labgrid Documentation

Release 0.1.dev127+g57f2eaa

Jan Luebbe, Rouven Czerwinski

Oct 30, 2019

CONTENTS

Labgrid is a 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.

CHAPTER

ONE

GETTING STARTED

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 virtualenv
```

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.

Note: If you are installing via pip and intend to use Serial over IP (RFC2217), it is highly recommended to uninstall pyserial after installation and replace it with the pyserial version from the labgrid project:

```
$ pip uninstall pyserial
$ pip install https://github.com/labgrid-project/pyserial/archive/v3.4.0.1.
↪zip#egg=pyserial
```

This pyserial version has two fixes for an Issue we found with Serial over IP multiplexers. Additionally it reduces the Serial over IP traffic considerably since the port is not reconfigured when labgrid changes the timeout (which is done inside the library a lot).

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 seperate 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:
```

(continues on next page)

(continued from previous page)

```
        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. crossbar is a network messaging framework for building distributed applications, which labgrid plugs into.

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.

Attention: The *ManagedFile* will create temporary uploads in the exporters `/var/cache/labgrid` directory. This directory needs to be created manually and should allow write access for users. The `/contrib` directory in the labgrid-project contains a tmpfiles configuration example to automatically create and clean the directory. It is also highly recommended to enable `fs.protected_regular=1` and `fs.protected_fifos=1` for kernels>=4.19, to protect the users from opening files not owned by them in world writeable sticky directories. For more information see this kernel commit.

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
```

See *Remote Access* for some more advanced features. 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 es 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*.

CHAPTER

TWO

OVERVIEW

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` (*bound/active*). 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)
```

(continues on next page)

(continued from previous page)

```
>>> 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.

The following sections describe the resposibilities of each component. See *Remote Access* for usage information.

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.

To support selecting a specific place from a group containing similar or identical hardware, key-value tags can be added to places and used for scheduling.

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 *aquired* 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.

2.2.5 Proxy Mechanism

Both client and exporter support the proxy mechanism which uses SSH to tunnel connections to a remote host. To enable and force proxy mode on the exporter use the `-i` or `--isolated` command line option. This indicates to clients that all connections to remote resources made available by this exporter need to be tunneled using a SSH connection. On the other hand, clients may need to access the remote infrastrucure using a SSH tunnel. In this case the `LG_PROXY` environment variable needs to be set to the remote host which should tunnel the connections. The client

than forwards all network traffic through SSH, even the coordinator connection is forwarded. This means that with `LG_PROXY` and `LG_CROSSBAR` labgrid can be used fully remotely with only a SSH connection as a requirement. One remaining issue here is the forward of UDP connections, which is currently not possible. UDP connections are used by some of the power backends in the form of SNMP.

CHAPTER

THREE

USAGE

3.1 Remote Access

As described in *Remote Resources and Places*, one of labgrid's main features is making access to boards connected to other hosts transparent for the client. To get started with remote access, take a look at *Setting Up the Distributed Infrastructure*.

3.1.1 Place Scheduling

When sharing places between developers or with CI jobs, it soon becomes necessary to manage who can access which places. Developers often just need any place which has one of a group of identical devices, while CI jobs should wait until the necessary place is free instead of failing.

To support these use-cases, the coordinator has support for reserving places by using a tag filter and an optional priority. First, the places have to be tagged with the relevant key-value pairs:

```
$ labgrid-client -p board-1 set-tags board=imx6-foo
$ labgrid-client -p board-2 set-tags board=imx6-foo
$ labgrid-client -p board-3 set-tags board=imx8m-bar
$ labgrid-client -v places
Place 'board-1':
  tags: bar=baz, board=imx6-foo, jlu=2, rcz=1
  matches:
    rl-test/Testport1/NetworkSerialPort
...
Place 'board-2':
  tags: board=imx6-foo
  matches:
    rl-test/Testport2/NetworkSerialPort
...
Place 'board-3':
  tags: board=imx8m-bar
  matches:
    rl-test/Testport3/NetworkSerialPort
...
```

Now, if you want to access any `imx6-foo` board, you could find that all are already in use by someone else:

```
$ labgrid-client who
User     Host     Place    Changed
rcz      dude     board-1  2019-08-06 12:14:38.446201
jenkins  worker1  board-2  2019-08-06 12:52:44.762131
```

In this case, you can create a reservation. You can specify any custom tags as part of the filter, as well as `name=<place-name>` to select only a specific place (even if it has no custom tags).

```
$ labgrid-client reserve board=imx6-foo
Reservation 'SP37P5OQRU':
  owner: rettich/jlu
  token: SP37P5OQRU
  state: waiting
  filters:
    main: board=imx6-foo
  created: 2019-08-06 12:56:49.779982
  timeout: 2019-08-06 12:57:49.779983
```

As soon as any matching place becomes free, the reservation state will change from `waiting` to `allocated`. Then, you can use the reservation token prefixed by `+` to refer to the allocated place for locking and usage. While a place is allocated for a reservation, only the owner of the reservation can lock that place.

```
$ labgrid-client wait SP37P5OQRU
owner: rettich/jlu
token: SP37P5OQRU
state: waiting
filters:
  main: board=imx6-foo
created: 2019-08-06 12:56:49.779982
timeout: 2019-08-06 12:58:14.900621
...
owner: rettich/jlu
token: SP37P5OQRU
state: allocated
filters:
  main: board=imx6-foo
allocations:
  main: board-2
created: 2019-08-06 12:56:49.779982
timeout: 2019-08-06 12:58:46.145851
$ labgrid-client -p +SP37P5OQRU lock
acquired place board-2
$ labgrid-client reservations
Reservation 'SP37P5OQRU':
  owner: rettich/jlu
  token: SP37P5OQRU
  state: acquired
  filters:
    main: board=imx6-foo
  allocations:
    main: board-2
  created: 2019-08-06 12:56:49.779982
  timeout: 2019-08-06 12:59:11.840780
$ labgrid-client -p +SP37P5OQRU console
```

When using reservation in a CI job or to save some typing, the `labgrid-client reserve` command supports a `--shell` command to print code for evaluating in the shell. This sets the `LG_TOKEN` environment variable, which is then automatically used by `wait` and expanded via `-p +`.

```
$ eval `labgrid-client reserve --shell board=imx6-foo`
$ echo $LG_TOKEN
ZDMZJZNLBF
$ labgrid-client wait
```

(continues on next page)

(continued from previous page)

```
owner: rettich/jlu
token: ZDMZJZNLBF
state: waiting
filters:
  main: board=imx6-foo
created: 2019-08-06 13:05:30.987072
timeout: 2019-08-06 13:06:44.629736
...
owner: rettich/jlu
token: ZDMZJZNLBF
state: allocated
filters:
  main: board=imx6-foo
allocations:
  main: board-1
created: 2019-08-06 13:05:30.987072
timeout: 2019-08-06 13:06:56.196684
$ labgrid-client -p + lock
acquired place board-1
$ labgrid-client -p + show
Place 'board-1':
  tags: bar=baz, board=imx6-foo, jlu=2, rcz=1
  matches:
    rettich/Testport1/NetworkSerialPort
  acquired: rettich/jlu
  acquired resources:
  created: 2019-07-29 16:11:52.006269
  changed: 2019-08-06 13:06:09.667682
  reservation: ZDMZJZNLBF
```

Finally, to avoid calling the `wait` command explicitly, you can add `--wait` to the `reserve` command, so it waits until the reservation is allocated before returning.

A reservation will time out after a short time, if it is neither refreshed nor used by locked places.

3.2 Library

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

3.2.1 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, is 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.

For more information on the environment configuration files and the usage of multiple drivers, see *Environment Configuration*.

3.3 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, or the LG_ENV environment variable) 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.

3.3.1 Command-Line Options

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.

Other labgrid-related pytest plugin options are:

`--lg-env=LG_ENV` (was `--env-config=ENV_CONFIG`) Specify a labgrid environment config file. This is equivalent to labgrid-client's `-c`/`--config`.

`--lg-coordinator=CROSSBAR_URL` Specify labgrid coordinator websocket URL. Defaults to `ws://127.0.0.1:20408/ws`. This is equivalent to labgrid-client's `-x`/`--crossbar`.

`--lg-log=[path to logfiles]` Path to store console log file. If option is specified without path the current working directory is used.

`--lg-colored-steps` Enables the ColoredStepReporter. Different events have different colors. The more colorful, the more important. In order to make less important output "blend into the background" different color schemes are available. See *LG_COLOR_SCHEME*.

`pytest --help` shows these options in a separate *labgrid* section.

3.3.2 Environment Variables

LG_ENV

Behaves like `LG_ENV` for *labgrid-client*.

LG_COLOR_SCHEME

Influences the color scheme used for the Colored Step Reporter. `dark` (default) is meant for dark terminal background. `light` is optimized for light terminal background. Takes effect only when used with `--lg-colored-steps`.

LG_PROXY

Specifies a SSH proxy host to be used for port forwards to access the coordinator and network resources.

3.3.3 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.

3.3.4 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 `conftest.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
```

3.3.5 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 `conftest.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 switch_off(target, strategy, capsys):
    with capsys.disabled():
        strategy.transition('off')

@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 ===========================
```

3.3.6 Feature Flags

Labgrid includes support for feature flags on a global and target scope. They will be concatenated and compared to a pytest mark on the test to decide whether the test can run with the available features.:

```
import pytest

@pytest.mark.lg_feature("camera")
def test_camera(target):
    [...]
```

together with an example environment configuration:

```
targets:
  main:
    features:
      - camera
    resources: {}
    drivers: {}
```

would run the above test, however the following configuration would skip the test because of the missing feature:

```
targets:
  main:
    features:
      - console
    resources: {}
    drivers: {}
```

This is also reported in the pytest execution as a skipped test with the reason being the missing feature.

For tests with multiple required features, pass them as a list to pytest::

```
import pytest

@pytest.mark.lg_feature(["camera", "console"])
def test_camera(target):
    [...]
```

Features do not have to be set per target, they can also be set via the global features key:

```
features:
  - camera
targets:
  main:
    features:
      - console
    resources: {}
    drivers: {}
```

This yaml would combine both the global and the target features.

3.3.7 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

3.4 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.

3.5 USB stick emulation

Labgrid makes it posible 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 retrived using the *put_file* and *get_file* methods. The *plug_in* and *plug_out* functions plug the emulated USB stick in and out.

3.6 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.

CHAPTER

FOUR

MANUAL PAGES

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

-v, --verbose increase verbosity

-P PROXY, --proxy PROXY proxy connections over ssh

CONFIGURATION FILE

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

ENVIRONMENT VARIABLES

Various labgrid-client commands use the following environment variable:

LG_PLACE

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

LG_TOKEN

This variable can be used to specify a reservation for the `wait` command and for the `+` place expansion.

LG_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_ENV

This variable can be used to specify the configuration file to use without using the `--config` option, the `--config` option overrides it.

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`.

LG_PROXY

This variable can be used to specify a SSH proxy hostname which should be used to connect to the coordinator and any resources which are normally accessed directly.

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

`who` List acquired places by user

`show` Show a place and related resources

`create` Add a new place (name supplied by -p parameter)

`delete` Delete an existing place

`add-alias` alias Add an alias to a place

`del-alias` alias Delete an alias from a place

`set-comment` comment Update or set the place comment

`set-tags` comment Set place tags (key=value)

`add-match` match Add one (or multiple) match pattern(s) to a place, see MATCHES

`del-match` match Delete one (or multiple) match pattern(s) from a place, see MATCHES

`add-named-match` match name Add one match pattern with a name to a place

`acquire (lock)` Acquire a place

`allow` user Allow another user to access 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

`io` action Interact with GPIO (OneWire, relays, . . .) devices, where action is one of high, low, get

`console (con)` Connect to the console

`fastboot` arg Run fastboot with argument

`bootstrap` filename Start a bootloader

`sd-mux` action Switch USB SD Muxer, where action is one of dut (device-under-test), host, off

`ssh` Connect via SSH

`telnet` Connect via telnet

`video` Start a video stream

`tmc` command Control a USB TMC device

`write-image` Write images onto block devices (USBSDMux, USB Sticks, . . .)

`reserve` filter Create a reservation

`cancel-reservation` token Cancel a pending reservation

`wait` token Wait for a reservation to be allocated

`reservations` List current reservations

ADDING NAMED RESOURCES

If a target contains multiple Resources of the same type, named matches need to be used to address the individual resources. In addition to the *match* taken by *add-match*, *add-named-match* also takes a name for the resource. The other client commands support the name as an optional parameter and will inform the user that a name is required if multiple resources are found, but no name is given.

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

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.

OPTIONS 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. The image paths can be either relative to the YAML file or absolute.

IMAGE KEYS

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

IMAGE EXAMPLE

Two configured images, one for the root filesystem, one for the bootloader:

```
images:
  root: "platform-v7a/images/root.img"
  boot: "platform-v7a/images/barebox.img"
```

TOOLS

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

TOOLS 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

TOOLS EXAMPLE

Configure the tool path for `imx-usb-loader`:

```
tools:
  imx-usb-loader: "/opt/labgrid-helper/imx-usb-loader"
```

IMPORTS

The `imports` key is a list of files or python modules which are imported by the environment after loading the configuration. Paths relative to the configuration file are also supported. This is useful to load drivers and strategy which are contained in your testsuite, since the import is done before instantiating the targets.

