.Enum`

This class describes the USBStick Status

```

unplugged = 0
plugged = 1
mounted = 2
__module__ = 'labgrid.external.usbstick'
__new__(value)

```

```
class labgrid.external.usbstick.USBStick(target, image_dir, image_name="")
```

Bases: `object`

The USBStick class provides an easy to use interface to describe a target as an USB Stick.

```
__attrs_post_init__()
```

```
plug_in()
```

Insert the USBStick

This function plugs the virtual USB Stick in, making it available to the connected computer.

```
plug_out()
```

Plugs out the USBStick

Plugs out the USBStick from the connected computer, does nothing if it is already unplugged

```
put_file(filename, destination="")
```

Put a file onto the USBStick Image

Puts a file onto the USB Stick, raises a `StateError` if it is not mounted on the host computer.

```
get_file(filename)
```

Gets a file from the USBStick Image

Gets a file from the USB Stick, raises a `StateError` if it is not mounted on the host computer.

```
upload_image(image)
```

Upload a complete image as a new USB Stick

This replaces the current USB Stick image, storing it permanently on the RiotBoard.

```
switch_image(image_name)
```

Switch between already uploaded images on the target.

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True,
```

```
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'USBStick' object>
__init__ (target, image_dir, image_name=")
__module__ = 'labgrid.external.usbstick'
__repr__ ()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
```

exception labgrid.external.usbstick.StateError (*msg*)

Bases: Exception

Exception which indicates a error in the state handling of the test

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=None, repr=True, c
__init__ (msg)
__module__ = 'labgrid.external.usbstick'
__repr__ ()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
```

labgrid.protocol package

Submodules

labgrid.protocol.bootstrapprotocol module

class labgrid.protocol.bootstrapprotocol.BootstrapProtocol

Bases: abc.ABC

load (*filename: str*)

```
__abstractmethods__ = frozenset({'load'})
```

```
__module__ = 'labgrid.protocol.bootstrapprotocol'
```

labgrid.protocol.commandprotocol module

class labgrid.protocol.commandprotocol.CommandProtocol

Bases: abc.ABC

Abstract class for the CommandProtocol

run (*command: str*)

Run a command

run_check (*command: str*)

Run a command, return str if succesful, ExecutionError otherwise

get_status ()

Get status of the Driver


```
wait_for()
    Wait for a shell command to return with the specified output

__abstractmethods__ = frozenset({'run', 'get_status', 'run_check', 'wait_for'})
__module__ = 'labgrid.protocol.commandprotocol'
```

labgrid.protocol.consoleprotocol module

```
class labgrid.protocol.consoleprotocol.ConsoleProtocol
    Bases: abc.ABC

    Abstract class for the ConsoleProtocol

    read()
        Read data from underlying port

    write(data: bytes)
        Write data to underlying port

    sendline(line: str)

    sendcontrol(char: str)

    expect(pattern: str)

    class Client
        Bases: abc.ABC

        get_console_matches()

        notify_console_match(pattern, match)

        __abstractmethods__ = frozenset({'get_console_matches', 'notify_console_match'})
        __module__ = 'labgrid.protocol.consoleprotocol'

        __abstractmethods__ = frozenset({'read', 'write'})
        __module__ = 'labgrid.protocol.consoleprotocol'
```

labgrid.protocol.digitaloutputprotocol module

```
class labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol
    Bases: abc.ABC

    Abstract class providing the OneWireProtocol interface

    get()
        Implementations should return the status of the OneWirePort.

    set(status)
        Implementations should set the status of the OneWirePort

    __abstractmethods__ = frozenset({'get', 'set'})
    __module__ = 'labgrid.protocol.digitaloutputprotocol'
```

labgrid.protocol.filesystemprotocol module

```
class labgrid.protocol.filesystemprotocol.FileSystemProtocol
    Bases: abc.ABC

    read (filename: str)

    write (filename: str, data: bytes, append: bool)

    __abstractmethods__ = frozenset({'read', 'write'})
    __module__ = 'labgrid.protocol.filesystemprotocol'
```

labgrid.protocol.filetransferprotocol module

```
class labgrid.protocol.filetransferprotocol.FileTransferProtocol
    Bases: abc.ABC

    put (filename: str, remotepath: str)

    get (filename: str, destination: str)

    __abstractmethods__ = frozenset({'put', 'get'})
    __module__ = 'labgrid.protocol.filetransferprotocol'
```

labgrid.protocol.infoprotocol module

```
class labgrid.protocol.infoprotocol.InfoProtocol
    Bases: abc.ABC

    Abstract class providing the InfoProtocol interface

    get_ip (interface: str = 'eth0')
        Implementations should return the IP-adress for the supplied interface.

    get_hostname ()
        Implementations should return the hostname for the supplied interface.

    get_service_status (service)
        Implementations should return the status of a service

    __abstractmethods__ = frozenset({'get_ip', 'get_service_status', 'get_hostname'})
    __module__ = 'labgrid.protocol.infoprotocol'
```

labgrid.protocol.linuxbootprotocol module

```
class labgrid.protocol.linuxbootprotocol.LinuxBootProtocol
    Bases: abc.ABC

    boot (name: str)

    await_boot ()

    reset ()

    __abstractmethods__ = frozenset({'reset', 'boot', 'await_boot'})
    __module__ = 'labgrid.protocol.linuxbootprotocol'
```

labgrid.protocol.mmioprotocol module

```
class labgrid.protocol.mmioprotocol.MMIOProtocol
    Bases: abc.ABC

    read (address: int, size: int, count: int) → bytes
    write (address: int, size: int, data: bytes) → None

    __abstractmethods__ = frozenset({'read', 'write'})
    __module__ = 'labgrid.protocol.mmioprotocol'
```

labgrid.protocol.powerprotocol module

```
class labgrid.protocol.powerprotocol.PowerProtocol
    Bases: abc.ABC

    on ()
    off ()
    cycle ()

    __abstractmethods__ = frozenset({'off', 'on', 'cycle'})
    __module__ = 'labgrid.protocol.powerprotocol'
```

labgrid.provider package

Submodules

labgrid.provider.fileprovider module

```
class labgrid.provider.fileprovider.FileProvider
    Bases: abc.ABC

    Abstract class for the FileProvider

    get (name: str) → dict
        Get a dictionary of target paths to local paths for a given name.

    list ()
        Get a list of names.

    __abstractmethods__ = frozenset({'list', 'get'})
    __module__ = 'labgrid.provider.fileprovider'
```

labgrid.provider.mediafileprovider module

```
class labgrid.provider.mediafileprovider.MediaFileProvider (groups={})
    Bases: labgrid.provider.fileprovider.FileProvider

    get (name)

    list ()

    __abstractmethods__ = frozenset ()
```

```
__attrs_attrs__ = (Attribute(name='groups', default={}, validator=<instance of validator function...
__init__(groups={})
__module__ = 'labgrid.provider.mediafileprovider'
__repr__()
    Automatically created by attrs.
```

labgrid.pytestplugin package

Submodules

labgrid.pytestplugin.fixtures module

```
labgrid.pytestplugin.fixtures.env(request)
    Return the environment configured in the supplied configuration file. It contains the targets contained in the
    configuration file.
labgrid.pytestplugin.fixtures.pytest_adoption(parser)
labgrid.pytestplugin.fixtures.target(env)
    Return the default target main configured in the supplied configuration file.
```

labgrid.pytestplugin.reporter module

```
labgrid.pytestplugin.reporter.bold(text)
labgrid.pytestplugin.reporter.under(text)
labgrid.pytestplugin.reporter.pytest_configure(config)
class labgrid.pytestplugin.reporter.StepReporter(terminalreporter, *, rewrite=False)
    Bases: object
    __init__(terminalreporter, *, rewrite=False)
    notify(event)
    pytest_runtest_logstart()
    pytest_runtest_logreport(report)
    __dict__ = mappingproxy({'_StepReporter__commit': <function StepReporter.__commit>, '_
    __module__ = 'labgrid.pytestplugin.reporter'
    __weakref__
        list of weak references to the object (if defined)
```

labgrid.remote package

Submodules

labgrid.remote.authenticator module

labgrid.remote.client module

The remote.client module contains the functionality to connect to a coordinator, acquire a place and interact with the connected resources

```
exception labgrid.remote.client.Error
    Bases: Exception
    __module__ = 'labgrid.remote.client'
    __weakref__
        list of weak references to the object (if defined)
```

```
exception labgrid.remote.client.UserError
    Bases: labgrid.remote.client.Error
    __module__ = 'labgrid.remote.client'
```

```
exception labgrid.remote.client.ServerError
    Bases: labgrid.remote.client.Error
    __module__ = 'labgrid.remote.client'
```

```
labgrid.remote.client.start_session(url, realm, extra)
```

```
labgrid.remote.client.find_role_by_place(config, place)
```

```
labgrid.remote.client.find_any_role_with_place(config)
```

```
labgrid.remote.client.main()
```

labgrid.remote.common module

```
class labgrid.remote.common.ResourceEntry(data, acquired=None)
    Bases: object
    __attrs_post_init__()
    avail
    cls
    params
    args
        arguments for resource construction
    extra
        extra resource information
    asdict()
    __attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
    __dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'ResourceEntry' of
```

```
__init__(data, acquired=None)
__module__ = 'labgrid.remote.common'
__repr__()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
class labgrid.remote.common.ResourceMatch(exporter, group, cls, name=None, re-
                                         name=None)
Bases: object
classmethod fromstr(pattern)
__repr__()
__str__()
ismatch(resource_path)
__attrs_attrs__ = (Attribute(name='exporter', default=NOTHING, validator=None, repr=Tr
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'ResourceMatch' of
__eq__(other)
__ge__(other)
    Automatically created by attrs.
__gt__(other)
    Automatically created by attrs.
__hash__ = None
__init__(exporter, group, cls, name=None, rename=None)
__le__(other)
    Automatically created by attrs.
__lt__(other)
    Automatically created by attrs.
__module__ = 'labgrid.remote.common'
__ne__(other)
    Check equality and either forward a NotImplemented or return the result negated.
__weakref__
    list of weak references to the object (if defined)
class labgrid.remote.common.Place(name, aliases=NOTHING, comment="",
                                  matches=NOTHING, acquired=None, ac-
                                  quired_resources=NOTHING, created=NOTHING,
                                  changed=NOTHING)
Bases: object
asdict()
show(level=0)
getmatch(resource_path)
    Return the ResourceMatch object for the given resource path or None if not found.
    A resource_path has the structure (exporter, group, cls, name).
```

hasmatch (*resource_path*)

Return True if this place as a ResourceMatch object for the given resource path.

A resource_path has the structure (exporter, group, cls, name).

touch ()

```
__attrs_attrs__ = (Attribute(name='name', default=NOTHING, validator=None, repr=True, c
```

```
__dict__ = mappingproxy({'getmatch': <function Place.getmatch>, 'touch': <function P
```

```
__init__(name, aliases=NOTHING, comment="", matches=NOTHING, acquired=None, ac-
quired_resources=NOTHING, created=NOTHING, changed=NOTHING)
```

```
__module__ = 'labgrid.remote.common'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

labgrid.remote.common.**enable_tcp_nodelay** (*session*)

asyncio/autobahn does not set TCP_NODELAY by default, so we need to do it like this for now.

labgrid.remote.config module

class labgrid.remote.config.**ResourceConfig** (*filename*)

Bases: object

```
__attrs_post_init__ ()
```

```
__attrs_attrs__ = (Attribute(name='filename', default=NOTHING, validator=<instance_of v
```

```
__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'ResourceConfig' object
```

```
__init__ (filename)
```

```
__module__ = 'labgrid.remote.config'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

labgrid.remote.coordinator module

The coordinator module coordinates exported resources and clients accessing them.

class labgrid.remote.coordinator.**Action**

Bases: enum.Enum

An enumeration.

```
ADD = 0
```

```
DEL = 1
```

```
UPD = 2
```

```
__module__ = 'labgrid.remote.coordinator'
```

```
__new__ (value)
```

```
class labgrid.remote.coordinator.RemoteSession
```

Bases: object

class encapsulating a session, used by ExporterSession and ClientSession

key

Key of the session

name

Name of the session

```
__attrs_attrs__ = (Attribute(name='coordinator', default=NOTHING, validator=None, repr
```

```
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'RemoteSession' of
```

```
__module__ = 'labgrid.remote.coordinator'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

```
class labgrid.remote.coordinator.ExporterSession(coordinator, session, authid)
```

Bases: *labgrid.remote.coordinator.RemoteSession*

An ExporterSession is opened for each Exporter connecting to the coordinator, allowing the Exporter to get and set resources

```
set_resource (groupname, resourcename, resource)
```

```
get_resources ()
```

Method invoked by the exporter, get a resource from the coordinator

```
__attrs_attrs__ = (Attribute(name='coordinator', default=NOTHING, validator=None, repr
```

```
__init__ (coordinator, session, authid)
```

```
__module__ = 'labgrid.remote.coordinator'
```

```
__repr__ ()
```

Automatically created by attrs.

```
class labgrid.remote.coordinator.ClientSession(coordinator, session, authid)
```

Bases: *labgrid.remote.coordinator.RemoteSession*

```
__attrs_attrs__ = (Attribute(name='coordinator', default=NOTHING, validator=None, repr
```

```
__init__ (coordinator, session, authid)
```

```
__module__ = 'labgrid.remote.coordinator'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.remote.exporter module

The remote.exporter module exports resources to the coordinator and makes them available to other clients on the same coordinator

```
labgrid.remote.exporter.get_free_port ()
```

Helper function to always return an unused port.


```
class labgrid.remote.exporter.ResourceExport (data, acquired=None, host='build-6919692-project-82349-labgrid')
```

Bases: *labgrid.remote.common.ResourceEntry*

Represents a local resource exported via a specific protocol.

The ResourceEntry attributes contain the information for the client.

```
__attrs_post_init__()
```

```
poll()
```

```
__attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
```

```
__init__(data, acquired=None, host='build-6919692-project-82349-labgrid')
```

```
__module__ = 'labgrid.remote.exporter'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.remote.exporter.USBSerialPortExport (data, acquired=None, host='build-6919692-project-82349-labgrid')
```

Bases: *labgrid.remote.exporter.ResourceExport*

ResourceExport for a USB SerialPort

```
__attrs_post_init__()
```

```
__del__()
```

```
__attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
```

```
__init__(data, acquired=None, host='build-6919692-project-82349-labgrid')
```

```
__module__ = 'labgrid.remote.exporter'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.remote.exporter.USBEthernetExport (data, acquired=None, host='build-6919692-project-82349-labgrid')
```

Bases: *labgrid.remote.exporter.ResourceExport*

ResourceExport for a USB ethernet interface

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
```

```
__init__(data, acquired=None, host='build-6919692-project-82349-labgrid')
```

```
__module__ = 'labgrid.remote.exporter'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.remote.exporter.USBGenericExport (data, acquired=None, host='build-6919692-project-82349-labgrid')
```

Bases: *labgrid.remote.exporter.ResourceExport*

ResourceExport for USB devices accessed directly from userspace

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
```

```
__init__(data, acquired=None, host='build-6919692-project-82349-labgrid')
```

```
__module__ = 'labgrid.remote.exporter'
__repr__ ()
    Automatically created by attrs.
class labgrid.remote.exporter.USBSigrokExport (data, acquired=None, host='build-
    6919692-project-82349-labgrid')
    Bases: labgrid.remote.exporter.USBGenericExport
    ResourceExport for USB devices accessed directly from userspace
    __attrs_post_init__ ()
    __attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
    __init__ (data, acquired=None, host='build-6919692-project-82349-labgrid')
    __module__ = 'labgrid.remote.exporter'
    __repr__ ()
        Automatically created by attrs.
class labgrid.remote.exporter.USBSDMuxExport (data, acquired=None, host='build-
    6919692-project-82349-labgrid')
    Bases: labgrid.remote.exporter.USBGenericExport
    ResourceExport for USB devices accessed directly from userspace
    __attrs_post_init__ ()
    __attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
    __init__ (data, acquired=None, host='build-6919692-project-82349-labgrid')
    __module__ = 'labgrid.remote.exporter'
    __repr__ ()
        Automatically created by attrs.
class labgrid.remote.exporter.USBPowerPortExport (data, acquired=None, host='build-
    6919692-project-82349-labgrid')
    Bases: labgrid.remote.exporter.USBGenericExport
    ResourceExport for ports on switchable USB hubs
    __attrs_post_init__ ()
    __attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
    __init__ (data, acquired=None, host='build-6919692-project-82349-labgrid')
    __module__ = 'labgrid.remote.exporter'
    __repr__ ()
        Automatically created by attrs.
class labgrid.remote.exporter.EthernetPortExport (data, acquired=None, host='build-
    6919692-project-82349-labgrid')
    Bases: labgrid.remote.exporter.ResourceExport
    ResourceExport for a ethernet interface
    __attrs_post_init__ ()
    __attrs_attrs__ = (Attribute(name='data', default=NOTHING, validator=None, repr=True,
    __eq__ (other)
```

```
__ge__ (other)
    Automatically created by attrs.

__gt__ (other)
    Automatically created by attrs.

__hash__ = None

__init__ (data, acquired=None, host='build-6919692-project-82349-labgrid')

__le__ (other)
    Automatically created by attrs.

__lt__ (other)
    Automatically created by attrs.

__module__ = 'labgrid.remote.exporter'

__ne__ (other)
    Check equality and either forward a NotImplemented or return the result negated.

__repr__ ()
    Automatically created by attrs.
```

```
labgrid.remote.exporter.main()
```

labgrid.resource package

Submodules

labgrid.resource.base module

```
class labgrid.resource.base.SerialPort (target, name, port=None, speed=115200)
```

Bases: [labgrid.resource.common.Resource](#)

The basic SerialPort describes port and speed

Parameters

- **port** (*str*) – port to connect to
- **speed** (*int*) – speed of the port, defaults to 115200

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, port=None, speed=115200)
```

```
__module__ = 'labgrid.resource.base'
```

```
__repr__ ()
```

Automatically created by attrs.

```
class labgrid.resource.base.EthernetInterface (target, name, ifname=None)
```

Bases: [labgrid.resource.common.Resource](#)

The basic EthernetInterface contains an interfacename

Parameters **ifname** (*str*) – name of the interface

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, ifname=None)
```

```
__module__ = 'labgrid.resource.base'
```

`__repr__()`
Automatically created by attrs.

class `labgrid.resource.base.EthernetPort` (*target, name, switch=None, interface=None*)

Bases: `labgrid.resource.common.Resource`

The basic EthernetPort describes a switch and interface

Parameters

- **switch** (*str*) – name of the switch
- **interface** (*str*) – name of the interface

`__attrs_attrs__` = (`Attribute`(name='target', default=NOTHING, validator=None, repr=True

`__eq__` (*other*)

`__ge__` (*other*)
Automatically created by attrs.

`__gt__` (*other*)
Automatically created by attrs.

`__hash__` = None

`__init__` (*target, name, switch=None, interface=None*)

`__le__` (*other*)
Automatically created by attrs.

`__lt__` (*other*)
Automatically created by attrs.

`__module__` = 'labgrid.resource.base'

`__ne__` (*other*)
Check equality and either forward a NotImplemented or return the result negated.

`__repr__()`
Automatically created by attrs.

labgrid.resource.common module

class `labgrid.resource.common.Resource` (*target, name*)

Bases: `labgrid.binding.BindingMixin`

Represents a resource which is used by drivers. It only stores information and does not implement any actual functionality.

Resources can exist without a target, but they must be bound to one before use.

Life cycle:

- create
- bind (n times)

`__attrs_post_init__()`

`command_prefix`

`parent`

get_managed_parent ()

For Resources which have been created at runtime, return the ManagedResource resource which created it.

Returns None otherwise.

