
LTD Platform

Release 2017.09.1

Oct 30, 2017

Contents

1	Zephyr microPlatform	3
1.1	LTD Zephyr microPlatform	3
2	Linux microPlatform	21
2.1	LTD Linux microPlatform	21
3	Basic IoT Gateway (BIG)	27
3.1	LTD Basic IoT Gateway	27
4	End-to-end Open Source IoT Demonstration Systems	31
4.1	End-to-end Demonstration Systems	31

This is the documentation for the Linaro Technologies Division.

Zephyr microPlatform

The *LTD Zephyr microPlatform* is an extensible software and hardware platform that makes it easier to develop, secure, and maintain Internet-connected embedded devices. The Zephyr microPlatform is based on the [Zephyr](#) real-time operating system.

LTD Zephyr microPlatform

The Zephyr microPlatform is an extensible software and hardware platform that makes it easier to develop, secure, and maintain Internet-connected embedded devices. The Zephyr microPlatform is based on the [Zephyr](#) real-time operating system.

Getting Started

All you need to get started is a development board supported by the Zephyr microPlatform, a computer to develop on, and an Internet connection.

Todo

Add link to a top-level “supported boards” page when that’s ready.

Get Hardware

Here’s what you’ll need:

- A development computer, running one of:
 - macOS (experimental; we test on Sierra, 10.12)
 - 64 bit Windows 10 Anniversary Update or later (experimental)
 - a 64 bit Linux distribution (we test on [Ubuntu 16.04.](#))

- A development board supported by the Zephyr microPlatform. We support the [96Boards Nitrogen](#), and other boards on a best effort basis.

Set up Build Environment

Before installing the the Zephyr microPlatform, you need to set up your workstation build environment. Instructions for each supported platform follow.

macOS

1. Install [HomeBrew](#).
2. Install dependencies for the Zephyr microPlatform:

```
brew install dtc python3 repo gpg
pip3 install --user ply pyyaml pycrypto pyasn1 ecdsa pyelftools
```

3. Install the tools you need to flash your board.

For [96Boards Nitrogen](#), you'll need [pyOCD](#), which you can install with [Python 2](#) from [HomeBrew](#):

```
brew install python
pip2 install --user pyOCD
export PATH=$PATH:$HOME/Library/Python/2.7/bin
```

Otherwise, check your board's documentation.

4. **Optional:** Set up [Git](#):

```
git config --global user.name "Your Full Name"
git config --global user.email "your-email-address@example.com"
```

5. **Optional:** If you want to build this documentation, you'll need some additional dependencies:

```
pip3 install --user sphinx sphinx_rtd_theme

# Replace "3.X" with the version number you have installed.
export PATH=$PATH:$HOME/Library/Python/3.X/bin
```

Your build environment is now ready; continue by following the steps in [Install Zephyr microPlatform](#).

Windows 10 (Experimental)

Windows versions supporting the Windows Subsystem for Linux have experimental support. These instructions will let you build binaries; however, flashing support is not yet documented.

1. Install the [Windows Subsystem for Linux](#), then open a Bash window to enter commands.
2. Change to your Windows user directory with a command like this:

```
cd /mnt/c/Users/YOUR-USER-NAME
```

You can press the Tab key after typing `/Users/` to see a list of user names.

Warning: Skipping this step means you won't be able to use the microPlatform with Windows tools like Explorer, graphical editors, etc.

As documented by Microsoft, [changing files in Linux directories using Windows tools](#) can damage your system.

3. We recommend making sure your Linux subsystem is up to date with these commands (which can take a while they first time they're run):

```
apt-get update
apt-get upgrade
```

4. Finish by following the Ubuntu instructions in the next section.

Linux

1. Install dependencies for the Zephyr microPlatform.

On Ubuntu 16.04:

```
sudo add-apt-repository ppa:linaro-maintainers/ltd
sudo apt-get update
sudo apt-get install genesis-dev
pip3 install --user pyelftools
```

On other distributions, see [Appendix: Zephyr microPlatform Dependencies](#).

2. Install the tools you need to flash your board.

For 96Boards Nitrogen, you'll need [pyOCD](#), which you can install with `pip`:

```
pip install --user pyOCD
```

On Linux platforms, you also need to install the following udev rules as root, then unplug and plug back in any boards you may have connected:

```
echo 'ATTR{idProduct}=="0204", ATTR{idVendor}=="0d28", MODE="0666", GROUP="plugdev
↪"' > /etc/udev/rules.d/50-cmsis-dap.rules
```

3. **Optional:** Set up Git:

```
git config --global user.name "Your Full Name"
git config --global user.email "your-email-address@example.com"
```

Your system is now ready to install the Zephyr microPlatform.

Install Zephyr microPlatform

Todo

Generate instructions for other manifest repository sources.

In these configurations, we need extra docs:

- Cache Git usernames and passwords you enter in memory for one hour; this allows `repo sync` to work unprompted in the next step. If you don't want to do this, see <https://git-scm.com/docs/gitcredentials> for alternatives.

```
git config --global credential.helper 'cache --timeout=3600'
```

- If you don't already have one, create a [GitHub](#) account (it's free).
- Make sure you can see the Zephyr microPlatform SDK manifest repository when you're logged in to your account (**needs link**).
- If you enabled [two-factor authentication](#) on your GitHub account, you also need a [personal access token](#). Give this token at least "repo" access, and make sure you keep a copy.
- When prompted by `repo init`, enter your GitHub username and password (or access token, if you use two-factor authentication).

To install the latest release, make an installation directory and install the Zephyr microPlatform there with `repo`:

```
mkdir genesis && cd genesis
repo init -u https://github.com/linaro-technologies/genesis-sdk-manifest
repo sync
```

Note: If you're new to `repo` and want to know more, see *Zephyr microPlatform and Repo Primer*.

Build an Application

Now that you've installed the Zephyr microPlatform, it's time to build a demonstration application.

Since one of the main features of the microPlatform is making it easy to build application binaries which are cryptographically checked by `mcuboot`, a secure bootloader, you'll first build a simple "Hello World" application provided by `mcuboot`.

If you're using 96Boards Nitrogen, run this from the `genesis` directory you made earlier:

```
./genesis build mcuboot/samples/zephyr/hello-world
```

If you're using another board, run this instead:

```
./genesis build -b your_board mcuboot/samples/zephyr/hello-world
```

Where `your_board` is Zephyr's name for your board. (Here's a [list of Zephyr boards](#), but some of them may not work with the Zephyr microPlatform.)

(If you want to know more, see *Build an Application: genesis build*.)

Flash the Application

Now you'll flash the application to your board.

If you're using 96Boards Nitrogen, plug it into your computer via USB, then run this from the the Zephyr microPlatform directory:

```
./genesis flash mcuboot/samples/zephyr/hello-world
```

If you're using another board, make sure it's connected, and use this instead:

```
./genesis flash -b your_board mcuboot/samples/zephyr/hello-world
```

Congratulations; you've just flashed a bootloader and cryptographically signed application binaries¹ you built in the previous step onto your board!

(If you want to know more, see *Flash an Application to a Device: genesis flash*.)

Test the Application

You're now ready to test the application itself.

If you're using a 96Boards Nitrogen:

- Make sure it's plugged into computer via USB. A serial port device (usually named `/dev/ttyACM0` on Linux, but the number may change if you've got other devices plugged in) will be created when the board enumerates.
- Open the device with your favorite serial console program² at 115200 baud.
- Reset the chip by pressing the RST button on the board.

You should see some messages printed in the serial console.

When you power on or reset the board:

1. The mcuboot bootloader runs first, and checks the cryptographic signature on the application binary.
2. If the signature is valid for the given binary, will run the application itself.
3. The application you just built will print a "Hello World" message on screen.

The combined output looks like this:

```
[MCUBOOT] [INF] main: Starting bootloader
[MCUBOOT] [INF] boot_status_source: Image 0: magic=good, copy_done=0xff, image_ok=0xff
[MCUBOOT] [INF] boot_status_source: Scratch: magic=unset, copy_done=0x23, image_
→ok=0xff
[MCUBOOT] [INF] boot_status_source: Boot source: slot 0
[MCUBOOT] [INF] boot_swap_type: Swap type: none
[MCUBOOT] [INF] main: Bootloader chainload address offset: 0x8000
[MCUBOOT] [WRN] zephyr_flash_area_warn_on_open: area 1 has 1 users
[MCUBOOT] [INF] main: Jumping to the first image slot
***** BOOTING ZEPHYR OS v1.8.99 - BUILD: Aug 15 2017 19:41:06 *****
Hello World from Zephyr on 96b_nitrogen!
```

If you're using another board, you may need to do something slightly different, but the basic idea is the same: connect a serial console at 115200 baud, and reset the chip.

That's it! You've successfully installed the Zephyr microPlatform, compiled an application, flashed it to a device, and seen it work.

¹ Since this tutorial is meant to help you get started, the binaries are signed with keys that aren't secret, and **are not suitable for production use**. When it's time to ship, see *Production Workflows* for more information.

² On Linux, with `picocom`:

```
picocom -b 115200 /dev/ttyACM0
```

On Linux or macOS, with `screen`:

```
screen /dev/ttyACM0 115200
```

To use PuTTY on another computer running Windows, see *Connecting to a local serial line* in the PuTTY documentation.

Onwards!

You're now ready to take your next steps.

Todo

Add links to next steps documents when they're ready.

Example of tutorials and reference docs:

- Zephyr microPlatform overview (different projects with links to their reference docs, how they tie together, e.g. description of boot process with links to mcuboot documentation).
 - Hardware peripheral tutorials (UART, SPI, etc.)
 - Internet connectivity with an Basic IoT Gateway
 - FOTA with hawkBit
-

Appendixes

Appendix: Zephyr microPlatform Dependencies

Here is a list of dependencies needed to install the Zephyr microPlatform with these instructions, which may be useful on other development platforms.

- Device tree compiler (dtc)
- Git
- GNU Make
- GCC and G++ with 32-bit application support
- bzip2
- Python 3 with the following packages:
 - setuptools
 - Sphinx
 - Sphinx RTD theme
 - PLY
 - PyYaml
 - Crypto
 - ECDSA
 - ASN.1
 - pyelftools
- Google Repo

Appendix: Zephyr microPlatform Development Container (Experimental)

You can install a Docker container based on Ubuntu 16.04 which provides a Zephyr microPlatform build environment. However, instructions for flashing binaries you build with this container are not yet provided.

1. Install Docker.
2. Fetch the container:

```
docker pull linarotechnologies/genesis-sdk:latest
```

3. **Optional:** Create a mount in your host environment to access the builds; see the [Docker documentation on data management](#) for more details.

On **macOS only**, you can just create a directory to contain the SDK sources and build artifacts in your host file system. For example:

```
mkdir genesis
```

4. Run the container as the `genesis-dev` user, granting it access to the host data area if you created one.

For example:

```
docker run -it -w /home/genesis-dev -u genesis-dev genesis-sdk
```

If you created a directory in your macOS environment, it's easier to run as the root user in the container:

```
docker run -it -v genesis:/root/genesis -w /root/genesis genesis-sdk
```

5. **Optional:** Set up Git inside the container:

```
git config --global user.name "Your Full Name"
git config --global user.email "your-email-address@example.com"
```

You can now follow the above instructions to *install the Zephyr microPlatform* inside the running container.

Branch Management

Todo

Add a few good diagrams.

This document defines the rules governing the branches in the Zephyr microPlatform Git repositories, and what you can expect from them.

Why Have Branching Rules?

The short answer is that it's the only way to keep things working while staying close to our upstream projects' latest versions.

The details are given below in *Appendix: Branch Management Rationale*.

Zephyr microPlatform and Repo Primer

Below sections describe the branches in the Zephyr microPlatform manifest and source code repositories, and how they are related. Before getting there, this section gives some background on how the Zephyr microPlatform uses Repo, which may make that explanation clearer.

As described in *Getting Started*, every Zephyr microPlatform installation contains multiple Git repositories, which are managed by a *manifest file* in a *Repo manifest repository*.

The name of the manifest repository is `genesis-sdk-manifest`. It's a Git repository, just like any of the source code repositories. While installing the Zephyr microPlatform, you passed `repo init` a URL for the manifest repository. The manifest repository is special, in that it contains an XML manifest file, named `manifest.xml`, which describes all of the other Git repositories in the Zephyr microPlatform installation. After `repo init`, you ran `repo sync`, which parsed the manifest file and cloned all of the other Zephyr microPlatform repositories as instructed by its contents.

The manifest file contains:

- a list of *remotes*, each of which specifies a base URL where other Zephyr microPlatform Git repositories are hosted.
- a list of *projects*, each of which specifies a Git repository to clone, along with a remote to pull it from, and a revision to check out in the local clone.

An example manifest repository, its manifest file, and the manifest file's contents are as follows.