IMPORTS EXAMPLE

Import a local *myfunctions.py* file:

```
imports:
  - myfunctions.py
```

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'
```

A sample configuration with *RemotePlace*, using the tools configuration and importing the local *mystrategy.py* file. The *MyStrategy* strategy is contained in the loaded local python file:

```
targets:
  main:
    resources:
      RemotePlace:
        name: test-place
    drivers:
      SerialDriver: {}
      ShellDriver:
        prompt: 'root@\w+:[^ ]+ '
        login_prompt: ' login: '
        username: 'root'
      MyStrategy: {}
      IMXUSBLoader: {}
tools:
  imx-usb-loader: "/opt/lg-tools/imx-usb-loader"
imports:
  - mystrategy.py
```

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
-i, --isolated	enable isolated mode (always request SSH forwards)
-n, --name	the public name of the exporter
--hostname	hostname (or IP) published for accessing resources
-d, --debug	enable debug mode

-i / –isolated

This option enables isolated mode, which causes all exported resources marked as requiring SSH connection forwarding. Isolated mode is useful when resources (such as NetworkSerialPorts) are not directly accessible from the clients. The client will then use SSH to create a port forward to the resource when needed.

-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 releated 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)

CHAPTER

FIVE

CONFIGURATION

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*

PDUDaemonPort

A PDUDaemonPort describes a PDU port accessible via PDUDaemon. As one PDUDaemon instance can control many PDUs, the instance name from the PDUDaemon configuration file needs to be specified.

```
PDUDaemonPort:
  host: pduserver
  pdu: apc-snmpv3-noauth
  index: 1
```

The example describes port 1 on the PDU configured as *apc-snmpv3-noauth*, with PDUDaemon running on the host *pduserver*.

- host (str): name of the host running the PDUDaemon
- pdu (str): name of the PDU in the configuration file
- index (int): index of the power port on the PDU

Used by:

- *PDUDaemonDriver*

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 `udevadm 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
- port (int): optional, port used by SSH (default 22)

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): optional, 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 RKUSBLoader

An RKUSBLoader resource describes a USB device in the rockchip loader state.

```
RKUSBLoader:
  match:
    'sys_name': '1-3'
```

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

Used by:

- *RKUSBDriver*

5.1.12 NetworkMXSUSBLoader

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

5.1.13 NetworkIMXUSBLoader

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

5.1.14 NetworkRKUSBLoader

A NetworkRKUSBLoader descibes an *RKUSBLoader* available on a remote computer.

5.1.15 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*

5.1.16 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*

5.1.17 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*

5.1.18 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

5.1.19 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): optional, 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*

5.1.20 NetworkSigrokUSBDevice

A NetworkSigrokUSBDevice resource describes a sigrok USB device connected to a host which is exported over the network. The SigrokDriver will access it via SSH.

```
NetworkSigrokUSBDevice:
  driver: fx2lafw
  channel: "D0=CLK,D1=DATA"
  match:
    'ID_PATH': 'pci-0000:06:00.0-usb-0:1.3.2:1.0'
  host: remote.example.computer
```

- driver (str): name of the sigrok driver to use
- channel (str): optional, 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*

5.1.21 SigrokUSBSerialDevice

A SigrokUSBSerialDevice resource describes a sigrok device which communicates of a USB serial port instead of being a USB device itself (see *SigrokUSBDevice* for that case).

```
SigrokUSBSerialDevice:
  driver: manson-hcs-3xxx
  match:
    '@ID_SERIAL_SHORT': P-00-02389
```

- driver (str): name of the sigrok driver to use

Used by:

- *SigrokPowerDriver*

5.1.22 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*

5.1.23 NetworkUSBSDMuxDevice

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

5.1.24 USBVideo

A *USBVideo* resource describes a USB video camera which is supported by a Video4Linux2 kernel driver.

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

Used by:

- *USBVideoDriver*

5.1.25 NetworkUSBVideo

A *NetworkUSBVideo* resource describes a *USBVideo* resource available on a remote computer.

5.1.26 USBTMC

A *USBTMC* resource describes an oscilloscope connected via the USB TMC protocol. The low-level communication is handled by the `usbtmc` kernel driver.

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

A udev rules file may be needed to allow access for non-root users:

```
DRIVERS=="usbtmc", MODE="0660", GROUP="plugdev"
```

Used by:

- *USBTMCDriver*

5.1.27 NetworkUSBTMC

A *NetworkUSBTMC* resource describes a *USBTMC* resource available on a remote computer.

5.1.28 Flashrom

A Flashrom resource is used to configure the parameters to a local installed flashrom instance. It is assumed that flashrom is installed on the host and the executable is configured in:

```
tools:
  flashrom: '/usr/sbin/flashrom'
```

The resource must configure which programmer to use and the parameters to the programmer. The programmer parameter is passed directly to the flashrom bin hence man(8) flashrom can be used for reference. Below an example where the local spidev is used.

```
Flashrom:
  programmer: 'linux_spi:dev=/dev/spidev0.1,spispeed=30000'
```

Used by:

- *FlashromDriver*

5.1.29 NetworkFlashRom

A NetworkFlashrom descibes an *Flashrom* available on a remote computer.

5.1.30 XenaManager

A XenaManager resource describe a Xena Manager instance which is the instance the Xena Driver must connect to in order to configure a Xena chassis.

```
XenaManager:
  hostname: "example.computer"
```

Used by:

- *XenaDriver*

5.1.31 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

5.1.32 DockerDaemon

A DockerDaemon describes where to contact a docker daemon process. DockerDaemon also participates in managing *NetworkService* instances created through interaction with that daemon.

```
DockerDaemon:
  docker_daemon_url: 'unix://var/run/docker.sock'
```

The example describes a docker daemon accessible via the '/var/run/docker.sock' unix socket. When used by a *DockerDriver*, the *DockerDriver* will first create a docker container which the DockerDaemon resource will subsequently use to create one/more *NetworkService* instances - as specified by *DockerDriver* configuration. Each *NetworkService* instance corresponds to a network service running inside the container.

Moreover, DockerDaemon will remove any hanging containers if DockerDaemon is used several times in a row - as is the case when executing test suites. Normally *DockerDriver* - when deactivated - cleans up the created docker

container; programming errors, keyboard interrupts or unix kill signals may lead to hanging containers, however; therefore auto-cleanup is important.

- docker_daemon_url (str): The url of the daemon to use for this target.

Used by:

- *DockerDriver*

5.1.33 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=:ffffff:
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:

port:

- *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:

console:

- *ConsoleProtocol*

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. It uses SSH's ServerAliveInterval option to detect failed connections.

If a shared SSH connection to the target is already open, it will reuse it when running commands. In that case, ServerAliveInterval should be set outside of labgrid, as it cannot be enabled for an existing connection.

Binds to:

networkservice:

- *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)
- **stderr_merge (bool): set to True to make *run()* return stderr merged with** stdout, and an empty list as second element.

5.2.4 UBootDriver

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

Binds to:

console:

- *ConsoleProtocol*

Implements:

- *CommandProtocol*

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

Arguments:

- prompt (regex): u-boot prompt to match
- autoboot (regex, default="stop autoboot"): autoboot message 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
- login_timeout (int): optional, timeout for login prompt detection in seconds (default 60)

5.2.5 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 is 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 is 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+"

- login_timeout (str): optional, timeout for the password/login detection

5.2.6 BareboxDriver

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

Binds to:

console:

- *ConsoleProtocol*

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
- password (str): optional, password to use for access to the shell
- login_timeout (int): optional, timeout for access to the shell

5.2.7 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

5.2.8 AndroidFastbootDriver

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

Binds to:

fastboot:

- *AndroidFastboot*

Implements:

- None (yet)

```
AndroidFastbootDriver:
  image: mylocal.image
  sparse_size: 100M
```

Arguments:

- image (str): filename of the image to upload to the device
- sparse_size (str): optional, sparse files greater than given size (see fastboot manpage -S option for allowed size suffixes). The default is the same as the fastboot default, which is computed after querying the target's `max-download-size` variable.

5.2.9 OpenOCDDriver

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

Note that OpenOCD supports specifying USB paths since a1b308ab which is not part of a release yet. The OpenOCDDriver passes the resource's USB path. Depending on which OpenOCD version is installed it is either used correctly or a warning is displayed and the first resource seen is used, which might be the wrong USB device. Consider updating your OpenOCD version when using multiple USB Blasters.

Binds to:

interface:

- *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

5.2.10 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.

5.2.11 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)

5.2.12 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

5.2.13 NetworkPowerDriver

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

Binds to:

port:

- *NetworkPowerPort*

Implements:

- *PowerProtocol*

```
NetworkPowerDriver:
  delay: 5.0
```

Arguments:

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

5.2.14 PDUDaemonDriver

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

Note: PDUDaemon processess commands in the background, so the actual state change may happen several seconds after calls to PDUDaemonDriver return.

Binds to:

port:

- *PDUDaemonPort*

Implements:

- *PowerProtocol*

```
PDUDaemonDriver:
  delay: 5
```

Arguments:

- delay (int): 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:

port:

- *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:

output:

- *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 GpioDigitalOutputDriver

The GpioDigitalOutputDriver writes a digital signal to a GPIO line.

This driver configures GPIO lines via *the sysfs kernel interface <https://www.kernel.org/doc/html/latest/gpio/sysfs.html>*. While the driver automatically exports the GPIO, it does not configure it in any other way than as an output.

Implements:

- *DigitalOutputProtocol*

```
GpioDigitalOutputDriver:
  index: 42
```

Arguments:

- index (int): The index of a GPIO line

5.2.19 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).

5.2.20 FileDigitalOutputDriver

The FileDigitalOutputDriver uses a file to write arbitrary string representations of booleans to a file and read from it.

The file is checked to exist at configuration time.

If the file's content does not match any of the representations reading defaults to False.

A prime example for using this driver is Linux's sysfs.

Implements:

- *DigitalOutputProtocol*

```
FileDigitalOutputDriver:
  filepath: "/sys/class/leds/myled/brightness"
```

Arguments:

- filepath (str): A file that is used for reads and writes.
- false_repr (str): A representation for False (default: "0n")
- true_repr (str): A representation for True (default: "1n")

5.2.21 ModbusCoilDriver

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

Binds to:

coil:

- *ModbusTCPCoil*

Implements:

- *DigitalOutputProtocol*

```
ModbusCoilDriver: {}
```

Arguments:

- None

5.2.22 MXSUSBDriver

A MXUSBDriver 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:

loader:

- *MXSUSBLoader*

- *NetworkMXSUSBLoader*

Implements:

- *BootstrapProtocol*

```
targets:
  main:
    drivers:
      MXSUSBDriver:
        image: mybootloaderkey

images:
  mybootloaderkey: path/to/mybootloader.img
```

Arguments:

- image (str): The key in *images* containing the path of an image to bootstrap onto the target

5.2.23 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:

loader:

- *IMXUSBLoader*
- *NetworkIMXUSBLoader*

Implements:

- *BootstrapProtocol*

```
targets:
  main:
    drivers:
      IMXUSBDriver:
        image: mybootloaderkey

images:
  mybootloaderkey: path/to/mybootloader.img
```

Arguments:

- image (str): The key in *images* containing the path of an image to bootstrap onto the target

5.2.24 RKUSBDriver

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

Binds to:

loader:

- *RKUSBLoader*
- *NetworkRKUSBLoader*

Implements:

- *BootstrapProtocol*

```
targets:
  main:
    drivers:
      RKUSBDriver:
        image: mybootloaderkey
        usb_loader: myloaderkey

images:
  mybootloaderkey: path/to/mybootloader.img
  myloaderkey: path/to/myloader.bin
```

Arguments:

- image (str): The key in *images* containing the path of an image to bootstrap onto the target
- usb_loader (srt): The key in *images* containing the path of an image to bootstrap onto the target

5.2.25 USBStorageDriver

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

Binds to:

storage:

- *USBMassStorage*

Implements:

- None (yet)

```
USBStorageDriver: {}
```

Arguments:

- None

5.2.26 NetworkUSBStorageDriver

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

Binds to:

- *USBMassStorage*
- *NetworkUSBMassStorage*

Implements:

- None (yet)

```
NetworkUSBStorageDriver:
  image: flashimage
```

```
images:
  flashimage: ../images/myusb.image
```

Arguments:

- image (str): filename of the image to write to the remote usb storage

5.2.27 OneWirePIODriver

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

Binds to:

port:

- *OneWirePIO*

Implements:

- *DigitalOutputProtocol*

```
OneWirePIODriver: {}
```

Arguments:

- None

5.2.28 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

5.2.29 SigrokDriver

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

Binds to:

sigrok:

- *SigrokUSBDevice*
- *SigrokDevice*
- *NetworkSigrokUSBDevice*

Implements:

- None yet

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

5.2.30 SigrokPowerDriver

The SigrokPowerDriver uses a *SigrokUSBSerialDevice* Resource to control a programmable power supply.

Binds to:

sigrok:

- *SigrokUSBSerialDevice*
- NetworkSigrokUSBSerialDevice

Implements:

- `PowerProtocol`

```
SigrokPowerDriver:
  delay: 3.0
```

Arguments:

- delay (float): optional delay in seconds between off and on, defaults to 3.0
- max_voltage (float): maximum allowed voltage for protection against accidental damage (optional, in volts)
- max_current (float): maximum allowed current for protection against accidental damage (optional, in ampere)

5.2.31 USBSDMuxDriver

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

5.2.32 USBVideoDriver

The `USBVideoDriver` is used to show a video stream from a remote USB video camera in a local window. It uses the GStreamer command line utility `gst-launch` on both sides to stream the video via an SSH connection to the exporter.

Binds to:

video:

- *USBVideo*
- *NetworkUSBVideo*

Implements:

- None yet

Although the driver can be used from Python code by calling the *stream()* method, it is currenly mainly useful for the `video` subcommand of `labgrid-client`. It supports the *Logitech HD Pro Webcam C920* with the USB ID 046d:082d, but other cameras can be added to *get_caps()* in `labgrid/driver/usbvideodriver.py`.

5.2.33 USBTMCDriver

The `USBTMCDriver` is used to control a oscilloscope via the USB TMC protocol.

Binds to:

tmc:

- *USBTMC*
- *NetworkUSBTMC*

Implements:

- None yet

Currently, it can be used by the `labgrid-client tmc` subcommands to show (and save) a screenshot, to show per channel measurements and to execute raw TMC commands. It only supports the *Keysight DSO-X 2000* series (with the USB ID 0957:1798), but more devices can be added by extending *on_activate()* in `labgrid/driver/usbtmcdriver.py` and writing a corresponding backend in `labgrid/driver/usbtmc/`.

5.2.34 FlashromDriver

The *FlashromDriver* is used to flash a rom, using the flashrom utility.