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name)
```

```
__module__ = 'labgrid.resource.common'
```

```
__repr__ ()
```

Automatically created by attrs.

class labgrid.resource.common.**NetworkResource** (target, name, host)

Bases: *labgrid.resource.common.Resource*

Represents a remote Resource available on another computer.

This stores a command_prefix to describe how to connect to the remote computer.

Parameters **host** (*str*) – remote host the resource is available on

command_prefix

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host)
```

```
__module__ = 'labgrid.resource.common'
```

```
__repr__ ()
```

Automatically created by attrs.

class labgrid.resource.common.**ResourceManager**

Bases: object

```
instances = {}
```

```
classmethod get ()
```

```
__attrs_post_init__ ()
```

```
on_resource_added (resource)
```

```
poll ()
```

```
__attrs_attrs__ = ()
```

```
__dict__ = mappingproxy({'on_resource_added': <function ResourceManager.on_resource_a
```

```
__init__ ()
```

```
__module__ = 'labgrid.resource.common'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

class labgrid.resource.common.**ManagedResource** (target, name)

Bases: *labgrid.resource.common.Resource*

Represents a resource which can appear and disappear at runtime. Every ManagedResource has a corresponding ResourceManager which handles these events.

```
manager_cls
    alias of ResourceManager

__attrs_post_init__()

poll()

__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__(target, name)

__module__ = 'labgrid.resource.common'

__repr__()
    Automatically created by attrs.

get_managed_parent()
```

labgrid.resource.ethernetport module

```
class labgrid.resource.ethernetport.SNMPSwitch(hostname)
```

Bases: object

SNMPSwitch describes a switch accessible over SNMP. This class implements functions to query ports and the forwarding database.

```
__attrs_post_init__()

update()
    Update port status and forwarding database status

    Returns None

__attrs_attrs__ = (Attribute(name='hostname', default=NOTHING, validator=<instance_of
__dict__ = mappingproxy({'_get_ports': <function SNMPSwitch._get_ports>, '__weakref__
__eq__(other)

__ge__(other)
    Automatically created by attrs.

__gt__(other)
    Automatically created by attrs.

__hash__ = None

__init__(hostname)

__le__(other)
    Automatically created by attrs.

__lt__(other)
    Automatically created by attrs.

__module__ = 'labgrid.resource.ethernetport'

__ne__(other)
    Check equality and either forward a NotImplemented or return the result negated.

__repr__()
    Automatically created by attrs.

__weakref__
    list of weak references to the object (if defined)
```

class labgrid.resource.ethernetport.**EthernetPortManager**

Bases: *labgrid.resource.common.ResourceManager*

The EthernetPortManager periodically polls the switch for new updates.

`__attrs_post_init__()`

`on_resource_added(resource)`

Handler to execute when the resource is added

Checks whether the resource can be managed by this Manager and starts the event loop.

Parameters **resource** (*Resource*) – resource to check against

Returns None

`poll()`

Updates the state with new information from the event loop

Returns None

`__attrs_attrs__ = ()`

`__eq__(other)`

`__ge__(other)`

Automatically created by attrs.

`__gt__(other)`

Automatically created by attrs.

`__hash__ = None`

`__init__()`

`__le__(other)`

Automatically created by attrs.

`__lt__(other)`

Automatically created by attrs.

`__module__ = 'labgrid.resource.ethernetport'`

`__ne__(other)`

Check equality and either forward a NotImplemented or return the result negated.

`__repr__()`

Automatically created by attrs.

class labgrid.resource.ethernetport.**SNMPEthernetPort** (*target, name, switch, interface*)

Bases: *labgrid.resource.common.ManagedResource*

SNMPEthernetPort describes an ethernet port which can be queried over SNMP.

Parameters

- **switch** (*str*) – hostname of the switch to query
- **interface** (*str*) – name of the interface to query

manager_cls

alias of *EthernetPortManager*

`__attrs_post_init__()`

`__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True`

`__eq__(other)`

```
__ge__ (other)
    Automatically created by attrs.

__gt__ (other)
    Automatically created by attrs.

__hash__ = None

__init__ (target, name, switch, interface)

__le__ (other)
    Automatically created by attrs.

__lt__ (other)
    Automatically created by attrs.

__module__ = 'labgrid.resource.ethernetport'

__ne__ (other)
    Check equality and either forward a NotImplemented or return the result negated.

__repr__ ()
    Automatically created by attrs.
```

labgrid.resource.modbus module

```
class labgrid.resource.modbus.ModbusTCPCoil (target, name, host, coil, invert=False)
```

Bases: `labgrid.resource.common.Resource`

This resource describes Modbus TCP coil.

Parameters

- **host** (*str*) – hostname of the Modbus TCP server e.g. “192.168.23.42:502”
- **coil** (*int*) – index of the coil e.g. 3
- **invert** (*bool*) – optional, whether the logic level is be inverted (active-low)

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True)
__init__ (target, name, host, coil, invert=False)
__module__ = 'labgrid.resource.modbus'
__repr__ ()
    Automatically created by attrs.
```

labgrid.resource.networkservice module

```
class labgrid.resource.networkservice.NetworkService (target, name, address, user-
name, password=”)
```

Bases: `labgrid.resource.common.Resource`

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True)
__init__ (target, name, address, username, password=”)
__module__ = 'labgrid.resource.networkservice'
__repr__ ()
    Automatically created by attrs.
```


labgrid.resource.onewireport module

class labgrid.resource.onewireport.**OneWirePIO** (*target, name, host, path, invert=False*)

Bases: *labgrid.resource.common.Resource*

This resource describes a Onewire PIO Port.

Parameters

- **host** (*str*) – hostname of the owserver e.g. localhost:4304
- **path** (*str*) – path to the port on the owserver e.g. 29.7D6913000000/PIO.0
- **invert** (*bool*) – optional, whether the logic level is be inverted (active-low)

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, path, invert=False)
```

```
__module__ = 'labgrid.resource.onewireport'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.resource.power module

class labgrid.resource.power.**NetworkPowerPort** (*target, name, model, host, index*)

Bases: *labgrid.resource.common.Resource*

The NetworkPowerPort describes a remotely switchable PowerPort

Parameters

- **model** (*str*) – model of the external power switch
- **host** (*str*) – host to connect to
- **index** (*str*) – index of the power port on the external switch

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, model, host, index)
```

```
__module__ = 'labgrid.resource.power'
```

```
__repr__ ()
```

Automatically created by attrs.

labgrid.resource.remote module

class labgrid.resource.remote.**RemotePlaceManager**

Bases: *labgrid.resource.common.ResourceManager*

```
__attrs_post_init__ ()
```

```
on_resource_added (resource)
```

```
poll ()
```

```
__attrs_attrs__ = ()
```

```
__init__ ()
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__()  
    Automatically created by attrs.
```

```
class labgrid.resource.remote.RemotePlace(target, name)
```

```
    Bases: labgrid.resource.common.ManagedResource
```

```
    manager_cls  
        alias of RemotePlaceManager
```

```
    __attrs_post_init__()
```

```
    __attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
    __init__(target, name)
```

```
    __module__ = 'labgrid.resource.remote'
```

```
    __repr__()  
    Automatically created by attrs.
```

```
class labgrid.resource.remote.RemoteUSBResource(target, name, host, busnum, devnum,  
                                                path, vendor_id, model_id)
```

```
    Bases: labgrid.resource.common.NetworkResource, labgrid.resource.common.  
           ManagedResource
```

```
    manager_cls  
        alias of RemotePlaceManager
```

```
    __attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
    __init__(target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
    __module__ = 'labgrid.resource.remote'
```

```
    __repr__()  
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkAndroidFastboot(target, name, host, busnum,  
                                                    devnum, path, vendor_id,  
                                                    model_id)
```

```
    Bases: labgrid.resource.remote.RemoteUSBResource
```

```
    __attrs_post_init__()
```

```
    __attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
    __init__(target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
    __module__ = 'labgrid.resource.remote'
```

```
    __repr__()  
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkIMXUSBLoader(target, name, host, busnum, devnum,  
                                                  path, vendor_id, model_id)
```

```
    Bases: labgrid.resource.remote.RemoteUSBResource
```

```
    __attrs_post_init__()
```

```
    __attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
    __init__(target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
    __module__ = 'labgrid.resource.remote'
```

```
    __repr__()  
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkMXSUSBLoader (target, name, host, busnum, devnum,  
                                                path, vendor_id, model_id)
```

```
Bases: labgrid.resource.remote.RemoteUSBResource
```

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__()
```

```
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkAlteraUSBBlaster (target, name, host, busnum,  
                                                       devnum, path, vendor_id,  
                                                       model_id)
```

```
Bases: labgrid.resource.remote.RemoteUSBResource
```

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__()
```

```
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkSigrokUSBDevice (target, name, host, busnum,  
                                                       devnum, path, vendor_id,  
                                                       model_id, driver=None, chan-  
                                                       nels=None)
```

```
Bases: labgrid.resource.remote.RemoteUSBResource
```

```
The NetworkSigrokUSBDevice describes a remotely accessible sigrok USB device
```

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id, driver=None, chan-  
          nels=None)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__()
```

```
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkUSBMassStorage (target, name, host, busnum,  
                                                       devnum, path, vendor_id,  
                                                       model_id)
```

```
Bases: labgrid.resource.remote.RemoteUSBResource
```

```
The NetworkUSBMassStorage describes a remotely accessible USB storage device
```

```
__attrs_post_init__()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__()
```

```
    Automatically created by attrs.
```

```
class labgrid.resource.remote.NetworkUSBSDMuxDevice (target, name, host, busnum,  
devnum, path, vendor_id,  
model_id, control_path=None)
```

Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkUSBSDMuxDevice describes a remotely accessible USBSDMux device

```
__attrs_post_init__ ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id, control_path=None)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__ ()
```

Automatically created by attrs.

```
class labgrid.resource.remote.NetworkUSBPowerPort (target, name, host, busnum, de-  
vnum, path, vendor_id, model_id,  
index=None)
```

Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkUSBPowerPort describes a remotely accessible USB hub port with power switching

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, host, busnum, devnum, path, vendor_id, model_id, index=None)
```

```
__module__ = 'labgrid.resource.remote'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__attrs_post_init__ ()
```

labgrid.resource.serialport module

```
class labgrid.resource.serialport.RawSerialPort (target, name, port=None,  
speed=115200)
```

Bases: *labgrid.resource.base.SerialPort, labgrid.resource.common.Resource*

RawSerialPort describes a serialport which is available on the local computer.

```
__attrs_post_init__ ()
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, port=None, speed=115200)
```

```
__module__ = 'labgrid.resource.serialport'
```

```
__repr__ ()
```

Automatically created by attrs.

```
class labgrid.resource.serialport.NetworkSerialPort (target, name, host, port,  
speed=115200, proto-  
col='rfc2217')
```

Bases: *labgrid.resource.common.NetworkResource*

A NetworkSerialPort is a remotely accessible serialport, usually accessed via rfc2217 or tcp raw.

Parameters

- **port** (*str*) – socket port to connect to

- **speed** (*int*) – speed of the port e.g. 9800
- **protocol** (*str*) – connection protocol: “raw” or “rfc2217”

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__(target, name, host, port, speed=115200, protocol='rfc2217')
__module__ = 'labgrid.resource.serialport'
__repr__()
    Automatically created by attrs.
```

labgrid.resource.sigrok module

```
class labgrid.resource.sigrok.SigrokDevice(target, name, driver='demo', chan-
    nels=None)
```

Bases: *labgrid.resource.common.Resource*

The SigrokDevice describes an attached sigrok device with driver and channel mapping

Parameters

- **driver** (*str*) – driver to use with sigrok
- **channels** (*str*) – a sigrok channel mapping as described in the sigrok-cli man page

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__(target, name, driver='demo', channels=None)
__module__ = 'labgrid.resource.sigrok'
__repr__()
    Automatically created by attrs.
```

labgrid.resource.udev module

```
class labgrid.resource.udev.UdevManager
```

Bases: *labgrid.resource.common.ResourceManager*

```
__attrs_post_init__()
on_resource_added(resource)
poll()
__attrs_attrs__ = ()
__init__()
__module__ = 'labgrid.resource.udev'
__repr__()
    Automatically created by attrs.
```

```
class labgrid.resource.udev.USBResource(target, name, match={}, device=None)
```

Bases: *labgrid.resource.common.ManagedResource*

```
manager_cls
    alias of UdevManager
__attrs_post_init__()
```

```
filter_match (device)
try_match (device)
update ()
busnum
devnum
path
vendor_id
model_id
read_attr (attribute)
    read uncached attribute value from sysfs

    pyudev currently supports only cached access to attributes, so we read directly from sysfs.
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name, match={}, device=None)
__module__ = 'labgrid.resource.udev'
__repr__ ()
    Automatically created by attrs.

class labgrid.resource.udev.USBSerialPort (target, name, match={}, device=None,
                                           port=None, speed=115200)
    Bases: labgrid.resource.udev.USBResource, labgrid.resource.base.SerialPort
    __attrs_post_init__ ()
    update ()
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name, match={}, device=None, port=None, speed=115200)
    __module__ = 'labgrid.resource.udev'
    __repr__ ()
        Automatically created by attrs.

class labgrid.resource.udev.USBMassStorage (target, name, match={}, device=None)
    Bases: labgrid.resource.udev.USBResource
    __attrs_post_init__ ()
    path
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
    __init__ (target, name, match={}, device=None)
    __module__ = 'labgrid.resource.udev'
    __repr__ ()
        Automatically created by attrs.

class labgrid.resource.udev.IMXUSBLoader (target, name, match={}, device=None)
    Bases: labgrid.resource.udev.USBResource
    filter_match (device)
    __attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, match={}, device=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.resource.udev.MXSUSBLoader(target, name, match={}, device=None)
```

Bases: *labgrid.resource.udev.USBResource*

```
filter_match(device)
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, match={}, device=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.resource.udev.AndroidFastboot(target, name, match={}, device=None, usb_vendor_id='1d6b', usb_product_id='0104')
```

Bases: *labgrid.resource.udev.USBResource*

```
filter_match(device)
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, match={}, device=None, usb_vendor_id='1d6b', usb_product_id='0104')
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.resource.udev.USBEthernetInterface(target, name, match={}, device=None, ifname=None)
```

Bases: *labgrid.resource.udev.USBResource*, *labgrid.resource.base.EthernetInterface*

```
__attrs_post_init__()
```

```
update()
```

```
if_state
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, match={}, device=None, ifname=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__()
```

Automatically created by attrs.

```
class labgrid.resource.udev.AlteraUSBBlaster(target, name, match={}, device=None)
```

Bases: *labgrid.resource.udev.USBResource*

```
filter_match(device)
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, match={}, device=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__ ()  
    Automatically created by attrs.
```

```
class labgrid.resource.udev.SigrokUSBDevice (target, name, match={}, device=None,  
                                             driver=None, channels=None)
```

Bases: *labgrid.resource.udev.USBResource*

The SigrokUSBDevice describes an attached sigrok device with driver and channel mapping, it is identified via usb using udev

Parameters

- **driver** (*str*) – driver to use with sigrok
- **channels** (*str*) – a sigrok channel mapping as described in the sigrok-cli man page

```
__attrs_post_init__ ()
```

```
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, match={}, device=None, driver=None, channels=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__ ()  
    Automatically created by attrs.
```

```
class labgrid.resource.udev.USBSDMuxDevice (target, name, match={}, device=None)
```

Bases: *labgrid.resource.udev.USBResource*

The USBSDMuxDevice describes an attached USBSDMux device, it is identified via USB using udev

```
__attrs_post_init__ ()
```

```
control_path
```

```
path
```

```
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__init__ (target, name, match={}, device=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__ ()  
    Automatically created by attrs.
```

```
class labgrid.resource.udev.USBPowerPort (target, name, match={}, device=None, in-  
                                          dex=None)
```

Bases: *labgrid.resource.udev.USBResource*

The USBPowerPort describes a single port on an USB hub which supports power control.

Parameters **index** (*int*) – index of the downstream port on the USB hub

```
__attrs_attrs__ = (Attribute (name='target', default=NOTHING, validator=None, repr=True
```

```
__attrs_post_init__ ()
```

```
__init__ (target, name, match={}, device=None, index=None)
```

```
__module__ = 'labgrid.resource.udev'
```

```
__repr__ ()  
    Automatically created by attrs.
```


labgrid.resource.ykushpowerport module

class labgrid.resource.ykushpowerport.YKUSHPowerPort (*target, name, serial, index*)

Bases: *labgrid.resource.common.Resource*

This resource describes a YEPKIT YKUSH switchable USB hub.

Parameters

- **serial** (*str*) – serial of the YKUSH device
- **index** (*int*) – port index

__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True

__init__ (*target, name, serial, index*)

__module__ = 'labgrid.resource.ykushpowerport'

__repr__ ()

Automatically created by attrs.

labgrid.strategy package

Submodules

labgrid.strategy.bareboxstrategy module

exception labgrid.strategy.bareboxstrategy.StrategyError (*msg*)

Bases: Exception

__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid

__init__ (*msg*)

__module__ = 'labgrid.strategy.bareboxstrategy'

__repr__ ()

Automatically created by attrs.

__weakref__

list of weak references to the object (if defined)

class labgrid.strategy.bareboxstrategy.Status

Bases: enum.Enum

An enumeration.

unknown = 0

barebox = 1

shell = 2

__module__ = 'labgrid.strategy.bareboxstrategy'

__new__ (*value*)

class labgrid.strategy.bareboxstrategy.BareboxStrategy (*target, name, status=<Status.unknown: 0>*)

Bases: *labgrid.strategy.common.Strategy*

BareboxStrategy - Strategy to switch to barebox or shell

```
bindings = {'barebox': <class 'labgrid.driver.bareboxdriver.BareboxDriver'>, 'power':
__attrs_post_init__ ()
transition (status, *, step)
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name, status=<Status.unknown: 0>)
__module__ = 'labgrid.strategy.bareboxstrategy'
__repr__ ()
    Automatically created by attrs.
```

labgrid.strategy.common module

```
class labgrid.strategy.common.Strategy (target, name)
```

Bases: *labgrid.driver.common.Driver*

Represents a strategy which places a target into a requested state by calling specific drivers. A strategy usually needs to know some details of a given target.