Since the `genesis-sdk-manifest` repository is a Git repository, it can, and does, contain multiple branches:

- one branch named `master`, which tracks *trunk development*, or the latest changes.
- a series of *monthly snapshot branches* named `YY.MM`, each of which tracks the state of development in month MM of year YY.

For example, the `17.05` monthly snapshot branch in the manifest repository contains a manifest file which tracks the work done in May 2017 for the Zephyr microPlatform source code repositories. Similarly, the `17.06` branch in the manifest repository contains a manifest tracking June 2017.

The other (non-manifest) Zephyr microPlatform Git repositories have branches named `ltd-YY.MM`. These contain development work for month MM of year YY.

Trunk Development

Note: The important things to know are:

- The `master` branch in the *manifest repository* tracks the **latest** monthly `ltd-YY.MM` branches in the other Zephyr microPlatform repositories.
 - Each month, Zephyr microPlatform repositories with upstreams, like Zephyr and mcuboot, **will rebase onto new upstream baseline commits** when new monthly branches are cut.
 - Currently, updates to Zephyr microPlatform repositories without upstreams are always *fast-forward*, even when new branches are cut. However, in the future, these may also rebase.
-

As described above, the `master` branch in the `genesis-sdk-manifest` repository tracks the very latest development.

Thus, to check out the very latest Zephyr microPlatform, you can run:

```
mkdir genesis && cd genesis
repo init -u https://github.com/linaro-technologies/genesis-sdk-manifest
repo sync
```

The `repo init` line clones a local manifest repository in `genesis/.repo/manifests`, and creates and checks out a branch called `default` that tracks `master` in the remote manifest repository. The `repo sync` line fetches the latest changes in this `master` branch, parses the resulting XML manifest file, and updates the local repositories based on its new contents.

Continuing the above example, in May 2017, the manifest file in the manifest repository's `master` branch might look like this:

```
<manifest>
  <remote name="ltd" fetch="https://github.com/linaro-technologies"/>

  <project name="zephyr" remote="ltd" revision="ltd-17.05"/>
  <project name="zephyr-fota-hawkbite" remote="ltd" revision="ltd-17.05"/>
  <!-- Other projects, etc. -->
</manifest>
```

Running `repo sync` again during the same month will fetch changes from the same upstream `ltd-17.05` branches, and attempt to rebase any locally checked out branches on top of them.

At the end of each month, the `master` branch in the manifest repository is updated so its manifest file synchronizes from the next month's branches.

Thus, in the beginning of June 2017, the manifest file is updated to look like this:

```
<manifest>
  <remote name="ltd" fetch="https://github.com/linaro-technologies"/>

  <project name="zephyr" remote="ltd" revision="ltd-17.06"/>
  <project name="zephyr-fota-hawkbite" remote="ltd" revision="ltd-17.06"/>
  <!-- Other projects, etc. -->
</manifest>
```

Running `repo sync` after this happens fetches and synchronizes your local trees with the `ltd-17.06` branches in each of the Zephyr microPlatform projects named in the manifest. (See `repo sync` for details.)

Warning: When this happens, **upstream Git history is rewritten** for Zephyr microPlatform repositories which have an upstream, like Zephyr and `mcuboot`. This happens because the next month's development branch is rebased onto a new baseline commit from upstream.

For more information, see *Extra Rules For Repositories with Upstreams*.

Monthly Snapshot Branches

Note: The important things to know are:

- Each `YY.MM` branch in the *manifest repository* tracks the monthly `ltd-YY.MM` branches in each of the other Zephyr microPlatform repositories.
- Running `repo sync` with this manifest branch results in **fast-forward** changes only in upstream repositories.
- At the end of the month, **upstream development stops** in all of these snapshot branches. You need to update to a newer manifest branch to get more recent changes.

As described above, the manifest repository has multiple `YY.MM` branches, each of which tracks development in month `MM` of year `YY`, e.g. `17.05` for May of 2017.

To check out one of these monthly snapshots, run:

```
mkdir genesis && cd genesis
repo init -b YY.MM -u https://github.com/linaro-technologies/genesis-sdk-manifest
repo sync
```

This clones local repositories tracking `ltd-YY.MM` branches. Running `repo sync` again later fetches the latest `ltd-YY.MM` branches from remote repositories, and attempts to `rebase` any locally checked out branches on top of the latest from upstream.

You can sync the latest changes to upstream repositories using the current month's snapshot branch. All updates to remote repositories will be fast-forward changes only. However, **updates will stop after the month ends** and trunk development continues on new branches.

You can continue using the Zephyr microPlatform at your site for as long as you'd like, even when you're using a monthly snapshot manifest branch. However, to fetch new updates from Linaro Technologies Division after the month ends, you need to update your manifest repository to sync from more recent development branches. You can do this using an existing Zephyr microPlatform installation directory; **you do not need to create a new Zephyr microPlatform directory to update your manifest repository branch.**

For example, if you have the `17.05` manifest branch checked out, and you want to update to `17.07`, you can run this from your existing Zephyr microPlatform installation directory:

```
repo init -b 17.07 -u https://github.com/linaro-technologies/genesis-sdk-manifest
repo sync
```

Warning: When changing manifest branches, you may synchronize based on upstream repository changes that are not fast-forward updates to what you have already cloned. This may rewrite Git history in your local repositories. Be careful!

You can use `repo sync -n` to fetch changes from the network only, without updating your working directories. See *Zephyr microPlatform and Repo Primer* for more information.

Monthly Baseline Rebases

As noted above, some repositories have their history rewritten when new monthly development branches are cut. This currently only happens to repositories which have upstreams, namely Zephyr and mcuboot.

For example, in May 2017, the `zephyr` repository tracked the `ltd-17.05` branch in the Linaro Technologies Division Zephyr Git tree. When development moved to the `ltd-17.06` branch in early June 2017, the `zephyr` repository was updated so that Linaro Technologies Division changes to the mainline Zephyr source code start at a new **baseline commit** in the upstream repository's mainline (master) branch.

When a new baseline commit is established, the history for the commits that LTD added to the upstream branch is rewritten and cleaned up (squashing commits, removing hacks that are no longer needed, etc.). See *Extra Rules For Repositories with Upstreams*, below, for rules which make it easy to see which commits those are.

What about Upstream Releases?

We don't currently take baseline commits in any LTD branches from upstream release branches. That is, both trunk development and monthly snapshots are based on commits in upstream master branches.

However, changes from upstream release branches may be cherry-picked or otherwise merged into monthly snapshot branches.

Extra Rules For Repositories with Upstreams

Note: The important thing to know is:

When Linaro Technologies Division adds patches to a repository with an upstream, we add an "LTD" tag in the Git shortlog to mark the commit as currently LTD-specific.

These tags are called "sauce tags".

Here is list of sauce tags, with a brief summary of their purposes:

- [LTD toup]: patches that want to go upstream, and revisions to them
- [LTD noup]: patches needed by LTD, but not for upstream
- [LTD mergeup]: merge commits from upstream into an LTD tree
- [LTD temp hack]: patches needed temporarily until some underlying code is fixed or refactored upstream
- [LTD fromtree]: patches cherry-picked from upstream (when they're only available in a newer version that can't be merged)
- [LTD fromlist]: patches propose for upstream that are under discussion and are still being merged, and revisions to them.

More detailed rules for each sauce tag follow below.

[LTD toup]

Use this for patches that are submitted upstream. Also use this for subsequent revisions to the LTD branch which follow upstream review, and make it possible to [autosquash](#) them together in the next baseline rebase.

For example, let's take this series posted upstream:

```
boards: arm: add sweet_new_board
samples: http_client: support sweet_new_board
```

The shortlogs in the master-upstream-dev branch should be:

```
[LTD toup] boards: arm: add sweet_new_board
[LTD toup] samples: http_client: support sweet_new_board
```

Then, after rebasing the review series in response to changes requested to the "add sweet_new_board" patch, add another commit to master-upstream-dev that makes the same change, like this:

```
[LTD toup] boards: arm: add sweet_new_board
[LTD toup] samples: http_client: support sweet_new_board
(other commits in between)
squash! [LTD toup] boards: arm: add sweet_new_board
```

When the patches are merged into upstream master and it's time to merge that into master-upstream-dev, first propose a revert, then do the merge, like so:

```
[LTD toup] boards: arm: add sweet_new_board
[LTD toup] samples: http_client: support sweet_new_board
(...)
squash! [LTD toup] boards: arm: add sweet_new_board
(...)
Revert "[LTD toup] samples: http_client: support sweet_new_board"
Revert "[LTD toup] boards: arm: add sweet_new_board"
(...)
Merge master into master-upstream-dev
```

[LTD noup]

Use this if the patch isn't upstreamable for whatever reason, but it's still needed in the LTD trees. Use good judgement between this and [LTD tempack].

[LTD mergeup]

Use this for merge commits from upstream into an LTD tree.

[LTD tempack]

Use this for patches which "get things working again", but are unacceptable to upstream, and will be dropped at some point when rebasing to a new baseline commit.

For example, use this if the patch wraps new code added upstream with `#if 0 ... #endif` because it broke something, while a better fix is being worked out.

[LTD fromtree]

When patches are cherry-picked from a later upstream version. **Do not rewrite upstream's history with this tag** when merging upstream master into LTD master-upstream-dev.

[LTD fromlist]

When you've cherry-picked a commit proposed for inclusion upstream. Note that if you want to include changes to that patch made during review, follow the same autosquash rules as [LTD toup].

Appendix: Branch Management Rationale

This section provides a rationale for why these rules exist.

There are two "types" of repository in an Zephyr microPlatform installation:

- Projects which have an external upstream, namely Zephyr and mcuboot.
- Projects which are developed for the Zephyr microPlatform, and which have no external upstream, like the one containing the documentation you're reading now.

Rather than cloning the upstream versions of the Zephyr and mcuboot repositories in an Zephyr microPlatform installation, Linaro Technologies Division maintains its own trees. This is for two reasons.

1. It allows us to keep track of known-good revisions that work well with the Zephyr microPlatform.
2. It gives us a place to carry out our own internal development on these repositories.

Changes flow in both directions between the LTD trees and the upstream trees. In one direction, we're constantly upstreaming these changes as we add features, fix bugs, etc. In the other, we're keeping track of what's going on upstream, and merging in new patches as they arrive and are tested. We also sometimes need to keep some temporary solutions or patches in our trees which aren't useful for upstream.

While all of this is going on in repositories with an upstream, the Zephyr microPlatform-only repositories are evolving too, both to use those new features added in Zephyr and mcuboot, and as they're being developed in their own right.

This gets complicated, and some extra process is necessary to keep things working smoothly over time.

The branching rules manage development in a way that allows:

- Users to see clearly what the differences are between the upstream and Zephyr microPlatform versions of each repository,
- Developers to stage local and integrate upstream changes into Zephyr microPlatform branches,
- Continuous Integration to track versions which should work together for testing and test report generation,
- Snapshots and releases to track the state of development over time, allowing comparisons between versions.

Development Workflows

This page describes the workflows for developing and deploying embedded applications with the Zephyr microPlatform. It assumes that the Zephyr microPlatform has successfully been installed as described in *Getting Started*.

Helper Script

After installing the Zephyr microPlatform repositories and build environment, the Zephyr and mcuboot build systems and other tools can be used directly. However, these interfaces can be hard to use when first developing applications. For this reason, the Zephyr microPlatform provides a helper script, also named `genesis`, which provides a higher-level interface.

The `genesis` utility is installed into the root of the Zephyr microPlatform tree by `repo sync`. It accepts multiple commands useful during development; they are documented below. Run `./genesis -h` from the Zephyr microPlatform installation directory for additional information.

Build an Application: `genesis build`

Warning: By default, mcuboot binaries and Zephyr microPlatform applications are built and signed with development keys which are not secret. While this makes development and testing more convenient, it is not suitable for production. See *Production Workflows* for more information.

It's not currently possible to generate mcuboot images that trust non-dev keys. As such, the `--signing-key` and `--signing-key-type` arguments to `genesis build` are misleading, as the mcuboot image won't trust the key used to sign the application. Don't use these for now.

Todo

Re-work after resolution of <https://trello.com/c/mSZPuXxG> and <https://projects.linaro.org/browse/LITE-147>

The top-level command is `genesis build`. By default, it takes a path to an application inside the Zephyr microPlatform installation directory, and builds a signed application image, as well as an mcuboot binary capable of loading that application image. (The default behavior can be changed through various options.)

To get help, run this from the Zephyr microPlatform root directory:

```
./genesis build -h
```

The `genesis build` command always builds out of tree; that is, build artifacts are never generated in the source code directories. By default, they are stored under `outdir` in the Zephyr microPlatform top-level directory.