```
FlashromDriver:
     image: 'foo'
images:
  foo: ../images/image_to_load.raw
```

Binds to:

flashrom_resource:

- *Flashrom*
- *NetworkFlashrom*

The FlashromDriver allows using the linux util "flashrom" to write directly to a ROM e.g. a NOR SPI flash. The assumption is that the device flashing the DUT e.g. an exporter is wired to the Flash to be flashed. The driver implements the bootstrap protocol. The driver uses tool configuration section and the key: flashrom to determine the path of the installed flashrom utility.

5.2.35 XenaDriver

The XenaDriver allows to use Xena networking tests equipment. Using the xenavalkyrie library a full API to control the tester is available.

Binds to:

xena_manager:

- *XenaManager*

The driver is supposed to work with all Xena products from the "Valkyrie Layer 2-3 Test platform" Currently tested on a *XenaCompact* chassis equipped with a *1 GE test module*.

5.2.36 DockerDriver

A DockerDriver binds to a *DockerDaemon* and is used to create and control one docker container.

The driver uses the docker python module to interact with the docker daemon.
For more information on the parameters see:
https://docker-py.readthedocs.io/en/stable/containers.html#container-objects

Binds to:

docker_daemon:

- *DockerDaemon*

Implements:

- *PowerProtocol*

```
DockerDriver:
  image_uri: "rastasheep/ubuntu-sshd:16.04"
  container_name: "ubuntu-lg-example"
  host_config: {"network_mode":"bridge"}
  network_services: [{"port":22,"username":"root","password":"root"}]
```

Arguments:

- image_uri (str): identifier of the docker image to use (may have a tag suffix)
- command (str): command to run in the container (optional, depends on image)
- volumes (list): list to configure volumes mounted inside the container (optional)
- container_name (str): name of the container
- environment (list): list of environment variables (optional)
- host_config (dict): dictionary of host configurations
- network_services (list): dictionaries that describe individual *NetworkService* instances that come alive when the container is created. The "address" argument which *NetworkService* also requires will be derived automatically upon container creation.

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 four states:

- unknown
- off
- 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 four states:

- unknown
- off
- 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.

5.3.4 DockerShellStrategy

A DockerShellStrategy has three states:

- unknown
- off
- shell

To transition to the shell state:

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

These commands would activate the docker driver which creates and starts a docker container. This will subsequently make *NetworkService* instance(s) available which can be used for e.g. ssh access.

Note: Transitioning to the "off" state will make any *NetworkService* instance(s) unresponsive - which may in turn invalidate ssh connection sharing. Therefore, during automated test suites, refrain from transitioning to the "off" state.

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 ColoredStepReporter

The ColoredStepReporter inherits from the StepReporter. The output is colored using ANSI color code sequences.

5.4.3 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
```

(continues on next page)

(continued from previous page)

```
  <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>
  - <python module>
```

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.

If you want to use multiple drivers of the same type, the resources and drivers need to be lists, e.g:

```
resources:
  RawSerialPort:
    port: '/dev/ttyS1'
drivers:
  SerialDriver: {}
```

becomes:

```
resources:
- RawSerialPort:
    port: '/dev/ttyS1'
- RawSerialPort:
    port: '/dev/ttyS2'
drivers:
- SerialDriver: {}
- SerialDriver: {}
```

This configuration doesn't specifiy which *RawSerialPort* to use for each *SerialDriver*, so it will cause an exception when instantiating the *Target*. To bind the correct driver to the correct resource, explicit `name` and `bindings` properties are used:

```
resources:
- RawSerialPort:
    name: 'foo'
    port: '/dev/ttyS1'
- RawSerialPort:
    name: 'bar'
    port: '/dev/ttyS2'
drivers:
- SerialDriver:
    name: 'foo_driver'
```

(continues on next page)

(continued from previous page)

```
    bindings:
      port: 'foo'
- SerialDriver:
    name: 'bar_driver'
    bindings:
      port: 'bar'
```

The property name for the binding (e.g. *port* in the example above) is documented for each individual driver under this chapter.

The YAML configuration file also supports templating for some substitutions, these are:

- LG_* variables, are replaced with their respective LG_* environment variable
- BASE is substituted with the base directory of the YAML file.

As an example:

```
targets:
  main:
    resources:
      RemotePlace:
        name: !template $LG_PLACE
tools:
  qemu_bin: !template "$BASE/bin/qemu-bin"
```

would resolve the qemu_bin path relative to the BASE dir of the YAML file and try to use the RemotePlace with the name set in the LG_PLACE environment variable.

See the *labgrid-device-config* man page for documentation on the top-level `options`, `images`, `tools`, and `examples` keys in 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: rl1, 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.

CHAPTER

SIX

DEVELOPMENT

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(eq=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(eq=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(eq=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(eq=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(eq=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(eq=False)
```

(continues on next page)

(continued from previous page)

```
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
    off = 1
    barebox = 2
    shell = 3
```

defines 3 custom states and the *unknown* state as the start point. These three states are handled in the transition function:

```
elif status == Status.off:
    self.target.deactivate(self.barebox)
    self.target.deactivate(self.shell)
    self.target.activate(self.power)
    self.power.off()
elif status == Status.barebox:
    self.transition(Status.off)
    # cycle power
    self.power.cycle()
    # interrupt barebox
    self.target.activate(self.barebox)
elif status == Status.shell:
    # tansition 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 than boot the linux kernel. The *off* states switch the power off.

6.5 Graph Strategies

Warning: This feature is experimental and brings much complexity to your project.

GraphStrategies are made for more complex strategies, with multiple, on each other depending, states. A GraphStrategy graph has to be a directed graph with one root state.

Using a GraphStrategy makes only sense if you have board states that are reachable by different ways. In this case GraphStrategies reduce state duplication.

6.5.1 Example

```
# conftest.py
from labgrid.strategy import GraphStrategy


class TestStrategy(GraphStrategy):
    def state_Unknown(self):
        pass

    @GraphStrategy.depends('Unknown')
    def state_Boot_via_NAND(self):
        pass

    @GraphStrategy.depends('Unknown')
    def state_Boot_via_NFS(self):
        pass
```

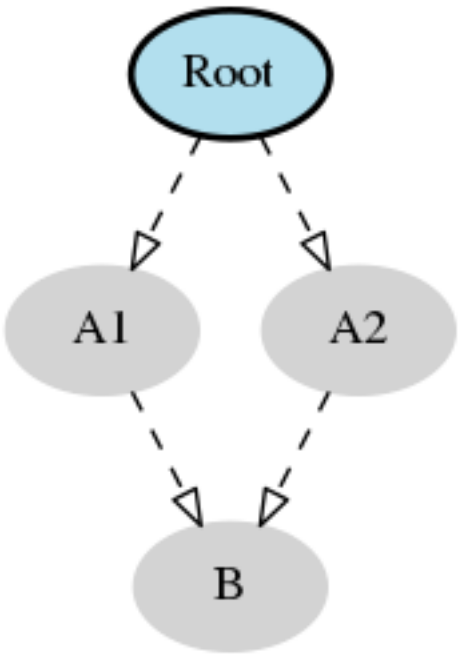
(continues on next page)

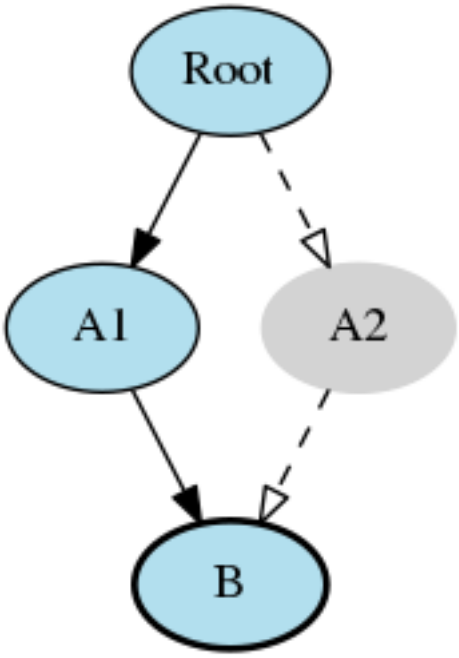
(continued from previous page)

```
    @GraphStrategy.depends('Boot_via_NAND', 'Boot_via_NFS')
    def state_BareBox(self):
        pass

    @GraphStrategy.depends('BareBox')
    def state_Linux_Shell(self):
        pass
```

```
# render graph to png
>>> graph_strategy.graph.render('filename')
'filename.png'
```





6.5.2 State

Every graph node describes a board state and how to reach it, A state has to be a class method following this prototype: *def state_$STATENAME(self):*. A state may not call *transition()* in its state definition.

6.5.3 Dependency

Every state, but the root state, can depend on other States, If a state has multiple dependencies, not all of them, but one, have to be reached before running the current state. When no via is used during a transition the order of the given dependencies decides which one gets called, where the first one has the highest priority and the last one the lowest. Dependencies are represented by graph edges.

6.5.4 Root State

Every GraphStrategy has to has to define exactly one root state. The root state defines the start of the graph and therefore the start of every transition. A state becomes a root state if it has no dependencies.

6.5.5 Transition

A transition describes a path, or a part of a path, through a GraphStrategy graph. Every State in the graph has a auto generated default path starting from the root state. So using the given example, the GraphStrategy would call the states *Unknown* , *Boot_via_NAND*, *BareBox*, and *Linux_Shell* in this order if *transition('Linux_Shell')* would be called. The GraphStrategy would prefer *Boot_via_NAND* over *Boot_via_NFS* because *Boot_via_NAND* is mentioned before *Boot_via_NFS* in the dependencies of *BareBox*. If you want to reach via *Boot_via_NFS* the call would look like this: *transition('Linux_Shell', via='Boot_via_NFS')*.

A transition can be incremental. If we trigger a transition *transition('BareBox')* first, the states *Unknown*, *Boot_via_NAND* and *BareBox* will be called in this order. If we trigger a transition *transition('Linux_Shell')* afterwards only *Linux_Shell* gets called. This happens because *Linux_Shell* is reachable from *BareBox* and the Strat-

egy holds state of the last walked path. But there is a catch! The second, incremental path must be *fully* incremental to the previous path! For example: Lets say we reached *BareBox* via *Boot_via_NFS*, (*transition('Barebox', via='Boot_via_NFS')*). If we trigger *transition('Linux_Shell')* afterwards the GraphStrategy would compare the last path *'Unknown', 'Boot_via_NFS', 'BareBox'* with the default path to *Linux_Shell* which would be *'Unknown', 'Boot_via_NAND', 'BareBox', 'Linux_Shell'*, and decides the path is not fully incremental and starts over by the root state. If we had given the second transition *Boot_via_NFS* like in the first transition the paths had been incremental.

6.6 SSHManager

Labgrid provides a SSHManager to allow connection reuse with control sockets. To use the SSHManager in your code, import it from *labgrid.util.ssh*:

```
from labgrid.util.ssh import sshmanager
```

you can now request or remove forwards:

```
from labgrid.util.ssh import sshmanager

localport = sshmanager.request_forward('somehost', 3000)

sshmanager.remove_forward('somehost', 3000)
```

or get and put files:

```
from labgrid.util.ssh import sshmanager

sshmanager.put_file('somehost', '/path/to/local/file', '/path/to/remote/file')
```

Note: The SSHManager will reuse existing Control Sockets and set up a keepalive loop to prevent timeouts of the socket during tests.

6.7 ManagedFile

While the *SSHManager* exposes a lower level interface to use SSH Connections, the ManagedFile provides a higher level interface for file upload to another host. It is meant to be used in conjunction with a remote resource, and store the file on the remote host with the following pattern:

```
/tmp/labgrid-<username>/<sha256sum>/<filename>
```

Additionally it provides *get_remote_path()* to retrieve the complete file path, to easily employ it for driver implementations. To use it in conjunction with a *Resource* and a file:

```
from labgrid.util.managedfile import ManagedFile

mf = ManagedFile(<your-file>, <your-resource>)
mf.sync_to_resource()
path = mf.get_remote_path()
```

Unless constructed with *ManagedFile(..., detect_nfs=False)*, ManagedFile employs the following heuristic to check if a file is on NFS and if so, foregoes the transfer and *get_remote_path()* just returns the local path (which is identical to the remote path in this case):

- check if GNU coreutils stat(1) with option –format exists on local and remote system
- check if inode number, total size and birth/modification timestamps match on local and remote system

6.8 ProxyManager

The proxymanager is used to open connections across proxies via an attribute in the resource. This allows gated testing networks by always using the exporter as an SSH gateway to proxy the connections using SSH Forwarding. Currently this is used in the *SerialDriver* for proxy connections.

Usage:

```
from labgrid.util.proxy import proxymanager

proxymanager.get_host_and_port(<resource>)
```

6.9 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.

6.9.1 Workflow

- Changes should be submitted via a GitHub pull request.
- Try to limit each commit to a single conceptual change.
- Add a signed-of-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.

6.9.2 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.

6.9.3 Documentation

- Use semantic linefeeds in .rst files.

6.9.4 Run Tests

```
$ tox -r
```

6.9.5 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:
>
> (a) 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
>
> (b) 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
>
> (c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
>
> (d) 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).

6.10 Ideas

6.10.1 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.

6.10.2 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.

6.10.3 CommandProtocol Support for Background Processes

Currently the CommandProtocol does not support long running processes well. An implementation should start a new process, return a handle and forbid running other processes in the foreground. The handle can be used to retrieve output from a command.