Life cycle: - create - bind (n times) - usage

TODO: This might also be just a driver?

```
__attrs_post_init__ ()
on_client_bound (client)
on_activate ()
on_deactivate ()
resolve_conflicts (client)
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
__init__ (target, name)
__module__ = 'labgrid.strategy.common'
__repr__ ()
    Automatically created by attrs.
```

labgrid.strategy.shellstrategy module

```
exception labgrid.strategy.shellstrategy.StrategyError (msg)
```

Bases: *Exception*

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
__init__ (msg)
__module__ = 'labgrid.strategy.shellstrategy'
__repr__ ()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
```

```
class labgrid.strategy.shellstrategy.Status
```

```
Bases: enum.Enum
```

```
An enumeration.
```

```
unknown = 0
```

```
off = 1
```

```
shell = 2
```

```
__module__ = 'labgrid.strategy.shellstrategy'
```

```
__new__(value)
```

```
class labgrid.strategy.shellstrategy.ShellStrategy(target, name, sta-
                                         tus=<Status.unknown: 0>)
```

```
Bases: labgrid.strategy.common.Strategy
```

```
ShellStrategy - Strategy to switch to shell
```

```
bindings = {'power': <class 'labgrid.protocol.powerprotocol.PowerProtocol'>, 'shell':
```

```
__attrs_post_init__()
```

```
transition(status, *, step)
```

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True
```

```
__init__(target, name, status=<Status.unknown: 0>)
```

```
__module__ = 'labgrid.strategy.shellstrategy'
```

```
__repr__()
```

```
Automatically created by attrs.
```

labgrid.strategy.ubootstrategy module

```
exception labgrid.strategy.ubootstrategy.StrategyError(msg)
```

```
Bases: Exception
```

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
```

```
__init__(msg)
```

```
__module__ = 'labgrid.strategy.ubootstrategy'
```

```
__repr__()
```

```
Automatically created by attrs.
```

```
__weakref__
```

```
list of weak references to the object (if defined)
```

```
class labgrid.strategy.ubootstrategy.Status
```

```
Bases: enum.Enum
```

```
An enumeration.
```

```
unknown = 0
```

```
uboot = 1
```

```
shell = 2
```

```
__module__ = 'labgrid.strategy.ubootstrategy'
```

```
__new__(value)
```

```
class labgrid.strategy.ubootstrategy.UBootStrategy (target, name, status=<Status.unknown: 0>)
    Bases: labgrid.strategy.common.Strategy
    UbootStrategy - Strategy to switch to uboot or shell
    bindings = {'power': <class 'labgrid.protocol.powerprotocol.PowerProtocol'>, 'shell':
    __attrs_post_init__()
    transition (status)
    __attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True)
    __init__ (target, name, status=<Status.unknown: 0>)
    __module__ = 'labgrid.strategy.ubootstrategy'
    __repr__ ()
        Automatically created by attrs.
```

labgrid.util package

Submodules

labgrid.util.dict module

```
labgrid.util.dict.diff_dict (old, new)
    Compares old and new dictionaries, yielding for each difference (key, old_value, new_value). None is used for missing values.
labgrid.util.dict.flat_dict (d)
```

labgrid.util.exceptions module

```
exception labgrid.util.exceptions.NoValidDriverError (msg)
    Bases: Exception
    __attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
    __init__ (msg)
    __module__ = 'labgrid.util.exceptions'
    __repr__ ()
        Automatically created by attrs.
    __weakref__
        list of weak references to the object (if defined)
```

labgrid.util.expect module

```
class labgrid.util.expect.PtxExpect (driver, logfile=None, timeout=30, cwd=None)
    Bases: pexpect.py_spawn.spawn
    labgrid Wrapper of the pexpect module.
    This class provides pexpect functionality for the ConsoleProtocol classes. driver: ConsoleProtocol object to be passed in
```

```
__init__(driver, logfile=None, timeout=30, cwd=None)

send(s)
    Write to underlying transport, return number of bytes written

read_nonblocking(size=1, timeout=-1)
    Expects needs a nonblocking read function, simply use pyserial with a timeout of 0

__module__ = 'labgrid.util.expect'
```

labgrid.util.marker module

```
labgrid.util.marker.gen_marker()
```

labgrid.util.qmp module

```
class labgrid.util.qmp.QMPMonitor(monitor_out, monitor_in)
    Bases: object

    __attrs_post_init__()

    execute(command)

    __attrs_attrs__ = (Attribute(name='monitor_out', default=NOTHING, validator=None, repr=
    __dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'QMPMonitor' obje
    __init__(monitor_out, monitor_in)

    __module__ = 'labgrid.util.qmp'

    __repr__()
        Automatically created by attrs.

    __weakref__
        list of weak references to the object (if defined)

exception labgrid.util.qmp.QMPError(msg)
    Bases: Exception

    __attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
    __init__(msg)

    __module__ = 'labgrid.util.qmp'

    __repr__()
        Automatically created by attrs.

    __weakref__
        list of weak references to the object (if defined)
```

labgrid.util.timeout module

```
class labgrid.util.timeout.Timeout(timeout=120.0)
    Bases: object

    Represents a timeout (as a deadline)

    __attrs_post_init__()
```

```
remaining
expired
__attrs_attrs__ = (Attribute(name='timeout', default=120.0, validator=<instance_of val
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'Timeout' objects:
__init__ (timeout=120.0)
__module__ = 'labgrid.util.timeout'
__repr__ ()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
```

labgrid.util.yaml module

```
labgrid.util.yaml.load(stream)
labgrid.util.yaml.dump(data, stream=None)
labgrid.util.yaml.resolve_templates(data, mapping)
    Iterate recursively over data and call substitute(mapping) on all Templates.
```

9.1.2 Submodules

9.1.3 labgrid.binding module

```
exception labgrid.binding.StateError(msg)
    Bases: Exception
    __attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
    __init__ (msg)
    __module__ = 'labgrid.binding'
    __repr__ ()
        Automatically created by attrs.
    __weakref__
        list of weak references to the object (if defined)
exception labgrid.binding.BindingError(msg)
    Bases: Exception
    __attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance_of valid
    __init__ (msg)
    __module__ = 'labgrid.binding'
    __repr__ ()
        Automatically created by attrs.
    __weakref__
        list of weak references to the object (if defined)
```

```
class labgrid.binding.BindingState
```

```
Bases: enum.Enum
```

An enumeration.

```
error = -1
```

```
idle = 0
```

```
bound = 1
```

```
active = 2
```

```
__module__ = 'labgrid.binding'
```

```
__new__ (value)
```

```
class labgrid.binding.BindingMixin (target, name)
```

```
Bases: object
```

Handles the binding and activation of drivers and their supplying resources and drivers.

One client can be bound to many suppliers, and one supplier can be bound by many clients.

Conflicting access to one supplier can be avoided by deactivating conflicting clients before activation (using the `resolve_conflicts` callback).

```
bindings = {}
```

```
__attrs_post_init__ ()
```

```
display_name
```

```
on_supplier_bound (supplier, name)
```

Called by the Target after a new supplier has been bound

```
on_client_bound (client)
```

Called by the Target after a new client has been bound

```
on_activate ()
```

Called by the Target when this object has been activated

```
on_deactivate ()
```

Called by the Target when this object has been deactivated

```
resolve_conflicts (client)
```

Called by the Target to allow this object to deactivate conflicting clients.

```
classmethod check_active (func)
```

```
class NamedBinding (value)
```

```
Bases: object
```

Marks a binding (or binding set) as requiring an explicit name.

```
__init__ (value)
```

```
__repr__ ()
```

```
__dict__ = mappingproxy({'__dict__': <attribute '__dict__' of 'NamedBinding' object>})
```

```
__module__ = 'labgrid.binding'
```

```
__weakref__
```

list of weak references to the object (if defined)

```
__attrs_attrs__ = (Attribute(name='target', default=NOTHING, validator=None, repr=True, eq=True, order=0, hash=None, init=True),)
```

```
__dict__ = mappingproxy({'__attrs_attrs__': (Attribute(name='target', default=NOTHING
__init__(target, name)
__module__ = 'labgrid.binding'
__repr__()
    Automatically created by attrs.
__weakref__
    list of weak references to the object (if defined)
```

9.1.4 labgrid.config module

Config convenience class

This class encapsulates access functions to the environment configuration

```
class labgrid.config.Config(filename)
```

Bases: object

```
__attrs_post_init__()
```

```
resolve_path(path)
```

Resolve an absolute path

Parameters *path* (*str*) – path to resolve

Returns the absolute path

Return type str

```
get_tool(tool)
```

Retrieve an entry from the tools subkey

Parameters *tool* (*str*) – the tool to retrieve the path for

Returns path to the requested tools

Return type str

```
get_image_path(kind)
```

Retrieve an entry from the images subkey

Parameters *kind* (*str*) – the kind of the image to retrieve the path for

Returns path to the image

Return type str

Raises `KeyError` – if the requested image can not be found in the configuration

```
get_path(kind)
```

Retrieve an entry from the paths subkey

Parameters *kind* (*str*) – the type of path to retrieve the path for

Returns path to the path

Return type str

Raises `KeyError` – if the requested image can not be found in the configuration

```
get_option(name, default=None)
```

Retrieve an entry from the options subkey

Parameters

- **name** (*str*) – name of the option
- **default** (*str*) – A default parameter in case the option can not be found

Returns value of the option or default parameter

Return type str

Raises `KeyError` – if the requested image can not be found in the configuration

set_option (*name*, *value*)

Set an entry in the options subkey

Parameters

- **name** (*str*) – name of the option
- **value** (*str*) – the new value

get_targets ()

get_imports ()

Helper function that returns the list of all imports

Returns List of files which should be imported

Return type List

get_paths ()

Helper function that returns the subdict of all paths

Returns Dictionary containing all path definitions

Return type Dict

get_images ()

Helper function that returns the subdict of all images

Returns Dictionary containing all image definitions

Return type Dict

```

__attrs_attrs__ = (Attribute(name='filename', default=NOTHING, validator=<instance_of
__dict__ = mappingproxy({'get_targets': <function Config.get_targets>, '__init__': <
__init__ (filename)
__module__ = 'labgrid.config'
__repr__ ()
Automatically created by attrs.
__weakref__
list of weak references to the object (if defined)

```

9.1.5 labgrid.consoleloggingreporter module

class labgrid.consoleloggingreporter.ConsoleLoggingReporter (*logpath*)

Bases: object

ConsoleLoggingReporter - Reporter that writes console log files

Parameters **logpath** (*str*) – path to store the logfiles in

instance = None

```
classmethod start (path)
    starts the ConsoleLoggingReporter

classmethod stop ()
    stops the ConsoleLoggingReporter

__init__ (logpath)

get_logfile (event)
    Returns the correct file handle from cache or creates a new file handle

notify (event)
    This is the callback function for steps

__dict__ = mappingproxy({'_stop': <function ConsoleLoggingReporter._stop>, '__weakref__': None})

__module__ = 'labgrid.consoleloggingreporter'

__weakref__
    list of weak references to the object (if defined)
```

9.1.6 labgrid.environment module

```
class labgrid.environment.Environment (config_file='config.yaml', interact=<built-in function input>)
```

Bases: object

An environment encapsulates targets.

```
__attrs_post_init__ ()
```

```
get_target (role: str = 'main') → labgrid.target.Target
    Returns the specified target or None if not found.
```

Each target is initialized as needed.

```
cleanup ()
```

```
__attrs_attrs__ = (Attribute(name='config_file', default='config.yaml', validator=<instance of validator function at 0x...>),)
```

```
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'Environment' object>,})
```

```
__init__ (config_file='config.yaml', interact=<built-in function input>)
```

```
__module__ = 'labgrid.environment'
```

```
__repr__ ()
```

Automatically created by attrs.

```
__weakref__
```

list of weak references to the object (if defined)

9.1.7 labgrid.exceptions module

```
exception labgrid.exceptions.NoConfigFoundError (msg)
```

Bases: Exception

```
__attrs_attrs__ = (Attribute(name='msg', default=NOTHING, validator=<instance of validator function at 0x...>),)
```

```
__init__ (msg)
```

```
__module__ = 'labgrid.exceptions'
```

`__repr__()`
Automatically created by attrs.

`__weakref__`
list of weak references to the object (if defined)

exception `labgrid.exceptions.NoSupplierFoundError` (*msg, filter=None*)

Bases: `Exception`

`__attrs_attrs__` = (`Attribute`(name='msg', default=NOTHING, validator=<instance of valid

`__init__` (*msg, filter=None*)

`__module__` = 'labgrid.exceptions'

`__repr__()`
Automatically created by attrs.

`__weakref__`
list of weak references to the object (if defined)

exception `labgrid.exceptions.InvalidConfigError` (*msg*)

Bases: `Exception`

`__attrs_attrs__` = (`Attribute`(name='msg', default=NOTHING, validator=<instance of valid

`__init__` (*msg*)

`__module__` = 'labgrid.exceptions'

`__repr__()`
Automatically created by attrs.

`__weakref__`
list of weak references to the object (if defined)

exception `labgrid.exceptions.NoDriverFoundError` (*msg, filter=None*)

Bases: `labgrid.exceptions.NoSupplierFoundError`

`__attrs_attrs__` = (`Attribute`(name='msg', default=NOTHING, validator=<instance of valid

`__init__` (*msg, filter=None*)

`__module__` = 'labgrid.exceptions'

`__repr__()`
Automatically created by attrs.

exception `labgrid.exceptions.NoResourceFoundError` (*msg, filter=None*)

Bases: `labgrid.exceptions.NoSupplierFoundError`

`__attrs_attrs__` = (`Attribute`(name='msg', default=NOTHING, validator=<instance of valid

`__init__` (*msg, filter=None*)

`__module__` = 'labgrid.exceptions'

`__repr__()`
Automatically created by attrs.

9.1.8 labgrid.factory module

class `labgrid.factory.TargetFactory`

Bases: `object`

```
__init__()  
reg_resource (cls)  
    Register a resource with the factory.  
  
    Returns the class to allow using it as a decorator.  
reg_driver (cls)  
    Register a driver with the factory.  
  
    Returns the class to allow using it as a decorator.  
normalize_config (config)  
make_resource (target, resource, name, args)  
make_driver (target, driver, name, args)  
make_target (name, config, *, env=None)  
__dict__ = mappingproxy({'normalize_config': <function TargetFactory.normalize_config>})  
__module__ = 'labgrid.factory'  
__weakref__  
    list of weak references to the object (if defined)
```

```
labgrid.factory.target_factory = <labgrid.factory.TargetFactory object>  
Global TargetFactory instance
```

This instance is used to register Resource and Driver classes so that Targets can be created automatically from YAML files.

9.1.9 labgrid.step module

```
class labgrid.step.Steps  
    Bases: object
```

```
__init__()  
get_current()  
get_new (title, tag, source)  
push (step)  
pop (step)  
subscribe (callback)  
unsubscribe (callback)  
notify (event)  
__dict__ = mappingproxy({'__weakref__': <attribute '__weakref__' of 'Steps' objects>})  
__module__ = 'labgrid.step'  
__weakref__  
    list of weak references to the object (if defined)
```

```
class labgrid.step.StepEvent (step, data, *, resource=None, stream=False)  
    Bases: object
```

```
__init__ (step, data, *, resource=None, stream=False)  
__str__()
```

merge (*other*)

age

__dict__ = `mappingproxy({'__dict__': <attribute '__dict__' of 'StepEvent' objects>, '...`

__module__ = 'labgrid.step'

__weakref__

list of weak references to the object (if defined)

class labgrid.step.**Step** (*title, level, tag, source*)

Bases: object

__init__ (*title, level, tag, source*)

__repr__ ()

__str__ ()

duration

status

is_active

is_done

start ()

skip (*reason*)

stop ()

__del__ ()

__dict__ = `mappingproxy({'__dict__': <attribute '__dict__' of 'Step' objects>, '...__del...`

__module__ = 'labgrid.step'

__weakref__

list of weak references to the object (if defined)

labgrid.step.**step** (*, *title=None, args=[], result=False, tag=None*)

9.1.10 labgrid.stepreporter module

class labgrid.stepreporter.**StepReporter**

Bases: object

instance = None

classmethod **start** ()

starts the StepReporter

classmethod **stop** ()

stops the StepReporter

__init__ ()

notify (*event*)

__dict__ = `mappingproxy({'__dict__': <attribute '__dict__' of 'StepReporter' objects>...`

__module__ = 'labgrid.stepreporter'

__weakref__
list of weak references to the object (if defined)

9.1.11 labgrid.target module

class labgrid.target.**Target** (*name, env=None*)
Bases: object

__attrs_attrs__ = (Attribute(name='name', default=NOTHING, validator=<instance_of vali

__attrs_post_init__ ()

__dict__ = mappingproxy({'__attrs_attrs__': (Attribute(name='name', default=NOTHING, v

__getitem__ (*key*)
Syntactic sugar to access drivers by class (optionally filtered by name).