Examples:

- To build an application `some-application` available in the Zephyr microPlatform tree, targeting the default board (`96b_nitrogen`):

```
./genesis build some-application
```

This generates artifacts under `outdir` like so:

```
outdir
- some-application
  - 96b_nitrogen
    - app
    - mcuboot
```

The application build for `96b_nitrogen` is in `outdir/some-application/96b_nitrogen/app`. The `mcuboot` build is in `mcuboot`, next to `app`.

- To build the same application for another board, e.g. `96b_carbon`, use the `-b` option:

```
./genesis build -b 96b_carbon some-application
```

The `-b` option can be used in any `genesis build` command to target other boards.

Running this after building for 96Boards Nitrogen as in the above example results in a parallel set of build artifacts, like so:

```
outdir
- some-application
  - 96b_carbon
    | - app
    | - mcuboot
  - 96b_nitrogen
    - app
    - mcuboot
```

- It's fine to build application sources in a subdirectory. For example, running:

```
./genesis build some-nested/application-name
```

will generate:

```
outdir
- some-nested
  - application-name
    - 96b_nitrogen
      - app
      - mcuboot
```

Note that the signed image in `96b_nitrogen/app` is named `application-name-96b_nitrogen-signed.bin`; i.e., just the base name of the application directory is used.

- To build or incrementally compile the application image only, not updating the `mcuboot` image, use `-o`:

```
./genesis build -o app some-application
```

- Similarly, to build or incrementally compile mcuboot only:

```
./genesis build -o mcuboot some-application
```

Configure an Application: `genesis configure`

The Zephyr RTOS uses a configuration system called Kconfig, which is borrowed from the Linux kernel. The `genesis configure` command lets you change the configuration database for an application build, using any of the Kconfig front-ends supported on your platform.

The top-level command is `genesis configure`.

This command can only be run after using `genesis build` to create the build directory, which contains the configuration database.

To get help, run this from the Zephyr microPlatform root directory:

```
./genesis configure -h
```

Example uses:

- To change the application configuration (not the mcuboot configuration) for `some-application` for the default board:

```
./genesis configure -o app some-application
```

- To change the mcuboot (not application) configuration for another board, `96b_carbon`:

```
./genesis configure -o mcuboot -b 96b_carbon some-application
```

If you don't specify `-o`, then `genesis configure` will let you change both the mcuboot and application configurations.

Note that `genesis configure` accepts many of the same options as *`genesis build`*.

For more information on Kconfig in Zephyr, see [Configuration Options Reference Guide](#).

Flash an Application to a Device: `genesis flash`

After building an application and mcuboot binary with *`genesis build`*¹, the `genesis flash` command can be used to flash it to a board, usually via USB.

The `genesis flash` command uses information about the board obtained from Zephyr's build system to choose a flashing utility, and run it with the correct arguments to flash mcuboot and the application binary to an attached board. Before using this command, make sure you can flash your board using the Zephyr `make flash` command as described in its [Zephyr documentation](#)².

To get help, run this from the Zephyr microPlatform root directory:

¹ It's possible to use `genesis flash` on directories not generated by `genesis build`, but it assumes an output directory hierarchy matching what *`genesis build`* creates, including the presence of a `Makefile.export`.

² If your board's Zephyr support does not include `make flash`, `genesis flash` will not work either. `genesis flash` exists because the Zephyr `make flash` target currently only allows flashing a single application binary to a board at a fixed address. This is not sufficient for the Zephyr microPlatform, which has a more complex flashing process due to the presence of a bootloader and an application, which must be flashed in different locations.

```
./genesis flash -h
```

Basic uses:

- To flash the artifacts for `some-application` to the default board:

```
./genesis flash some-application
```

- To flash to a different board, `96b_carbon`:

```
./genesis flash -b 96b_carbon some-application
```

- To flash to a particular board, given the device ID supported by its underlying flashing utility:

```
./genesis flash -d SOME_BOARD_ID some-application
```

The command also accepts an `-e` argument, which can be used to pass extra arguments to the flashing utility.

Create an Application

Todo

fill this in when it's possible.

<https://trello.com/c/Yj5vW4zf>

<https://projects.linaro.org/browse/LITE-91>

<https://projects.linaro.org/browse/LITE-125>

Debug a Running Application

Todo

improve this.

Attach a debugger in the host environment to the device, and provide the ELF binaries to it for symbol tables. On boards which support CMSIS-DAP, `pyOCD` is the recommended solution.

Integrate an External Dependency

Todo

user-friendly instructions, post-CMake transition.

Integrating external dependencies with Zephyr is currently not straightforward. One approach is to copy them into your application repository, either directly or as submodules.

Additional information is available in the Zephyr [Application Development Primer](#).

Use Repo to Manage Git Repositories

Note: After first installing the Zephyr microPlatform, use of Repo is optional. Since Repo is essentially a wrapper around Git, it's possible to use `git` commands directly in individual repositories as well.

The Zephyr microPlatform uses the Repo tool to manage its Git repositories. In *Install Zephyr microPlatform*, you used this tool to clone these Git repositories into an Zephyr microPlatform installation directory on a development computer.

After the installation, you can continue to use Repo to manage local branches and fetch upstream changes. Importantly, you can use:

- `repo start` to create local Git branches in multiple repositories.
- `repo status` to get status output about each Zephyr microPlatform repository (this is similar to `git status`, but operates on all repositories).
- `repo diff` to get a diff of unstaged changes in each Git repository (this is similar to `git diff`, but operates on all repositories).
- `repo sync` to fetch remote changes from all Zephyr microPlatform repositories, and rebase local Git branches on top of them (alternatively, use `repo sync -n` to fetch changes only, without rebasing).

See the [Repo command reference](#) for more details. However, note that because the **Zephyr microPlatform does not use Gerrit** as a Git repository server, `repo` commands which expect a Gerrit server are not applicable to an Zephyr microPlatform installation. For example, instead of using `repo upload`, use `git push`.

You can also run `repo help <command>` to get usage for each `repo` command; for example, use `repo help sync` to get help on `repo sync`.

Production Workflows

Placeholder for production workflows.

Todo

Write this section.

- Minimum sane key management policies
 - Building production-ready mcuboot and application images (blocker: <https://trello.com/c/mSZPuXxG>)
 - Disabling JTAG/SWD or making physical access harder and other issues discussed in the threat model.
-

Linux microPlatform

The *LTD Linux microPlatform* is a minimal Linux distribution for deploying and managing multi-tenant, containerized applications on Linux devices. The Linux microPlatform is built using OpenEmbedded and adding a select set of board support package layers for enabling some popular development boards. Though the Linux kernel and software used for any on of the microPlatform builds may contain out-of-tree patches or features, a fundamental goal is to run as close to the tip, or latest software, as possible so that users of the Linux microPlatform can benefit from the latest changes.

LTD Linux microPlatform

The Linux microPlatform is an extensible software and hardware platform that makes it easier to develop, secure, and maintain Internet-connected embedded devices. The Linux microPlatform is based on the [Open Embedded](#), [Linux Kernel](#), [Docker](#), and other open source software creating an intentionally designed minimal distribution for embedded systems.

Getting Started

All you need to get started is a gateway device supported by the Linux microPlatform, a computer, and an Internet connection.

Get Hardware

Here's what you'll need:

- A computer to develop on. This can be running Windows, Mac OS X, or Linux.
- A gateway device supported by the Linux microPlatform. We currently support the [96Boards HiKey](#), and assume you have a [96Boards UART Serial Adapter](#) for console access.

Get Installation Dependencies

To install the Linux microPlatform on your device, you'll need Python 2, pySerial, drivers for FTDI serial port devices, and Android's fastboot tool.

Windows

- Install the latest [Python 2](#) release for Windows.
- Install [pySerial](#) and the [FTDI drivers](#).
- Install fastboot with the latest [Android SDK Platform Tools for Windows](#).

Mac OS X

- Python 2 is installed by default by Apple.
- Install [pySerial](#) and the [FTDI drivers](#).
- Install fastboot with the latest [Android SDK Platform Tools for Mac](#).
- Optionally, install the latest [Ansible release for OS X](#). This will make it easier to deploy containers on your device.

Linux

On Debian-based Linux distributions, including Ubuntu, run:

```
sudo apt-get install python-serial fastboot
```

Optionally, install Ansible, to make it easier to deploy containers on your device:

```
sudo apt-get install ansible
```

On other Linux distributions:

- Python 2 may be installed by default, and should be available in your package manager if not. You can also [install Python from source](#).
- [pySerial](#) is also likely available via your package manager or pip.
- Most distribution kernels provide FTDI USB serial port device support.
- Install fastboot using your package manager or the latest [Android SDK Platform Tools for Linux](#).
- You may optionally install Ansible using instructions on its [Installation page](#).

Get prebuilt images

Fetch the following files from the latest build for 96Boards HiKey:

- [bootloader/hisi-idt.py](#)
- [bootloader/l-loader.bin](#)
- [bootloader/fip.bin](#)
- [bootloader/nvme.img](#)

- the `boot-XXXX.uefi.img` and `rpb-ltd-gateway-image-hikey-YYYY.rootfs.img.gz` files from the [latest HiKey build artifacts page](#).

Uncompress the `rpb-ltd-gateway-image-hikey-YYYY.rootfs.img.gz` file, obtaining `rpb-ltd-gateway-image-hikey-YYYY.rootfs.img`.

(If you can't use a HiKey, you can try the same files from [builds for other boards](#), but these may not be functional.)

Set Your Board Up For Flashing

If you're using a 96Boards HiKey, do this by putting it into "Recovery Mode" as follows:

1. Remove power from the HiKey.
2. Remove both jumpers from the 2x3 header at the top left of the board (J601 on LeMaker HiKeys).

The board should now look like this:



3. Use the jumpers to connect pins 1 and 2, as well as pins 3 and 4, on the 2x3 header.

The board should now look like this:

4. Connect the HiKey to your PC via USB.
5. Power on the HiKey.

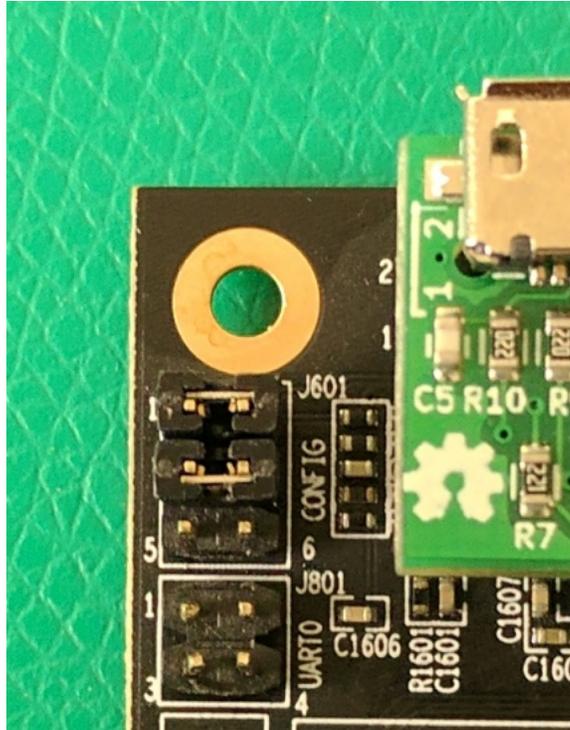
See [HiKey board recovery documentation](#) for more information on Recovery Mode.

If you're using a different 96Boards CE board, check its [96boards.org documentation](#) for instructions on how to reflash the bootloader, install fastboot support, and flash images via fastboot.

If you're not using a 96Boards board, refer to your vendor's documentation for similar instructions, or contact your vendor directly.

Flash Images To Your Board

These instructions assume you're using HiKey.



Windows

First, make sure that the directories containing the `python.exe` and `fastboot.exe` executables are on your `PATH` environment variable.

Now run the following, replacing `XXXX` and `YYYY` appropriately for the files you downloaded previously:

```
python.exe hisi-idt.py --img1=l-loader.bin
timeout 3 > NUL
fastboot.exe flash fastboot fip.bin
fastboot.exe flash nvme nvme.img
fastboot.exe flash boot boot-XXXX.uefi.img
fastboot.exe flash system rpb-ltd-gateway-image-hikey-YYYY.rootfs.img
```

Mac OS X and Linux

Note: On Linux, the `hisi-idt.py` script searches for a serial port device provided by your HiKey in `/dev/serial/by-id`. Some HiKey boards have non-Roman characters in their serial devices' names, which confuse the script and cause it to fail.

If this happens, passing the script `-d /dev/ttyUSBx`, where `/dev/ttyUSBx` is the absolute path pointed to by the symlink in `/dev/serial/by-id`, should resolve the issue.