CHAPTER

SEVEN

DESIGN DECISIONS

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 (Hawkbit)
- bootstrap services:
  - fastboot
  - imxusbloader

7.3 Further Goals

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

CHAPTER

EIGHT

CHANGES

8.1 Release 0.3.0 (unreleased)

8.1.1 New Features in 0.3.0

- The new *FileDigitalOutputDriver* respresents a digital signal with a file.
- The new *GpioDigitalOutputDriver* controls the state of a GPIO via the sysfs interface.
- Crossbar and autobahn have been updated to 19.3.3 and 19.3.5 respectively.
- The InfoDriver was removed. The functions have been integrated into the labgridhelper library, please use the library for the old functionality.
- labgrid-client `write-image` subcommand: labgrid client now has a `write-image` command to write images onto block devices.
- `labgrid-client ssh` now also uses port from NetworkService resource if available
- The SSHDriver's keyfile attribute is now specified relative to the config file just like the images are.
- The ShellDriver's keyfile attribute is now specified relative to the config file just like the images are.
- `labgrid-client -P <PROXY>` and the `LG_PROXY` enviroment variable can be used to access the coordinator and network resources via that SSH proxy host. Drivers which run commands via SSH to the exporter still connect directly, allowing custom configuration in the user's `.ssh/config` as needed. Note that not all drivers have been updated to use the ProxyManager yet.
- Flashrom support added, by hard-wiring e.g. an exporter to the DUT, the ROM on the DUT can be written directly. The flashrom driver implements the bootstrap protocol.
- AndroidFastbootDriver now supports 'getvar' and 'oem getenv' subcommands.
- The coordinator now updates the resource acquired state at the exporter. Accordingly, the exporter now starts ser2net only when a resources is aquired. Furthermore, resource conflicts between places are now detected.
- The binding dictionary can now supports type name strings in addition to the types themselves, avoiding the need to import a specific protocol or driver in some cases.

8.1.2 Breaking changes in 0.3.0

- *ManagedFile* now saves the files in a different directory on the exporter. Previously `/tmp` was used, labgrid now uses `/var/cache/labgrid`. A tmpfiles example configuration for systemd is provided in the `/contrib` directory. It is also highly recommended to enable `fs.protected_regular=1` and `fs.protected_fifos=1` for kernels>=4.19. This requires user intervention after the upgrade to create the directory and setup the cleanup job.

- `@attr.s(cmp=False)` is deprecated and all classes have been moved to `@attr.s(eq=False)`, this release requires attrs version 19.2.0

8.2 Release 0.2.0 (released Jan 4, 2019)

8.2.1 New Features in 0.2.0

- A colored StepReporter was added and can be used with `pytest --lg-colored-steps`.
- `labgrid-client` can now use the last changed information to sort listed resources and places.
- `labgrid-client ssh` now uses ip/user/password from NetworkService resource if available
- The pytest plugin option `--lg-log` enables logging of the serial traffic into a file (see below).
- The environement files can contain feature flags which can be used to control which tests are run in pytest.
- `LG_*` variables from the OS environment can be used in the config file with the `!template` directive.
- The new "managed file" support takes a local file and synchronizes it to a resource on a remote host. If the resource is not a *NetworkResource*, the local file is used instead.
- ProxyManager: a class to automatically create ssh forwardings to proxy connections over the exporter
- SSHManager: a global manager to multiplex connections to different exporters
- The target now saves it's attached drivers, resources and protocols in a lookup table, avoiding the need of importing many Drivers and Protocols (see *Syntactic sugar for Targets*)
- When multiple Drivers implement the same Protocol, the best one can be selected using a priority (see below).
- The new subcommand `labgrid-client monitor` shows resource or places changes as they happen, which is useful during development or debugging.
- 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.
- Experimental: The labgrid-autoinstall tool was added (see below).

8.2.2 New and Updated Drivers

- The new *DigitalOutputResetDriver* adapts a driver implementing the DigitalOutputProtocol to the ResetProtocol.
- The new *ModbusCoilDriver* support outputs on a ModbusTCP device.
- The new *NetworkUSBStorageDriver* allows writing to remote USB storage devices (such as SD cards or memory sticks connected to a mux).
- The new *QEMUDriver* runs a system image in QEmu and implements the *ConsoleProtocol* and *PowerProtocol*. This allows using labgrid without any real hardware.

- The new *QuartusHPSDriver* controls the "Quartus Prime Programmer and Tools" to flash a target's QSPI.
- The new *SerialPortDigitalOutputDriver* controls the state of a GPIO using the control lines of a serial port.
- The new *SigrokDriver* uses a (local or remote) device supported by sigrok to record samples.
- The new *SmallUBootDriver* supports the extremely limited U-Boot found in cheap WiFi routers.
- The new *USBSDMuxDriver* controls a Pengutronix USB-SD-Mux device.
- The new *USBTMCDriver* can fetch measurements and screenshots from the "Keysight DSOX2000 series" and the "Tektronix TDS 2000 series".
- The new *USBVideoDriver* can stream video from a remote H.264 UVC (USB Video Class) camera using gstreamer over SSH. Currently, configuration for the "Logitech HD Pro Webcam C920" exists.
- The new *XenaDriver* allows interacting with Xena network testing equipment.
- The new *YKUSHPowerDriver* and *USBPowerDriver* support software-controlled USB hubs.
- 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 *IMXUSBLoader* now recognizes more USB IDs.
- The *OpenOCDDriver* is now more flexible with loading configuration files.
- The *NetworkPowerDriver* now additionally supports:
  - 24 port "Gude Expert Power Control 8080"
  - 8 port "Gude Expert Power Control 8316"
  - NETIO 4 models (via telnet)
  - a simple REST interface
- The *SerialDriver* now supports using plain TCP instead of RFC 2217, which is needed from some console servers.
- The *ShellDriver* has been improved:
  - It supports configuring the various timeouts used during the login process.
  - It can use xmodem to transfer file from and to the target.

8.2.3 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.

8.2.4 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.

8.2.5 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)
```

(continues on next page)

(continued from previous page)

8.2.6 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. See the documentation for details.

8.2.7 ConsoleLogging Reporter

The ConsoleLoggingReporter can be used with the pytest plugin or the library. It records the Data send from a DUT to the computer running labgrid. The logfile contains a header with the name of the device from the environment configuration and a timestamp.

When using the library, the reporter can be started with:

```
from labgrid.consoleloggingreporter import ConsoleLoggingReporter

ConsoleLoggingReporter.start(".")
```

where "." is the output directory.

The pytest plugin accepts the `--lg-log` commandline option, either with or without an output path.

8.2.8 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. Interally, 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.

Contributions from: Ahmad Fatoum, Bastian Krause, Björn Lässig, Chris Fiege, Enrico Joerns, Esben Haabendal, Felix Lampe, Florian Scherf, Georg Hofmann, Jan Lübbe, Jan Remmet, Johannes Nau, Kasper Revsbech, Kjeld Flarup, Laurentiu Palcu, Oleksij Rempel, Roland Hieber, Rouven Czerwinski, Stanley Phoong Cheong Kwan, Steffen Trumtrar, Tobi Gschwendtner, Vincent Prince

8.3 Release 0.1.0 (released May 11, 2017)

This is the initial release of labgrid.

CHAPTER

NINE

MODULES

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*)
Initialize self. See help(type(self)) for accurate signature.

run()
Method to be run in sub-process; can be overridden in sub-class

run_once()

__module__ = 'labgrid.autoinstall.main'

class labgrid.autoinstall.main.**Manager**(*env*, *args*)
Bases: object

__init__(*env*, *args*)
Initialize self. See help(type(self)) for accurate signature.

configure()

start()

join()

__dict__ = mappingproxy({'__module__': 'labgrid.autoinstall.main', '__init__': <func

__module__ = 'labgrid.autoinstall.main'

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

labgrid.autoinstall.main.**main**()

labgrid.driver package

Subpackages

labgrid.driver.power package

Submodules

labgrid.driver.power.apc module

labgrid.driver.power.apc.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.apc.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.digipower module

labgrid.driver.power.digipower.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.digipower.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.gude module

labgrid.driver.power.gude.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.gude.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.gude24 module

labgrid.driver.power.gude24.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.gude24.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.gude8316 module

labgrid.driver.power.gude8316.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.gude8316.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.netio module

labgrid.driver.power.netio.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.netio.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.netio_kshell module

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

labgrid.driver.power.netio_kshell.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.netio_kshell.**power_get**(*host*, *port*, *index*)

labgrid.driver.power.simplerest module

Simple rest interface for Power Port. Used for ex. misc Raspberry Pi configs Author: Kjeld Flarup <kfa@deif.com>

The URL given in hosts in exporter.yaml must replace {value} with '0' or '1' It is optional whether to use {index} or not.

NetworkPowerPort: model: simplerest host: 'http://172.17.180.53:9999/relay/{index}/{value}' index: 0

labgrid.driver.power.simplerest.**power_set**(*host*, *port*, *index*, *value*)

labgrid.driver.power.simplerest.**power_get**(*host*, *port*, *index*)

labgrid.driver.usbtmc package

Submodules

labgrid.driver.usbtmc.keysight_dsox2000 module

labgrid.driver.usbtmc.keysight_dsox2000.**get_channel_info**(*driver*, *channel*)

labgrid.driver.usbtmc.keysight_dsox2000.**get_channel_values**(*driver*, *channel*)

labgrid.driver.usbtmc.keysight_dsox2000.**get_screenshot_png**(*driver*)

labgrid.driver.usbtmc.tektronix_tds2000 module

labgrid.driver.usbtmc.tektronix_tds2000.**get_channel_info**(*driver*, *channel*)

labgrid.driver.usbtmc.tektronix_tds2000.**get_channel_values**(*driver*, *channel*)

labgrid.driver.usbtmc.tektronix_tds2000.**get_screenshot_tiff**(*driver*)

Submodules

labgrid.driver.bareboxdriver module

class labgrid.driver.bareboxdriver.**BareboxDriver**(*target*, *name*, *prompt=''*, *autoboot='stop autoboot'*, *interrupt='n'*, *bootstring='Linux version \d'*, *password=''*, *login_timeout=60*)

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
- **bootstring** (*str*) – string that indicates that the Kernel is booting
- **password** (*str*) – optional, password to use for access to the shell

- **login_timeout** (*int*) – optional, timeout for access to the shell

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

__attrs_post_init__()

on_activate()
Activate the BareboxDriver

This function tries to login if not already active

on_deactivate()
Deactivate the BareboxDriver

Simply sets the internal status to 0

run(*cmd: str*, ***, *timeout: int = 30*)
Run a command

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 succesfully 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 version \\d'*, *password=''*, *login_timeout=60*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Wait until the pattern is detected in the output of cmd. Raises ExecutionError when the timeout expires.

Parameters

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

- **pattern** (*str*) – pattern as a string to look for in the output
- **timeout** (*float*) – timeout for the pattern detection
- **sleepduration** (*int*) – sleep time between the runs of the cmd

poll_until_success(*cmd*, ***, *expected=0*, *tries=None*, *timeout=30.0*, *sleepduration=1*)
Poll a command until a specific exit code is detected. Takes a timeout and the number of tries to run the cmd. The sleepduration argument sets the duration between runs of the cmd.

Parameters

- **cmd** (*str*) – command to run on the shell
- **expected** (*int*) – exitcode to detect
- **tries** (*int*) – number of tries, can be None for infinite tries
- **timeout** (*float*) – timeout for the exitcode detection
- **sleepduration** (*int*) – sleep time between the runs of the cmd

Returns whether the command finally executed sucessfully

Return type bool

run_check(*cmd: str*, ***, *timeout=30*, *codec='utf-8'*, *decodeerrors='strict'*)
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({'__module__': 'labgrid.driver.commandmixin', '__doc__': '\n

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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({'__module__': 'labgrid.driver.consoleexpectmixin', '__doc__'

__module__ = 'labgrid.driver.consoleexpectmixin'

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

labgrid.driver.deditecrelaisdriver module

class labgrid.driver.deditecrelaisdriver.**DeditecRelaisDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol*

bindings = {'relais': {<class 'labgrid.resource.remote.NetworkDeditecRelais8'>, <clas

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.deditecrelaisdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.dockerdriver module

Class for connecting to a docker daemon running on the host machine.

class labgrid.driver.dockerdriver.**DockerDriver**(*target*, *name*, *image_uri=None*, *command=None*, *volumes=None*, *container_name=None*, *environment=None*, *host_config=None*, *network_services=None*)
Bases: *labgrid.protocol.powerprotocol.PowerProtocol*, *labgrid.driver.common.Driver*

The DockerDriver is used to create docker containers. This is done via communication with a docker daemon.

When a container is created the container is labeled with an cleanup strategy identifier. Currently only one strategy is implemented. This strategy simply deletes all labgrid created containers before each test run. This is to ensure cleanup of dangling containers from crashed tests or hanging containers.

Image pruning is not done by the driver.

For detailed information about the arguments see the "Docker SDK for Python" documentation https://docker-py.readthedocs.io/en/stable/containers.html#container-objects

Parameters **bindings** (*dict*) – The labgrid bindings

Args passed to docker.create_container: image_uri (str): The uri of the image to fetch command (str): The command to execute once container has been created volumes (list): The volumes to declare environment (list): Docker environment variables to set host_config (dict): Docker host configuration parameters network_services (list): Sequence of dicts each specifying a network service that the docker container exposes.

bindings = {'docker_daemon': {<class 'labgrid.resource.docker.DockerDaemon'>}}

__attrs_post_init__()

on_activate()
On activation: 1. Import docker module (_client and _container remain available) 2. Connect to the docker daemon 3. Pull requested image from docker registry if needed 4. Create the new container according to parameters from conf

on_deactivate()
Remove container after use

on()
Start the container created during activation

off()
Stop the container created during activation

cycle()
Cycle the docker container by stopping and starting it

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *image_uri=None*, *command=None*, *volumes=None*, *container_name=None*, *environment=None*, *host_config=None*, *network_services=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.dockerdriver'

__repr__()
Automatically created by attrs.

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*, *txdelay=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()
Called by the Target when this object has been deactivated

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *cmd*, *txdelay=0.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*, *timeout=None*)
Run a command

run_check(**args*)
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]

get_status()
Get status of the Driver

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.fake'

__repr__()
Automatically created by attrs.

labgrid.driver.fastbootdriver module

class labgrid.driver.fastbootdriver.**AndroidFastbootDriver**(*target*, *name*, *image=None*, *sparse_size=None*)
Bases: *labgrid.driver.common.Driver*

bindings = {'fastboot': {<class 'labgrid.resource.udev.AndroidFastboot'>, <class 'lab

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

__call__(**args*)
Call self as a function.

boot(*filename*)

flash(*partition*, *filename*)

run(*cmd*)

continue_boot()

getvar(*var*)
Return variable value via 'fastboot getvar <var>'.

oem_getenv(*var*)
Return barebox environment variable value via 'fastboot oem getenv <var>'.

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

__init__(*target*, *name*, *image=None*, *sparse_size=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.fastbootdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.filedigitaloutput module

class labgrid.driver.filedigitaloutput.**FileDigitalOutputDriver**(*target*, *name*, *filepath*, *false_repr='0\n'*, *true_repr='1\n'*)
Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol*

Two arbitrary string values false_repr and true_repr are defined as representations for False and True. These values are written to a file and read from it. If the file's content does not match any of the representations it defaults to False. A prime example for using this driver is Linux's sysfs.

bindings = {}

__attrs_post_init__()

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *filepath*, *false_repr='0\n'*, *true_repr='1\n'*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.filedigitaloutput'

__repr__()
Automatically created by attrs.

labgrid.driver.flashromdriver module

class labgrid.driver.flashromdriver.**FlashromDriver**(*target*, *name*, *image=None*)
Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.bootstrapprotocol.BootstrapProtocol*

The Flashrom driver used the flashrom utility to write an image to a raw rom. The driver is a pure wrapper of the flashrom utility

bindings = {'flashrom_resource': {<class 'labgrid.resource.flashrom.NetworkFlashrom'>

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

__call__(**args*)
Call self as a function.

load(*filename=None*)

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.flashromdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.gpiodriver module

All GPIO-related drivers

class labgrid.driver.gpiodriver.**GpioDigitalOutputDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol*

bindings = {'gpio': {<class 'labgrid.resource.remote.NetworkSysfsGPIO'>, <class 'labg

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.gpiodriver'

__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__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.modbusdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.networkusbstoragedriver module

class labgrid.driver.networkusbstoragedriver.**Mode**
Bases: enum.Enum

An enumeration.

DD = 1

BMAPTOOL = 2

__module__ = 'labgrid.driver.networkusbstoragedriver'

class labgrid.driver.networkusbstoragedriver.**NetworkUSBStorageDriver**(*target*, *name*, *image=None*)

Bases: *labgrid.driver.common.Driver*

bindings = {'storage': {<class 'labgrid.resource.remote.NetworkUSBSDMuxDevice'>, <cla

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

write_image(*filename=None*, *mode=<Mode.DD: 1>*)

get_size()

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

__init__(*target*, *name*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.networkusbstoragedriver'

__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.onewireport.OneWirePIO'>}

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.onewiredriver'

__repr__()
Automatically created by attrs.

labgrid.driver.openocddriver module

class labgrid.driver.openocddriver.**OpenOCDDriver**(*target*, *name*, *config*, *search=[]*, *image=None*)

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

bindings = {'interface': {<class 'labgrid.resource.remote.NetworkAlteraUSBBlaster'>,

__attrs_post_init__()

resolve_path_str_or_list(*path*)

load(*filename=None*)

execute(*commands: list*)

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *config*, *search=[]*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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__() → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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_activate()
Called by the Target when this object has been activated

on()

off()

cycle()

get()

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *delay=2.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.powerdriver'

__repr__()
Automatically created by attrs.

class labgrid.driver.powerdriver.**DigitalOutputPowerDriver**(*target*, *name*, *delay=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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 'la

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.powerdriver'

__repr__()
Automatically created by attrs.

class labgrid.driver.powerdriver.**PDUDaemonDriver**(*target*, *name*, *delay=5*)
Bases: *labgrid.driver.common.Driver*, *labgrid.driver.powerdriver.PowerResetMixin*, *labgrid.protocol.powerprotocol.PowerProtocol*

PDUDaemonDriver - Driver using a PDU port available via pdudaemon

bindings = {'port': 'PDUDaemonPort'}

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on()

off()

cycle()

get()

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *delay=5*) → None
Initialize self. See help(type(self)) for accurate signature.

__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()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.qemudriver'

__repr__()
Automatically created by attrs.

labgrid.driver.quartushpsdriver module

class labgrid.driver.quartushpsdriver.**QuartusHPSDriver**(*target*, *name*, *image=None*)
Bases: *labgrid.driver.common.Driver*

bindings = {'interface': {<class 'labgrid.resource.remote.NetworkAlteraUSBBlaster'>,

__attrs_post_init__()

flash(*filename=None*, *address=0*)

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

__init__(*target*, *name*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.quartushpsdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.resetdriver module

class labgrid.driver.resetdriver.**DigitalOutputResetDriver**(*target*, *name*, *delay=1.0*)
Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.resetprotocol.ResetProtocol*

DigitalOutputResetDriver - Driver using a DigitalOutput to reset the target

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

__attrs_post_init__()

reset()

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *delay=1.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.resetdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.serialdigitaloutput module

class labgrid.driver.serialdigitaloutput.**SerialPortDigitalOutputDriver**(*target*, *name*, *signal*)

Bases: *labgrid.driver.common.Driver*, *labgrid.protocol.digitaloutputprotocol.DigitalOutputProtocol*

Controls the state of a GPIO using the control lines of a serial port.

This driver uses the flow-control pins of a serial port (for example an USB-UART-dongle) to control some external power switch. You may connect some kind of relay board to the flow control pins.

The serial port should NOT be used for serial communication at the same time. This will probably reset the flow-control signals.

Usable signals are DTR and RTS.

bindings = {'serial': <class 'labgrid.driver.serialdriver.SerialDriver'>}

__attrs_post_init__()

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

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *signal*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.serialdigitaloutput'

__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.serialport.NetworkSerialPort'>, <class

message = 'The installed pyserial version does not contain important RFC2217 fixes.\nY

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

run(*cmd*, *timeout=30.0*, *codec='utf-8'*, *decodeerrors='strict'*)
Run a command

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

put_ssh_key(*keyfile_path*)

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 ***ExecutionError*** – if something 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 ***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 not 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 ***ExecutionError*** – if something 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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.shelldriver'

__repr__()
Automatically created by attrs.

labgrid.driver.sigrokdriver module

class labgrid.driver.sigrokdriver.**SigrokCommon**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.sigrokdriver'

__repr__()
Automatically created by attrs.

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

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.sigrok.SigrokDevice'>, <class 'labgri

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

stop()

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

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.sigrokdriver'

__repr__()
Automatically created by attrs.

class labgrid.driver.sigrokdriver.**SigrokPowerDriver**(*target*, *name*, *delay=3.0*, *max_voltage=None*, *max_current=None*)

Bases: *labgrid.driver.sigrokdriver.SigrokCommon*, *labgrid.driver.powerdriver.PowerResetMixin*, *labgrid.protocol.powerprotocol.PowerProtocol*

The SigrokPowerDriverDriver uses sigrok-cli to control a PSU and collect measurements.

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

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

on()

off()

cycle()

set_voltage_target(*value*)

set_current_limit(*value*)

get()

measure()

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *delay=3.0*, *max_voltage=None*, *max_current=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.sigrokdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.smallubootdriver module

class labgrid.driver.smallubootdriver.**SmallUBootDriver**(*target*, *name*, *prompt=''*, *autoboot='stop autoboot'*, *password=''*, *interrupt='n'*, *init_commands=NOTHING*, *password_prompt='enter Password:'*, *boot_expression='U-Boot 20\d+'*, *bootstring='Linux version \d'*, *boot_secret='a'*, *login_timeout=60.0*)