```
>>> target = Target('main')
>>> console = FakeConsoleDriver(target, 'console')
>>> target.activate(console)
>>> target[FakeConsoleDriver]
FakeConsoleDriver(target=Target(name='main', ...), name='console', ...)
>>> target[FakeConsoleDriver, 'console']
FakeConsoleDriver(target=Target(name='main', ...), name='console', ...)
```

__init__ (*name, env=None*)

__module__ = 'labgrid.target'

__repr__ ()
Automatically created by attrs.

__weakref__
list of weak references to the object (if defined)

activate (*client*)
Activate the client by activating all bound suppliers. This may require deactivating other clients.

await_resources (*resources, timeout=None, avail=True*)
Poll the given resources and wait until they are (un-)available.

Parameters

- **resources** (*List*) – the resources to poll
- **timeout** (*float*) – optional timeout
- **avail** (*bool*) – optionally wait until the resources are unavailable with `avail=False`

bind (*bindable*)

bind_driver (*client*)
Bind the driver to all suppliers (resources and other drivers).

Currently, we only support binding all suppliers at once.

bind_resource (*resource*)
Bind the resource to this target.

cleanup ()
Clean up connected drivers and resources in reversed order

deactivate (*client*)

Recursively deactivate the client's clients and itself.

This is needed to ensure that no client has an inactive supplier.

get_active_driver (*cls, *, name=None*)

Helper function to get the active driver of the target. Returns the active driver found, otherwise None.

Arguments: *cls* – driver-class to return as a resource name – optional name to use as a filter

get_driver (*cls, *, name=None, activate=True*)

Helper function to get a driver of the target. Returns the first valid driver found, otherwise None.

Arguments: *cls* – driver-class to return as a resource name – optional name to use as a filter *activate* – activate the driver (default True)

get_resource (*cls, *, name=None, await=True*)

Helper function to get a resource of the target. Returns the first valid resource found, otherwise None.

Arguments: *cls* – resource-class to return as a resource name – optional name to use as a filter *await* – wait for the resource to become available (default True)

interact (*msg*)

set_binding_map (*mapping*)

Configure the binding name mapping for the next driver only.

update_resources ()

Iterate over all relevant managers and deactivate any active but unavailable resources.

CHAPTER 10

Indices and Tables

- genindex
- modindex
- search

|

- labgrid, 69
- labgrid.autoinstall, 69
- labgrid.autoinstall.main, 69
- labgrid.binding, 122
- labgrid.config, 124
- labgrid.consoleloggingreporter, 125
- labgrid.driver, 70
 - labgrid.driver.bareboxdriver, 71
 - labgrid.driver.commandmixin, 72
 - labgrid.driver.common, 72
 - labgrid.driver.consoleexpectmixin, 73
 - labgrid.driver.exception, 73
 - labgrid.driver.externalconsoledriver, 74
 - labgrid.driver.fake, 74
 - labgrid.driver.fastbootdriver, 75
 - labgrid.driver.infodriver, 76
 - labgrid.driver.modbusdriver, 76
 - labgrid.driver.onewiredriver, 77
 - labgrid.driver.openocddriver, 77
 - labgrid.driver.power, 70
 - labgrid.driver.power.apc, 70
 - labgrid.driver.power.digipower, 70
 - labgrid.driver.power.gude, 70
 - labgrid.driver.power.gude24, 70
 - labgrid.driver.power.netio, 70
 - labgrid.driver.power.netio_kshell, 70
 - labgrid.driver.powerdriver, 78
 - labgrid.driver.qemudriver, 80
 - labgrid.driver.quartushpsdriver, 82
 - labgrid.driver.serialdriver, 82
 - labgrid.driver.shelldriver, 82
 - labgrid.driver.sigrokdriver, 85
 - labgrid.driver.smallubootdriver, 85
 - labgrid.driver.sshdriver, 86
 - labgrid.driver.ubootdriver, 87
 - labgrid.driver.usbloader, 88
 - labgrid.driver.usbstorage, 89
 - labgrid.environment, 126
 - labgrid.exceptions, 126
 - labgrid.external, 89
 - labgrid.external.hawkbit, 89
 - labgrid.external.usbstick, 91
 - labgrid.factory, 127
 - labgrid.protocol, 92
 - labgrid.protocol.bootstrapprotocol, 92
 - labgrid.protocol.commandprotocol, 92
 - labgrid.protocol.consoleprotocol, 93
 - labgrid.protocol.digitaloutputprotocol, 93
 - labgrid.protocol.filesystemprotocol, 94
 - labgrid.protocol.filetransferprotocol, 94
 - labgrid.protocol.infoprotocol, 94
 - labgrid.protocol.linuxbootprotocol, 94
 - labgrid.protocol.mmioprotocol, 95
 - labgrid.protocol.powerprotocol, 95
 - labgrid.provider, 95
 - labgrid.provider.fileprovider, 95
 - labgrid.provider.mediafileprovider, 95
 - labgrid.pytestplugin, 96
 - labgrid.pytestplugin.fixtures, 96
 - labgrid.pytestplugin.reporter, 96
 - labgrid.remote, 97
 - labgrid.remote.authenticator, 97
 - labgrid.remote.client, 97
 - labgrid.remote.common, 97
 - labgrid.remote.config, 99
 - labgrid.remote.coordinator, 99
 - labgrid.remote.exporter, 100
 - labgrid.resource, 103
 - labgrid.resource.base, 103
 - labgrid.resource.common, 104
 - labgrid.resource.ethernetport, 106
 - labgrid.resource.modbus, 108
 - labgrid.resource.networkservice, 108
 - labgrid.resource.onewireport, 109
 - labgrid.resource.power, 109

labgrid.resource.remote, 109
labgrid.resource.serialport, 112
labgrid.resource.sigrok, 113
labgrid.resource.udev, 113
labgrid.resource.ykushpowerport, 117
labgrid.step, 128
labgrid.stepreporter, 129
labgrid.strategy, 117
labgrid.strategy.bareboxstrategy, 117
labgrid.strategy.common, 118
labgrid.strategy.shellstrategy, 118
labgrid.strategy.ubootstrategy, 119
labgrid.target, 130
labgrid.util, 120
labgrid.util.dict, 120
labgrid.util.exceptions, 120
labgrid.util.expect, 120
labgrid.util.marker, 121
labgrid.util.qmp, 121
labgrid.util.timeout, 121
labgrid.util.yaml, 122

Symbols

<code>__abstractmethods__</code>	(lab- grid.driver.bareboxdriver.BareboxDriver attribute), 71	<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.NetworkPowerDriver attribute), 79
<code>__abstractmethods__</code>	(lab- grid.driver.externalconsoledriver.ExternalConsoleDriver attribute), 74	<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.PowerResetMixin attribute), 78
<code>__abstractmethods__</code>	(lab- grid.driver.fake.FakeCommandDriver attribute), 75	<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.USBPowerDriver attribute), 80
<code>__abstractmethods__</code>	(lab- grid.driver.fake.FakeConsoleDriver attribute), 74	<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.YKUSHPowerDriver attribute), 80
<code>__abstractmethods__</code>	(lab- grid.driver.fake.FakeFileTransferDriver attribute), 75	<code>__abstractmethods__</code>	(lab- grid.driver.qemudriver.QEMUDriver attribute), 81
<code>__abstractmethods__</code>	(lab- grid.driver.fake.FakePowerDriver attribute), 75	<code>__abstractmethods__</code>	(lab- grid.driver.serialdriver.SerialDriver attribute), 82
<code>__abstractmethods__</code>	(lab- grid.driver.infodriver.InfoDriver attribute), 76	<code>__abstractmethods__</code>	(lab- grid.driver.shelldriver.ShellDriver attribute), 84
<code>__abstractmethods__</code>	(lab- grid.driver.modbusdriver.ModbusCoilDriver attribute), 77	<code>__abstractmethods__</code>	(lab- grid.driver.smallubootdriver.SmallUBootDriver attribute), 86
<code>__abstractmethods__</code>	(lab- grid.driver.onewiredriver.OneWirePIODriver attribute), 77	<code>__abstractmethods__</code>	(lab- grid.driver.sshdriver.SSHDriver attribute), 87
<code>__abstractmethods__</code>	(lab- grid.driver.openocddriver.OpenOCDDriver attribute), 77	<code>__abstractmethods__</code>	(lab- grid.driver.ubootdriver.UBootDriver attribute), 88
<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.DigitalOutputPowerDriver attribute), 79	<code>__abstractmethods__</code>	(lab- grid.driver.usbloader.IMXUSBDriver attribute), 89
<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.ExternalPowerDriver attribute), 78	<code>__abstractmethods__</code>	(lab- grid.driver.usbloader.MXSUSBDriver attribute), 88
<code>__abstractmethods__</code>	(lab- grid.driver.powerdriver.ManualPowerDriver attribute), 78	<code>__abstractmethods__</code>	(lab- grid.protocol.bootstrapprotocol.BootstrapProtocol attribute), 92
		<code>__abstractmethods__</code>	(lab- attribute), 78

grid.protocol.commandprotocol.CommandProtocol attribute), 74
 attribute), 93
 __abstractmethods__ (lab-
 grid.protocol.consoleprotocol.ConsoleProtocol __attrs_attrs__ (labgrid.driver.fake.FakeFileTransferDriver
 attribute), 93 attribute), 75
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.fake.FakePowerDriver at-
 grid.protocol.consoleprotocol.ConsoleProtocol.Client attribute), 76
 attribute), 93 __attrs_attrs__ (labgrid.driver.fastbootdriver.AndroidFastbootDriver
 __abstractmethods__ (lab- attribute), 76
 grid.protocol.digitaloutputprotocol.DigitalOutputProtocol __attrs_attrs__ (labgrid.driver.infodriver.InfoDriver at-
 attribute), 93 attribute), 77
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.modbusdriver.ModbusCoilDriver
 attribute), 93 attribute), 77
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.onewiredriver.OneWirePIODriver
 grid.protocol.filesystemprotocol.FileSystemProtocol attribute), 77
 attribute), 94 __attrs_attrs__ (labgrid.driver.openocddriver.OpenOCDDriver
 __abstractmethods__ (lab- attribute), 77
 grid.protocol.filetransferprotocol.FileTransferProtocol __attrs_attrs__ (labgrid.driver.powerdriver.DigitalOutputPowerDriver
 attribute), 94 attribute), 79
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.powerdriver.ExternalPowerDriver
 grid.protocol.infoprotocol.InfoProtocol at- attribute), 78
 tribute), 94 __attrs_attrs__ (labgrid.driver.powerdriver.ManualPowerDriver
 __abstractmethods__ (lab- attribute), 78
 grid.protocol.linuxbootprotocol.LinuxBootProtocol __attrs_attrs__ (labgrid.driver.powerdriver.NetworkPowerDriver
 attribute), 94 attribute), 79
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.powerdriver.PowerResetMixin
 grid.protocol.mmioprotocol.MMIOProtocol attribute), 78
 attribute), 95 __attrs_attrs__ (labgrid.driver.powerdriver.USBPowerDriver
 __abstractmethods__ (lab- attribute), 80
 grid.protocol.powerprotocol.PowerProtocol __attrs_attrs__ (labgrid.driver.powerdriver.YKUSHPowerDriver
 attribute), 95 attribute), 80
 __abstractmethods__ (lab- __attrs_attrs__ (labgrid.driver.qemudriver.QEMUDriver
 grid.provider.fileprovider.FileProvider at- attribute), 81
 tribute), 95 __attrs_attrs__ (labgrid.driver.quartushpsdriver.QuartusHPSDriver
 __abstractmethods__ (lab- attribute), 82
 grid.provider.mediafileprovider.MediaFileProvider __attrs_attrs__ (labgrid.driver.serialdriver.SerialDriver at-
 attribute), 95 attribute), 82
 __attrs_attrs__ (labgrid.binding.BindingError attribute), __attrs_attrs__ (labgrid.driver.shelldriver.ShellDriver at-
 122 attribute), 84
 __attrs_attrs__ (labgrid.binding.BindingMixin attribute), __attrs_attrs__ (labgrid.driver.sigrokdriver.SigrokDriver
 123 attribute), 85
 __attrs_attrs__ (labgrid.binding.StateError attribute), 122 __attrs_attrs__ (labgrid.driver.smallbootdriver.SmallUBootDriver
 __attrs_attrs__ (labgrid.config.Config attribute), 125 attribute), 86
 __attrs_attrs__ (labgrid.driver.bareboxdriver.BareboxDriver __attrs_attrs__ (labgrid.driver.sshdriver.SSHDriver
 attribute), 71 attribute), 87
 __attrs_attrs__ (labgrid.driver.common.Driver attribute), __attrs_attrs__ (labgrid.driver.ubootdriver.UBootDriver
 72 attribute), 88
 __attrs_attrs__ (labgrid.driver.exception.CleanUpError __attrs_attrs__ (labgrid.driver.usbloader.IMXUSBDriver
 attribute), 73 attribute), 89
 __attrs_attrs__ (labgrid.driver.exception.ExecutionError __attrs_attrs__ (labgrid.driver.usbloader.MXSUSBDriver
 attribute), 73 attribute), 88
 __attrs_attrs__ (labgrid.driver.externalconsoledriver.ExternalConsoleDriver __attrs_attrs__ (labgrid.driver.usbstorage.USBStorageDriver
 attribute), 74 attribute), 89
 __attrs_attrs__ (labgrid.driver.fake.FakeCommandDriver __attrs_attrs__ (labgrid.environment.Environment at-
 attribute), 75 attribute), 126
 __attrs_attrs__ (labgrid.driver.fake.FakeConsoleDriver __attrs_attrs__ (labgrid.exceptions.InvalidConfigError at-

- tribute), 127
- __attrs_attrs__ (labgrid.exceptions.NoConfigFoundError attribute), 126
- __attrs_attrs__ (labgrid.exceptions.NoDriverFoundError attribute), 127
- __attrs_attrs__ (labgrid.exceptions.NoResourceFoundError attribute), 127
- __attrs_attrs__ (labgrid.exceptions.NoSupplierFoundError attribute), 127
- __attrs_attrs__ (labgrid.external.hawkbite.HawkbitError attribute), 90
- __attrs_attrs__ (labgrid.external.hawkbite.HawkbitTestClient attribute), 90
- __attrs_attrs__ (labgrid.external.usbstick.StateError attribute), 92
- __attrs_attrs__ (labgrid.external.usbstick.USBStick attribute), 91
- __attrs_attrs__ (labgrid.provider.mediafileprovider.MediaFileProvider attribute), 95
- __attrs_attrs__ (labgrid.remote.common.Place attribute), 99
- __attrs_attrs__ (labgrid.remote.common.ResourceEntry attribute), 97
- __attrs_attrs__ (labgrid.remote.common.ResourceMatch attribute), 98
- __attrs_attrs__ (labgrid.remote.config.ResourceConfig attribute), 99
- __attrs_attrs__ (labgrid.remote.coordinator.ClientSession attribute), 100
- __attrs_attrs__ (labgrid.remote.coordinator.ExporterSession attribute), 100
- __attrs_attrs__ (labgrid.remote.coordinator.RemoteSession attribute), 100
- __attrs_attrs__ (labgrid.remote.exporter.EthernetPortExport attribute), 102
- __attrs_attrs__ (labgrid.remote.exporter.ResourceExport attribute), 101
- __attrs_attrs__ (labgrid.remote.exporter.USBEthernetExport attribute), 101
- __attrs_attrs__ (labgrid.remote.exporter.USBGenericExport attribute), 101
- __attrs_attrs__ (labgrid.remote.exporter.USBPowerPortExport attribute), 102
- __attrs_attrs__ (labgrid.remote.exporter.USBSDMuxExport attribute), 102
- __attrs_attrs__ (labgrid.remote.exporter.USBSerialPortExport attribute), 101
- __attrs_attrs__ (labgrid.remote.exporter.USBSigrokExport attribute), 102
- __attrs_attrs__ (labgrid.resource.base.EthernetInterface attribute), 103
- __attrs_attrs__ (labgrid.resource.base.EthernetPort attribute), 104
- __attrs_attrs__ (labgrid.resource.base.SerialPort attribute), 103
- __attrs_attrs__ (labgrid.resource.common.ManagedResource attribute), 106
- __attrs_attrs__ (labgrid.resource.common.NetworkResource attribute), 105
- __attrs_attrs__ (labgrid.resource.common.Resource attribute), 105
- __attrs_attrs__ (labgrid.resource.common.ResourceManager attribute), 105
- __attrs_attrs__ (labgrid.resource.ethernetport.EthernetPortManager attribute), 107
- __attrs_attrs__ (labgrid.resource.ethernetport.SNMPEthernetPort attribute), 107
- __attrs_attrs__ (labgrid.resource.ethernetport.SNMPSwitch attribute), 106
- __attrs_attrs__ (labgrid.resource.modbus.ModbusTCPCoil attribute), 108
- __attrs_attrs__ (labgrid.resource.networkservice.NetworkService attribute), 108
- __attrs_attrs__ (labgrid.resource.onewirereport.OneWirePIO attribute), 109
- __attrs_attrs__ (labgrid.resource.power.NetworkPowerPort attribute), 109
- __attrs_attrs__ (labgrid.resource.remote.NetworkAlteraUSBBlaster attribute), 111
- __attrs_attrs__ (labgrid.resource.remote.NetworkAndroidFastboot attribute), 110
- __attrs_attrs__ (labgrid.resource.remote.NetworkIMXUSBLoader attribute), 110
- __attrs_attrs__ (labgrid.resource.remote.NetworkMXSUSBLoader attribute), 111
- __attrs_attrs__ (labgrid.resource.remote.NetworkSigrokUSBDevice attribute), 111
- __attrs_attrs__ (labgrid.resource.remote.NetworkUSBMassStorage attribute), 111
- __attrs_attrs__ (labgrid.resource.remote.NetworkUSBPowerPort attribute), 112
- __attrs_attrs__ (labgrid.resource.remote.NetworkUSBSDMuxDevice attribute), 112
- __attrs_attrs__ (labgrid.resource.remote.RemotePlace attribute), 110
- __attrs_attrs__ (labgrid.resource.remote.RemotePlaceManager attribute), 109
- __attrs_attrs__ (labgrid.resource.remote.RemoteUSBResource attribute), 110
- __attrs_attrs__ (labgrid.resource.serialport.NetworkSerialPort attribute), 113
- __attrs_attrs__ (labgrid.resource.serialport.RawSerialPort attribute), 112
- __attrs_attrs__ (labgrid.resource.sigrok.SigrokDevice attribute), 113
- __attrs_attrs__ (labgrid.resource.udev.AlteraUSBBlaster attribute), 115
- __attrs_attrs__ (labgrid.resource.udev.AndroidFastboot attribute), 115

attribute), 115	method), 72
__attrs_attrs__ (labgrid.resource.udev.IMXUSBLoader attribute), 114	__attrs_post_init__() (labgrid.driver.common.Driver method), 72
__attrs_attrs__ (labgrid.resource.udev.MXSUSBLoader attribute), 115	__attrs_post_init__() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
__attrs_attrs__ (labgrid.resource.udev.SigrokUSBDevice attribute), 116	__attrs_post_init__() (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
__attrs_attrs__ (labgrid.resource.udev.USBEthernetInterface attribute), 115	__attrs_post_init__() (labgrid.driver.fake.FakeConsoleDriver method), 74
__attrs_attrs__ (labgrid.resource.udev.USBMassStorage attribute), 114	__attrs_post_init__() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
__attrs_attrs__ (labgrid.resource.udev.USBPowerPort attribute), 116	__attrs_post_init__() (labgrid.driver.infodriver.InfoDriver method), 76
__attrs_attrs__ (labgrid.resource.udev.USBResource attribute), 114	__attrs_post_init__() (labgrid.driver.modbusdriver.ModbusCoilDriver method), 76
__attrs_attrs__ (labgrid.resource.udev.USBSDMuxDevice attribute), 116	__attrs_post_init__() (labgrid.driver.onewiredriver.OneWirePIODriver method), 77
__attrs_attrs__ (labgrid.resource.udev.USBSerialPort attribute), 114	__attrs_post_init__() (labgrid.driver.openocddriver.OpenOCDDriver method), 77
__attrs_attrs__ (labgrid.resource.udev.UdevManager attribute), 113	__attrs_post_init__() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
__attrs_attrs__ (labgrid.resource.ykushpowerport.YKUSHPowerPort attribute), 117	__attrs_post_init__() (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
__attrs_attrs__ (labgrid.strategy.bareboxstrategy.BareboxStrategy attribute), 118	__attrs_post_init__() (labgrid.driver.powerdriver.PowerResetMixin method), 78
__attrs_attrs__ (labgrid.strategy.bareboxstrategy.StrategyError attribute), 117	__attrs_post_init__() (labgrid.driver.powerdriver.USBPowerDriver method), 80
__attrs_attrs__ (labgrid.strategy.common.Strategy attribute), 118	__attrs_post_init__() (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
__attrs_attrs__ (labgrid.strategy.shellstrategy.ShellStrategy attribute), 119	__attrs_post_init__() (labgrid.driver.qemudriver.QEMUDriver method), 81
__attrs_attrs__ (labgrid.strategy.shellstrategy.StrategyError attribute), 118	__attrs_post_init__() (labgrid.driver.quartushpsdriver.QuartusHPSDriver method), 82
__attrs_attrs__ (labgrid.strategy.ubootstrategy.StrategyError attribute), 119	__attrs_post_init__() (labgrid.driver.serialdriver.SerialDriver method), 82
__attrs_attrs__ (labgrid.strategy.ubootstrategy.UBootStrategy attribute), 120	__attrs_post_init__() (labgrid.driver.shelldriver.ShellDriver method), 83
__attrs_attrs__ (labgrid.target.Target attribute), 130	__attrs_post_init__() (labgrid.driver.commandmixin.CommandMixin method), 71
__attrs_attrs__ (labgrid.util.exceptions.NoValidDriverError attribute), 120	
__attrs_attrs__ (labgrid.util.qmp.QMPError attribute), 121	
__attrs_attrs__ (labgrid.util.qmp.QMPMonitor attribute), 121	
__attrs_attrs__ (labgrid.util.timeout.Timeout attribute), 122	
__attrs_post_init__() (labgrid.binding.BindingMixin method), 123	
__attrs_post_init__() (labgrid.config.Config method), 124	
__attrs_post_init__() (labgrid.driver.bareboxdriver.BareboxDriver method), 71	
__attrs_post_init__() (labgrid.driver.commandmixin.CommandMixin method), 71	

grid.driver.sigrokdriver.SigrokDriver method), 85	grid.resource.common.ManagedResource method), 106