Run the following, replacing `XXXX` and `YYYY` appropriately for the files you downloaded previously:

```
python2 hisi-idt.py --img1=l-loader.bin
sleep 2
fastboot flash fastboot fip.bin
```

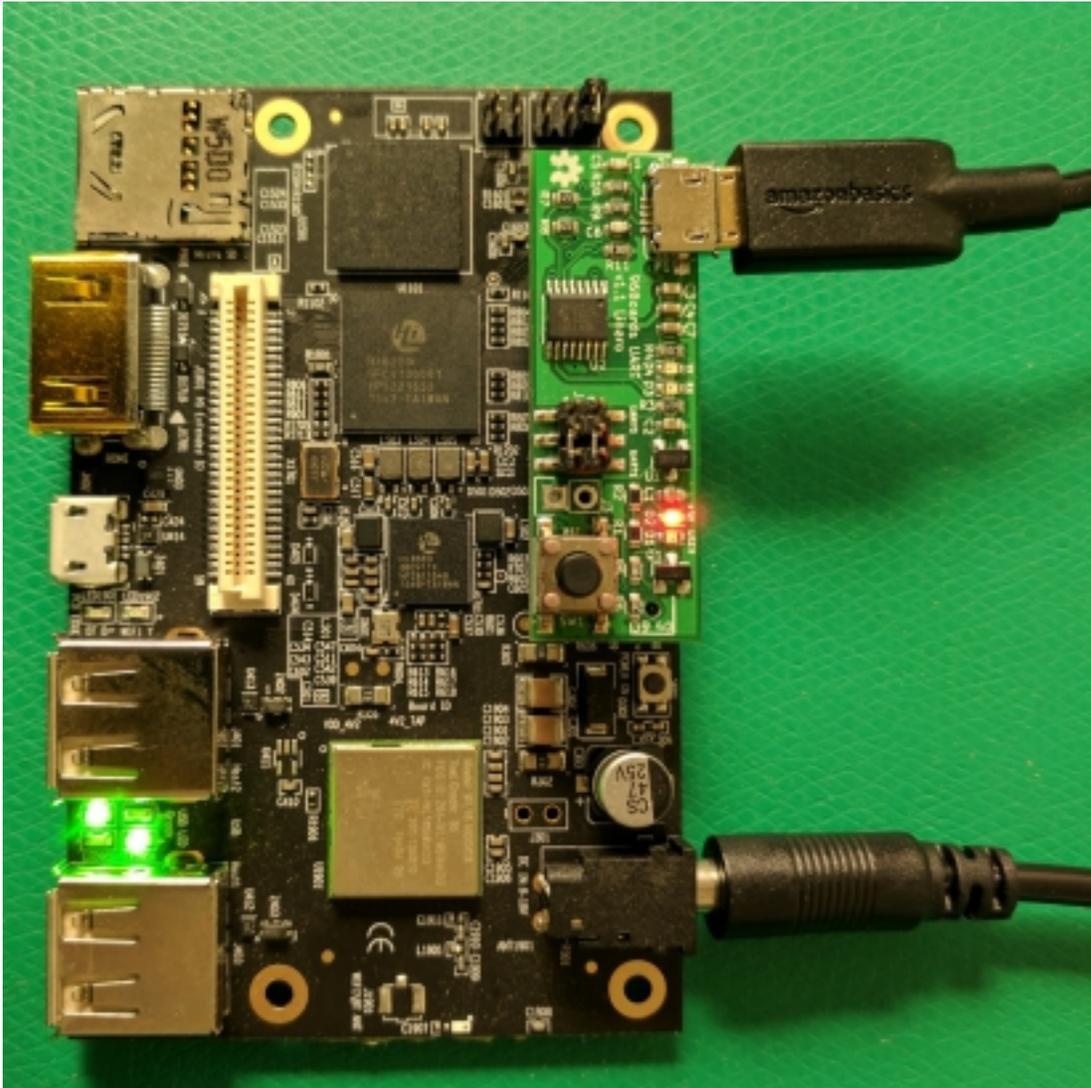
```
fastboot flash nvme nvme.img
fastboot flash boot boot-XXXX.uefi.img
fastboot flash system rpb-ltd-gateway-image-hikey-YYYY.rootfs.img
```

Boot the Board

Now that you've flashed the board, it's time to boot it. If you're using a HiKey, follow these instructions.

1. Remove the jumper connecting pins 3 and 4 from the 2x3 header you used when putting the board in Recovery Mode.
2. Install the 96Boards UART Serial Adapter board on the board. Make sure the USB connector faces outward from the board, or you will damage or break both HiKey and the UART Serial Adapter.
3. Connect the UART Serial Adapter to your host PC via USB.
4. Apply power to the HiKey via the barrel jack connector.

Your board should look like this:



At the serial console, the following login prompt should appear after the board finishes booting:

```
Reference-Platform-Build-X11 2.0+linaro hikey ttyAMA3
hikey login:
```

Enter `linaro` for the username, and `linaro` for the password. You will be dropped into a normal user shell, and should now change the password. The `linaro` user may use `sudo` to obtain root access on the device.

That's it! You've successfully installed the Linux microPlatform onto your device, and booted into its console.

Onwards!

At this point your device is ready to run Docker containers. If you would like to configure the device as a Basic IoT Gateway, follow the instructions at [Getting Started](#).

You're now ready to take your next steps. This will take the form of deploying containerized applications to your device.

One of the greatest advantages of using Cerberus is that it makes it easier to deploy and manage container-based applications. What's more, unlike other container-based embedded device platforms, Cerberus allows you to deploy **multiple applications to the same gateway, each running at the same time in its own container**. This is called **multitenancy**.

Check out the Linaro Technologies Division [Gateway Containers](#) repository for example Docker containers, along with instructions for how to get them running on your board. Start with the top-level [gateway-containers README.md](#), and move on to the subdirectories for containers which interest you.

If you installed Ansible earlier, you can also use Ansible playbooks to deploy the containers; these are available in the [gateway-ansible](#) repository. (While Ansible isn't supported on Windows, you can run [Ubuntu in a Docker container](#) and run Ansible from Ubuntu.)

Basic IoT Gateway (BIG)

The *LTD Basic IoT Gateway* is built from the Linux microPlatform and specific containers are added to enable gateway functionality, such as IPv6/IPv4 routing and MQTT message brokering.

LTD Basic IoT Gateway

The Basic IoT Gateway is simply the *LTD Linux microPlatform* configured with a few key containers that configure the device to behave as a basic IoT gateway device.

Getting Started

All you need to get started is a gateway device supported by the LTD Basic IoT Gateway, a computer, and an Internet connection.

Start with a base Linux microPlatform

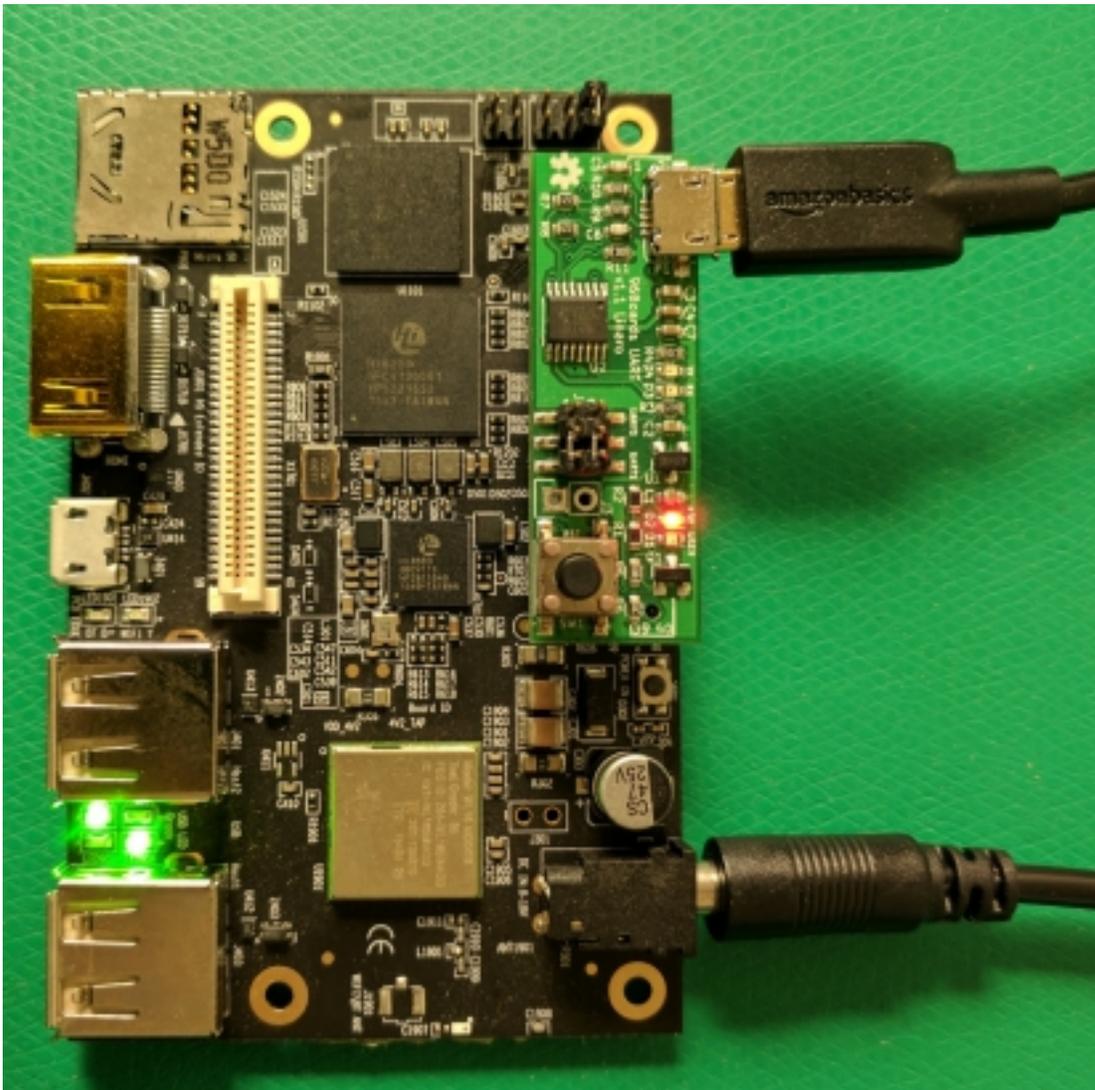
follow the instructions to setup your target hardware with the Linux microPlatform software following the instructions at *Getting Started*.

Boot the Board

1. Connect the UART Serial Adapter to your host PC via USB.
2. Apply power to the HiKey via the barrel jack connector.

Your board should look like this:

At the serial console, the following login prompt should appear after the board finishes booting:



```
Reference-Platform-Build-X11 2.0+linaro hikey ttyAMA3
hikey login:
```

Enter `linaro` for the username, and `linaro` for the password. You will be dropped into a normal user shell, and should now change the password. The `linaro` user may use `sudo` to obtain root access on the device.

Load Gateway Containers

Now to deploy some key containerized applications to your device.

One of the greatest advantages of using the Linux microPlatform is that it makes it easier to deploy and manage container-based applications. What's more, unlike other container-based embedded device platforms, the Linux microPlatform allows you to deploy **multiple applications to the same gateway, each running at the same time in its own container**. This is called **multitenancy**.

Check out the Linaro Technologies Division [Gateway Containers](#) repository for example Docker containers, along with instructions for how to get them running on your board. Start with the top-level [gateway-containers README.md](#), and move on to the subdirectories for containers which interest you.

If you installed Ansible earlier, you can also use Ansible playbooks to deploy the containers; these are available in the [gateway-ansible](#) repository. (While Ansible isn't supported on Windows, you can run [Ubuntu in a Docker container](#) and run Ansible from Ubuntu.)

Whitelist Setup for IoT Gateway

Instructions follow setting up a 6LoWPAN Bluetooth device whitelist.

Note: Prior to starting this walk through, please power off any IoT devices in your area.

Enable the whitelist feature

To enable the whitelist, simply enable the whitelist function by modifying the `bluetooth_6lowpand.conf`. You'll want to set `USE_WL` to 1 and add a `WL=MAC_ADDRESS` line for each device you wish to whitelist.

How to Find Devices for the Whitelist

Now that the whitelist is enabled, you can find the beaconing devices using the following command:

```
sudo hcitool lscan
```

While leaving this command running, power on the IoT device you wish to add to the whitelist. You should see an additional line appear as each device is powered on.

The following is an example of the output from this command:

```
LE Scan ...
D6:E7:D2:E8:6C:9F (unknown)
D6:E7:D2:E8:6C:9F Linaro IPSP node
```

Write down all of the "Linaro IPSP node" Bluetooth addresses, as you will need these for the next steps.

Disable the whitelist feature

To turn off the whitelist feature, set USE_WL to '0' in bluetooth_6lowpand.conf

End-to-end Open Source IoT Demonstration Systems

End-to-end Demonstration Systems shows you how to combine the Zephyr microPlatform and the Linux microPlatform along with server/cloud applications to create complete systems.