Bases: *labgrid.driver.ubootdriver.UBootDriver*

SmallUBootDriver is meant as a driver for UBoot with only little functionality compared to standard a standard UBoot. Especially is 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 is 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=''*, *autoboot='stop autoboot'*, *password=''*, *interrupt='\n'*, *init_commands=NOTHING*, *password_prompt='enter Password:'*, *boot_expression='U-Boot 20\\d+'*, *bootstring='Linux version \\d'*, *boot_secret='a'*, *login_timeout=60.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__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=''*, *stderr_merge=False*)
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'

priorities = {<class 'labgrid.protocol.commandprotocol.CommandProtocol'>: 10, <class

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

run(*cmd*, *codec='utf-8'*, *decodeerrors='strict'*, *timeout=None*)
Run a command

get_status()
The SSHDriver is always connected, return 1

put(*filename*, *remotepath=''*)

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

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *keyfile=''*, *stderr_merge=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__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=''*, *autoboot='stop autoboot'*, *password=''*, *interrupt='n'*, *init_commands=NOTHING*, *password_prompt='enter Password:'*, *boot_expression='U-Boot 20\d+'*, *bootstring='Linux version \d'*, *login_timeout=30*)
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
- **login_timeout** (*int*) – optional, timeout for login prompt detection

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*, *timeout=30*)
Runs the specified command on the shell and returns the output.

Parameters

- **cmd** (*str*) – command to run on the shell
- **timeout** (*int*) – optional, how long to wait for completion

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 succesfully 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=''*, *autoboot='stop autoboot'*, *password=''*, *interrupt='\n'*, *init_commands=NOTHING*, *password_prompt='enter Password:'*, *boot_expression='U-Boot 20\\d+'*, *bootstring='Linux version \\d'*, *login_timeout=30*) → None
Initialize self. See help(type(self)) for accurate signature.

__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.remote.NetworkMXSUSBLoader'>, <class

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

load(*filename=None*)

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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.remote.NetworkIMXUSBLoader'>, <class

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

load(*filename=None*)

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *image=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbloader'

__repr__()
Automatically created by attrs.

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

bindings = {'loader': {<class 'labgrid.resource.udev.RKUSBLoader'>, <class 'labgrid.r

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

load(*filename=None*)

__abstractmethods__ = frozenset({})

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

__init__(*target*, *name*, *image=None*, *usb_loader=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbloader'

__repr__()
Automatically created by attrs.

labgrid.driver.usbsdmuxdriver module

class labgrid.driver.usbsdmuxdriver.**USBSDMuxDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*

The USBSDMuxDriver uses the usbsdmux tool (https://github.com/pengutronix/usbsdmux) to control the USB-SD-Mux hardware

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

bindings = {'mux': {<class 'labgrid.resource.remote.NetworkUSBSDMuxDevice'>, <class '

__attrs_post_init__()

set_mode(*mode*)

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbsdmuxdriver'

__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()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

write_image(*filename*)

get_size()

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbstorage'

__repr__()
Automatically created by attrs.

labgrid.driver.usbtmcdriver module

class labgrid.driver.usbtmcdriver.**USBTMCDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*

bindings = {'tmc': {<class 'labgrid.resource.udev.USBTMC'>, <class 'labgrid.resource.

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

command(*cmd*)

query(*cmd*, *binary=False*, *raw=False*)

identify()

get_channel_info(*channel*)

get_channel_values(*channel*)

get_screenshot()

get_bool(*cmd*)

get_int(*cmd*)

get_decimal(*cmd*)

get_str(*cmd*)

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbtmcdriver'

__repr__()
Automatically created by attrs.

labgrid.driver.usbvideodriver module

class labgrid.driver.usbvideodriver.**USBVideoDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*

bindings = {'video': {<class 'labgrid.resource.remote.NetworkUSBVideo'>, <class 'labg

get_caps()

select_caps(*hint=None*)

get_pipeline()

stream(*caps_hint=None*)

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.usbvideodriver'

__repr__()
Automatically created by attrs.

labgrid.driver.xenadriver module

class labgrid.driver.xenadriver.**XenaDriver**(*target*, *name*)
Bases: *labgrid.driver.common.Driver*

Xena Driver

bindings = {'xena_manager': <class 'labgrid.resource.xenamanager.XenaManager'>}

__attrs_post_init__()

on_activate()
Called by the Target when this object has been activated

on_deactivate()
Called by the Target when this object has been deactivated

get_session()

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.driver.xenadriver'

__repr__()
Automatically created by attrs.

labgrid.external package

Submodules

labgrid.external.hawkbit module

class labgrid.external.hawkbit.**HawkbitTestClient**(*host*, *port*, *username*, *password*, *version=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({'__module__': 'labgrid.external.hawkbit', '__attrs_post_init

__init__(*host*, *port*, *username*, *password*, *version=1.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.external.hawkbit'

__repr__()
Automatically created by attrs.

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

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

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

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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'

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({'__module__': 'labgrid.external.usbstick', '__doc__': 'The

__init__(*target*, *image_dir*, *image_name=''*) → None
Initialize self. See help(type(self)) for accurate signature.

__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, e

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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

abstract load(*filename: str*)

__abstractmethods__ = frozenset({'load'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.bootstrapprotocol', 'load':

__module__ = 'labgrid.protocol.bootstrapprotocol'

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

labgrid.protocol.commandprotocol module

class labgrid.protocol.commandprotocol.**CommandProtocol**
Bases: abc.ABC

Abstract class for the CommandProtocol

abstract run(*command: str*)
Run a command

abstract run_check(*command: str*)
Run a command, return str if succesful, ExecutionError otherwise

abstract get_status()
Get status of the Driver

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

abstract poll_until_success()
Repeatedly call a shell command until it succeeds

__abstractmethods__ = frozenset({'get_status', 'poll_until_success', 'run', 'run_check

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.commandprotocol', '__doc__':

__module__ = 'labgrid.protocol.commandprotocol'

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

labgrid.protocol.consoleprotocol module

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

Abstract class for the ConsoleProtocol

abstract read()
Read data from underlying port

abstract write(*data: bytes*)
Write data to underlying port

sendline(*line: str*)

sendcontrol(*char: str*)

expect(*pattern: str*)

class Client
Bases: abc.ABC

abstract get_console_matches()

abstract notify_console_match(*pattern*, *match*)

__abstractmethods__ = frozenset({'get_console_matches', 'notify_console_match'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.consoleprotocol', 'get_co

__module__ = 'labgrid.protocol.consoleprotocol'

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

__abstractmethods__ = frozenset({'read', 'write'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.consoleprotocol', '__doc__':

__module__ = 'labgrid.protocol.consoleprotocol'

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

labgrid.protocol.digitaloutputprotocol module

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

Abstract class providing the OneWireProtocol interface

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

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

__abstractmethods__ = frozenset({'get', 'set'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.digitaloutputprotocol', '__d

__module__ = 'labgrid.protocol.digitaloutputprotocol'

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

labgrid.protocol.filesystemprotocol module

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

abstract read(*filename: str*)

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

__abstractmethods__ = frozenset({'read', 'write'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.filesystemprotocol', 'read':

__module__ = 'labgrid.protocol.filesystemprotocol'

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

labgrid.protocol.filetransferprotocol module

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

abstract put(*filename: str*, *remotepath: str*)

abstract get(*filename: str*, *destination: str*)

__abstractmethods__ = frozenset({'get', 'put'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.filetransferprotocol', 'put'

__module__ = 'labgrid.protocol.filetransferprotocol'

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

labgrid.protocol.infoprotocol module

class labgrid.protocol.infoprotocol.**InfoProtocol**
Bases: abc.ABC

Abstract class providing the InfoProtocol interface

abstract get_ip(*interface: str = 'eth0'*)
Implementations should return the IP-adress for the supplied interface.

abstract get_hostname()
Implementations should return the hostname for the supplied interface.

abstract get_service_status(*service*)
Implementations should return the status of a service

__abstractmethods__ = frozenset({'get_hostname', 'get_ip', 'get_service_status'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.infoprotocol', '__doc__': '

__module__ = 'labgrid.protocol.infoprotocol'

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

labgrid.protocol.linuxbootprotocol module

class labgrid.protocol.linuxbootprotocol.**LinuxBootProtocol**
Bases: abc.ABC

abstract boot(*name: str*)

abstract await_boot()

abstract reset()

__abstractmethods__ = frozenset({'await_boot', 'boot', 'reset'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.linuxbootprotocol', 'boot':

__module__ = 'labgrid.protocol.linuxbootprotocol'

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

labgrid.protocol.mmioprotocol module

class labgrid.protocol.mmioprotocol.**MMIOProtocol**
Bases: abc.ABC

abstract read(*address: int*, *size: int*, *count: int*) → bytes

abstract write(*address: int*, *size: int*, *data: bytes*) → None

__abstractmethods__ = frozenset({'read', 'write'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.mmioprotocol', 'read': <fun

__module__ = 'labgrid.protocol.mmioprotocol'

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

labgrid.protocol.powerprotocol module

class labgrid.protocol.powerprotocol.**PowerProtocol**
Bases: abc.ABC

abstract on()

abstract off()

abstract cycle()

__abstractmethods__ = frozenset({'cycle', 'off', 'on'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.powerprotocol', 'on': <func

__module__ = 'labgrid.protocol.powerprotocol'

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

labgrid.protocol.resetprotocol module

class labgrid.protocol.resetprotocol.**ResetProtocol**
Bases: abc.ABC

abstract reset()

__abstractmethods__ = frozenset({'reset'})

__dict__ = mappingproxy({'__module__': 'labgrid.protocol.resetprotocol', 'reset': <f

__module__ = 'labgrid.protocol.resetprotocol'

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

labgrid.provider package

Submodules

labgrid.provider.fileprovider module

class labgrid.provider.fileprovider.**FileProvider**
Bases: abc.ABC

Abstract class for the FileProvider

abstract get(*name: str*) → dict
Get a dictionary of target paths to local paths for a given name.

abstract list()
Get a list of names.

__abstractmethods__ = frozenset({'get', 'list'})

__dict__ = mappingproxy({'__module__': 'labgrid.provider.fileprovider', '__doc__': '

__module__ = 'labgrid.provider.fileprovider'

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

labgrid.provider.mediafileprovider module

class labgrid.provider.mediafileprovider.**MediaFileProvider**(*groups={}*)
Bases: *labgrid.provider.fileprovider.FileProvider*

get(*name*)
Get a dictionary of target paths to local paths for a given name.

list()
Get a list of names.

__abstractmethods__ = frozenset({})

__attrs_attrs__ = (Attribute(name='groups', default={}, validator=<instance_of validat

__init__(*groups={}*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.provider.mediafileprovider'

__repr__()
Automatically created by attrs.

labgrid.pytestplugin package

Submodules

labgrid.pytestplugin.fixtures module

labgrid.pytestplugin.fixtures.**pytest_addoption**(*parser*)

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.**target**(*env*)
Return the default target *main* configured in the supplied configuration file.

labgrid.pytestplugin.hooks module

labgrid.pytestplugin.hooks.**pytest_configure**(*config*)

labgrid.pytestplugin.hooks.**pytest_collection_modifyitems**(*config*, *items*)
This function matches function feature flags with those found in the environment and disables the item if no match is found

labgrid.pytestplugin.reporter module

class labgrid.pytestplugin.reporter.**StepReporter**(*terminalreporter*, *, *rewrite=False*)
Bases: object

__init__(*terminalreporter*, *, *rewrite=False*)
Initialize self. See help(type(self)) for accurate signature.

notify(*event*)

pytest_runtest_logstart()

pytest_runtest_logreport(*report*)

__dict__ = mappingproxy({'__module__': 'labgrid.pytestplugin.reporter', '__init__':

__module__ = 'labgrid.pytestplugin.reporter'

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

class labgrid.pytestplugin.reporter.**ColoredStepReporter**(*terminalreporter*, *, *rewrite=False*)
Bases: *labgrid.pytestplugin.reporter.StepReporter*

EVENT_COLORS_DARK = {'cycle$|on$|off$': 246, 'expect$': 8, 'run': 10, 'state_': 51

EVENT_COLORS_LIGHT = {'cycle$|on$|off$': 8, 'expect$': 250, 'run': 10, 'state_': 5

__init__(*terminalreporter*, *, *rewrite=False*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.pytestplugin.reporter'

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*)
Bases: object

__attrs_post_init__()

property acquired

property avail

property cls

property params

property args
arguments for resource construction

property extra
extra resource information

asdict()

update(*data*)
apply updated information from the exporter on the coordinator

acquire(*place_name*)

release()

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

__dict__ = mappingproxy({'__module__': 'labgrid.remote.common', '__attrs_post_init__'

__init__(*data*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*, *rename=None*)
Bases: object

classmethod fromstr(*pattern*)

__repr__()
Return repr(self).

__str__()
Return str(self).

ismatch(*resource_path*)
Return True if this matches the given resource

__attrs_attrs__ = (Attribute(name='exporter', default=NOTHING, validator=None, repr=Tr

__dict__ = mappingproxy({'__module__': 'labgrid.remote.common', 'fromstr': <classmet

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*exporter*, *group*, *cls*, *name=None*, *rename=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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=''*, *tags=NOTHING*, *matches=NOTHING*, *acquired=None*, *acquired_resources=NOTHING*, *allowed=NOTHING*, *created=NOTHING*, *changed=NOTHING*, *reservation=None*)
Bases: object

asdict()

update(*config*)

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,

__dict__ = mappingproxy({'__module__': 'labgrid.remote.common', 'asdict': <function

__init__(*name*, *aliases=NOTHING*, *comment=''*, *tags=NOTHING*, *matches=NOTHING*, *acquired=None*, *acquired_resources=NOTHING*, *allowed=NOTHING*, *created=NOTHING*, *changed=NOTHING*, *reservation=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.common'

__repr__()
Automatically created by attrs.

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

class labgrid.remote.common.**ReservationState**
Bases: enum.Enum

An enumeration.

waiting = 0

allocated = 1

acquired = 2

expired = 3

invalid = 4

__module__ = 'labgrid.remote.common'

class labgrid.remote.common.**Reservation**(*owner*, *token=NOTHING*, *state='waiting'*, *prio=0.0*, *filters=NOTHING*, *allocations=NOTHING*, *created=NOTHING*, *timeout=NOTHING*)

Bases: object

asdict()

refresh(*delta=60*)

property expired

show(*level=0*)

__attrs_attrs__ = (Attribute(name='owner', default=NOTHING, validator=<instance_of val

__dict__ = mappingproxy({'__module__': 'labgrid.remote.common', 'asdict': <function

__init__(*owner*, *token=NOTHING*, *state='waiting'*, *prio=0.0*, *filters=NOTHING*, *allocations=NOTHING*, *created=NOTHING*, *timeout=NOTHING*) → None

Initialize self. See help(type(self)) for accurate signature.

__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

__dict__ = mappingproxy({'__module__': 'labgrid.remote.config', '__attrs_post_init__'

__init__(*filename*) → None

Initialize self. See help(type(self)) for accurate signature.

__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'

class labgrid.remote.coordinator.**RemoteSession**
Bases: object

class encapsulating a session, used by ExporterSession and ClientSession

property key
Key of the session

property name
Name of the session

__attrs_attrs__ = (Attribute(name='coordinator', default=NOTHING, validator=None, repr

__dict__ = mappingproxy({'__module__': 'labgrid.remote.coordinator', '__doc__': 'cla

__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*, *resourcedata*)

get_resources()
Method invoked by the client, get the resources from the coordinator

__attrs_attrs__ = (Attribute(name='coordinator', default=NOTHING, validator=None, repr

__init__(*coordinator*, *session*, *authid*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.coordinator'

__repr__()
Automatically created by attrs.

class labgrid.remote.coordinator.**ResourceImport**(*data*, *, *path*)
Bases: *labgrid.remote.common.ResourceEntry*

Represents a local resource exported from an exporter.

The ResourceEntry attributes contain the information for the client.

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

__init__(*data*, *, *path*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.coordinator'

__repr__()
Automatically created by attrs.

labgrid.remote.coordinator.**locked**(*func*)

labgrid.remote.exporter module

The remote.exporter module exports resources to the coordinator and makes them available to other clients on the same coordinator

class labgrid.remote.exporter.**ResourceExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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__()

start()

stop()

poll()

acquire(**args*, ***kwargs*)

release(**args*, ***kwargs*)

