Oryx Linux Documentation

Release 0.2.0

Togán Labs

Jun 27, 2017

CONTENTS

CHAPTER

ONE

ORYX LINUX

1.1 Introduction

1.1.1 Summary

This documentation covers the 0.2.0 version of Oryx Linux.

Oryx Linux is a Linux® distribution targeted at embedded applications and based on the work of The Yocto Project and OpenEmbedded.

1.1.2 Motivation

The Oryx Linux project is primarily motivated by a desire to incorporate a lightweight Linux container implementation into the OpenEmbedded build system whilst maintaining the benefits of both systems. The key word here is 'lightweight': we're avoiding fully-integrated systems such as Docker which are targeted at cloud computing deployments rather than embedded deployments. Instead we're using `runc`, the lightweight container runtime which sits at the heart of Docker, without any of the surrounding tools such as `containerd` and `docker` itself. This gives us the flexibility to address the needs of the embedded use-case.

One of the main aims of this project is to provide a developer workflow which is familiar to existing OpenEmbedded users. You should not be required to learn a new build system or method of creating images (such as Docker and its corresponding Dockerfile syntax) in order to incorporate the benefits of containers into an embedded Linux product. Keeping the focus on the OpenEmbedded workflow ensures that we retain all the benefits of this system, such as the excellent license compliance tooling, the extensible SDK and a proper cross-compilation environment. Other methods of creating container-based Linux systems are typically targeted at cloud computing deployments and don't address these issues that crop up when shipping an embedded Linux product.

The benefits of Linux containers have been discussed at length elsewhere so we won't cover the general benefits here. However, it's worth mentioning the additional benefits that we get in the embedded world:

- The ability to isolate applications requiring access to specialised hardware from those which just use 'normal' Linux interfaces such as the network and filesystems.
- The ability to mix legacy software which is dependent on specific older versions of system libraries with an up-to-date and secure base system. This is especially relevant in the embedded space where legacy applications abound.
- The ability to update and restart a full application stack cleanly and quickly by restarting a container guest instead of rebooting the whole device. For devices with long startup times there can be significant benefit here.

1.1.3 Support

For support requests, bug reports or other feedback please open an issue in the Togán Labs bug tracker or contact us at support@toganlabs.com.

1.1.4 Notation

The following notation is used for arguments:

- `ARGUMENT`: A required argument.
- `[ARGUMENT]`: An optional argument.
- `ARGUMENTS...`: One or more required arguments which are not parsed further by `oryxcmd`. This is typically used for arguments which are passed through to another application.

1.1.5 Copyright and Trademark notices

1.2 Using Oryx Linux

This section describes how to install and use Oryx Linux on an embedded device.

1.2.1 Supported Platforms

This release of Oryx Linux supports all features on two demonstration platforms:

- qemux86: This is a 32-bit x86 system emulated using qemu.
- Raspberry Pi: This is the popular 32-bit ARM based system. Images for both the original "raspberrypi" model and the newer "raspberrypi3" model are provided.

1.2.2 Installation and Getting Started

qemux86

Download the following files from the v0.2.0 release to run Oryx Linux in qemu:

- Kernel image
- Rootfs image

The rootfs image must first be decompressed:

```
unxz oryx-native-host-qemux86.ext4.xz
```

To launch qemu:

```
qemu-system-i386 -kernel bzImage-qemux86.bin -hda oryx-native-host-qemux86.ext4 -
↪append "root=/dev/hda"
```

For further details on the configuration and use of qemu, see the qemu documentation.

Raspberry Pi

For the original "raspberrypi" model, download the following file from the v0.2.0 release:

- SD card image [raspberrypi]

For the newer "raspberrypi3" model, download the following file from the v0.2.0 release:

- SD card image [raspberrypi3]

Once the appropriate SD card image has been downloaded, it must first be decompressed:

```
unxz oryx-native-host-raspberrypi.rpi-sdimg.xz
```

The uncompressed SD card image should then be written to an appropriate SD card (in this example the target SD card appears in the system as `/dev/sdb` but this should be replaced by the correct path for the system in use):

```
dd if=oryx-native-host-raspberrypi.rpi-sdimg of=/dev/sdb bs=1M
```

The SD card may then be removed and placed into the Raspberry Pi device itself.

1.2.3 Adding Guest Containers

One the Oryx Linux host system has been set up, the *oryxcmd* tool may be used to create guest containers.