__attrs_post_init__() (labgrid.driver.sshdriver.SSHDriver method), 86	__attrs_post_init__() (labgrid.resource.common.Resource method), 104
__attrs_post_init__() (labgrid.driver.ubootdriver.UBootDriver method), 87	__attrs_post_init__() (labgrid.resource.common.ResourceManager method), 105
__attrs_post_init__() (labgrid.driver.usbloader.IMXUSBDriver method), 89	__attrs_post_init__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
__attrs_post_init__() (labgrid.driver.usbloader.MXSUSBDriver method), 88	__attrs_post_init__() (labgrid.resource.ethernetport.SNMPEthernetPort method), 107
__attrs_post_init__() (labgrid.driver.usbstorage.USBStorageDriver method), 89	__attrs_post_init__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
__attrs_post_init__() (labgrid.environment.Environment method), 126	__attrs_post_init__() (labgrid.resource.remote.NetworkAlteraUSBBlaster method), 111
__attrs_post_init__() (labgrid.external.hawkbite.HawkbiteTestClient method), 89	__attrs_post_init__() (labgrid.resource.remote.NetworkAndroidFastboot method), 110
__attrs_post_init__() (labgrid.external.usbstick.USBStick method), 91	__attrs_post_init__() (labgrid.resource.remote.NetworkIMXUSBLoader method), 110
__attrs_post_init__() (labgrid.remote.common.ResourceEntry method), 97	__attrs_post_init__() (labgrid.resource.remote.NetworkMXSUSBLoader method), 111
__attrs_post_init__() (labgrid.remote.config.ResourceConfig method), 99	__attrs_post_init__() (labgrid.resource.remote.NetworkSigrokUSBDevice method), 111
__attrs_post_init__() (labgrid.remote.exporter.EthernetPortExport method), 102	__attrs_post_init__() (labgrid.resource.remote.NetworkUSBMassStorage method), 111
__attrs_post_init__() (labgrid.remote.exporter.ResourceExport method), 101	__attrs_post_init__() (labgrid.resource.remote.NetworkUSBPowerPort method), 112
__attrs_post_init__() (labgrid.remote.exporter.USBEthernetExport method), 101	__attrs_post_init__() (labgrid.resource.remote.NetworkUSBSDMuxDevice method), 112
__attrs_post_init__() (labgrid.remote.exporter.USBGenericExport method), 101	__attrs_post_init__() (labgrid.resource.remote.RemotePlace method), 110
__attrs_post_init__() (labgrid.remote.exporter.USBPowerPortExport method), 102	__attrs_post_init__() (labgrid.resource.remote.RemotePlaceManager method), 109
__attrs_post_init__() (labgrid.remote.exporter.USBSDMuxExport method), 102	__attrs_post_init__() (labgrid.resource.serialport.RawSerialPort method), 112
__attrs_post_init__() (labgrid.remote.exporter.USBSerialPortExport method), 101	__attrs_post_init__() (labgrid.resource.udev.SigrokUSBDevice method), 116
__attrs_post_init__() (labgrid.remote.exporter.USBSigrokExport method), 102	__attrs_post_init__() (lab-
__attrs_post_init__() (lab-	

grid.resource.udev.USBEthernetInterface method), 115
 __attrs_post_init__() (labgrid.resource.udev.USBMassStorage method), 114
 __attrs_post_init__() (labgrid.resource.udev.USBPowerPort method), 116
 __attrs_post_init__() (labgrid.resource.udev.USBResource method), 113
 __attrs_post_init__() (labgrid.resource.udev.USBSDMuxDevice method), 116
 __attrs_post_init__() (labgrid.resource.udev.USBSerialPort method), 114
 __attrs_post_init__() (labgrid.resource.udev.UdevManager method), 113
 __attrs_post_init__() (labgrid.strategy.bareboxstrategy.BareboxStrategy method), 118
 __attrs_post_init__() (labgrid.strategy.common.Strategy method), 118
 __attrs_post_init__() (labgrid.strategy.shellstrategy.ShellStrategy method), 119
 __attrs_post_init__() (labgrid.strategy.ubootstrategy.UBootStrategy method), 120
 __attrs_post_init__() (labgrid.target.Target method), 130
 __attrs_post_init__() (labgrid.util.qmp.QMPMonitor method), 121
 __attrs_post_init__() (labgrid.util.timeout.Timeout method), 121
 __call__() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
 __del__() (labgrid.remote.exporter.USBSerialPortExport method), 101
 __del__() (labgrid.step.Step method), 129
 __dict__ (labgrid.autoinstall.main.Manager attribute), 69
 __dict__ (labgrid.binding.BindingMixin attribute), 123
 __dict__ (labgrid.binding.BindingMixin.NamedBinding attribute), 123
 __dict__ (labgrid.config.Config attribute), 125
 __dict__ (labgrid.consoleloggingreporter.ConsoleLoggingReporter attribute), 126
 __dict__ (labgrid.driver.commandmixin.CommandMixin attribute), 72
 __dict__ (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin attribute), 73
 __dict__ (labgrid.environment.Environment attribute), 126
 __dict__ (labgrid.external.hawkbit.HawkbitTestClient attribute), 90
 __dict__ (labgrid.external.usbstick.USBStick attribute), 91
 __dict__ (labgrid.factory.TargetFactory attribute), 128
 __dict__ (labgrid.pytestplugin.reporter.StepReporter attribute), 96
 __dict__ (labgrid.remote.common.Place attribute), 99
 __dict__ (labgrid.remote.common.ResourceEntry attribute), 97
 __dict__ (labgrid.remote.common.ResourceMatch attribute), 98
 __dict__ (labgrid.remote.config.ResourceConfig attribute), 99
 __dict__ (labgrid.remote.coordinator.RemoteSession attribute), 100
 __dict__ (labgrid.resource.common.ResourceManager attribute), 105
 __dict__ (labgrid.resource.ethernetport.SNMPSwitch attribute), 106
 __dict__ (labgrid.step.Step attribute), 129
 __dict__ (labgrid.step.StepEvent attribute), 129
 __dict__ (labgrid.step.Steps attribute), 128
 __dict__ (labgrid.stepreporter.StepReporter attribute), 129
 __dict__ (labgrid.target.Target attribute), 130
 __dict__ (labgrid.util.qmp.QMPMonitor attribute), 121
 __dict__ (labgrid.util.timeout.Timeout attribute), 122
 __eq__() (labgrid.remote.common.ResourceMatch method), 98
 __eq__() (labgrid.remote.exporter.EthernetPortExport method), 102
 __eq__() (labgrid.resource.base.EthernetPort method), 104
 __eq__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
 __eq__() (labgrid.resource.ethernetport.SNMPEthernetPort method), 107
 __eq__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
 __ge__() (labgrid.remote.common.ResourceMatch method), 98
 __ge__() (labgrid.remote.exporter.EthernetPortExport method), 102
 __ge__() (labgrid.resource.base.EthernetPort method), 104
 __ge__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
 __ge__() (labgrid.resource.ethernetport.SNMPEthernetPort method), 107
 __ge__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
 __getitem__() (labgrid.target.Target method), 130
 __gt__() (labgrid.remote.common.ResourceMatch

- method), 98
- `__gt__()` (labgrid.remote.exporter.EthernetPortExport method), 103
- `__gt__()` (labgrid.resource.base.EthernetPort method), 104
- `__gt__()` (labgrid.resource.ethernetport.EthernetPortManager method), 107
- `__gt__()` (labgrid.resource.ethernetport.SNMPEthernetPort method), 108
- `__gt__()` (labgrid.resource.ethernetport.SNMPSwitch method), 106
- `__hash__` (labgrid.remote.common.ResourceMatch attribute), 98
- `__hash__` (labgrid.remote.exporter.EthernetPortExport attribute), 103
- `__hash__` (labgrid.resource.base.EthernetPort attribute), 104
- `__hash__` (labgrid.resource.ethernetport.EthernetPortManager attribute), 107
- `__hash__` (labgrid.resource.ethernetport.SNMPEthernetPort attribute), 108
- `__hash__` (labgrid.resource.ethernetport.SNMPSwitch attribute), 106
- `__init__()` (labgrid.autoinstall.main.Handler method), 69
- `__init__()` (labgrid.autoinstall.main.Manager method), 69
- `__init__()` (labgrid.binding.BindingError method), 122
- `__init__()` (labgrid.binding.BindingMixin method), 124
- `__init__()` (labgrid.binding.BindingMixin.NamedBinding method), 123
- `__init__()` (labgrid.binding.StateError method), 122
- `__init__()` (labgrid.config.Config method), 125
- `__init__()` (labgrid.consoleloggingreporter.ConsoleLoggingReporter method), 126
- `__init__()` (labgrid.driver.bareboxdriver.BareboxDriver method), 71
- `__init__()` (labgrid.driver.common.Driver method), 72
- `__init__()` (labgrid.driver.exception.CleanUpError method), 73
- `__init__()` (labgrid.driver.exception.ExecutionError method), 73
- `__init__()` (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
- `__init__()` (labgrid.driver.fake.FakeCommandDriver method), 75
- `__init__()` (labgrid.driver.fake.FakeConsoleDriver method), 74
- `__init__()` (labgrid.driver.fake.FakeFileTransferDriver method), 75
- `__init__()` (labgrid.driver.fake.FakePowerDriver method), 75
- `__init__()` (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
- `__init__()` (labgrid.driver.infodriver.InfoDriver method), 76
- `__init__()` (labgrid.driver.modbusdriver.ModbusCoilDriver method), 77
- `__init__()` (labgrid.driver.onewiredriver.OneWirePIODriver method), 77
- `__init__()` (labgrid.driver.openocddriver.OpenOCDDriver method), 77
- `__init__()` (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
- `__init__()` (labgrid.driver.powerdriver.ExternalPowerDriver method), 78
- `__init__()` (labgrid.driver.powerdriver.ManualPowerDriver method), 78
- `__init__()` (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
- `__init__()` (labgrid.driver.powerdriver.PowerResetMixin method), 78
- `__init__()` (labgrid.driver.powerdriver.USBPowerDriver method), 80
- `__init__()` (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
- `__init__()` (labgrid.driver.qemudriver.QEMUDriver method), 81
- `__init__()` (labgrid.driver.quartushpsdriver.QuartusHPSDriver method), 82
- `__init__()` (labgrid.driver.serialdriver.SerialDriver method), 82
- `__init__()` (labgrid.driver.shelldriver.ShellDriver method), 85
- `__init__()` (labgrid.driver.sigrokdriver.SigrokDriver method), 85
- `__init__()` (labgrid.driver.smallubootdriver.SmallUBootDriver method), 86
- `__init__()` (labgrid.driver.sshdriver.SSHDriver method), 87
- `__init__()` (labgrid.driver.ubootdriver.UBootDriver method), 88
- `__init__()` (labgrid.driver.usbloader.IMXUSBDriver method), 89
- `__init__()` (labgrid.driver.usbloader.MXSUSBDriver method), 88
- `__init__()` (labgrid.driver.usbstorage.USBStorageDriver method), 89
- `__init__()` (labgrid.environment.Environment method), 126
- `__init__()` (labgrid.exceptions.InvalidConfigError method), 127
- `__init__()` (labgrid.exceptions.NoConfigFoundError method), 126
- `__init__()` (labgrid.exceptions.NoDriverFoundError method), 127
- `__init__()` (labgrid.exceptions.NoResourceFoundError method), 127
- `__init__()` (labgrid.exceptions.NoSupplierFoundError method), 127

- `__init__()` (labgrid.external.hawkbit.HawkbiError method), 90
- `__init__()` (labgrid.external.hawkbit.HawkbitTestClient method), 90
- `__init__()` (labgrid.external.usbstick.StateError method), 92
- `__init__()` (labgrid.external.usbstick.USBStick method), 92
- `__init__()` (labgrid.factory.TargetFactory method), 127
- `__init__()` (labgrid.provider.mediafileprovider.MediaFileProvider method), 96
- `__init__()` (labgrid.pytestplugin.reporter.StepReporter method), 96
- `__init__()` (labgrid.remote.common.Place method), 99
- `__init__()` (labgrid.remote.common.ResourceEntry method), 97
- `__init__()` (labgrid.remote.common.ResourceMatch method), 98
- `__init__()` (labgrid.remote.config.ResourceConfig method), 99
- `__init__()` (labgrid.remote.coordinator.ClientSession method), 100
- `__init__()` (labgrid.remote.coordinator.ExporterSession method), 100
- `__init__()` (labgrid.remote.exporter.EthernetPortExport method), 103
- `__init__()` (labgrid.remote.exporter.ResourceExport method), 101
- `__init__()` (labgrid.remote.exporter.USBEthernetExport method), 101
- `__init__()` (labgrid.remote.exporter.USBGenericExport method), 101
- `__init__()` (labgrid.remote.exporter.USBPowerPortExport method), 102
- `__init__()` (labgrid.remote.exporter.USBSDMuxExport method), 102
- `__init__()` (labgrid.remote.exporter.USBSerialPortExport method), 101
- `__init__()` (labgrid.remote.exporter.USBSigrokExport method), 102
- `__init__()` (labgrid.resource.base.EthernetInterface method), 103
- `__init__()` (labgrid.resource.base.EthernetPort method), 104
- `__init__()` (labgrid.resource.base.SerialPort method), 103
- `__init__()` (labgrid.resource.common.ManagedResource method), 106
- `__init__()` (labgrid.resource.common.NetworkResource method), 105
- `__init__()` (labgrid.resource.common.Resource method), 105
- `__init__()` (labgrid.resource.common.ResourceManager method), 105
- `__init__()` (labgrid.resource.ethernetport.EthernetPortManager method), 107
- `__init__()` (labgrid.resource.ethernetport.SNMPEthernetPort method), 108
- `__init__()` (labgrid.resource.ethernetport.SNMPSwitch method), 106
- `__init__()` (labgrid.resource.modbus.ModbusTCPCoil method), 108
- `__init__()` (labgrid.resource.networkservice.NetworkService method), 108
- `__init__()` (labgrid.resource.onewireport.OneWirePIO method), 109
- `__init__()` (labgrid.resource.power.NetworkPowerPort method), 109
- `__init__()` (labgrid.resource.remote.NetworkAlteraUSBBlaster method), 111
- `__init__()` (labgrid.resource.remote.NetworkAndroidFastboot method), 110
- `__init__()` (labgrid.resource.remote.NetworkIMXUSBLoader method), 110
- `__init__()` (labgrid.resource.remote.NetworkMXSUSBLoader method), 111
- `__init__()` (labgrid.resource.remote.NetworkSigrokUSBDevice method), 111
- `__init__()` (labgrid.resource.remote.NetworkUSBMassStorage method), 111
- `__init__()` (labgrid.resource.remote.NetworkUSBPowerPort method), 112
- `__init__()` (labgrid.resource.remote.NetworkUSBSDMuxDevice method), 112
- `__init__()` (labgrid.resource.remote.RemotePlace method), 110
- `__init__()` (labgrid.resource.remote.RemotePlaceManager method), 109
- `__init__()` (labgrid.resource.remote.RemoteUSBResource method), 110
- `__init__()` (labgrid.resource.serialport.NetworkSerialPort method), 113
- `__init__()` (labgrid.resource.serialport.RawSerialPort method), 112
- `__init__()` (labgrid.resource.sigrok.SigrokDevice method), 113
- `__init__()` (labgrid.resource.udev.AlteraUSBBlaster method), 115
- `__init__()` (labgrid.resource.udev.AndroidFastboot method), 115
- `__init__()` (labgrid.resource.udev.IMXUSBLoader method), 114
- `__init__()` (labgrid.resource.udev.MXSUSBLoader method), 115
- `__init__()` (labgrid.resource.udev.SigrokUSBDevice method), 116
- `__init__()` (labgrid.resource.udev.USBEthernetInterface method), 115
- `__init__()` (labgrid.resource.udev.USBMassStorage method), 115

method), 114
 __init__() (labgrid.resource.udev.USBPowerPort method), 116
 __init__() (labgrid.resource.udev.USBResource method), 114
 __init__() (labgrid.resource.udev.USBSDMuxDevice method), 116
 __init__() (labgrid.resource.udev.USBSerialPort method), 114
 __init__() (labgrid.resource.udev.UdevManager method), 113
 __init__() (labgrid.resource.ykushpowerport.YKUSHPowerPort method), 117
 __init__() (labgrid.step.Step method), 129
 __init__() (labgrid.step.StepEvent method), 128
 __init__() (labgrid.step.Steps method), 128
 __init__() (labgrid.stepreporter.StepReporter method), 129
 __init__() (labgrid.strategy.bareboxstrategy.BareboxStrategy method), 118
 __init__() (labgrid.strategy.bareboxstrategy.StrategyError method), 117
 __init__() (labgrid.strategy.common.Strategy method), 118
 __init__() (labgrid.strategy.shellstrategy.ShellStrategy method), 119
 __init__() (labgrid.strategy.shellstrategy.StrategyError method), 118
 __init__() (labgrid.strategy.ubootstrategy.StrategyError method), 119
 __init__() (labgrid.strategy.ubootstrategy.UBootStrategy method), 120
 __init__() (labgrid.target.Target method), 130
 __init__() (labgrid.util.exceptions.NoValidDriverError method), 120
 __init__() (labgrid.util.expect.PtxExpect method), 120
 __init__() (labgrid.util.qmp.QMPErrror method), 121
 __init__() (labgrid.util.qmp.QMPMonitor method), 121
 __init__() (labgrid.util.timeout.Timeout method), 122
 __le__() (labgrid.remote.common.ResourceMatch method), 98
 __le__() (labgrid.remote.exporter.EthernetPortExport method), 103
 __le__() (labgrid.resource.base.EthernetPort method), 104
 __le__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
 __le__() (labgrid.resource.ethernetport.SNMPETHERNETPort method), 108
 __le__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
 __lt__() (labgrid.remote.common.ResourceMatch method), 98
 __lt__() (labgrid.remote.exporter.EthernetPortExport method), 103
 __lt__() (labgrid.resource.base.EthernetPort method), 104
 __lt__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
 __lt__() (labgrid.resource.ethernetport.SNMPETHERNETPort method), 108
 __lt__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
 __module__ (labgrid.autoinstall.main.Handler attribute), 69
 __module__ (labgrid.autoinstall.main.Manager attribute), 69
 __module__ (labgrid.binding.BindingError attribute), 122
 __module__ (labgrid.binding.BindingMixin attribute), 124
 __module__ (labgrid.binding.BindingMixin.NamedBinding attribute), 123
 __module__ (labgrid.binding.BindingState attribute), 123
 __module__ (labgrid.binding.StateError attribute), 122
 __module__ (labgrid.config.Config attribute), 125
 __module__ (labgrid.consoleloggingreporter.ConsoleLoggingReporter attribute), 126
 __module__ (labgrid.driver.bareboxdriver.BareboxDriver attribute), 72
 __module__ (labgrid.driver.commandmixin.CommandMixin attribute), 72
 __module__ (labgrid.driver.common.Driver attribute), 72
 __module__ (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin attribute), 73
 __module__ (labgrid.driver.exception.CleanUpError attribute), 73
 __module__ (labgrid.driver.exception.ExecutionError attribute), 73
 __module__ (labgrid.driver.externalconsoledriver.ExternalConsoleDriver attribute), 74
 __module__ (labgrid.driver.fake.FakeCommandDriver attribute), 75
 __module__ (labgrid.driver.fake.FakeConsoleDriver attribute), 74
 __module__ (labgrid.driver.fake.FakeFileTransferDriver attribute), 75
 __module__ (labgrid.driver.fake.FakePowerDriver attribute), 75
 __module__ (labgrid.driver.fastbootdriver.AndroidFastbootDriver attribute), 76
 __module__ (labgrid.driver.infodriver.InfoDriver attribute), 76
 __module__ (labgrid.driver.modbusdriver.ModbusCoilDriver attribute), 77
 __module__ (labgrid.driver.onewiredriver.OneWirePIODriver attribute), 77
 __module__ (labgrid.driver.openocddriver.OpenOCDDriver

- attribute), 77
- __module__ (labgrid.driver.powerdriver.DigitalOutputPowerDriver attribute), 79
- __module__ (labgrid.driver.powerdriver.ExternalPowerDriver attribute), 79
- __module__ (labgrid.driver.powerdriver.ManualPowerDriver attribute), 78
- __module__ (labgrid.driver.powerdriver.NetworkPowerDriver attribute), 79
- __module__ (labgrid.driver.powerdriver.PowerResetMixin attribute), 78
- __module__ (labgrid.driver.powerdriver.USBPowerDriver attribute), 80
- __module__ (labgrid.driver.powerdriver.YKUSHPowerDriver attribute), 80