With these systems, you can publish sensor data from devices to the cloud and perform firmware over the air (FOTA) updates of the device firmware.

End-to-end Demonstration Systems

The IoT Foundry provides IoT platform demonstration systems with end-to-end connectivity. It is comprised of:

- Containers for cloud data and device management systems,
- a Basic IoT Gateway based on the *LTD Linux microPlatform*, and
- demonstration apps based on the *LTD Zephyr microPlatform*, which support cloud data and device management systems, and connect to the cloud through the gateway.

We continuously test and maintain these systems, and keep them synchronized with the upstream projects they depend on.

The two end-to-end demonstration systems we currently support are:

HTTP / MQTT via Hawkbit

hawkBit FOTA and MQTT Demonstration System

Todo

add binary install instructions for all boards

Todo

add autogenerated flash layout for all boards

Warning: Technology demonstration system only.

While the system described below works as documented, it is **unstable**, and its behavior may change incompatibly in the future. It is also **not supported**.

Overview

This page documents how to set up and use a demonstration system containing IoT devices and an IoT gateway, which can publish sensor data from devices to the cloud and perform firmware over the air (FOTA) updates of the device firmware.

A block diagram of this system is shown here. One or more IoT devices can connect to the network through the same gateway.

Using this demonstration system, you can:

- See live temperature readings from your devices appear in the web console provided by a cloud MQTT broker, CloudMQTT.
- Upload a cryptographically signed firmware image to a device management server, hawkBit.
- Use hawkBit to install a firmware image onto an IoT device via over the air update. The device will boot the update after checking its cryptographic signature.

Get the Hardware

To set up this system, you will need a Linux or macOS workstation computer, one or more IoT devices, and an IoT gateway.

We currently recommend:

- [96Boards Nitrogen](#) as an IoT device
- [96Boards HiKey](#) as an IoT gateway, with [UART Serial Mezzanine](#) for console access

Source for other boards is provided on a best-effort basis.

Prepare the System

This is broken down into the following steps.

1. *Set up hawkBit*
2. *Set up CloudMQTT*
3. *Install the Linux microPlatform*
4. *Set Up the IoT Gateway*
5. *Install the Zephyr microPlatform*

- 6. Set Up the IoT Device(s)

1. Set up hawkBit

Required Equipment: workstation which supports [Docker](#).

Run a demonstration-grade hawkBit server:

```
docker run -dit --name hawkbit -p 8080:8080 linarotechnologies/hawkbit-
↪update-server
```

Warning: This hawkBit container contains an official `hawkbit-update-server` artifact build from Maven; however, it is for **demonstration purposes only**, and should not be deployed in production as-is.

Among other potential issues, the server has an insecure default administrative username/password pair. For more information, see the official documentation on [building and running hawkBit](#) and [hawkBit security](#).

This container can take approximately 40 seconds for the application to start for the first time.

After running the hawkBit container, visit <http://localhost:8080/UI> to load the administrative interface, and log in with the default username and password (admin/admin).

Your browser window should look like this:

The screenshot shows the hawkBit administrative interface. On the left is a dark sidebar with the title 'Software Provisioning' and a user profile for 'DEFAULT admin...'. Below the profile are navigation links: Deployment, Rollout, Target Filters, Distributions, Upload, and System Config. At the bottom of the sidebar is a 'Documentation' link. The main content area is titled 'Deployment Management' and features several panels:

- Filters:** A 'Simple Filter' section showing 'NO TAG' and a 'Filter by Status' section with color-coded buttons (grey, green, yellow, red, blue). Below that is a 'Filter by Overdu' slider and a 'Custom Filter' button.
- Targets:** A table with columns 'Name' and 'Vers'. Below the table are 'Target' and 'Distribution set' sections, each with 'Details' and 'Description' tabs and navigation arrows.
- Distributions:** A table with columns 'Name' and 'Vers'.
- Artifact Details of:** A table with columns 'Active', 'Distribution', 'Date and tir', 'Statu', 'Force', and 'Actions'.

 At the bottom of the main content area, it says 'Total Targets : 0' and has two buttons: 'Drop here to delete' and 'No actions'.

Note: For convenience, you may want to adjust the “Polling Time” in the “System Config” area. This will

instruct your IoT devices to check for updates more frequently. The default is 5 minutes; the minimum value is 30 seconds.

Your hawkBit container is now ready for use.

2. Set up CloudMQTT

Required Equipment: workstation computer.

First, create a [CloudMQTT](#) account. The free CloudMQTT plan is enough to run this demo.

After logging in to your account, go to your [CloudMQTT Control Panel](#), and create a new instance. Then click on the “Details” button next to the new instance in your control panel. Record the following information about the instance:

- CLOUDMQTT_SERVER: the URL of the server
- CLOUDMQTT_PORT: the port to connect to on the server
- CLOUDMQTT_USER: the auto-generated username
- CLOUDMQTT_PASSWORD: the auto-generated password

3. Install the Linux microPlatform

Required Equipment: IoT gateway and workstation to flash the board.

Follow the Linux microPlatform [Getting Started](#) to set up a [96Boards HiKey](#) gateway for container-based application deployment.

If you don't have a HiKey, the Getting Started guide contains information for other boards, provided on a best-effort basis.

4. Set Up the IoT Gateway

Required Equipment: IoT gateway and workstation to run [Ansible](#).

You'll now use Ansible to set up your IoT gateway to act as a network proxy for your IoT device to publish sensor data to CloudMQTT, and fetch updates from hawkBit.

- Follow the Basic IoT Gateway [Getting Started](#) guide to log into the Basic IoT Gateway console and change the password for the `linaro` user. The default password is `linaro`.
- Connect your IoT gateway to the network.

You can connect a HiKey to a local WiFi network¹ from its serial console as follows:

```
sudo nmcli device wifi connect <NetworkSSID> password <NetworkPassword>
```

After connecting to the network, record the IP address of your gateway, `GATEWAY_IP_ADDRESS`, which you can obtain when using WiFi with:

```
ip addr show wlan0 | grep -o 'inet [.0-9]*'
```

(If you're using Ethernet, `ip addr show` will show all IP addresses on the system.)

¹ You can also use a USB Ethernet dongle.

- If you don't already have one, you now need to create an SSH key on your workstation. If you've never done this before, the [GitHub guide to SSH keys](#) has useful instructions.
- Copy your SSH key to the gateway in order to control it with Ansible. Do this with `ssh-copy-id`:

```
ssh-copy-id linaro@GATEWAY_IP_ADDRESS
```

Use the new password for the `linaro` account you set earlier.

- **Install Ansible**, which will let you install and control containers on your IoT gateway via SSH from your workstation.
- Clone the `gateway-ansible` repository, which contains an Ansible playbook to set up the gateway for this system:

```
git clone https://github.com/linaro-technologies/gateway-ansible
```

- From the `gateway-ansible` repository, deploy the gateway containers using the gateway's IP address and CloudMQTT information you recorded earlier:

```
ansible-playbook -e "mqttuser=CLOUDMQTT_USER mqttpass=CLOUDMQTT_PASSWORD \
↩ \
    mqttthost=CLOUDMQTT_SERVER mqttport=CLOUDMQTT_PORT \
    gitci=WORKSTATION_IP_ADDRESS tag=latest" \
-i GATEWAY_IP_ADDRESS, -u linaro iot-gateway.yml \
--tags cloud
```

`WORKSTATION_IP_ADDRESS` in the above command line is the IP address of the system which is running the hawkBit server you set up earlier. **The comma after `GATEWAY_IP_ADDRESS` is mandatory.**

5. Install the Zephyr microPlatform

Required Equipment: workstation to install the Zephyr microPlatform development environment, and IoT device to test installation.

Follow the installation steps in the Zephyr microPlatform *Getting Started* guide.

6. Set Up the IoT Device(s)

Required Equipment: IoT device and workstation to flash the device.

If you're using [96Boards Nitrogen](#), build and flash the demonstration application:

```
./genesis build -b 96b_nitrogen zephyr-fota-samples/dm-hawkbit-mqtt
./genesis flash -b 96b_nitrogen zephyr-fota-samples/dm-hawkbit-mqtt
```

Flashing this board requires pyOCD. To install:

```
pip install --user pyOCD
```

If you don't have `pip` installed, see the [pip Installation](#) documentation. On Linux platforms, you also need to install the following `udev` rules as root, then unplug and plug back in any boards you may have connected:

```
echo 'ATTR{idProduct}=="0204", ATTR{idVendor}=="0d28", MODE="0666", GROUP=
↪"plugdev"' > /etc/udev/rules.d/50-cmsis-dap.rules
```

If you don't have a Nitrogen, information for other boards is provided on a best-effort basis in *Additional IoT Devices*.

Use the System

Now that your system is fully set up, it's time to check that sensor data are being sent to the cloud, and do a FOTA update.

Cloud Sensor Updates

From your [CloudMQTT Control Panel](#), load your instance's Details page and click the "WebSocket UI" button to get a live view of data being sent to the server. You should see new data appear every few seconds; it will look like this:

Received messages	
Topic	Message
iot-2/type/96b_nitrogen/id/96b_nitrogen-4c1906d0/evt/status/fmt/json	{"d":{"mcutemp":26}}
iot-2/type/96b_nitrogen/id/96b_nitrogen-4c1906d0/evt/status/fmt/json	{"d":{"mcutemp":26}}
iot-2/type/96b_nitrogen/id/96b_nitrogen-4c1906d0/evt/status/fmt/json	{"d":{"mcutemp":27}}

Fig. 4.1: MQTT messages from 96Boards Nitrogen appearing in CloudMQTT WebSocket UI.

You can now connect other subscribers to this CloudMQTT instance, which can act on the data.

FOTA Updates

Now let's perform a FOTA update. In the hawkBit server UI, you should see the 96Boards device show up in the "Targets" pane. It will look like this:

It's time to upload a firmware binary to the server, and update it using this UI. To make uploading the binaries to the demonstration hawkBit server easier, download this Python script to your Zephyr microPlatform installation directory:

<https://raw.githubusercontent.com/linaro-technologies/hawkbit/master/hawkbit.py>

Todo

hawkbit.py should be a versioned part of the release

From the Zephyr microPlatform installation directory:

Targets		
Name		
96b_nitrogen-4c1906d0		

Fig. 4.2: 96Boards Nitrogen registered with hawkBit.

```
python /path/to/hawkbit.py \
    -ds 'http://localhost:8080/rest/v1/distributionsets' \
    -sm 'http://localhost:8080/rest/v1/softwaremodules' \
    -d 'Nitrogen End-to-end IoT system' \
    -f outdir/zephyr-fota-samples/dm-hawkbit-mqtt/96b_nitrogen/
↪ app/dm-hawkbit-mqtt-96b_nitrogen-signed.bin \
    -sv "1.0" -p "Linaro" -n "Nitrogen E2E preview" -t os
```

Above, 1.0 is an arbitrary version number.

You will see an update in the hawkBit UI for the new image:

Distributions		
Name	Versi	
Nitrogen E2E preview	1.0	...

Fig. 4.3: Distribution Set representing a signed firmware binary.

You'll now update the device. Before doing so, you can connect to its serial console via USB at 115200 baud to see log messages during the upgrade (which should be at `/dev/ttyACM0` or so on Linux systems).

- Click on the distribution you uploaded, and drag it over the line in “Targets” for your IoT Device.
- You’ll next need to confirm the action. Click a button towards the bottom of your screen labeled “You Have Actions”. This should now have a “1” at its top right, since you’ve assigned the distribution to your Nitrogen:

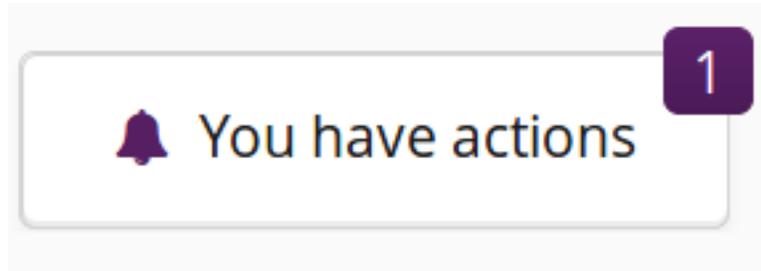


Fig. 4.4: Click this button.

- A screen will appear. Select “Save Assign” on this screen:

Your IoT devices will poll the hawkBit server periodically and will fetch the update the next time they poll.

Note: By default, devices wait five minutes between polls. If you don’t want to wait and are using 96Boards Nitrogen, you can press the “RST” button on the board to reset it; it will check for updates shortly after booting.