Firstly, the appropriate official source for this release should be configured:

- qemux86:

  ```
  oryxcmd add_source oryx http://downloads.toganlabs.com/oryx/distro/0.2.0/qemux86
  ```

- raspberrypi:

  ```
  oryxcmd add_source oryx http://downloads.toganlabs.com/oryx/distro/0.2.0/
  ↪raspberrypi
  ```

- raspberrypi3:

  ```
  oryxcmd add_source oryx http://downloads.toganlabs.com/oryx/distro/0.2.0/
  ↪raspberrypi3
  ```

Once this source is configured, a guest container can be created from one of the following images:

- `minimal` image:

  ```
  oryxcmd add_guest test oryx:minimal
  ```

- `full-cmdline` image:

  ```
  oryxcmd add_guest test oryx:full-cmdline
  ```

The guest image may then be booted using `runc` as follows:

```
oryxcmd runc test run test
```

For further details, see the full documentation for the *oryxcmd* tool.

1.3 oryx-apps

oryx-apps is a collection of applications which implement the core functionality of the Oryx Linux distro. However, oryx-apps is also available independently of Oryx Linux and so these applications may be re-used and integrated into other Linux distros if desired.

1.3.1 oryxcmd

`oryxcmd` is the core of the "host" application profile within Oryx Linux. It is responsible for the management of guest containers and the sources from which container images may be obtained. As a command-line application it has both an interactive mode and a non-interactive mode.

Interactive Mode

In the interactive mode, `oryxcmd` is started without specifying a command:

```
$ oryxcmd
Welcome to oryxcmd (oryx-apps v0.1.0)
oryxcmd>
```

At the `oryxcmd` prompt, any of the supported *Commands* may be executed. For example:

```
oryxcmd> list_sources
oryx
```

To leave interactive mode, use the `exit` command:

```
oryxcmd> exit
```

Non-interactive Mode

In the non-interactive mode, `oryxcmd` is executed with a command specified as an argument. The specified command will be executed and then `oryxcmd` will exit. For example:

```
$ oryxcmd list_sources
oryx
```

Any of the supported *Commands* may be executed in this way.

Command Line Arguments

The following command line arguments are supported by `oryxcmd`:

- `-v`, `--verbose`: Print verbose debug messages during operation. This argument is usable for both interactive and non-interactive mode.

- `-h`, `--help`: Print help messages and exit.
- `-V`, `--version`: Print version string and exit.

Commands

add_source

Register a new source from which images may be fetched.

Usage:

```
add_source NAME URL
```

Arguments:

- `NAME`: An identifier which may be used to reference this source in future commands.
- `URL`: The root URL under which image archives may be found.

Example:

```
oryxcmd> add_source oryx http://downloads.toganlabs.com/oryx/distro/0.2.0/raspberrypi3
Added source "oryx" with URL "http://downloads.toganlabs.com/oryx/distro/0.2.0/
↪raspberrypi3"
```

remove_source

Remove a previously registered source.

Usage:

```
remove_source NAME
```

Arguments:

- `NAME`: The identifier of the source to remove.

Example:

```
oryxcmd> remove_source oryx
Removed source "oryx"
```

list_sources

List all currently registered sources.

Usage:

```
list_sources
```

This command has no arguments.

Example:

```
oryxcmd> list_sources
oryx
```

show_source

Show details of a previously registered source in JSON format.

Usage:

```
show_source NAME
```

Arguments:

- `NAME`: The identifier of the source to show.

Example:

```
oryxcmd> show_source oryx
{
    "url": "http://downloads.toganlabs.com/oryx/distro/0.2.0/raspberrypi3"
}
```

add_guest

Create a new guest container from an image.

Usage:

```
add_guest NAME IMAGE
```

Arguments:

- `NAME`: An identifier which may be used to reference this source in future commands.
- `IMAGE`: A fully-qualified reference to an image which is available from one of the sources which has been configured. The format of this reference is `<source>:<image_name>`:
  - `source`: The identifier of a registered source.
  - `image_name`: The name of an image which is available within the identified source. The image name typically matches the name of an *Application Profile* which has been built for the system on which `oryxcmd` is running.

Example:

```
oryxcmd> add_guest test oryx:minimal
Added guest "test" from image "oryx:minimal"
```

remove_guest

Delete an existing guest container.

Usage:

```
remove_guest NAME
```

Arguments:

- `NAME`: The identifier of the guest container to remove.

Example:

```
oryxcmd> remove_guest test
Removed guest "test"
```

list_guests

List all currently registered guests.

Usage:

```
list_guests
```

This command has no arguments.

Example:

```
oryxcmd> list_guests
test
```

show_guest

Show details of a previously registered guest in JSON format.

Usage:

```
show_guest NAME
```

Arguments:

- `NAME`: The identifier of the guest to show.