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

__init__(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBSerialPortExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
→ None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBEthernetExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
→ None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBGenericExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
→ None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBSigrokExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBSDMuxExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBPowerPortExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**USBDeditecRelaisExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
Bases: *labgrid.remote.exporter.USBGenericExport*

ResourceExport for outputs on deditec relais

__attrs_post_init__()

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

__init__(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

class labgrid.remote.exporter.**EthernetPortExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
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*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
→ None
Initialize self. See help(type(self)) for accurate signature.

__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.

class labgrid.remote.exporter.**GPIOGenericExport**(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
Bases: *labgrid.remote.exporter.ResourceExport*

ResourceExport for GPIO lines accessed directly from userspace

__attrs_post_init__()

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

__init__(*data*, *host='build-9884079-project-82349-labgrid'*, *proxy=None*, *proxy_required=False*)
→ None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.exporter'

__repr__()
Automatically created by attrs.

labgrid.remote.exporter.**main**()

labgrid.remote.scheduler module

class labgrid.remote.scheduler.**TagSet**(*name*, *tags*)
Bases: object

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

__dict__ = mappingproxy({'__module__': 'labgrid.remote.scheduler', '__dict__': <attr

__init__(*name*, *tags*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.remote.scheduler'

__repr__()
Automatically created by attrs.

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

labgrid.remote.scheduler.**schedule_step**(*places*, *filters*)
Find the filters that can be directly allocated without overlap.

labgrid.remote.scheduler.**schedule_overlaps**(*places*, *filters*)
Iterate schedule_step until no more allocations are found.

labgrid.remote.scheduler.**schedule**(*places*, *filters*)

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*target*, *name*, *switch=None*, *interface=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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.

class labgrid.resource.base.**SysfsGPIO**(*target*, *name*, *index=None*)
Bases: *labgrid.resource.common.Resource*

The basic SysfsGPIO contains an index

Parameters **index** (*int*) – index of target gpio line.

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

__init__(*target*, *name*, *index=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.base'

__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__()

property command_prefix

property 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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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

property command_prefix

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

__init__(*target*, *name*, *host*) → None
Initialize self. See help(type(self)) for accurate signature.

__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({'__module__': 'labgrid.resource.common', 'instances': {}, '

__init__() → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.common'

__repr__()
Automatically created by attrs.

get_managed_parent()
For Resources which have been created at runtime, return the ManagedResource resource which created it.

Returns None otherwise.

labgrid.resource.docker module

Auxiliary classes that assist DockerDriver. Specifically, DockerDaemon and DockerManager will create the NetworkResource instance that is declared in the specification (e.g. yaml) of DockerDriver.

class labgrid.resource.docker.**DockerConstants**
Bases: object

Class constants for handling container cleanup

DOCKER_LG_CLEANUP_LABEL = 'lg_cleanup'

DOCKER_LG_CLEANUP_TYPE_AUTO = 'auto'

__dict__ = mappingproxy({'__module__': 'labgrid.resource.docker', '__doc__': 'Class

__module__ = 'labgrid.resource.docker'

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

class labgrid.resource.docker.**DockerManager**
Bases: *labgrid.resource.common.ResourceManager*

The DockerManager is responsible for cleaning up dangling containers and managing the managed NetworkService resources. Different docker daemons for different targets are allowed, but there has to be only one for each target.

__attrs_post_init__()

on_resource_added(*resource*)
If the resource added is a DockerDaemon, make sure this is the only one added for this target. If it is, create a docker client to be used to communicate with the docker daemon.

poll()
Ask associated DockerDaemon resource to check if associated NetworkService has come up.

__attrs_attrs__ = ()

__init__() → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.docker'

__repr__()
Automatically created by attrs.

class labgrid.resource.docker.**DockerDaemon**(*target*, *name*, *docker_daemon_url*)
Bases: *labgrid.resource.common.ManagedResource*

A resource identifying a docker daemon

docker_daemon_url = None
The docker network service is a managed resource mapping a container name to a network service.

manager_cls
alias of *DockerManager*

__attrs_post_init__()

on_client_bound(*client*)
Each time a docker driver binds to the DockerDaemon resource the docker driver network services list is iterated and for each network service defined a NetworkService resource instance is created with the parameters from the configuration file.

on_poll(*docker_client*)
Check if associated NetworkService has come up.

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

__init__(*target*, *name*, *docker_daemon_url*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.docker'

__repr__()
Automatically created by attrs.

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({'__module__': 'labgrid.resource.ethernetport', '__doc__': '

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*hostname*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__() → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*target*, *name*, *switch*, *interface*) → None
Initialize self. See help(type(self)) for accurate signature.

__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.flashrom module

class labgrid.resource.flashrom.**Flashrom**(*target*, *name*, *programmer*)
Bases: *labgrid.resource.common.Resource*

Programmer is the programmer parameter described in man(8) of flashrom

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

__init__(*target*, *name*, *programmer*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.flashrom'

__repr__()
Automatically created by attrs.

class labgrid.resource.flashrom.**NetworkFlashrom**(*target*, *name*, *host*, *programmer*)
Bases: *labgrid.resource.common.NetworkResource*

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

__init__(*target*, *name*, *host*, *programmer*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.flashrom'

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.modbus'

__repr__()
Automatically created by attrs.

labgrid.resource.networkservice module

class labgrid.resource.networkservice.**NetworkService**(*target*, *name*, *address*, *username*, *password=''*, *port=22*)
Bases: *labgrid.resource.common.Resource*

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

__init__(*target*, *name*, *address*, *username*, *password=''*, *port=22*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.power'

__repr__()
Automatically created by attrs.

class labgrid.resource.power.**PDUDaemonPort**(*target*, *name*, *host*, *pdu*, *index*)
Bases: *labgrid.resource.common.Resource*

The PDUDaemonPort describes a port on a PDU accessible via PDUDaemon

Parameters

- **host** (*str*) – name of the host running the PDUDaemon
- **pdu** (*str*) – name of the PDU in the configuration file
- **index** (*int*) – index of the power port on the PDU

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

__init__(*target*, *name*, *host*, *pdu*, *index*) → None
Initialize self. See help(type(self)) for accurate signature.

__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__() → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkRKUSBLoader**(*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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*, *channels=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*, *channels=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkSigrokUSBSerialDevice**(*target*, *name*, *host*, *busnum*, *devnum*, *path*, *vendor_id*, *model_id*, *driver=None*, *channels=None*)

Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkSigrokUSBSerialDevice 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*, *channels=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkUSBPowerPort**(*target*, *name*, *host*, *busnum*, *devnum*, *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_post_init__()

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

__init__(*target*, *name*, *host*, *busnum*, *devnum*, *path*, *vendor_id*, *model_id*, *index=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkUSBVideo**(*target*, *name*, *host*, *busnum*, *devnum*, *path*, *vendor_id*, *model_id*)
Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkUSBVideo describes a remotely accessible USB video 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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkUSBTMC**(*target*, *name*, *host*, *busnum*, *devnum*, *path*, *vendor_id*, *model_id*)
Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkUSBTMC describes a remotely accessible USB TMC 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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkDeditecRelais8**(*target*, *name*, *host*, *busnum*, *devnum*, *path*, *vendor_id*, *model_id*, *index=None*, *invert=False*)
Bases: *labgrid.resource.remote.RemoteUSBResource*

The NetworkDeditecRelais8 describes a remotely accessible USB relais port

__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*, *index=None*, *invert=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

class labgrid.resource.remote.**NetworkSysfsGPIO**(*target*, *name*, *host*, *index*)
Bases: *labgrid.resource.common.NetworkResource*, *labgrid.resource.common.ManagedResource*

manager_cls
The NetworkSysfsGPIO describes a remotely accessible gpio line

alias of *RemotePlaceManager*

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

__attrs_post_init__()

__init__(*target*, *name*, *host*, *index*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.remote'

__repr__()
Automatically created by attrs.

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.serialport'

__repr__()
Automatically created by attrs.

class labgrid.resource.serialport.**NetworkSerialPort**(*target*, *name*, *host*, *port*, *speed=115200*, *protocol='rfc2217'*)
Bases: *labgrid.resource.common.NetworkResource*

A NetworkSerialPort is a remotely accessable 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'*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.serialport'

__repr__()
Automatically created by attrs.

labgrid.resource.sigrok module

class labgrid.resource.sigrok.**SigrokDevice**(*target*, *name*, *driver='demo'*, *channels=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 desribed in the sigrok-cli man page

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

__init__(*target*, *name*, *driver='demo'*, *channels=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.sigrok'

__repr__()
Automatically created by attrs.

labgrid.resource.suggest module

class labgrid.resource.suggest.**Suggester**(*args*)
Bases: object

__init__(*args*)
Initialize self. See help(type(self)) for accurate signature.

suggest_callback(*resource*, *meta*, *suggestions*)

run()

__dict__ = mappingproxy({'__module__': 'labgrid.resource.suggest', '__init__': <func

__module__ = 'labgrid.resource.suggest'

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

labgrid.resource.suggest.**main**()

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__() → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBResource**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)
Bases: *labgrid.resource.common.ManagedResource*

manager_cls
alias of *UdevManager*

__attrs_post_init__()

filter_match(*device*)

suggest_match(*device*)

try_match(*device*)

update()

property busnum

property devnum

property path

property vendor_id

property 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=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBSerialPort**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *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=NOTHING*, *device=None*, *suggest=False*, *port=None*, *speed=115200*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBMassStorage**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)

Bases: *labgrid.resource.udev.USBResource*

__attrs_post_init__()

property path

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**IMXUSBLoader**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)

Bases: *labgrid.resource.udev.USBResource*

filter_match(*device*)

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**RKUSBLoader**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)

Bases: *labgrid.resource.udev.USBResource*

filter_match(*device*)

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**MXSUSBLoader**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)

Bases: *labgrid.resource.udev.USBResource*

filter_match(*device*)

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**AndroidFastboot**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *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=NOTHING*, *device=None*, *suggest=False*, *usb_vendor_id='1d6b'*, *usb_product_id='0104'*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBEthernetInterface**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *ifname=None*)

Bases: *labgrid.resource.udev.USBResource*, *labgrid.resource.base.EthernetInterface*

__attrs_post_init__()

update()

property if_state

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *ifname=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**AlteraUSBBlaster**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)

Bases: *labgrid.resource.udev.USBResource*

filter_match(*device*)

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**SigrokUSBDevice**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *driver=None*, *channels=None*)

Bases: *labgrid.resource.udev.USBResource*

The SigrokUSBDevice describes an attached sigrok device with driver and optional channel mapping, it is identified via usb using udev.

This is used for devices which communicate over a custom USB protocol.

Parameters

- **driver** (*str*) – driver to use with sigrok
- **channels** (*str*) – a sigrok channel mapping as desribed in the sigrok-cli man page

__attrs_post_init__()

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *driver=None*, *channels=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**SigrokUSBSerialDevice**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *driver=None*, *channels=None*)

Bases: *labgrid.resource.udev.USBResource*

The SigrokUSBSerialDevice describes an attached sigrok device with driver and optional channel mapping, it is identified via usb using udev.

This is used for devices which communicate over an emulated serial device.

Parameters

- **driver** (*str*) – driver to use with sigrok
- **channels** (*str*) – a sigrok channel mapping as desribed in the sigrok-cli man page

__attrs_post_init__()

property path

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *driver=None*, *channels=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBSDMuxDevice**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *control_path=None*, *disk_path=None*)

Bases: *labgrid.resource.udev.USBResource*

The USBSDMuxDevice describes an attached USBSDMux device, it is identified via USB using udev

__attrs_post_init__()

property avail

poll()

property path

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *control_path=None*, *disk_path=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBPowerPort**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *index=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_post_init__()

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *index=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBVideo**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)
Bases: *labgrid.resource.udev.USBResource*

__attrs_post_init__()

property path

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**USBTMC**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*)
Bases: *labgrid.resource.udev.USBResource*

__attrs_post_init__()

property path

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()
Automatically created by attrs.

class labgrid.resource.udev.**DeditecRelais8**(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *index=None*, *invert=False*)

Bases: *labgrid.resource.udev.USBResource*

__attrs_post_init__()

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

__init__(*target*, *name*, *match=NOTHING*, *device=None*, *suggest=False*, *index=None*, *invert=False*) → None

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.udev'

__repr__()

Automatically created by attrs.

property path

labgrid.resource.xenamanager module

class labgrid.resource.xenamanager.**XenaManager**(*target*, *name*, *hostname*)

Bases: *labgrid.resource.common.Resource*

Hostname/IP identifying the manageent address of the xena tester

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

__init__(*target*, *name*, *hostname*) → None

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.xenamanager'

__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*) → None

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.resource.ykushpowerport'

__repr__()

Automatically created by attrs.

labgrid.strategy package

Submodules

labgrid.strategy.bareboxstrategy module

class labgrid.strategy.bareboxstrategy.**Status**
Bases: enum.Enum

An enumeration.

unknown = 0

off = 1

barebox = 2

shell = 3

__module__ = 'labgrid.strategy.bareboxstrategy'

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>*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.bareboxstrategy'

__repr__()
Automatically created by attrs.

labgrid.strategy.common module

exception labgrid.strategy.common.**StrategyError**(*msg*)
Bases: Exception

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

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.common'

__repr__()
Automatically created by attrs.

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

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*)
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.

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

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.common'

__repr__()
Automatically created by attrs.

labgrid.strategy.dockerstrategy module

class labgrid.strategy.dockerstrategy.**Status**
Bases: enum.Enum

The possible states of a docker container

unknown = 0

gone = 1

accessible = 2

__module__ = 'labgrid.strategy.dockerstrategy'

class labgrid.strategy.dockerstrategy.**DockerStrategy**(*target*, *name*, *status=<Status.unknown: 0>*)
Bases: *labgrid.strategy.common.Strategy*