While hawkBit is waiting for the device to download and install the update, a yellow circle will appear next to it in the targets list:

When the device informs hawkBit that the download has been successfully installed, this will turn into a green check box:

If you’re connected to the device’s serial console, look for output like this while the update is being downloaded and installed:

```
[0031200] [fota/hawkbit] [INF] hawkbit_report_update_status: Reporting_
↪action ID feedback: success
[0031210] [fota/hawkbit] [DBG] hawkbit_query:

POST /DEFAULT/controller/v1/96b_nitrogen-4c1906d0/deploymentBase/1/feedback_
↪HTTP/1.1
Host: gitci.com:8080
Content-Type: application/json
Content-Length: 78
Connection: close

{"id": "1", "status": {"result": {"finished": "success"}, "execution": "proceeding"}
↪}
[0031570] [fota/tcp] [DBG] tcp_received_cb: FIN received, closing network_
↪context
[0031580] [fota/hawkbit] [DBG] hawkbit_query: Hawkbit query completed
[0031690] [fota/hawkbit] [INF] hawkbit_install_update: Starting the download_
↪and flash process
[0032990] [fota/hawkbit] [DBG] hawkbit_download_cb: 1%
[0033740] [fota/hawkbit] [DBG] hawkbit_download_cb: 2%
[0034440] [fota/hawkbit] [DBG] hawkbit_download_cb: 3%
```

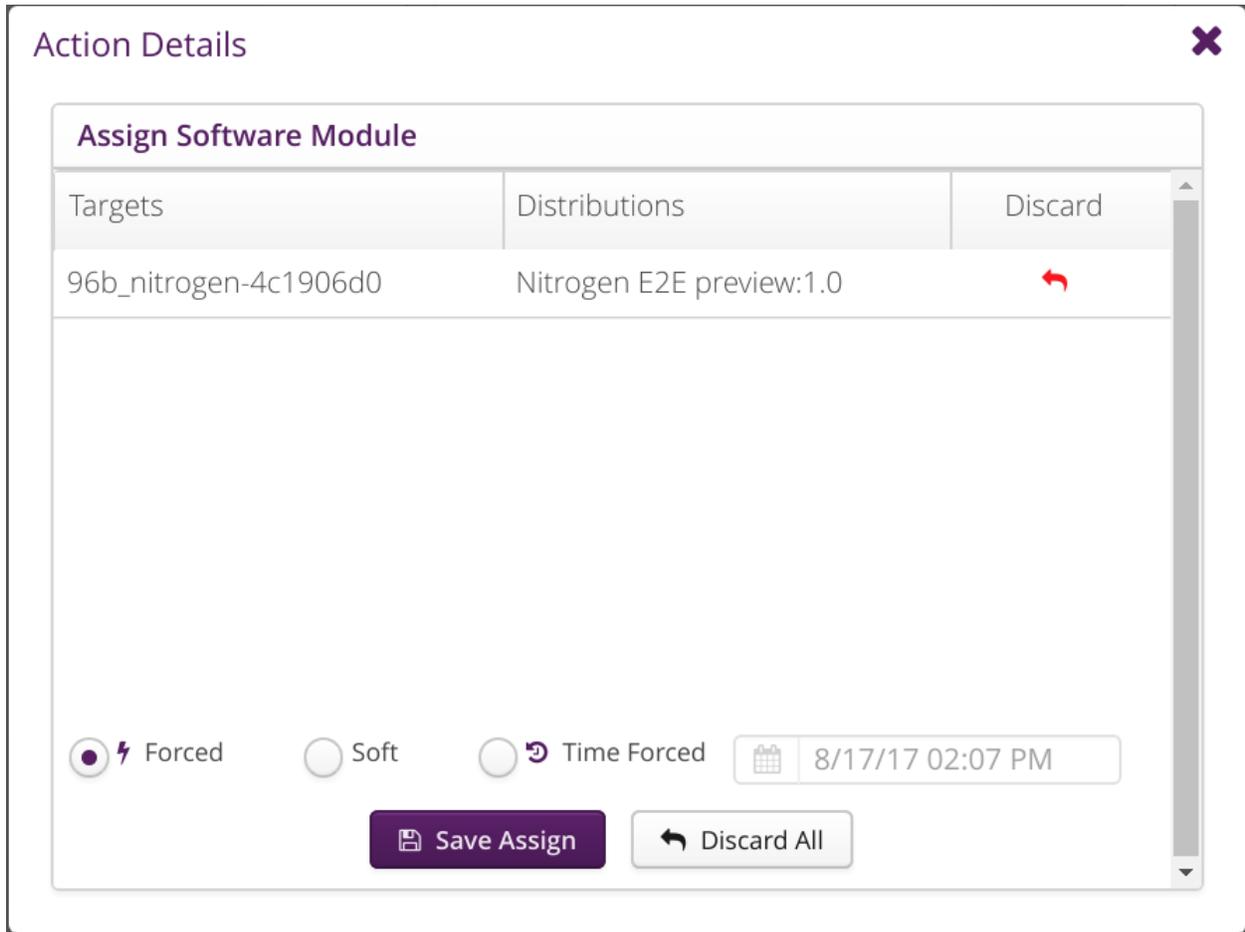


Fig. 4.5: Choose "Save Assign".



Fig. 4.6: Waiting for 96Boards Nitrogen to update.



Fig. 4.7: 96Boards Nitrogen successfully updated.

```

[0035290] [fota/hawkbit] [DBG] hawkbit_download_cb: 4%
[...]
[0627620] [fota/hawkbit] [DBG] hawkbit_download_cb: 98%
[0628470] [fota/hawkbit] [DBG] hawkbit_download_cb: 99%
[0629060] [fota/hawkbit] [INF] hawkbit_install_update: Download: downloaded_
↳bytes 212992
[0629070] [fota/hawkbit] [INF] hawkbit_ddi_poll: Triggering OTA update.
[0629180] [fota/hawkbit] [INF] hawkbit_ddi_poll: Image id 4 flashed_
↳successfully, rebooting now
[MCUBOOT] [INF] main: Starting bootloader
[MCUBOOT] [INF] boot_status_source: Image 0: magic=good, copy_done=0xff,_
↳image_ok=0xff
[MCUBOOT] [INF] boot_status_source: Scratch: magic=unset, copy_done=0x23,_
↳image_ok=0xff
[MCUBOOT] [INF] boot_status_source: Boot source: slot 0
[MCUBOOT] [INF] boot_swap_type: Swap type: none
[MCUBOOT] [INF] main: Bootloader chainload address offset: 0x8000
[MCUBOOT] [WRN] zephyr_flash_area_warn_on_open: area 1 has 1 users
[MCUBOOT] [INF] main: Jumping to the first image slot
***** BOOTING ZEPHYR OS v1.8.99 - BUILD: Aug  3 2017 13:28:24 *****
[0000000] [fota/main] [INF] main: Linaro FOTA example application

[Startup output omitted]

{"id": "1", "status": {"result": {"finished": "success"}, "execution": "closed"}}

```

Congratulations! You've just done your first FOTA update using this system.

Known Issues

Issues and observations are logged within Linaro's [Bugzilla issue tracker](#).

Appendix to hawkBit FOTA and MQTT Demonstration System

Warning: Content in this section is provided on a best-effort basis.

This document contains additional information related to the *hawkBit FOTA and MQTT Demonstration System*.

Additional IoT Devices

96Boards Carbon

In order to use the 96Boards Carbon with this demonstration system, you'll first need to set up a secondary chip on the board that provides Bluetooth functionality.

There are two chips on the Carbon that need firmware, an STM32 and an nRF51. The STM32 runs the bootloader and main application. The nRF51 is the secondary chip which needs to be flashed first to provide Bluetooth to the STM32, which doesn't support it on its own.

Build the application for the secondary chip from the Zephyr microPlatform installation directory:

```
./genesis build -b 96b_carbon_nrf51 -c prj_96b_carbon_nrf51.conf \
--skip-signature zephyr/samples/bluetooth/hci_spi/
```

This creates a firmware binary for the nRF51 at the following location:

```
outdir/zephyr/samples/bluetooth/hci_spi/96b_carbon_nrf51/app/zephyr.elf
```

Flashing the nRF51 device requires an external SWD flashing tool, such as the [Black Magic Debug Probe](#) or [Segger JLink](#).

Before flashing this file, first **put your Carbon in DFU mode** by unplugging it, then plugging it back in with the BOOT0 button pressed. This ensures the STM32 firmware does not interfere with the nRF51. Then follow the [Zephyr 96b_carbon_nrf51 flashing instructions](#) to flash the binary to the nRF51.

Now run this from the Zephyr microPlatform installation directory to build the main application:

```
./genesis build -b 96b_carbon zephyr-fota-samples/dm-hawkbitt-mqtt
```

You'll need to install dfu-util to flash this binary. A recent version of dfu-util is required; depending on your host operating system, you may need to build this from source.

- dfu-util URL: <http://dfu-util.sourceforge.net/>
- Git Repository: [git://git.code.sf.net/p/dfu-util/dfu-util](https://git.code.sf.net/p/dfu-util/dfu-util)

To flash the STM32, first put your Carbon into DFU mode again. Then, from the Zephyr microPlatform installation directory, run:

```
./genesis flash -b 96b_carbon zephyr-fota-samples/dm-hawkbitt-mqtt
```

FRDM-K64F

Building for [FRDM-K64F](#) requires some configuration information which depends on your local network:

- An IP address to use for the IoT gateway
- Whether the board should use DHCP, or a static IP address

This information must be written to the file `zephyr-fota-samples/dm-hawkbitt-mqtt/boards/frdm_k64f-local.conf` in the Zephyr microPlatform installation directory.

To use DHCP, with gateway IP address A.B.C.D, create the file with the following contents.

```
CONFIG_NET_DHCPV4=y
CONFIG_NET_APP_PEER_IPV4_ADDR="A.B.C.D"
```

To use a static IP address X.Y.Z.W for the FRDM-K64F instead, use this.

```
CONFIG_NET_APP_MY_IPV4_ADDR="X.Y.Z.W"
CONFIG_NET_APP_PEER_IPV4_ADDR="A.B.C.D"
```

Now you can build the binaries. From the Zephyr microPlatform installation directory:

```
./genesis build -b frdm_k64f zephyr-fota-samples/dm-hawkbitt-mqtt
```

Flashing this board requires pyOCD. To install:

```
pip install --user pyOCD
```

If you don't have pip installed, see the [pip Installation](#) documentation. On Linux platforms, you also need to install the following udev rules as root, then unplug and plug back in any boards you may have connected:

```
echo 'ATTR{idProduct}=="0204", ATTR{idVendor}=="0d28", MODE="0666", GROUP=
↪"plugdev"' > /etc/udev/rules.d/50-cmsis-dap.rules
```

To flash the binaries, plug the K64F into your system via the USB connector labeled "SDA USB". Then, from the Zephyr microPlatform installation directory:

```
./genesis flash -b frdm_k64f zephyr-fota-samples/dm-hawkbit-mqtt
```

Additional hawkBit Information

This section contains additional information for more complex use cases or further development.

- Upstream Github: <https://github.com/eclipse/hawkbit>
- Data model: <https://github.com/eclipse/hawkbit/wiki/Data-model>
- Docker container: <https://github.com/linaro-technologies/extra-containers/tree/master/hawkbit-update-server>
- Docker Hub: <https://hub.docker.com/r/linarotechnologies/hawkbit-update-server>

Hawkbit Restrictions

Target can only install a distribution set.

Hawkbit's default configuration for OS/Firmware forces only one Software Module per Distribution Set. The software module can contain several artifacts, as long they don't contain the same file name.