- __module__ (labgrid.driver.qemudriver.QEMUDriver attribute), 81
- __module__ (labgrid.driver.quartushpsdriver.QuartusHPSDriver attribute), 82
- __module__ (labgrid.driver.serialdriver.SerialDriver attribute), 82
- __module__ (labgrid.driver.shelldriver.ShellDriver attribute), 85
- __module__ (labgrid.driver.sigrokdriver.SigrokDriver attribute), 85
- __module__ (labgrid.driver.smallubootdriver.SmallUBootDriver attribute), 86
- __module__ (labgrid.driver.sshdriver.SSHDriver attribute), 87
- __module__ (labgrid.driver.ubootdriver.UBootDriver attribute), 88
- __module__ (labgrid.driver.usbloader.IMXUSBDriver attribute), 89
- __module__ (labgrid.driver.usbloader.MXSUSBDriver attribute), 88
- __module__ (labgrid.driver.usbstorage.USBStorageDriver attribute), 89
- __module__ (labgrid.environment.Environment attribute), 126
- __module__ (labgrid.exceptions.InvalidConfigError attribute), 127
- __module__ (labgrid.exceptions.NoConfigFoundError attribute), 126
- __module__ (labgrid.exceptions.NoDriverFoundError attribute), 127
- __module__ (labgrid.exceptions.NoResourceFoundError attribute), 127
- __module__ (labgrid.exceptions.NoSupplierFoundError attribute), 127
- __module__ (labgrid.external.hawkbit.HawkbiError attribute), 91
- __module__ (labgrid.external.hawkbit.HawkbitTestClient attribute), 90
- __module__ (labgrid.external.usbstick.StateError attribute), 92
- __module__ (labgrid.external.usbstick.USBStatus attribute), 91
- __module__ (labgrid.external.usbstick.USBStick attribute), 92
- __module__ (labgrid.factory.TargetFactory attribute), 128
- __module__ (labgrid.protocol.bootstrapprotocol.BootstrapProtocol attribute), 92
- __module__ (labgrid.protocol.commandprotocol.CommandProtocol attribute), 93
- __module__ (labgrid.protocol.consoleprotocol.ConsoleProtocol attribute), 93
- __module__ (labgrid.protocol.consoleprotocol.ConsoleProtocol.Client attribute), 93
- __module__ (labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol attribute), 93
- __module__ (labgrid.protocol.filesystemprotocol.FileSystemProtocol attribute), 94
- __module__ (labgrid.protocol.filetransferprotocol.FileTransferProtocol attribute), 94
- __module__ (labgrid.protocol.infoprotocol.InfoProtocol attribute), 94
- __module__ (labgrid.protocol.linuxbootprotocol.LinuxBootProtocol attribute), 94
- __module__ (labgrid.protocol.mmioprotocol.MMIOProtocol attribute), 95
- __module__ (labgrid.protocol.powerprotocol.PowerProtocol attribute), 95
- __module__ (labgrid.provider.fileprovider.FileProvider attribute), 95
- __module__ (labgrid.provider.mediafileprovider.MediaFileProvider attribute), 96
- __module__ (labgrid.pytestplugin.reporter.StepReporter attribute), 96
- __module__ (labgrid.remote.client.Error attribute), 97
- __module__ (labgrid.remote.client.ServerError attribute), 97
- __module__ (labgrid.remote.client.UserError attribute), 97
- __module__ (labgrid.remote.common.Place attribute), 99
- __module__ (labgrid.remote.common.ResourceEntry attribute), 98
- __module__ (labgrid.remote.common.ResourceMatch attribute), 98
- __module__ (labgrid.remote.config.ResourceConfig attribute), 99
- __module__ (labgrid.remote.coordinator.Action attribute), 99
- __module__ (labgrid.remote.coordinator.ClientSession attribute), 100
- __module__ (labgrid.remote.coordinator.ExporterSession attribute), 100
- __module__ (labgrid.remote.coordinator.RemoteSession attribute), 100

- __module__ (labgrid.remote.exporter.EthernetPortExport attribute), 103
- __module__ (labgrid.remote.exporter.ResourceExport attribute), 101
- __module__ (labgrid.remote.exporter.USBEthernetExport attribute), 101
- __module__ (labgrid.remote.exporter.USBGenericExport attribute), 101
- __module__ (labgrid.remote.exporter.USBPowerPortExport attribute), 102
- __module__ (labgrid.remote.exporter.USBSDMuxExport attribute), 102
- __module__ (labgrid.remote.exporter.USBSerialPortExport attribute), 101
- __module__ (labgrid.remote.exporter.USBSigrokExport attribute), 102
- __module__ (labgrid.resource.base.EthernetInterface attribute), 103
- __module__ (labgrid.resource.base.EthernetPort attribute), 104
- __module__ (labgrid.resource.base.SerialPort attribute), 103
- __module__ (labgrid.resource.common.ManagedResource attribute), 106
- __module__ (labgrid.resource.common.NetworkResource attribute), 105
- __module__ (labgrid.resource.common.Resource attribute), 105
- __module__ (labgrid.resource.common.ResourceManager attribute), 105
- __module__ (labgrid.resource.ethernetport.EthernetPortManager attribute), 107
- __module__ (labgrid.resource.ethernetport.SNMPEthernetPort attribute), 108
- __module__ (labgrid.resource.ethernetport.SNMPSwitch attribute), 106
- __module__ (labgrid.resource.modbus.ModbusTCPCoil attribute), 108
- __module__ (labgrid.resource.networkservice.NetworkService attribute), 108
- __module__ (labgrid.resource.onewirereport.OneWirePIO attribute), 109
- __module__ (labgrid.resource.power.NetworkPowerPort attribute), 109
- __module__ (labgrid.resource.remote.NetworkAlteraUSBBlaster attribute), 111
- __module__ (labgrid.resource.remote.NetworkAndroidFastboot attribute), 110
- __module__ (labgrid.resource.remote.NetworkIMXUSBLoader attribute), 110
- __module__ (labgrid.resource.remote.NetworkMXSUSBLoader attribute), 111
- __module__ (labgrid.resource.remote.NetworkSigrokUSBDevice attribute), 111
- __module__ (labgrid.resource.remote.NetworkUSBMassStorage attribute), 111
- __module__ (labgrid.resource.remote.NetworkUSBPowerPort attribute), 112
- __module__ (labgrid.resource.remote.NetworkUSBSDMuxDevice attribute), 112
- __module__ (labgrid.resource.remote.RemotePlace attribute), 110
- __module__ (labgrid.resource.remote.RemotePlaceManager attribute), 109
- __module__ (labgrid.resource.remote.RemoteUSBResource attribute), 110
- __module__ (labgrid.resource.serialport.NetworkSerialPort attribute), 113
- __module__ (labgrid.resource.serialport.RawSerialPort attribute), 112
- __module__ (labgrid.resource.sigrok.SigrokDevice attribute), 113
- __module__ (labgrid.resource.udev.AlteraUSBBlaster attribute), 115
- __module__ (labgrid.resource.udev.AndroidFastboot attribute), 115
- __module__ (labgrid.resource.udev.IMXUSBLoader attribute), 115
- __module__ (labgrid.resource.udev.MXSUSBLoader attribute), 115
- __module__ (labgrid.resource.udev.SigrokUSBDevice attribute), 116
- __module__ (labgrid.resource.udev.USBEthernetInterface attribute), 115
- __module__ (labgrid.resource.udev.USBMassStorage attribute), 114
- __module__ (labgrid.resource.udev.USBPowerPort attribute), 116
- __module__ (labgrid.resource.udev.USBResource attribute), 114
- __module__ (labgrid.resource.udev.USBSDMuxDevice attribute), 116
- __module__ (labgrid.resource.udev.USBSerialPort attribute), 114
- __module__ (labgrid.resource.udev.UdevManager attribute), 113
- __module__ (labgrid.resource.ykushpowerport.YKUSHPowerPort attribute), 117
- __module__ (labgrid.step.Step attribute), 129
- __module__ (labgrid.step.StepEvent attribute), 129
- __module__ (labgrid.step.Steps attribute), 128
- __module__ (labgrid.stepreporter.StepReporter attribute), 129
- __module__ (labgrid.strategy.bareboxstrategy.BareboxStrategy attribute), 118
- __module__ (labgrid.strategy.bareboxstrategy.Status attribute), 117
- __module__ (labgrid.strategy.bareboxstrategy.StrategyError attribute), 117

- attribute), 117
- __module__ (labgrid.strategy.common.Strategy attribute), 118
- __module__ (labgrid.strategy.shellstrategy.ShellStrategy attribute), 119
- __module__ (labgrid.strategy.shellstrategy.Status attribute), 119
- __module__ (labgrid.strategy.shellstrategy.StrategyError attribute), 118
- __module__ (labgrid.strategy.ubootstrategy.Status attribute), 119
- __module__ (labgrid.strategy.ubootstrategy.StrategyError attribute), 119
- __module__ (labgrid.strategy.ubootstrategy.UBootStrategy attribute), 120
- __module__ (labgrid.target.Target attribute), 130
- __module__ (labgrid.util.exceptions.NoValidDriverError attribute), 120
- __module__ (labgrid.util.expect.PtxExpect attribute), 121
- __module__ (labgrid.util.qmp.QMPErrror attribute), 121
- __module__ (labgrid.util.qmp.QMPMonitor attribute), 121
- __module__ (labgrid.util.timeout.Timeout attribute), 122
- __ne__() (labgrid.remote.common.ResourceMatch method), 98
- __ne__() (labgrid.remote.exporter.EthernetPortExport method), 103
- __ne__() (labgrid.resource.base.EthernetPort method), 104
- __ne__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
- __ne__() (labgrid.resource.ethernetport.SNMPEthernetPort method), 108
- __ne__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
- __new__() (labgrid.binding.BindingState method), 123
- __new__() (labgrid.external.usbstick.USBStatus method), 91
- __new__() (labgrid.remote.coordinator.Action method), 99
- __new__() (labgrid.strategy.bareboxstrategy.Status method), 117
- __new__() (labgrid.strategy.shellstrategy.Status method), 119
- __new__() (labgrid.strategy.ubootstrategy.Status method), 119
- __repr__() (labgrid.binding.BindingError method), 122
- __repr__() (labgrid.binding.BindingMixin method), 124
- __repr__() (labgrid.binding.BindingMixin.NamedBinding method), 123
- __repr__() (labgrid.binding.StateError method), 122
- __repr__() (labgrid.config.Config method), 125
- __repr__() (labgrid.driver.bareboxdriver.BareboxDriver method), 72
- __repr__() (labgrid.driver.common.Driver method), 72
- __repr__() (labgrid.driver.exception.CleanUpError method), 73
- __repr__() (labgrid.driver.exception.ExecutionError method), 73
- __repr__() (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
- __repr__() (labgrid.driver.fake.FakeCommandDriver method), 75
- __repr__() (labgrid.driver.fake.FakeConsoleDriver method), 74
- __repr__() (labgrid.driver.fake.FakeFileTransferDriver method), 75
- __repr__() (labgrid.driver.fake.FakePowerDriver method), 75
- __repr__() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
- __repr__() (labgrid.driver.infodriver.InfoDriver method), 76
- __repr__() (labgrid.driver.modbusdriver.ModbusCoilDriver method), 77
- __repr__() (labgrid.driver.onewiredriver.OneWirePIODriver method), 77
- __repr__() (labgrid.driver.openocddriver.OpenOCDDriver method), 77
- __repr__() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
- __repr__() (labgrid.driver.powerdriver.ExternalPowerDriver method), 79
- __repr__() (labgrid.driver.powerdriver.ManualPowerDriver method), 78
- __repr__() (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
- __repr__() (labgrid.driver.powerdriver.PowerResetMixin method), 78
- __repr__() (labgrid.driver.powerdriver.USBPowerDriver method), 80
- __repr__() (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
- __repr__() (labgrid.driver.qemudriver.QEMUDriver method), 81
- __repr__() (labgrid.driver.quartushpsdriver.QuartusHPSDriver method), 82
- __repr__() (labgrid.driver.serialdriver.SerialDriver method), 82
- __repr__() (labgrid.driver.shelldriver.ShellDriver method), 85
- __repr__() (labgrid.driver.sigrokdriver.SigrokDriver method), 85
- __repr__() (labgrid.driver.smallubootdriver.SmallUBootDriver method), 86
- __repr__() (labgrid.driver.sshdriver.SSHDriver method), 87
- __repr__() (labgrid.driver.ubootdriver.UBootDriver

- method), 88
- __repr__() (labgrid.driver.usbloader.IMXUSBDriver method), 89
- __repr__() (labgrid.driver.usbloader.MXSUSBDriver method), 88
- __repr__() (labgrid.driver.usbstorage.USBStorageDriver method), 89
- __repr__() (labgrid.environment.Environment method), 126
- __repr__() (labgrid.exceptions.InvalidConfigError method), 127
- __repr__() (labgrid.exceptions.NoConfigFoundError method), 126
- __repr__() (labgrid.exceptions.NoDriverFoundError method), 127
- __repr__() (labgrid.exceptions.NoResourceFoundError method), 127
- __repr__() (labgrid.exceptions.NoSupplierFoundError method), 127
- __repr__() (labgrid.external.hawkbit.HawkbiError method), 91
- __repr__() (labgrid.external.hawkbit.HawkbitTestClient method), 90
- __repr__() (labgrid.external.usbstick.StateError method), 92
- __repr__() (labgrid.external.usbstick.USBStick method), 92
- __repr__() (labgrid.provider.mediafileprovider.MediaFileProvider method), 96
- __repr__() (labgrid.remote.common.Place method), 99
- __repr__() (labgrid.remote.common.ResourceEntry method), 98
- __repr__() (labgrid.remote.common.ResourceMatch method), 98
- __repr__() (labgrid.remote.config.ResourceConfig method), 99
- __repr__() (labgrid.remote.coordinator.ClientSession method), 100
- __repr__() (labgrid.remote.coordinator.ExporterSession method), 100
- __repr__() (labgrid.remote.coordinator.RemoteSession method), 100
- __repr__() (labgrid.remote.exporter.EthernetPortExport method), 103
- __repr__() (labgrid.remote.exporter.ResourceExport method), 101
- __repr__() (labgrid.remote.exporter.USBEthernetExport method), 101
- __repr__() (labgrid.remote.exporter.USBGenericExport method), 102
- __repr__() (labgrid.remote.exporter.USBPowerPortExport method), 102
- __repr__() (labgrid.remote.exporter.USBSDMuxExport method), 102
- __repr__() (labgrid.remote.exporter.USBSerialPortExport method), 101
- __repr__() (labgrid.remote.exporter.USBSigrokExport method), 102
- __repr__() (labgrid.resource.base.EthernetInterface method), 103
- __repr__() (labgrid.resource.base.EthernetPort method), 104
- __repr__() (labgrid.resource.base.SerialPort method), 103
- __repr__() (labgrid.resource.common.ManagedResource method), 106
- __repr__() (labgrid.resource.common.NetworkResource method), 105
- __repr__() (labgrid.resource.common.Resource method), 105
- __repr__() (labgrid.resource.common.ResourceManager method), 105
- __repr__() (labgrid.resource.ethernetport.EthernetPortManager method), 107
- __repr__() (labgrid.resource.ethernetport.SNMPEthernetPort method), 108
- __repr__() (labgrid.resource.ethernetport.SNMPSwitch method), 106
- __repr__() (labgrid.resource.modbus.ModbusTCPCoil method), 108
- __repr__() (labgrid.resource.networkservice.NetworkService method), 108
- __repr__() (labgrid.resource.onewireport.OneWirePIO method), 109
- __repr__() (labgrid.resource.power.NetworkPowerPort method), 109
- __repr__() (labgrid.resource.remote.NetworkAlteraUSBBlaster method), 111
- __repr__() (labgrid.resource.remote.NetworkAndroidFastboot method), 110
- __repr__() (labgrid.resource.remote.NetworkIMXUSBLoader method), 110
- __repr__() (labgrid.resource.remote.NetworkMXSUSBLoader method), 111
- __repr__() (labgrid.resource.remote.NetworkSigrokUSBDevice method), 111
- __repr__() (labgrid.resource.remote.NetworkUSBMassStorage method), 111
- __repr__() (labgrid.resource.remote.NetworkUSBPowerPort method), 112
- __repr__() (labgrid.resource.remote.NetworkUSBSDMuxDevice method), 112
- __repr__() (labgrid.resource.remote.RemotePlace method), 110
- __repr__() (labgrid.resource.remote.RemotePlaceManager method), 110
- __repr__() (labgrid.resource.remote.RemoteUSBResource method), 110

- `__repr__()` (labgrid.resource.serialport.NetworkSerialPort method), 113
- `__repr__()` (labgrid.resource.serialport.RawSerialPort method), 112
- `__repr__()` (labgrid.resource.sigrok.SigrokDevice method), 113
- `__repr__()` (labgrid.resource.udev.AlteraUSBBlaster method), 115
- `__repr__()` (labgrid.resource.udev.AndroidFastboot method), 115
- `__repr__()` (labgrid.resource.udev.IMXUSBLoader method), 115
- `__repr__()` (labgrid.resource.udev.MXSUSBLoader method), 115
- `__repr__()` (labgrid.resource.udev.SigrokUSBDevice method), 116
- `__repr__()` (labgrid.resource.udev.USBEthernetInterface method), 115
- `__repr__()` (labgrid.resource.udev.USBMassStorage method), 114
- `__repr__()` (labgrid.resource.udev.USBPowerPort method), 116
- `__repr__()` (labgrid.resource.udev.USBResource method), 114
- `__repr__()` (labgrid.resource.udev.USBSDMuxDevice method), 116
- `__repr__()` (labgrid.resource.udev.USBSerialPort method), 114
- `__repr__()` (labgrid.resource.udev.UdevManager method), 113
- `__repr__()` (labgrid.resource.ykushpowerport.YKUSHPowerPort method), 117
- `__repr__()` (labgrid.step.Step method), 129
- `__repr__()` (labgrid.strategy.bareboxstrategy.BareboxStrategy method), 118
- `__repr__()` (labgrid.strategy.bareboxstrategy.StrategyError method), 117
- `__repr__()` (labgrid.strategy.common.Strategy method), 118
- `__repr__()` (labgrid.strategy.shellstrategy.ShellStrategy method), 119
- `__repr__()` (labgrid.strategy.shellstrategy.StrategyError method), 118
- `__repr__()` (labgrid.strategy.ubootstrategy.StrategyError method), 119
- `__repr__()` (labgrid.strategy.ubootstrategy.UBootStrategy method), 120
- `__repr__()` (labgrid.target.Target method), 130
- `__repr__()` (labgrid.util.exceptions.NoValidDriverError method), 120
- `__repr__()` (labgrid.util.qmp.QMPErrror method), 121
- `__repr__()` (labgrid.util.qmp.QMPMonitor method), 121
- `__repr__()` (labgrid.util.timeout.Timeout method), 122
- `__str__()` (labgrid.remote.common.ResourceMatch method), 98
- `__str__()` (labgrid.step.Step method), 129
- `__str__()` (labgrid.step.StepEvent method), 128
- `__weakref__` (labgrid.autoinstall.main.Manager attribute), 69
- `__weakref__` (labgrid.binding.BindingError attribute), 122
- `__weakref__` (labgrid.binding.BindingMixin attribute), 124
- `__weakref__` (labgrid.binding.BindingMixin.NamedBinding attribute), 123
- `__weakref__` (labgrid.binding.StateError attribute), 122
- `__weakref__` (labgrid.config.Config attribute), 125
- `__weakref__` (labgrid.consoleloggingreporter.ConsoleLoggingReporter attribute), 126
- `__weakref__` (labgrid.driver.commandmixin.CommandMixin attribute), 72
- `__weakref__` (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin attribute), 73
- `__weakref__` (labgrid.driver.exception.CleanUpError attribute), 74
- `__weakref__` (labgrid.driver.exception.ExecutionError attribute), 73
- `__weakref__` (labgrid.environment.Environment attribute), 126
- `__weakref__` (labgrid.exceptions.InvalidConfigError attribute), 127
- `__weakref__` (labgrid.exceptions.NoConfigFoundError attribute), 127
- `__weakref__` (labgrid.exceptions.NoSupplierFoundError attribute), 127
- `__weakref__` (labgrid.external.hawkbit.HawkbiError attribute), 91
- `__weakref__` (labgrid.external.hawkbit.HawkbitTestClient attribute), 90
- `__weakref__` (labgrid.external.usbstick.StateError attribute), 92
- `__weakref__` (labgrid.external.usbstick.USBStick attribute), 92
- `__weakref__` (labgrid.factory.TargetFactory attribute), 128
- `__weakref__` (labgrid.pytestplugin.reporter.StepReporter attribute), 96
- `__weakref__` (labgrid.remote.client.Error attribute), 97
- `__weakref__` (labgrid.remote.common.Place attribute), 99
- `__weakref__` (labgrid.remote.common.ResourceEntry attribute), 98
- `__weakref__` (labgrid.remote.common.ResourceMatch attribute), 98
- `__weakref__` (labgrid.remote.config.ResourceConfig attribute), 99
- `__weakref__` (labgrid.remote.coordinator.RemoteSession attribute), 100

- `__weakref__` (labgrid.resource.common.ResourceManager attribute), 105
 - `__weakref__` (labgrid.resource.ethernetport.SNMPSwitch attribute), 106
 - `__weakref__` (labgrid.step.Step attribute), 129
 - `__weakref__` (labgrid.step.StepEvent attribute), 129