DockerStrategy enables the user to directly transition to a state where a fresh docker container has been created and is ready for access (e.g. shell access via SSH if the docker image runs an SSH daemon).

bindings = {'docker_driver': <class 'labgrid.driver.dockerdriver.DockerDriver'>}

__attrs_post_init__()

transition(*status*)

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

__init__(*target*, *name*, *status=<Status.unknown: 0>*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.dockerstrategy'

__repr__()
Automatically created by attrs.

labgrid.strategy.graphstrategy module

exception labgrid.strategy.graphstrategy.**GraphStrategyError**(*msg*)
Bases: *labgrid.strategy.common.StrategyError*

__module__ = 'labgrid.strategy.graphstrategy'

exception labgrid.strategy.graphstrategy.**InvalidGraphStrategyError**(*msg*)
Bases: *labgrid.strategy.graphstrategy.GraphStrategyError*

__module__ = 'labgrid.strategy.graphstrategy'

exception labgrid.strategy.graphstrategy.**GraphStrategyRuntimeError**(*msg*)
Bases: *labgrid.strategy.graphstrategy.GraphStrategyError*

__module__ = 'labgrid.strategy.graphstrategy'

class labgrid.strategy.graphstrategy.**GraphStrategy**(*target*, *name*)
Bases: *labgrid.strategy.common.Strategy*

__attrs_post_init__()

invalidate()

transition(*state*, *via=None*)

find_abs_path(*state*, *via=None*)

find_rel_path(*path*)

property graph

classmethod depends(**dependencies*)

__module__ = 'labgrid.strategy.graphstrategy'

labgrid.strategy.shellstrategy module

class labgrid.strategy.shellstrategy.**Status**
Bases: enum.Enum

An enumeration.

unknown = 0

off = 1

shell = 2

__module__ = 'labgrid.strategy.shellstrategy'

class labgrid.strategy.shellstrategy.**ShellStrategy**(*target*, *name*, *status=<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>*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.shellstrategy'

__repr__()
Automatically created by attrs.

labgrid.strategy.ubootstrategy module

class labgrid.strategy.ubootstrategy.**Status**
Bases: enum.Enum

An enumeration.

unknown = 0

off = 1

uboot = 2

shell = 3

__module__ = 'labgrid.strategy.ubootstrategy'

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>*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.strategy.ubootstrategy'

__repr__()
Automatically created by attrs.

labgrid.util package

Subpackages

labgrid.util.agents package

Submodules

labgrid.util.agents.deditec_relais8 module

labgrid.util.agents.dummy module

labgrid.util.agents.dummy.**handle_neg**(*value*)

labgrid.util.agents.sysfsgpio module

This module implements switching GPIOs via sysfs GPIO kernel interface.

Takes an integer property 'index' which refers to the already exported GPIO device.

class labgrid.util.agents.sysfsgpio.**GpioDigitalOutput**(***kwargs*)

Bases: object

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

__del__()

get()

set(*status*)

__dict__ = mappingproxy({'__module__': 'labgrid.util.agents.sysfsgpio', '_gpio_sysfs_

__module__ = 'labgrid.util.agents.sysfsgpio'

__weakref__

list of weak references to the object (if defined)

labgrid.util.agents.sysfsgpio.**handle_set**(*index*, *status*)

labgrid.util.agents.sysfsgpio.**handle_get**(*index*)

Submodules

labgrid.util.agent module

labgrid.util.agent.**b2s**(*b*)

labgrid.util.agent.**s2b**(*s*)

class labgrid.util.agent.**Agent**

Bases: object

__init__()

Initialize self. See help(type(self)) for accurate signature.

send(*data*)

register(*name*, *func*)

load(*name*, *source*)

list()

run()

__dict__ = mappingproxy({'__module__': 'labgrid.util.agent', '__init__': <function A

__module__ = 'labgrid.util.agent'

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

labgrid.util.agent.**handle_test**(**args*, ***kwargs*)

labgrid.util.agent.**handle_error**(*message*)

labgrid.util.agent.**handle_usbtmc**(*index*, *cmd*, *read=False*)

labgrid.util.agent.**main**()

labgrid.util.agentwrapper module

labgrid.util.agentwrapper.**b2s**(*b*)

labgrid.util.agentwrapper.**s2b**(*s*)

exception labgrid.util.agentwrapper.**AgentError**
Bases: Exception

__module__ = 'labgrid.util.agentwrapper'

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

exception labgrid.util.agentwrapper.**AgentException**
Bases: Exception

__module__ = 'labgrid.util.agentwrapper'

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

class labgrid.util.agentwrapper.**MethodProxy**(*wrapper*, *name*)
Bases: object

__init__(*wrapper*, *name*)
Initialize self. See help(type(self)) for accurate signature.

__call__(**args*, ***kwargs*)
Call self as a function.

__dict__ = mappingproxy({'__module__': 'labgrid.util.agentwrapper', '__init__': <fun

__module__ = 'labgrid.util.agentwrapper'

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

class labgrid.util.agentwrapper.**ModuleProxy**(*wrapper*, *name*)
Bases: object

__init__(*wrapper*, *name*)
Initialize self. See help(type(self)) for accurate signature.

__getattr__(*name*)

__dict__ = mappingproxy({'__module__': 'labgrid.util.agentwrapper', '__init__': <fun

__module__ = 'labgrid.util.agentwrapper'

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

class labgrid.util.agentwrapper.**AgentWrapper**(*host=None*)
Bases: object

__init__(*host=None*)
Initialize self. See help(type(self)) for accurate signature.

__del__()

__getattr__(*name*)

call(*method*, **args*, ***kwargs*)

__dict__ = mappingproxy({'__module__': 'labgrid.util.agentwrapper', '__init__': <fun

__module__ = 'labgrid.util.agentwrapper'

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

load(*name*)

close()

labgrid.util.atomic module

labgrid.util.atomic.**atomic_replace**(*filename*, *data*)

labgrid.util.dict module

This module contains helper functions for working with dictionaries.

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.dict.**filter_dict**(*d*, *cls*, *warn=False*)
Returns a copy a dictionary which only contains the attributes defined on an attrs class.

labgrid.util.dict.**find_dict**(*d*, *key*)
Recursively search for a key in a dictionary

Parameters

- **d** (*dict*) – The dictionary to recursively search through
- **key** (*str*) – The key to search for

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*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Bases: pexpect.pty_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*)
Initializes a pexpect spawn instanse with required configuration

send(*s*)
Write to underlying transport, return number of bytes written

read_nonblocking(*size=1*, *timeout=-1*)
Pexpects needs a nonblocking read function, simply use pyserial with a timeout of 0.

__module__ = 'labgrid.util.expect'

labgrid.util.helper module

labgrid.util.helper.**get_free_port**()
Helper function to always return an unused port.

labgrid.util.helper.**get_user**()
Get the username of the current user.

class labgrid.util.helper.**ProcessWrapper**(*callbacks=NOTHING*)
Bases: object

check_output(*command*)
Run a command and supply the output to callback functions

register(*callback*)
Register a callback with the ProcessWrapper

unregister(*callback*)
Unregister a callback with the ProcessWrapper

enable_logging()
Enables process output to the logging interface. Loglevel is logging.INFO.

enable_print()
Enables process output to print.

__attrs_attrs__ = (Attribute(name='callbacks', default=Factory(factory=<class 'list'>,

__dict__ = mappingproxy({'__module__': 'labgrid.util.helper', 'check_output': <funct

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*callbacks=NOTHING*) → None
Initialize self. See help(type(self)) for accurate signature.

__le__(*other*)
Automatically created by attrs.

__lt__(*other*)
Automatically created by attrs.

__module__ = 'labgrid.util.helper'

__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)

labgrid.util.managedfile module

class labgrid.util.managedfile.**ManagedFile**(*local_path*, *resource*, *detect_nfs=True*)
Bases: object

The ManagedFile allows the synchronisation of a file to a remote host. It has to be created with the to be synced file and the target resource as argument:

:: from labgrid.util.managedfile import ManagedFile

ManagedFile("/tmp/examplefile", <your-resource>)

Synchronisation is done with the sync_to_resource method.

__attrs_post_init__()

sync_to_resource()
sync the file to the host specified in a resource

Raises ***ExecutionError*** – if the SSH connection/copy fails

get_remote_path()
Retrieve the remote file path

Returns path to the file on the remote host

Return type str

get_hash()
Retrieve the hash of the file

Returns SHA256 hexdigest of the file

Return type str

__attrs_attrs__ = (Attribute(name='local_path', default=NOTHING, validator=<instance_o

__dict__ = mappingproxy({'__module__': 'labgrid.util.managedfile', '__doc__': ' The

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*local_path*, *resource*, *detect_nfs=True*) → None
Initialize self. See help(type(self)) for accurate signature.

__le__(*other*)
Automatically created by attrs.

__lt__(*other*)
Automatically created by attrs.

__module__ = 'labgrid.util.managedfile'

__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)

labgrid.util.marker module

labgrid.util.marker.**gen_marker**()

labgrid.util.proxy module

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({'__module__': 'labgrid.util.qmp', '__attrs_post_init__': <f

__init__(*monitor_out*, *monitor_in*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.util.qmp'

__repr__()
Automatically created by attrs.

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

labgrid.util.ssh module

class labgrid.util.ssh.**SSHConnection**(*host*)
Bases: object

SSHConnections are individual connections to hosts managed by a control socket. In addition to command execution this class also provides an interface to manage port forwardings. These are used in the remote infrastructure to tunnel multiple connections over one SSH link.

A public identity infrastructure is assumed, no extra username or passwords are supported.

__attrs_post_init__()

get_prefix()

run(*command*, *, *codec='utf-8'*, *decodeerrors='strict'*, *force_tty=False*, *stderr_merge=False*, *stderr_loglevel=None*, *stdout_loglevel=None*)
Run a command over the SSHConnection

Parameters

- **command** (*string*) – The command to run
- **codec** (*string, optional*) – output encoding. Defaults to "utf-8".
- **decodeerrors** (*string, optional*) – behavior on decode errors. Defaults to "strict". Refer to stdtypes' bytes.decode for details.
- **force_tty** (*bool, optional*) – force allocate a tty (ssh -tt). Defaults to False
- **stderr_merge** (*bool, optional*) – merge ssh subprocess stderr into stdout. Defaults to False.
- **stdout_loglevel** (*int, optional*) – log stdout with specific log level as well. Defaults to None, i.e. don't log.
- **stderr_loglevel** (*int, optional*) – log stderr with specific log level as well. Defaults to None, i.e. don't log.

Returns (stdout, stderr, returncode)

run_check(*command*, *, *codec='utf-8'*, *decodeerrors='strict'*, *force_tty=False*, *stderr_merge=False*, *stderr_loglevel=None*, *stdout_loglevel=None*)
Runs a command over the SSHConnection returns the output if successful, raises ExecutionError otherwise.

Except for the means of returning the value, this is equivalent to run.

Parameters

- **command** (*string*) – The command to run
- **codec** (*string, optional*) – output encoding. Defaults to "utf-8".
- **decodeerrors** (*string, optional*) – behavior on decode errors. Defaults to "strict". Refer to stdtypes' bytes.decode for details.
- **force_tty** (*bool, optional*) – force allocate a tty (ssh -tt). Defaults to False
- **stderr_merge** (*bool, optional*) – merge ssh subprocess stderr into stdout. Defaults to False.
- **stdout_loglevel** (*int, optional*) – log stdout with specific log level as well. Defaults to None, i.e. don't log.
- **stderr_loglevel** (*int, optional*) – log stderr with specific log level as well. Defaults to None, i.e. don't log.

Returns

stdout of the executed command if successful and otherwise an ExecutionError Exception

Return type List[str]

get_file(*remote_file*, *local_file*)
Get a file from the remote host

put_file(*local_file*, *remote_path*)
Put a file onto the remote host

add_port_forward(*remote_host*, *remote_port*)
forward command

remove_port_forward(*remote_host*, *remote_port*)
cancel command

connect()

disconnect()

isconnected()

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

__dict__ = mappingproxy({'__module__': 'labgrid.util.ssh', '__doc__': 'SSHConnection

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*host*) → None
Initialize self. See help(type(self)) for accurate signature.

__le__(*other*)
Automatically created by attrs.

__lt__(*other*)
Automatically created by attrs.

__module__ = 'labgrid.util.ssh'

__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)

exception labgrid.util.ssh.**ForwardError**(*msg*)
Bases: Exception

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

__eq__(*other*)
Return self==value.

__ge__(*other*)
Automatically created by attrs.

__gt__(*other*)
Automatically created by attrs.

__hash__ = None

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__le__(*other*)
Automatically created by attrs.

__lt__(*other*)
Automatically created by attrs.

__module__ = 'labgrid.util.ssh'

__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)

labgrid.util.timeout module

class labgrid.util.timeout.**Timeout**(*timeout=120.0*)
Bases: object

Reperents a timeout (as a deadline)

__attrs_post_init__()

property remaining

property expired

__attrs_attrs__ = (Attribute(name='timeout', default=120.0, validator=<instance_of val

__dict__ = mappingproxy({'__module__': 'labgrid.util.timeout', '__doc__': 'Reperents

__init__(*timeout=120.0*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.util.timeout'

__repr__()
Automatically created by attrs.

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

labgrid.util.yaml module

This module contains the custom YAML load and dump functions and associated loader and dumper

labgrid.util.yaml.**load**(*stream*)
Wrapper for yaml load function with custom loader.

labgrid.util.yaml.**dump**(*data*, *stream=None*)
Wrapper for yaml dump function with custom dumper.

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 valida

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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'

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__()

property display_name

on_supplier_bound(*supplier*)
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*)
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Return repr(self).

__dict__ = mappingproxy({'__module__': 'labgrid.binding', '__doc__': '\n Marks a

__module__ = 'labgrid.binding'

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

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

__dict__ = mappingproxy({'__module__': 'labgrid.binding', '__doc__': '\n Handles the

__init__(*target*, *name*) → None
Initialize self. See help(type(self)) for accurate signature.

__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

get_features()

__attrs_attrs__ = (Attribute(name='filename', default=NOTHING, validator=<instance_of

__dict__ = mappingproxy({'__module__': 'labgrid.config', '__attrs_post_init__': <fun

__init__(*filename*) → None
Initialize self. See help(type(self)) for accurate signature.

__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*)
Initialize self. See help(type(self)) for accurate signature.

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({'__module__': 'labgrid.consoleloggingreporter', '__doc__':

__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.

get_features()

get_target_features()

cleanup()

__attrs_attrs__ = (Attribute(name='config_file', default='config.yaml', validator=<ins

__dict__ = mappingproxy({'__module__': 'labgrid.environment', '__doc__': 'An environ

__init__(*config_file='config.yaml'*, *interact=<built-in function input>*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*, *filter=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*, *filter=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__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 valida

__init__(*msg*, *filter=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.exceptions'

__repr__()
Automatically created by attrs.

9.1.8 labgrid.factory module

class labgrid.factory.**TargetFactory**
Bases: object

__init__()
Initialize self. See help(type(self)) for accurate signature.

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.

static normalize_config(*config*)

make_resource(*target*, *resource*, *name*, *args*)

make_driver(*target*, *driver*, *name*, *args*)

make_target(*name*, *config*, *, *env=None*)

__dict__ = mappingproxy({'__module__': 'labgrid.factory', '__init__': <function Targ

__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__()
Initialize self. See help(type(self)) for accurate signature.

get_current()

get_new(*title*, *tag*, *source*)

push(*step*)

pop(*step*)

subscribe(*callback*)

unsubscribe(*callback*)

notify(*event*)

__dict__ = mappingproxy({'__module__': 'labgrid.step', '__init__': <function Steps.__

__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*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return str(self).

__setitem__(*k*, *v*)

merge(*other*)

property age

__dict__ = mappingproxy({'__module__': 'labgrid.step', '__init__': <function StepEve

__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*)
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Return repr(self).

__str__()
Return str(self).

property duration

property status

property is_active

property is_done

start()

skip(*reason*)

stop()

__del__()

__dict__ = mappingproxy({'__module__': 'labgrid.step', '__init__': <function Step.__

__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__()
Initialize self. See help(type(self)) for accurate signature.

static notify(*event*)

__dict__ = mappingproxy({'__module__': 'labgrid.stepreporter', 'instance': None, 'st

__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_post_init__()

interact(*msg*)

update_resources()
Iterate over all relevant resources and deactivate any active but unavailable resources.

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

get_resource(*cls*, *, *name=None*, *wait_avail=True*)
Helper function to get a resource of the target. Returns the first valid resource found, otherwise a NoResourceFoundError is raised.

Arguments: cls – resource-class to return as a resource name – optional name to use as a filter wait_avail – wait for the resource to become available (default True)

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)

__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', ...)
```

set_binding_map(*mapping*)
Configure the binding name mapping for the next driver only.

bind_resource(*resource*)
Bind the resource to this target.

bind_driver(*client*)
Bind the driver to all suppliers (resources and other drivers).

Currently, we only support binding all suppliers at once.

bind(*bindable*)

activate(*client*)
Activate the client by activating all bound suppliers. This may require deactivating other clients.

deactivate(*client*)
Recursively deactivate the client's clients and itself.

This is needed to ensure that no client has an inactive supplier.

deactivate_all_drivers()
Deactivates all drivers in reversed order they were activated

cleanup()
Clean up conntected drivers and resources in reversed order

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

__dict__ = mappingproxy({'__module__': 'labgrid.target', '__attrs_post_init__': <fun

__init__(*name*, *env=None*) → None
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'labgrid.target'

__repr__()
Automatically created by attrs.

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

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y