Rest API

The authentication header is a simply HTTP BA implementation. To use the protected APIs, first generate your authentication credentials:

```
echo -n "admin:admin" | base64
```

Then add your credentials as part of HTTP header:

```
curl ... -H 'Authorization: Basic YWRtaW46YWRtaW4='
```

Rest Commands:

Targets:

- Reference: <http://sp.apps.bosch-iot-cloud.com/documentation/rest-api/targets-api-guide.html>
- List all targets:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets' -s -H
↪'Authorization: Basic YWRtaW46YWRtaW4=' | jq .
```

- Create a new target (security token generated by the server):

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets' -i -H
↪'Authorization: Basic YWRtaW46YWRtaW4=' -H 'Content-Type: application/
↪json' -X POST -d '[ {
"controllerId" : "nitrogen123",
"name" : "nitrogen123",
"description" : "Nitrogen Description"
} ]'
```

- Create a new target specifying a custom security token:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets' -i -H
↪'Authorization: Basic YWRtaW46YWRtaW4=' -H 'Content-Type: application/
↪json' -X POST -d '[ {
"controllerId" : "nitrogen123",
"name" : "nitrogen123",
"description" : "Nitrogen Description",
"securityToken" : "2345678DGGDGFTDzztgfg"
} ]'
```

- Delete a target:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets/
↪nitrogen123' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -X DELETE
```

- Retrieve details from a single target::

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets/
↪nitrogen123' -s -H 'Authorization: Basic YWRtaW46YWRtaW4=' | jq .
```

- Assign a distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets/
↪nitrogen123/assignedDS' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -
↪H 'Content-Type: application/json' -X POST -d '{
"forcetime" : 1472465267347,
"id" : 1,
"type" : "timeforced"
}'
```

- Retrieve assigned distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets/
↪nitrogen123/assignedDS' -s -H 'Authorization: Basic YWRtaW46YWRtaW4='
↪| jq .
```

Distribution Sets:

- Reference: <http://sp.apps.bosch-iot-cloud.com/documentation/rest-api/distributionsets-api-guide.html>
- List all distribution sets:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets' -s -H 'Authorization: Basic YWRtaW46YWRtaW4=' | jq .
```

- Create a new distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -H
↪'Content-Type: application/json' -X POST -d '[ {
"name" : "Zephyr 17.01",
"description" : "Zephyr 17.01 Build",
"version" : "1.5",
"requiredMigrationStep" : false,
"type" : "os"
} ]'
```

- Delete a distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets/1' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -X_
↪DELETE
```

- Retrieve assigned software modules:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets/1/assignedSM' -s -H 'Authorization: Basic_
↪YWRtaW46YWRtaW4=' | jq .
```

- Assign a software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets/1/assignedSM' -i -H 'Authorization: Basic_
↪YWRtaW46YWRtaW4=' -H 'Content-Type: application/json' -X POST -d '[ {
↪"id" : 1 } ]'
```

- Retrieve assigned targets to a distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets/1/assignedTargets' -s -H 'Authorization: Basic_
↪YWRtaW46YWRtaW4=' | jq .
```

- Assign targets to a distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets/1/assignedTargets' -i -H 'Authorization: Basic_
↪YWRtaW46YWRtaW4=' -H 'Content-Type: application/json' -X POST -d '[
{ "id" : 1 },
{ "id" : 2 },
{ "id" : 3 }
]'
```

Software Modules:

- Reference: <http://sp.apps.bosch-iot-cloud.com/documentation/rest-api/softwaremodules-api-guide.html>
- List all software modules:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/softwaremodules
↪' -s -H 'Authorization: Basic YWRtaW46YWRtaW4=' | jq .
```

- Create a new software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/softwaremodules
↪' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -H 'Content-Type:
↪application/json' -X POST -d '[ {
" name " : "Zephyr Firmware",
" vendor " : "Linaro",
" description " : "Firmware Images for Zephyr",
" type " : "os",
" version " : "1.5"
} ]'
```

- Delete a software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪softwaremodules/1' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -X
↪DELETE
```

- List artifacts from a software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪softwaremodules/2/artifacts' -s -H 'Authorization: Basic
↪YWRtaW46YWRtaW4=' | jq .
```

- Upload a new artifact to a software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪softwaremodules/2/artifacts' -i -H 'Authorization: Basic
↪YWRtaW46YWRtaW4=' -H 'Content-Type: multipart/form-data' -X POST -F
↪'file=@/tmp/zephyr.hex'
```

- Download an artifact:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪softwaremodules/2/artifacts/1/download' -s -H 'Authorization: Basic
↪YWRtaW46YWRtaW4=' -H 'Accept: application/octet-stream'
```

Rollout:

- Reference: <http://sp.apps.bosch-iot-cloud.com/documentation/rest-api/rollout-api-guide.html>

Bootstrap Example

- Create new target:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets' -i -H
↪'Authorization: Basic YWRtaW46YWRtaW4=' -H 'Content-Type: application/
↪json' -X POST -d '[ {
" controllerId " : "nitrogen123",
" name " : "nitrogen123",
" description " : "Nitrogen Description"
} ]'
```

- Create new distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/
↪distributionsets' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -H
↪'Content-Type: application/json' -X POST -d '[ {
" name " : "Zephyr 17.01",
" description " : "Zephyr 17.01 Build",
" version " : "1.5",
```

```
"requiredMigrationStep" : false,  
"type" : "os"  
} ]'
```

- Create new software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/softwaremodules  
↪' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -H 'Content-Type: ↪  
↪application/json' -X POST -d '{  
"name" : "Zephyr Firmware",  
"vendor" : "Linaro",  
"description" : "Firmware Images for Zephyr",  
"type" : "os",  
"version" : "1.5"  
} ]'
```

- Upload artifact to the software module:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/  
↪softwaremodules/2/artifacts' -i -H 'Authorization: Basic ↪  
↪YWRtaW46YWRtaW4=' -H 'Content-Type: multipart/form-data' -X POST -F  
↪'file=@/tmp/zephyr.hex'
```

- Assign software module to the distribution set:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/  
↪distributionsets/1/assignedSM' -i -H 'Authorization: Basic ↪  
↪YWRtaW46YWRtaW4=' -H 'Content-Type: application/json' -X POST -d '{  
↪"id" : 1 } ]'
```

- Assign distribution set to the desired target:

```
curl 'http://your-hawkbit-server.example.com:8080/rest/v1/targets/  
↪nitrogen123/assignedDS' -i -H 'Authorization: Basic YWRtaW46YWRtaW4=' -  
↪H 'Content-Type: application/json' -X POST -d '{  
"forcetime" : 1472465267347,  
"id" : 1,  
"type" : "timeforced"  
}'
```

Now create a rollout.

CoAP / LWM2M via Leshan

LWM2M FOTA and Data Demonstration System

Todo

add binary install instructions for all boards

Todo

add autogenerated flash layout for all boards

Warning: Technology demonstration system only.

While the system described below works as documented, it may be **unstable**, and its behavior may change in the future.

Overview

This page documents how to set up and use an end-to-end IoT demonstration system using OMA Lightweight M2M (LWM2M). The system contains Zephyr-based IoT devices, an IoT gateway, and a web application, Leshan, that is used as the LWM2M server. With Leshan you can issue commands, query data and perform firmware over the air (FOTA) updates on the IoT device(s).

A block diagram of this system is shown here, and though it is not explicitly shown, one or more IoT devices can connect to the network through the same gateway.

Using this demonstration system, you can:

- See live data readings from your devices using the Leshan Web application.
- Send commands to the device, such as turning on and off the USR LED.
- Use Leshan to transfer the firmware image onto an IoT device and then initiate a firmware update.

Get the Hardware

To set up this system, you will need a Linux or macOS workstation computer, one or more IoT devices, and an IoT gateway.

We currently recommend:

- [96Boards Nitrogen](#) as an IoT device
- [96Boards HiKey](#) as an IoT gateway, with [UART Serial Mezzanine](#) for console access

Source for other boards is provided on a best-effort basis.

Prepare the System

This is broken down into the following steps.

1. *Set up Leshan*
2. *Install the Linux microPlatform*
3. *Set up the IoT Gateway*
4. *Install the Zephyr microPlatform*
5. *Set up the IoT Device(s)*

1. Set up Leshan

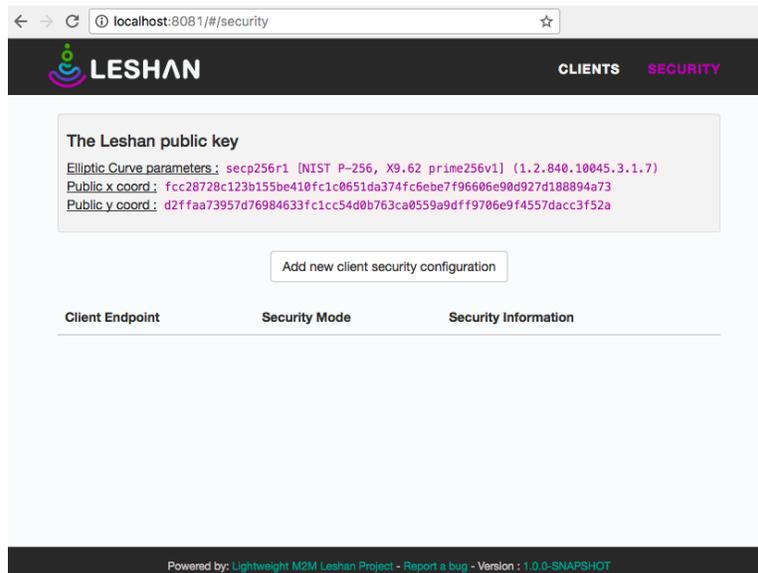
Required Equipment: workstation which supports [Docker](#).

Start the Linaro Technologies container with the following command:

```
docker run --restart=always -d -t -p 5683:5683/udp -p 5684:5684/udp \
  --read-only --tmpfs=/tmp -p 8081:8080 \
  --name leshan linarotechnologies/leshan:latest
```

After running the Leshan container, visit <http://localhost:8081/> to load the Leshan web interface.

You can also browse the Security tab:



Your Leshan container is now ready for use.

2. Install the Linux microPlatform

Required Equipment: IoT gateway and workstation to flash the board.

Follow the Linux microPlatform [Getting Started](#) guide to set up a [96Boards HiKey](#) gateway for container-based application deployment.

If you don't have a HiKey, the Getting Started Guide contains information for other boards, provided on a best-effort basis.

3. Set up the IoT Gateway

Required Equipment: IoT gateway device and workstation to run [Ansible](#).

You'll now use Ansible to set up your IoT gateway to act as a network proxy for your IoT device to communicate with the Leshan server you set up earlier.

- Follow the Basic IoT Gateway [Getting Started](#) guide to log into the Basic IoT Gateway console and change the password for the `linaro` user. The default password is `linaro`.

- Connect your IoT gateway to the network.

You can connect a HiKey to a local WiFi network¹ from its serial console as follows:

```
sudo nmcli device wifi connect <NetworkSSID> password <NetworkPassword>
```

After connecting to the network, record the IP address of your gateway, GATEWAY_IP_ADDRESS, which you can obtain when using WiFi with:

```
ip addr show wlan0 | grep -o 'inet [.0-9]*'
```

(If you're using Ethernet, `ip addr show` will show all IP addresses on the system.)

- If you don't already have one, you now need to create an SSH key on your workstation. If you've never done this before, the [GitHub guide to SSH keys](#) has useful instructions.
- Copy your SSH key to the gateway in order to control it with Ansible. Do this with `ssh-copy-id`:

```
ssh-copy-id linaro@GATEWAY_IP_ADDRESS
```

Use the new password for the `linaro` account you set earlier.

- Install **Ansible**, which will let you install and control containers on your IoT gateway via SSH from your workstation.
- Clone the `gateway-ansible` repository, which contains an Ansible playbook to set up the gateway for this system:

```
git clone https://github.com/linaro-technologies/gateway-ansible
```

- From the `gateway-ansible` repository, deploy the gateway containers to your IoT gateway:

```
ansible-playbook -e "gitci=WORKSTATION_IP_ADDRESS tag=latest" \
    -i GATEWAY_IP_ADDRESS, -u linaro iot-gateway.yml \
    --tags gateway
```

WORKSTATION_IP_ADDRESS in the above command line is the IP address of the system which is running the Leshan server you set up earlier. **The comma after GATEWAY_IP_ADDRESS is mandatory.**

4. Install the Zephyr microPlatform

Required Equipment: workstation to install the Zephyr microPlatform development environment, and IoT device to test installation.

Follow the installation steps in the Zephyr microPlatform *Getting Started* guide.

5. Set up the IoT Device(s)

Required Equipment: IoT device and workstation to flash the device.

If you're using [96Boards Nitrogen](#), build and flash the demonstration application:

```
./genesis build -b 96b_nitrogen zephyr-fota-samples/dm-lwm2m
./genesis flash -b 96b_nitrogen zephyr-fota-samples/dm-lwm2m
```

¹ You can also use a USB Ethernet dongle.

Flashing this board requires pyOCD. To install:

```
pip install --user pyOCD
```

If you don't have pip installed, see the [pip Installation](#) documentation. On Linux platforms, you also need to install the following udev rules as root, then unplug and plug back in any boards you may have connected:

```
echo 'ATTR{idProduct}=="0204", ATTR{idVendor}=="0d28", MODE="0666", GROUP=
↪"plugdev"' > /etc/udev/rules.d/50-cmsis-dap.rules
```

If you don't have a Nitrogen, information for other boards is provided on a best-effort basis in *Additional IoT Devices*.

Use the System

Now that your system is fully set up, it's time to check that sensor data are being sent to the cloud, and do a FOTA update.

Note: The Leshan user web interface is a simple web application, which does not provide a complete end-to-end device management system. Leshan's simplicity makes it a perfect demonstration and prototyping system for LWM2M devices.

Retrieve Data

When a device registers with the Leshan server, Leshan will automatically render known object types on the web interface. You can interact with the objects by scrolling and clicking the buttons for the objects.

- Read device information

To read the device information, simple scroll down to the corresponding device information object and select the 'READ' button. If Leshan is able to communicate with your device you will see all of the available device information.

The screenshot shows the Leshan web interface with a dark header containing the Leshan logo and navigation tabs for 'CLIENTS' and 'SECURITY'. The main content area is titled 'Device' and displays a list of objects for instance 0. The objects are organized into two columns. The left column lists various system and hardware objects, and the right column shows their corresponding values. Each object has a set of control buttons: 'Observe', 'Read', 'Write', and 'Delete'. The 'Read' button for the 'Manufacturer' object is highlighted in green. The values shown include manufacturer 'Zephyr', model number '96b_nitrogen', serial number 'a0c29b1b', firmware version '1.0', battery level '0', memory free '-1', and device type 'Zephyr OMA-LWM2M Client'.

Object	Value
Instance 0	/0/0
Manufacturer	Zephyr
Model Number	96b_nitrogen
Serial Number	a0c29b1b
Firmware Version	1.0
Reboot	
Factory Reset	
Available Power Sources	
Power Source Voltage	
Power Source Current	
Battery Level	0
Memory Free	-1
Error Code	
Reset Error Code	
Current Time	1970-01-01T00:02:50Z
UTC Offset	
Timezone	
Supported Binding and Modes	
Device Type	U
Hardware Version	Zephyr OMA-LWM2M Client
Software Version	nRF52832_QFAA
Battery Status	1.8.99
Memory Total	0
ExtDevInfo	208

- Read current state of temperature and light objects

To read the current status of the Temperature and Light objects, scroll down to the light and temperature objects and select the READ button. You will see the state of these objects on the device similar to this figure.

The screenshot shows the LESHAN interface with two main sections: Temperature and Light Control. The Temperature section (ID: /3303) lists several properties for Instance 0, including Min Measured Value, Max Measured Value, Min Range Value, Max Range Value, Reset Min and Max Measured Values, Sensor Value, and Sensor Units. The Light Control section (ID: /3311) lists properties for Instance 0, including Sensor Units, Colour, Cumulative active power, Power Factor, On/Off, Dimmer, and On Time. Both sections have a 'Create New Instance' button and a table of actions (Observe, Read, Write, Delete) for each property. The bottom status bar indicates 'Powered by: Lightweight M2M Leshan Project - Report a bug - Version: 1.0.0-SNAPSHOT' and 'CoAP messages'.

- Change state of the light object

To change the state of an object, simply use the leshan interface and select the 'write' button to bring up the appropriate interface for changing data.

The screenshot shows the LESHAN interface with the 'Update Instance 0 of Light Control' dialog box open. The dialog box has four input fields: Colour, On/Off (with 'true' entered), Dimmer, and On Time. There are 'Cancel' and 'Update' buttons at the bottom right. The background shows the same Light Control object list as the previous screenshot. The bottom status bar is the same as the previous screenshot.

FOTA Updates

Updating the firmware is provided by the LWM2M firmware update object.

- Initiate the firmware transfer

To start the firmware update, we will first 'write' the location of the file in the "Package URI" field. Once you send this message, the file will begin transferring to the target.

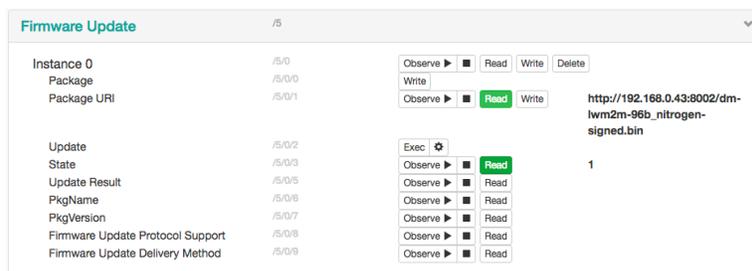
Note:

- the length of the Package URI field must be < 255 characters

- The URI must be hosted where it is routable from your device. The URI can be either coap:// or http://



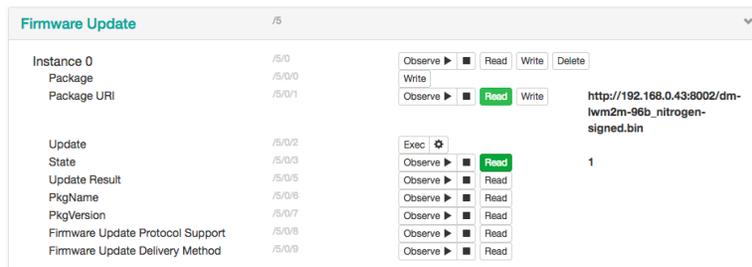
- Monitor the target for a completed transfer



- Execute the update and monitor the firmware update state

The state values and their meanings are:

- State == 0: Idle
- State == 1: Downloading
- State == 2: Downloaded
- State == 3: Updating



- After the device downloads the update file (State == 2), initiate the update by clicking on the 'exec' button.

When the update execution is complete, the device will restart.

- Congratulations! You've just done your first FOTA update using this system.

Known Issues

Issues and observations are logged within Linaro's [Bugzilla issue tracker](#).

Appendix to LWM2M FOTA and Data Demonstration System

Warning: Content in this section is provided on a best-effort basis.

This document contains additional information related to the *LWM2M FOTA and Data Demonstration System*.

Additional IoT Devices

96Boards Carbon

In order to use the 96Boards Carbon with this demonstration system, you'll first need to set up a secondary chip on the board that provides Bluetooth functionality.

There are two chips on the Carbon that need firmware, an STM32 and an nRF51. The STM32 runs the bootloader and main application. The nRF51 is the secondary chip which needs to be flashed first to provide Bluetooth to the STM32, which doesn't support it on its own.

Build the application for the secondary chip from the Zephyr microPlatform installation directory:

```
./genesis build -b 96b_carbon_nrf51 -c prj_96b_carbon_nrf51.conf \
  --skip-signature zephyr/samples/bluetooth/hci_spi/
```

This creates a firmware binary for the nRF51 at the following location:

```
outdir/zephyr/samples/bluetooth/hci_spi/96b_carbon_nrf51/app/zephyr.elf
```

Flashing the nRF51 device requires an external SWD flashing tool, such as the [Black Magic Debug Probe](#) or [Segger JLink](#).

Before flashing this file, first **put your Carbon in DFU mode** by unplugging it, then plugging it back in with the BOOT0 button pressed. This ensures the STM32 firmware does not interfere with the nRF51. Then follow the Zephyr [96b_carbon_nrf51 flashing instructions](#) to flash the binary to the nRF51.

Now run this from the Zephyr microPlatform installation directory to build the main application:

```
./genesis build -b 96b_carbon zephyr-fota-samples/dm-lwm2m
```

You'll need to install dfu-util to flash this binary. A recent version of dfu-util is required; depending on your host operating system, you may need to build this from source.

- dfu-util URL: <http://dfu-util.sourceforge.net/>
- Git Repository: [git://git.code.sf.net/p/dfu-util/dfu-util](https://git.code.sf.net/p/dfu-util/dfu-util)

To flash the STM32, first put your Carbon into DFU mode again. Then, from the Zephyr microPlatform installation directory, run:

```
./genesis flash -b 96b_carbon zephyr-fota-samples/dm-lwm2m
```

FRDM-K64F

Building for **FRDM-K64F** requires some configuration information which depends on your local network:

- An IP address to use for the IoT gateway
- Whether the board should use DHCP, or a static IP address

This information must be written to the file `zephyr-fota-samples/dm-lwm2m/boards/frdm_k64f-local.conf` in the Zephyr microPlatform installation directory.

To use DHCP, with gateway IP address A.B.C.D, create the file with the following contents.

```
CONFIG_NET_DHCPV4=y
CONFIG_NET_APP_PEER_IPV4_ADDR="A.B.C.D"
```

To use a static IP address X.Y.Z.W for the FRDM-K64F instead, use this.

```
CONFIG_NET_APP_MY_IPV4_ADDR="X.Y.Z.W"
CONFIG_NET_APP_PEER_IPV4_ADDR="A.B.C.D"
```

In addition, `zephyr-fota-samples/dm-lwm2m/boards/frdm_k64f-local.conf` must contain a line which specifies the IP address of the COAP proxy. In this case, that's just the IP address of your gateway device. To use IP address L.M.N.O, add a line like this after the other networking configuration:

```
CONFIG_LWM2M_FIRMWARE_UPDATE_PULL_COAP_PROXY_ADDR="L.M.N.O"
```

Now you can build the binaries. From the Zephyr microPlatform installation directory:

```
./genesis build -b frdm_k64f zephyr-fota-samples/dm-lwm2m
```

Flashing this board requires pyOCD. To install:

```
pip install --user pyOCD
```

If you don't have pip installed, see the [pip Installation](#) documentation. On Linux platforms, you also need to install the following udev rules as root, then unplug and plug back in any boards you may have connected:

```
echo 'ATTR{idProduct}=="0204", ATTR{idVendor}=="0d28", MODE="0666", GROUP=
↪"plugdev"' > /etc/udev/rules.d/50-cmsis-dap.rules
```

To flash the binaries, plug the K64F into your system via the USB connector labeled "SDA USB". Then, from the Zephyr microPlatform installation directory:

```
./genesis flash -b frdm_k64f zephyr-fota-samples/dm-lwm2m
```

Additional Leshan Information

This section contains additional information for more complex use cases or further development.

- Upstream Github:
- Data model:
- Docker container: <https://github.com/linaro-technologies>

-
- Docker Hub: <https://hub.docker.com/r/linarotechnologies/>
-

Todo

Add “advanced and special topics” Zephyr microPlatform docs:

- Support a new device
 - Build developer docs
 - Add features to the tooling (“internals” docs, generally)
 - HLL support (uPython, JerryScript)
 - Cloud service integration
-

Documentation-wide TODO List**Todo**

Add “advanced and special topics” Zephyr microPlatform docs:

- Support a new device
 - Build developer docs
 - Add features to the tooling (“internals” docs, generally)
 - HLL support (uPython, JerryScript)
 - Cloud service integration
-

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/index.rst`, line 69.)

Todo

add binary install instructions for all boards

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/iotfoundry/hawkbit-howto.rst`, line 9.)

Todo

add autogenerated flash layout for all boards

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/iotfoundry/hawkbit-howto.rst`, line 10.)

Todo

hawkbit.py should be a versioned part of the release

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/iotfoundry/hawkbit-howto.rst`, line 283.)

Todo

add binary install instructions for all boards

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/iotfoundry/lwm2m-howto.rst, line 9.)

Todo

add autogenerated flash layout for all boards

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/iotfoundry/lwm2m-howto.rst, line 10.)

Todo

Add a few good diagrams.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/branching.rst, line 6.)

Todo

Add link to a top-level “supported boards” page when that’s ready.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/getting-started.rst, line 12.)

Todo

Generate instructions for other manifest repository sources.

In these configurations, we need extra docs:

- Cache Git usernames and passwords you enter in memory for one hour; this allows `repo sync` to work unprompted in the next step. If you don’t want to do this, see <https://git-scm.com/docs/gitcredentials> for alternatives.

```
git config --global credential.helper 'cache --timeout=3600'
```

- If you don’t already have one, create a [GitHub](#) account (it’s free).
 - Make sure you can see the Zephyr microPlatform SDK manifest repository when you’re logged in to your account (**needs link**).
 - If you enabled [two-factor authentication](#) on your GitHub account, you also need a [personal access token](#). Give this token at least “repo” access, and make sure you keep a copy.
 - When prompted by `repo init`, enter your GitHub username and password (or access token, if you use two-factor authentication).
-

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/getting-started.rst, line 173.)

Todo

Add links to next steps documents when they're ready.

Example of tutorials and reference docs:

- Zephyr microPlatform overview (different projects with links to their reference docs, how they tie together, e.g. description of boot process with links to mcuboot documentation).
- Hardware peripheral tutorials (UART, SPI, etc.)
- Internet connectivity with an Basic IoT Gateway
- FOTA with hawkBit

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/getting-started.rst, line 316.)

Todo

Write this section.

- Minimum sane key management policies
- Building production-ready mcuboot and application images (blocker: <https://trello.com/c/mSZPuXxG>)
- Disabling JTAG/SWD or making physical access harder and other issues discussed in the threat model.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/prod-workflow.rst, line 8.)

Todo

Re-work after resolution of <https://trello.com/c/mSZPuXxG> and <https://projects.linaro.org/browse/LITE-147>

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/workflow.rst, line 47.)

Todo

fill this in when it's possible.

<https://trello.com/c/Yj5vW4zf> <https://projects.linaro.org/browse/LITE-91> <https://projects.linaro.org/browse/LITE-125>

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/workflow.rst, line 219.)

Todo

improve this.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/workflow.rst, line 228.)

Todo

user-friendly instructions, post-CMake transition.

(The original entry is located in `/home/docs/checkouts/readthedocs.org/user_builds/ltd-docs/checkouts/ltd-17.09/source/zephyr/workflow.rst`, line 238.)