 - `__weakref__` (labgrid.step.Steps attribute), 128
 - `__weakref__` (labgrid.stepreporter.StepReporter attribute), 129
 - `__weakref__` (labgrid.strategy.bareboxstrategy.StrategyError attribute), 117
 - `__weakref__` (labgrid.strategy.shellstrategy.StrategyError attribute), 118
 - `__weakref__` (labgrid.strategy.ubootstrategy.StrategyError attribute), 119
 - `__weakref__` (labgrid.target.Target attribute), 130
 - `__weakref__` (labgrid.util.exceptions.NoValidDriverError attribute), 120
 - `__weakref__` (labgrid.util.qmp.QMPError attribute), 121
 - `__weakref__` (labgrid.util.qmp.QMPMonitor attribute), 121
 - `__weakref__` (labgrid.util.timeout.Timeout attribute), 122
- ## A
- Action (class in labgrid.remote.coordinator), 99
 - activate() (labgrid.target.Target method), 130
 - active (labgrid.binding.BindingState attribute), 123
 - ADD (labgrid.remote.coordinator.Action attribute), 99
 - add_artifact() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - add_distributionset() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - add_rollout() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - add_swmodule() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - add_target() (labgrid.external.hawkbit.HawkbitTestClient method), 89
 - age (labgrid.step.StepEvent attribute), 129
 - AlteraUSBBlaster (class in labgrid.resource.udev), 115
 - analyze() (labgrid.driver.sigrokdriver.SigrokDriver method), 85
 - AndroidFastboot (class in labgrid.resource.udev), 115
 - AndroidFastbootDriver (class in labgrid.driver.fastbootdriver), 75
 - args (labgrid.remote.common.ResourceEntry attribute), 97
 - asdict() (labgrid.remote.common.Place method), 98
 - asdict() (labgrid.remote.common.ResourceEntry method), 97
 - assign_target() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - avail (labgrid.remote.common.ResourceEntry attribute), 97
 - await_boot() (labgrid.driver.bareboxdriver.BareboxDriver method), 71
 - await_boot() (labgrid.driver.ubootdriver.UBootDriver method), 88
 - await_boot() (labgrid.protocol.linuxbootprotocol.LinuxBootProtocol method), 94
 - await_resources() (labgrid.target.Target method), 130
- ## B
- barebox (labgrid.strategy.bareboxstrategy.Status attribute), 117
 - BareboxDriver (class in labgrid.driver.bareboxdriver), 71
 - BareboxStrategy (class in labgrid.strategy.bareboxstrategy), 117
 - bind() (labgrid.target.Target method), 130
 - bind_driver() (labgrid.target.Target method), 130
 - bind_resource() (labgrid.target.Target method), 130
 - BindingError, 122
 - BindingMixin (class in labgrid.binding), 123
 - BindingMixin.NamedBinding (class in labgrid.binding), 123
 - bindings (labgrid.binding.BindingMixin attribute), 123
 - bindings (labgrid.driver.bareboxdriver.BareboxDriver attribute), 71
 - bindings (labgrid.driver.fastbootdriver.AndroidFastbootDriver attribute), 75
 - bindings (labgrid.driver.infodriver.InfoDriver attribute), 76
 - bindings (labgrid.driver.modbusdriver.ModbusCoilDriver attribute), 76
 - bindings (labgrid.driver.onewiredriver.OneWirePIODriver attribute), 77
 - bindings (labgrid.driver.openocddriver.OpenOCDDriver attribute), 77
 - bindings (labgrid.driver.powerdriver.DigitalOutputPowerDriver attribute), 79
 - bindings (labgrid.driver.powerdriver.NetworkPowerDriver attribute), 79
 - bindings (labgrid.driver.powerdriver.USBPowerDriver attribute), 80
 - bindings (labgrid.driver.powerdriver.YKUSHPowerDriver attribute), 80
 - bindings (labgrid.driver.quartushpsdriver.QuartusHPSDriver attribute), 82
 - bindings (labgrid.driver.serialdriver.SerialDriver attribute), 82
 - bindings (labgrid.driver.shelldriver.ShellDriver attribute), 83
 - bindings (labgrid.driver.sigrokdriver.SigrokDriver attribute), 85
 - bindings (labgrid.driver.sshdriver.SSHDriver attribute), 86

- bindings (labgrid.driver.ubootdriver.UBootDriver attribute), 87
 - bindings (labgrid.driver.usbloader.IMXUSBDriver attribute), 89
 - bindings (labgrid.driver.usbloader.MXSUSBDriver attribute), 88
 - bindings (labgrid.driver.usbstorage.USBStorageDriver attribute), 89
 - bindings (labgrid.strategy.bareboxstrategy.BareboxStrategy attribute), 117
 - bindings (labgrid.strategy.shellstrategy.ShellStrategy attribute), 119
 - bindings (labgrid.strategy.ubootstrategy.UBootStrategy attribute), 120
 - BindingState (class in labgrid.binding), 122
 - bold() (in module labgrid.pytestplugin.reporter), 96
 - boot() (labgrid.driver.bareboxdriver.BareboxDriver method), 71
 - boot() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
 - boot() (labgrid.driver.smallubootdriver.SmallUBootDriver method), 86
 - boot() (labgrid.driver.ubootdriver.UBootDriver method), 88
 - boot() (labgrid.protocol.linuxbootprotocol.LinuxBootProtocol method), 94
 - BootstrapProtocol (class in labgrid.protocol.bootstrapprotocol), 92
 - bound (labgrid.binding.BindingState attribute), 123
 - busnum (labgrid.resource.udev.USBResource attribute), 114
- ## C
- capture() (labgrid.driver.sigrokdriver.SigrokDriver method), 85
 - check_active() (labgrid.binding.BindingMixin class method), 123
 - check_file() (in module labgrid.driver.common), 73
 - cleanup() (labgrid.environment.Environment method), 126
 - cleanup() (labgrid.target.Target method), 130
 - CleanUpError, 73
 - ClientSession (class in labgrid.remote.coordinator), 100
 - close() (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
 - close() (labgrid.driver.fake.FakeConsoleDriver method), 74
 - close() (labgrid.driver.serialdriver.SerialDriver method), 82
 - cls (labgrid.remote.common.ResourceEntry attribute), 97
 - command_prefix (labgrid.resource.common.NetworkResource attribute), 105
 - command_prefix (labgrid.resource.common.Resource attribute), 104
 - CommandMixin (class in labgrid.driver.commandmixin), 72
 - CommandProtocol (class in labgrid.protocol.commandprotocol), 92
 - Config (class in labgrid.config), 124
 - configure() (labgrid.autoinstall.main.Manager method), 69
 - ConsoleExpectMixin (class in labgrid.driver.consoleexpectmixin), 73
 - ConsoleLoggingReporter (class in labgrid.consoleloggingreporter), 125
 - ConsoleProtocol (class in labgrid.protocol.consoleprotocol), 93
 - ConsoleProtocol.Client (class in labgrid.protocol.consoleprotocol), 93
 - continue_boot() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
 - control_path (labgrid.resource.udev.USBSDMuxDevice attribute), 116
 - cycle() (labgrid.driver.fake.FakePowerDriver method), 75
 - cycle() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
 - cycle() (labgrid.driver.powerdriver.ExternalPowerDriver method), 78
 - cycle() (labgrid.driver.powerdriver.ManualPowerDriver method), 78
 - cycle() (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
 - cycle() (labgrid.driver.powerdriver.USBPowerDriver method), 80
 - cycle() (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
 - cycle() (labgrid.driver.qemudriver.QEMUDriver method), 81
 - cycle() (labgrid.protocol.powerprotocol.PowerProtocol method), 95
- ## D
- deactivate() (labgrid.target.Target method), 130
 - DEL (labgrid.remote.coordinator.Action attribute), 99
 - delete() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - delete_artifact() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - delete_distributionset() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - delete_swmodule() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - delete_target() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - devnum (labgrid.resource.udev.USBResource attribute), 114

- diff_dict() (in module labgrid.util.dict), 120
- DigitalOutputPowerDriver (class in labgrid.driver.powerdriver), 79
- DigitalOutputProtocol (class in labgrid.protocol.digitaloutputprotocol), 93
- display_name (labgrid.binding.BindingMixin attribute), 123
- Driver (class in labgrid.driver.common), 72
- dump() (in module labgrid.util.yaml), 122
- duration (labgrid.step.Step attribute), 129
- ## E
- enable_tcp_nodelay() (in module labgrid.remote.common), 99
- env() (in module labgrid.pytestplugin.fixtures), 96
- Environment (class in labgrid.environment), 126
- Error, 97
- error (labgrid.binding.BindingState attribute), 123
- EthernetInterface (class in labgrid.resource.base), 103
- EthernetPort (class in labgrid.resource.base), 104
- EthernetPortExport (class in labgrid.remote.exporter), 102
- EthernetPortManager (class in labgrid.resource.ethernetport), 106
- execute() (labgrid.util.qmp.QMPMonitor method), 121
- ExecutionError, 73
- expect() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
- expect() (labgrid.protocol.consoleprotocol.ConsoleProtocol method), 93
- expired (labgrid.util.timeout.Timeout attribute), 122
- ExporterSession (class in labgrid.remote.coordinator), 100
- ExternalConsoleDriver (class in labgrid.driver.externalconsoledriver), 74
- ExternalPowerDriver (class in labgrid.driver.powerdriver), 78
- extra (labgrid.remote.common.ResourceEntry attribute), 97
- ## F
- FakeCommandDriver (class in labgrid.driver.fake), 74
- FakeConsoleDriver (class in labgrid.driver.fake), 74
- FakeFileTransferDriver (class in labgrid.driver.fake), 75
- FakePowerDriver (class in labgrid.driver.fake), 75
- FileProvider (class in labgrid.provider.fileprovider), 95
- FileSystemProtocol (class in labgrid.protocol.filesystemprotocol), 94
- FileTransferProtocol (class in labgrid.protocol.filetransferprotocol), 94
- filter_match() (labgrid.resource.udev.AlteraUSBBlaster method), 115
- filter_match() (labgrid.resource.udev.AndroidFastboot method), 115
- filter_match() (labgrid.resource.udev.IMXUSBLoader method), 114
- filter_match() (labgrid.resource.udev.MXSUSBLoader method), 115
- filter_match() (labgrid.resource.udev.USBResource method), 113
- find_any_role_with_place() (in module labgrid.remote.client), 97
- find_role_by_place() (in module labgrid.remote.client), 97
- flash() (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
- flash() (labgrid.driver.quartushpsdriver.QuartusHPSDriver method), 82
- flat_dict() (in module labgrid.util.dict), 120
- fromstr() (labgrid.remote.common.ResourceMatch class method), 98
- ## G
- gen_marker() (in module labgrid.util.marker), 121
- get() (in module labgrid.driver.power.apc), 70
- get() (in module labgrid.driver.power.digipower), 70
- get() (in module labgrid.driver.power.gude), 70
- get() (in module labgrid.driver.power.gude24), 70
- get() (in module labgrid.driver.power.netio), 70
- get() (in module labgrid.driver.power.netio_kshell), 70
- get() (labgrid.driver.fake.FakeFileTransferDriver method), 75
- get() (labgrid.driver.modbusdriver.ModbusCoilDriver method), 77
- get() (labgrid.driver.onewiredriver.OneWirePIODriver method), 77
- get() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
- get() (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
- get() (labgrid.driver.powerdriver.USBPowerDriver method), 80
- get() (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
- get() (labgrid.driver.shelldriver.ShellDriver method), 84
- get() (labgrid.driver.sshdriver.SSHDriver method), 87
- get() (labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol method), 93
- get() (labgrid.protocol.filetransferprotocol.FileTransferProtocol method), 94
- get() (labgrid.provider.fileprovider.FileProvider method), 95
- get() (labgrid.provider.mediafileprovider.MediaFileProvider method), 95
- get() (labgrid.resource.common.ResourceManager class method), 105
- get_active_driver() (labgrid.target.Target method), 131

- get_bytes() (labgrid.driver.shelldriver.ShellDriver method), 83
 get_console_matches() (labgrid.protocol.consoleprotocol.ConsoleProtocol.Client method), 93
 get_current() (labgrid.step.Steps method), 128
 get_driver() (labgrid.target.Target method), 131
 get_endpoint() (labgrid.external.hawkbite.HawkbitTestClient method), 90
 get_file() (labgrid.external.usbstick.USBStick method), 91
 get_free_port() (in module labgrid.remote.exporter), 100
 get_hostname() (labgrid.driver.infodriver.InfoDriver method), 76
 get_hostname() (labgrid.protocol.infoprotocol.InfoProtocol method), 94
 get_image_path() (labgrid.config.Config method), 124
 get_images() (labgrid.config.Config method), 125
 get_imports() (labgrid.config.Config method), 125
 get_ip() (labgrid.driver.infodriver.InfoDriver method), 76
 get_ip() (labgrid.protocol.infoprotocol.InfoProtocol method), 94
 get_logfile() (labgrid.consoleloggingreporter.ConsoleLoggingReporter method), 126
 get_managed_parent() (labgrid.resource.common.ManagedResource method), 106
 get_managed_parent() (labgrid.resource.common.Resource method), 104
 get_new() (labgrid.step.Steps method), 128
 get_option() (labgrid.config.Config method), 124
 get_path() (labgrid.config.Config method), 124
 get_paths() (labgrid.config.Config method), 125
 get_priority() (labgrid.driver.common.Driver method), 72
 get_resource() (labgrid.target.Target method), 131
 get_resources() (labgrid.remote.coordinator.ExporterSession method), 100
 get_service_status() (labgrid.driver.infodriver.InfoDriver method), 76
 get_service_status() (labgrid.protocol.infoprotocol.InfoProtocol method), 94
 get_size() (labgrid.driver.usbstorage.USBStorageDriver method), 89
 get_status() (labgrid.driver.bareboxdriver.BareboxDriver method), 71
 get_status() (labgrid.driver.fake.FakeCommandDriver method), 75
 get_status() (labgrid.driver.shelldriver.ShellDriver method), 83
 get_status() (labgrid.driver.sshdriver.SSHDriver method), 86
 get_status() (labgrid.driver.ubootdriver.UBootDriver method), 88
 get_status() (labgrid.protocol.commandprotocol.CommandProtocol method), 92
 get_target() (labgrid.environment.Environment method), 126
 get_targets() (labgrid.config.Config method), 125
 get_tool() (labgrid.config.Config method), 124
 getmatch() (labgrid.remote.common.Place method), 98
- ## H
- Handler (class in labgrid.autoinstall.main), 69
 hasmatch() (labgrid.remote.common.Place method), 98
 HawkbiError, 90
 HawkbitTestClient (class in labgrid.external.hawkbite), 89
- ## I
- idle (labgrid.binding.BindingState attribute), 123
 if_state (labgrid.resource.udev.USBEthernetInterface attribute), 115
 IMXUSBDriver (class in labgrid.driver.usbloader), 88
 IMXUSBLoader (class in labgrid.resource.udev), 114
 InfoDriver (class in labgrid.driver.infodriver), 76
 InfoProtocol (class in labgrid.protocol.infoprotocol), 94
 instance (labgrid.consoleloggingreporter.ConsoleLoggingReporter attribute), 125
 instance (labgrid.stepreporter.StepReporter attribute), 129
 instances (labgrid.resource.common.ResourceManager attribute), 105
 interact() (labgrid.target.Target method), 131
 InvalidConfigError, 127
 is_active (labgrid.step.Step attribute), 129
 is_done (labgrid.step.Step attribute), 129
 ismatch() (labgrid.remote.common.ResourceMatch method), 98
- ## J
- join() (labgrid.autoinstall.main.Manager method), 70
- ## K
- key (labgrid.remote.coordinator.RemoteSession attribute), 100
- ## L
- labgrid (module), 69
 labgrid.autoinstall (module), 69
 labgrid.autoinstall.main (module), 69
 labgrid.binding (module), 122
 labgrid.config (module), 124
 labgrid.consoleloggingreporter (module), 125
 labgrid.driver (module), 70
 labgrid.driver.bareboxdriver (module), 71
 labgrid.driver.commandmixin (module), 72
 labgrid.driver.common (module), 72

- labgrid.driver.consoleexpectmixin (module), 73
 - labgrid.driver.exception (module), 73
 - labgrid.driver.externalconsoledriver (module), 74
 - labgrid.driver.fake (module), 74
 - labgrid.driver.fastbootdriver (module), 75
 - labgrid.driver.infodriver (module), 76
 - labgrid.driver.modbusdriver (module), 76
 - labgrid.driver.onewiredriver (module), 77
 - labgrid.driver.openocddriver (module), 77
 - labgrid.driver.power (module), 70
 - labgrid.driver.power.apc (module), 70
 - labgrid.driver.power.digipower (module), 70
 - labgrid.driver.power.gude (module), 70
 - labgrid.driver.power.gude24 (module), 70
 - labgrid.driver.power.netio (module), 70
 - labgrid.driver.power.netio_kshell (module), 70
 - labgrid.driver.powerdriver (module), 78
 - labgrid.driver.qemudriver (module), 80
 - labgrid.driver.quartushpsdriver (module), 82
 - labgrid.driver.serialdriver (module), 82
 - labgrid.driver.shelldriver (module), 82
 - labgrid.driver.sigrokdriver (module), 85
 - labgrid.driver.smallubootdriver (module), 85
 - labgrid.driver.sshdriver (module), 86
 - labgrid.driver.ubootdriver (module), 87
 - labgrid.driver.usbloader (module), 88
 - labgrid.driver.usbstorage (module), 89
 - labgrid.environment (module), 126
 - labgrid.exceptions (module), 126
 - labgrid.external (module), 89
 - labgrid.external.hawkbite (module), 89
 - labgrid.external.usbstick (module), 91
 - labgrid.factory (module), 127
 - labgrid.protocol (module), 92
 - labgrid.protocol.bootstrapprotocol (module), 92
 - labgrid.protocol.commandprotocol (module), 92
 - labgrid.protocol.consoleprotocol (module), 93
 - labgrid.protocol.digitaloutputprotocol (module), 93
 - labgrid.protocol.filesystemprotocol (module), 94
 - labgrid.protocol.filetransferprotocol (module), 94
 - labgrid.protocol.infoprotocol (module), 94
 - labgrid.protocol.linuxbootprotocol (module), 94
 - labgrid.protocol.mmioprotocol (module), 95
 - labgrid.protocol.powerprotocol (module), 95
 - labgrid.provider (module), 95
 - labgrid.provider.fileprovider (module), 95
 - labgrid.provider.mediafileprovider (module), 95
 - labgrid.pytestplugin (module), 96
 - labgrid.pytestplugin.fixtures (module), 96
 - labgrid.pytestplugin.reporter (module), 96
 - labgrid.remote (module), 97
 - labgrid.remote.authenticator (module), 97
 - labgrid.remote.client (module), 97
 - labgrid.remote.common (module), 97
 - labgrid.remote.config (module), 99
 - labgrid.remote.coordinator (module), 99
 - labgrid.remote.exporter (module), 100
 - labgrid.resource (module), 103
 - labgrid.resource.base (module), 103
 - labgrid.resource.common (module), 104
 - labgrid.resource.ethernetport (module), 106
 - labgrid.resource.modbus (module), 108
 - labgrid.resource.networkservice (module), 108
 - labgrid.resource.onewireport (module), 109
 - labgrid.resource.power (module), 109
 - labgrid.resource.remote (module), 109
 - labgrid.resource.serialport (module), 112
 - labgrid.resource.sigrok (module), 113
 - labgrid.resource.udev (module), 113
 - labgrid.resource.ykushpowerport (module), 117
 - labgrid.step (module), 128
 - labgrid.stepreporter (module), 129
 - labgrid.strategy (module), 117
 - labgrid.strategy.bareboxstrategy (module), 117
 - labgrid.strategy.common (module), 118
 - labgrid.strategy.shellstrategy (module), 118
 - labgrid.strategy.ubootstrategy (module), 119
 - labgrid.target (module), 130
 - labgrid.util (module), 120
 - labgrid.util.dict (module), 120
 - labgrid.util.exceptions (module), 120
 - labgrid.util.expect (module), 120
 - labgrid.util.marker (module), 121
 - labgrid.util.qmp (module), 121
 - labgrid.util.timeout (module), 121
 - labgrid.util.yaml (module), 122
 - LinuxBootProtocol (class in labgrid.protocol.linuxbootprotocol), 94
 - list() (labgrid.provider.fileprovider.FileProvider method), 95
 - list() (labgrid.provider.mediafileprovider.MediaFileProvider method), 95
 - load() (in module labgrid.util.yaml), 122
 - load() (labgrid.driver.openocddriver.OpenOCDDriver method), 77
 - load() (labgrid.driver.usbloader.IMXUSBDriver method), 89
 - load() (labgrid.driver.usbloader.MXSUSBDriver method), 88
 - load() (labgrid.protocol.bootstrapprotocol.BootstrapProtocol method), 92
- ## M
- main() (in module labgrid.autoinstall.main), 70
 - main() (in module labgrid.remote.client), 97
 - main() (in module labgrid.remote.exporter), 103
 - make_driver() (labgrid.factory.TargetFactory method), 128

- make_resource() (labgrid.factory.TargetFactory method), 128
- make_target() (labgrid.factory.TargetFactory method), 128
- ManagedResource (class in labgrid.resource.common), 105
- Manager (class in labgrid.autoinstall.main), 69
- manager_cls (labgrid.resource.common.ManagedResource attribute), 105
- manager_cls (labgrid.resource.ethernetport.SNMPEthernetPort attribute), 107
- manager_cls (labgrid.resource.remote.RemotePlace attribute), 110
- manager_cls (labgrid.resource.remote.RemoteUSBResource attribute), 110
- manager_cls (labgrid.resource.udev.USBResource attribute), 113
- ManualPowerDriver (class in labgrid.driver.powerdriver), 78
- MediaFileProvider (class in labgrid.provider.mediafileprovider), 95
- merge() (labgrid.step.StepEvent method), 129
- MMIOProtocol (class in labgrid.protocol.mmioprotocol), 95
- ModbusCoilDriver (class in labgrid.driver.modbusdriver), 76
- ModbusTCPCoil (class in labgrid.resource.modbus), 108
- model_id (labgrid.resource.udev.USBResource attribute), 114
- monitor_command() (labgrid.driver.qemudriver.QEMUDriver method), 81
- mounted (labgrid.external.usbstick.USBStatus attribute), 91
- MXSUSBDriver (class in labgrid.driver.usbloader), 88
- MXSUSBLoader (class in labgrid.resource.udev), 115
- ## N
- name (labgrid.remote.coordinator.RemoteSession attribute), 100
- NetworkAlteraUSBBlaster (class in labgrid.resource.remote), 111
- NetworkAndroidFastboot (class in labgrid.resource.remote), 110
- NetworkIMXUSBLoader (class in labgrid.resource.remote), 110
- NetworkMXSUSBLoader (class in labgrid.resource.remote), 110
- NetworkPowerDriver (class in labgrid.driver.powerdriver), 79
- NetworkPowerPort (class in labgrid.resource.power), 109
- NetworkResource (class in labgrid.resource.common), 105
- NetworkSerialPort (class in labgrid.resource.serialport), 112
- NetworkService (class in labgrid.resource.networkservice), 108
- NetworkSigrokUSBDevice (class in labgrid.resource.remote), 111
- NetworkUSBMassStorage (class in labgrid.resource.remote), 111
- NetworkUSBPowerPort (class in labgrid.resource.remote), 112
- NetworkUSBSDMuxDevice (class in labgrid.resource.remote), 112
- NoConfigFoundError, 126
- NoDriverFoundError, 127
- NoResourceFoundError, 127
- normalize_config() (labgrid.factory.TargetFactory method), 128
- NoSupplierFoundError, 127
- notify() (labgrid.consoleloggingreporter.ConsoleLoggingReporter method), 126
- notify() (labgrid.pytestplugin.reporter.StepReporter method), 96
- notify() (labgrid.step.Steps method), 128
- notify() (labgrid.stepreporter.StepReporter method), 129
- notify_console_match() (labgrid.protocol.consoleprotocol.ConsoleProtocol.Client method), 93
- NoValidDriverError, 120
- ## O
- off (labgrid.strategy.shellstrategy.Status attribute), 119
- off() (labgrid.driver.fake.FakePowerDriver method), 75
- off() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
- off() (labgrid.driver.powerdriver.ExternalPowerDriver method), 78
- off() (labgrid.driver.powerdriver.ManualPowerDriver method), 78
- off() (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
- off() (labgrid.driver.powerdriver.USBPowerDriver method), 80
- off() (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
- off() (labgrid.driver.qemudriver.QEMUDriver method), 81
- off() (labgrid.protocol.powerprotocol.PowerProtocol method), 95
- on() (labgrid.driver.fake.FakePowerDriver method), 75
- on() (labgrid.driver.powerdriver.DigitalOutputPowerDriver method), 79
- on() (labgrid.driver.powerdriver.ExternalPowerDriver method), 78

- [on\(\)](#) (labgrid.driver.powerdriver.ManualPowerDriver method), 78
[on\(\)](#) (labgrid.driver.powerdriver.NetworkPowerDriver method), 79
[on\(\)](#) (labgrid.driver.powerdriver.USBPowerDriver method), 80
[on\(\)](#) (labgrid.driver.powerdriver.YKUSHPowerDriver method), 80
[on\(\)](#) (labgrid.driver.qemudriver.QEMUDriver method), 81
[on\(\)](#) (labgrid.protocol.powerprotocol.PowerProtocol method), 95
[on_activate\(\)](#) (labgrid.binding.BindingMixin method), 123
[on_activate\(\)](#) (labgrid.driver.bareboxdriver.BareboxDriver method), 71
[on_activate\(\)](#) (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
[on_activate\(\)](#) (labgrid.driver.qemudriver.QEMUDriver method), 81
[on_activate\(\)](#) (labgrid.driver.serialdriver.SerialDriver method), 82
[on_activate\(\)](#) (labgrid.driver.shelldriver.ShellDriver method), 83
[on_activate\(\)](#) (labgrid.driver.sigrokdriver.SigrokDriver method), 85
[on_activate\(\)](#) (labgrid.driver.sshdriver.SSHDriver method), 86
[on_activate\(\)](#) (labgrid.driver.ubootdriver.UBootDriver method), 87
[on_activate\(\)](#) (labgrid.driver.usbloader.IMXUSBDriver method), 89
[on_activate\(\)](#) (labgrid.driver.usbloader.MXSUSBDriver method), 88
[on_activate\(\)](#) (labgrid.driver.usbstorage.USBStorageDriver method), 89
[on_activate\(\)](#) (labgrid.strategy.common.Strategy method), 118
[on_client_bound\(\)](#) (labgrid.binding.BindingMixin method), 123
[on_client_bound\(\)](#) (labgrid.strategy.common.Strategy method), 118
[on_deactivate\(\)](#) (labgrid.binding.BindingMixin method), 123
[on_deactivate\(\)](#) (labgrid.driver.bareboxdriver.BareboxDriver method), 71
[on_deactivate\(\)](#) (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
[on_deactivate\(\)](#) (labgrid.driver.fastbootdriver.AndroidFastbootDriver method), 76
[on_deactivate\(\)](#) (labgrid.driver.qemudriver.QEMUDriver method), 81
[on_deactivate\(\)](#) (labgrid.driver.quartushpsdriver.QuartusHPSDriver method), 82
[on_deactivate\(\)](#) (labgrid.driver.serialdriver.SerialDriver method), 82
[on_deactivate\(\)](#) (labgrid.driver.shelldriver.ShellDriver method), 83
[on_deactivate\(\)](#) (labgrid.driver.sigrokdriver.SigrokDriver method), 85
[on_deactivate\(\)](#) (labgrid.driver.sshdriver.SSHDriver method), 86
[on_deactivate\(\)](#) (labgrid.driver.ubootdriver.UBootDriver method), 87
[on_deactivate\(\)](#) (labgrid.driver.usbloader.IMXUSBDriver method), 89
[on_deactivate\(\)](#) (labgrid.driver.usbloader.MXSUSBDriver method), 88
[on_deactivate\(\)](#) (labgrid.driver.usbstorage.USBStorageDriver method), 89
[on_deactivate\(\)](#) (labgrid.strategy.common.Strategy method), 118
[on_resource_added\(\)](#) (labgrid.resource.common.ResourceManager method), 105
[on_resource_added\(\)](#) (labgrid.resource.ethernetport.EthernetPortManager method), 107
[on_resource_added\(\)](#) (labgrid.resource.remote.RemotePlaceManager method), 109
[on_resource_added\(\)](#) (labgrid.resource.udev.UdevManager method), 113
[on_supplier_bound\(\)](#) (labgrid.binding.BindingMixin method), 123
[OneWirePIO](#) (class in labgrid.resource.onewireport), 109
[OneWirePIODriver](#) (class in labgrid.driver.onewiredriver), 77
[open\(\)](#) (labgrid.driver.externalconsoledriver.ExternalConsoleDriver method), 74
[open\(\)](#) (labgrid.driver.fake.FakeConsoleDriver method), 74
[open\(\)](#) (labgrid.driver.serialdriver.SerialDriver method), 82
[OpenOCDDriver](#) (class in labgrid.driver.openocddriver), 77
- ## P
- [params](#) (labgrid.remote.common.ResourceEntry attribute), 97
[parent](#) (labgrid.resource.common.Resource attribute), 104
[path](#) (labgrid.resource.udev.USBMassStorage attribute), 114
[path](#) (labgrid.resource.udev.USBResource attribute), 114
[path](#) (labgrid.resource.udev.USBSDMuxDevice attribute), 116

- Place (class in labgrid.remote.common), 98
 - plug_in() (labgrid.external.usbstick.USBStick method), 91
 - plug_out() (labgrid.external.usbstick.USBStick method), 91
 - plugged (labgrid.external.usbstick.USBStatus attribute), 91
 - poll() (labgrid.remote.exporter.ResourceExport method), 101
 - poll() (labgrid.resource.common.ManagedResource method), 106
 - poll() (labgrid.resource.common.ResourceManager method), 105
 - poll() (labgrid.resource.ethernetport.EthernetPortManager method), 107
 - poll() (labgrid.resource.remote.RemotePlaceManager method), 109
 - poll() (labgrid.resource.udev.UdevManager method), 113
 - pop() (labgrid.step.Steps method), 128
 - post() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - post_binary() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - post_json() (labgrid.external.hawkbit.HawkbitTestClient method), 90
 - PowerProtocol (class in labgrid.protocol.powerprotocol), 95
 - PowerResetMixin (class in labgrid.driver.powerdriver), 78
 - priorities (labgrid.driver.powerdriver.PowerResetMixin attribute), 78
 - PtxExpect (class in labgrid.util.expect), 120
 - push() (labgrid.step.Steps method), 128
 - put() (labgrid.driver.fake.FakeFileTransferDriver method), 75
 - put() (labgrid.driver.shelldriver.ShellDriver method), 83
 - put() (labgrid.driver.sshdriver.SSHDriver method), 87
 - put() (labgrid.protocol.filetransferprotocol.FileTransferProtocol method), 94
 - put_bytes() (labgrid.driver.shelldriver.ShellDriver method), 83
 - put_file() (labgrid.external.usbstick.USBStick method), 91
 - put_ssh_key() (labgrid.driver.shelldriver.ShellDriver method), 83
 - pytest_addoption() (in module labgrid.pytestplugin.fixtures), 96
 - pytest_configure() (in module labgrid.pytestplugin.reporter), 96
 - pytest_runtest_logreport() (labgrid.pytestplugin.reporter.StepReporter method), 96
 - pytest_runtest_logstart() (labgrid.pytestplugin.reporter.StepReporter method), 96
 - Python Enhancement Proposals PEP 8, 59
- ## Q
- QEMUDriver (class in labgrid.driver.qemudriver), 80
 - QMPErrors, 121
 - QMPMonitor (class in labgrid.util.qmp), 121
 - QuartusHPSDriver (class in labgrid.driver.quartushpsdriver), 82
- ## R
- RawSerialPort (class in labgrid.resource.serialport), 112
 - read() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
 - read() (labgrid.protocol.consoleprotocol.ConsoleProtocol method), 93
 - read() (labgrid.protocol.filesystemprotocol.FileSystemProtocol method), 94
 - read() (labgrid.protocol.mmioprotocol.MMIOProtocol method), 95
 - read_attr() (labgrid.resource.udev.USBResource method), 114
 - read_nonblocking() (labgrid.util.expect.PtxExpect method), 121
 - reg_driver() (labgrid.factory.TargetFactory method), 128
 - reg_resource() (labgrid.factory.TargetFactory method), 128
 - remaining (labgrid.util.timeout.Timeout attribute), 121
 - RemotePlace (class in labgrid.resource.remote), 110
 - RemotePlaceManager (class in labgrid.resource.remote), 109
 - RemoteSession (class in labgrid.remote.coordinator), 100
 - RemoteUSBResource (class in labgrid.resource.remote), 110
 - reset() (labgrid.driver.bareboxdriver.BareboxDriver method), 71
 - reset() (labgrid.driver.powerdriver.PowerResetMixin method), 78
 - reset() (labgrid.driver.ubootdriver.UBootDriver method), 88
 - reset() (labgrid.protocol.linuxbootprotocol.LinuxBootProtocol method), 94
 - resolve_conflicts() (labgrid.binding.BindingMixin method), 123
 - resolve_conflicts() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
 - resolve_conflicts() (labgrid.strategy.common.Strategy method), 118
 - resolve_path() (labgrid.config.Config method), 124
 - resolve_templates() (in module labgrid.util.yaml), 122
 - Resource (class in labgrid.resource.common), 104
 - ResourceConfig (class in labgrid.remote.config), 99

- ResourceEntry (class in labgrid.remote.common), 97
- ResourceExport (class in labgrid.remote.exporter), 100
- ResourceManager (class in labgrid.resource.common), 105
- ResourceMatch (class in labgrid.remote.common), 98
- run() (labgrid.autoinstall.main.Handler method), 69
- run() (labgrid.driver.bareboxdriver.BareboxDriver method), 71
- run() (labgrid.driver.fake.FakeCommandDriver method), 75
- run() (labgrid.driver.shelldriver.ShellDriver method), 83
- run() (labgrid.driver.sshdriver.SSHDriver method), 86
- run() (labgrid.driver.ubootdriver.UBootDriver method), 87
- run() (labgrid.protocol.commandprotocol.CommandProtocol method), 92
- run_check() (labgrid.driver.commandmixin.CommandMixin method), 72
- run_check() (labgrid.driver.fake.FakeCommandDriver method), 75
- run_check() (labgrid.protocol.commandprotocol.CommandProtocol method), 92
- run_once() (labgrid.autoinstall.main.Handler method), 69
- run_script() (labgrid.driver.shelldriver.ShellDriver method), 84
- run_script_file() (labgrid.driver.shelldriver.ShellDriver method), 84
- S**
- send() (labgrid.util.expect.PtxExpect method), 121
- sendcontrol() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
- sendcontrol() (labgrid.protocol.consoleprotocol.ConsoleProtocol method), 93
- sendline() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73
- sendline() (labgrid.protocol.consoleprotocol.ConsoleProtocol method), 93
- SerialDriver (class in labgrid.driver.serialdriver), 82
- SerialPort (class in labgrid.resource.base), 103
- ServerError, 97
- set() (in module labgrid.driver.power.apc), 70
- set() (in module labgrid.driver.power.digipower), 70
- set() (in module labgrid.driver.power.gude), 70
- set() (in module labgrid.driver.power.gude24), 70
- set() (in module labgrid.driver.power.netio), 70
- set() (in module labgrid.driver.power.netio_kshell), 70
- set() (labgrid.driver.modbusdriver.ModbusCoilDriver method), 77
- set() (labgrid.driver.onewiredriver.OneWirePIODriver method), 77
- set() (labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol method), 93
- set_binding_map() (labgrid.target.Target method), 131
- set_option() (labgrid.config.Config method), 125
- set_resource() (labgrid.remote.coordinator.ExporterSession method), 100
- shell (labgrid.strategy.bareboxstrategy.Status attribute), 117
- shell (labgrid.strategy.shellstrategy.Status attribute), 119
- shell (labgrid.strategy.ubootstrategy.Status attribute), 119
- ShellDriver (class in labgrid.driver.shelldriver), 82
- ShellStrategy (class in labgrid.strategy.shellstrategy), 119
- show() (labgrid.remote.common.Place method), 98
- SigrokDevice (class in labgrid.resource.sigrok), 113
- SigrokDriver (class in labgrid.driver.sigrokdriver), 85
- SigrokUSBDevice (class in labgrid.resource.udev), 116
- skip() (labgrid.step.Step method), 129
- SmallUBootDriver (class in labgrid.driver.smallubootdriver), 85
- SNMPEthernetPort (class in labgrid.resource.ethernetport), 107
- SNMPSwitch (class in labgrid.resource.ethernetport), 106
- SSHDriver (class in labgrid.driver.sshdriver), 86
- start() (labgrid.autoinstall.main.Manager method), 70
- start() (labgrid.consoleloggingreporter.ConsoleLoggingReporter class method), 125
- start() (labgrid.step.Step method), 129
- start() (labgrid.stepreporter.StepReporter class method), 129
- start_rollout() (labgrid.external.hawkbit.HawkbitTestClient method), 90
- start_session() (in module labgrid.remote.client), 97
- StateError, 92, 122
- Status (class in labgrid.strategy.bareboxstrategy), 117
- Status (class in labgrid.strategy.shellstrategy), 118
- Status (class in labgrid.strategy.ubootstrategy), 119
- status (labgrid.step.Step attribute), 129
- Step (class in labgrid.step), 129
- step() (in module labgrid.step), 129
- StepEvent (class in labgrid.step), 128
- StepReporter (class in labgrid.pytestplugin.reporter), 96
- StepReporter (class in labgrid.stepreporter), 129
- Steps (class in labgrid.step), 128
- stop() (labgrid.consoleloggingreporter.ConsoleLoggingReporter class method), 126
- stop() (labgrid.driver.sigrokdriver.SigrokDriver method), 85
- stop() (labgrid.step.Step method), 129
- stop() (labgrid.stepreporter.StepReporter class method), 129
- Strategy (class in labgrid.strategy.common), 118
- StrategyError, 117–119
- subscribe() (labgrid.step.Steps method), 128
- switch_image() (labgrid.external.usbstick.USBStick method), 91

T

Target (class in labgrid.target), 130

target() (in module labgrid.pytestplugin.fixtures), 96

target_factory (in module labgrid.factory), 128

TargetFactory (class in labgrid.factory), 127

Timeout (class in labgrid.util.timeout), 121

touch() (labgrid.remote.common.Place method), 99

transition() (labgrid.strategy.bareboxstrategy.BareboxStrategy method), 118

transition() (labgrid.strategy.shellstrategy.ShellStrategy method), 119

transition() (labgrid.strategy.ubootstrategy.UBootStrategy method), 120

try_match() (labgrid.resource.udev.USBResource method), 114

U

uboot (labgrid.strategy.ubootstrategy.Status attribute), 119

UBootDriver (class in labgrid.driver.ubootdriver), 87

UBootStrategy (class in labgrid.strategy.ubootstrategy), 119

UdevManager (class in labgrid.resource.udev), 113

under() (in module labgrid.pytestplugin.reporter), 96

unknown (labgrid.strategy.bareboxstrategy.Status attribute), 117

unknown (labgrid.strategy.shellstrategy.Status attribute), 119

unknown (labgrid.strategy.ubootstrategy.Status attribute), 119

unplugged (labgrid.external.usbstick.USBStatus attribute), 91

unsubscribe() (labgrid.step.Steps method), 128

UPD (labgrid.remote.coordinator.Action attribute), 99

update() (labgrid.resource.ethernetport.SNMPSwitch method), 106

update() (labgrid.resource.udev.USBEthernetInterface method), 115

update() (labgrid.resource.udev.USBResource method), 114

update() (labgrid.resource.udev.USBSerialPort method), 114

update_resources() (labgrid.target.Target method), 131

upload_image() (labgrid.external.usbstick.USBStick method), 91

USBEthernetExport (class in labgrid.remote.exporter), 101

USBEthernetInterface (class in labgrid.resource.udev), 115

USBGenericExport (class in labgrid.remote.exporter), 101

USBMassStorage (class in labgrid.resource.udev), 114

USBPowerDriver (class in labgrid.driver.powerdriver), 80

USBPowerPort (class in labgrid.resource.udev), 116

USBPowerPortExport (class in labgrid.remote.exporter), 102

USBResource (class in labgrid.resource.udev), 113

USBSDMuxDevice (class in labgrid.resource.udev), 116

USBSDMuxExport (class in labgrid.remote.exporter), 102

USBSerialPort (class in labgrid.resource.udev), 114

USBSerialPortExport (class in labgrid.remote.exporter), 101

USBSigrokExport (class in labgrid.remote.exporter), 102

USBStatus (class in labgrid.external.usbstick), 91

USBStick (class in labgrid.external.usbstick), 91

USBStorageDriver (class in labgrid.driver.usbstorage), 89

UserError, 97

V

vendor_id (labgrid.resource.udev.USBResource attribute), 114

W

wait_for() (labgrid.driver.commandmixin.CommandMixin method), 72

wait_for() (labgrid.protocol.commandprotocol.CommandProtocol method), 92

write() (labgrid.driver.consoleexpectmixin.ConsoleExpectMixin method), 73

write() (labgrid.protocol.consoleprotocol.ConsoleProtocol method), 93

write() (labgrid.protocol.filesystemprotocol.FileSystemProtocol method), 94

write() (labgrid.protocol.mmioprotocol.MMIOProtocol method), 95

write_image() (labgrid.driver.usbstorage.USBStorageDriver method), 89

Y

YKUSHPowerDriver (class in labgrid.driver.powerdriver), 79

YKUSHPowerPort (class in labgrid.resource.ykushpowerport), 117