Example:

```
oryxcmd> show_guest test
{
    "image": {
        "APPLICATION_PROFILE": "minimal",
        "ARCHIVE": "oryx-guest-minimal-raspberrypi3.tar.xz",
        "DISTRO": "oryx",
        "MACHINE": "raspberrypi3",
        "SYSTEM_PROFILE": "guest",
        "VERSION": "dev"
    },
    "image_name": "minimal",
    "path": "/var/lib/oryx-guests/test",
    "source": {
        "url": "http://downloads.toganlabs.com/oryx/distro/0.2.0/raspberrypi3"
    },
    "source_name": "oryx"
}
```

runc

Execute `runc` for an existing guest container. See the documentation of `runc` for further details.

Usage:

```
runc NAME ARGS...
```

Arguments:

- `NAME`: The identifier of the guest container for which 'runc' will be executed.
- `ARGS...`: Command line arguments passed through to the 'runc' application.

Example:

```
oryxcmd> runc test run test
/ # ps
PID   USER     TIME   COMMAND
    1 root       0:00 python3 /sbin/oryx-guest-init
    7 root       0:00 sh
    8 root       0:00 ps
/ # hostname
test
/ # exit
```

help

List available commands with "help" or detailed help with "help cmd".

Usage:

```
help [CMD]
```

Arguments:

- `CMD`: The name of a supported command. If this argument is given, detailed help for the chosen command is printed.

Example:

```
oryxcmd> help

Documented commands (type help <topic>):
========================================
add_guest   exit  list_guests   remove_guest   runc        show_source
add_source  help  list_sources  remove_source  show_guest  version

Miscellaneous help topics:
==========================
arguments
```

version

Display version information.

Usage:

```
version
```

This command has no arguments.

Example:

```
oryxcmd> version
oryxcmd (oryx-apps v0.1.0)
```

exit

Exit the interactive oryxcmd shell.

Usage:

```
exit
```

This command has no arguments.

Example:

```
oryxcmd> exit
```

1.3.2 oryx-guest-init

`oryx-guest-init` is a cut-down init system suitable for use in a guest container. It is currently very basic and simply starts a shell for interactive use and reaps zombie processes in the background. When the shell is exited, the guest container is shutdown.

1.4 Building Oryx Linux Images

Oryx Linux introduces two major new concepts to the OpenEmbedded build system: these are *System Profiles* and *Application Profiles*. This section will also discuss how these concepts are integrated into the *OpenEmbedded Recipes* in the `meta-oryx` layer.

1.4.1 System Profiles

A system profile complements the OpenEmbedded machine selection and essentially specifies how the image we are building will be deployed onto the selected machine. Many platforms may be booted in multiple ways (local boot from flash memory vs remote boot via tftp for instance) and a system profile may be used to specify a boot mechanism. Additionally, an image may run under different virtualisation methods on a given platform and a system profile may be used to specify the chosen method. In each case the system profile will ensure that the correct build artifacts are produced to match how the image will be used. As system profiles are orthogonal to machine selection, consistent boot or virtualisation methods may be enforced across multiple platforms.

Two system profiles are provided in the initial Oryx release:

- `native`: This profile indicates that the image will run "bare metal" on the chosen platform. Build artifacts suitable for writing to an SD card, USB stick or embedded flash memory are produced and are then compressed to save space. When possible, u-boot is enabled to provide greater boot-time flexibility.
- `guest`: This profile indicates that the image will run as a container guest under runc. No bootloader or kernel is compiled for this profile. Build artifacts are always compressed tar archives of a rootfs, ready for installation onto a host system.

The system profile is determined by the `ORYX_SYSTEM_PROFILE` variable.

1.4.2 Application Profiles

An application profile specifies the use-case of a given image and typically corresponds to a particular software package or package group. The configurability here is greater than a traditional OpenEmbedded image recipe though, as the application profile may set `PACKAGECONFIG` values and other options to be applied to all components within an image. So it's possible to build a lightweight configuration of a library for one application profile but then enable additional options when building for a different application profile.

Here are two of the major application profiles provided in the initial Oryx release:

- `full-cmdline`: The profile simply combines the OpenEmbedded full-cmdline package group with an SSH server.
- `host`: This profile includes runc and other tools needed to setup Linux containers. It provides a host environment for images built using the guest system profile described above.

It's expected that Oryx will be enhanced by the addition of many more application profiles in future releases.

The application profile is determined by the `ORYX_APPLICATION_PROFILE` variable.

1.4.3 OpenEmbedded Recipes

oryx-image

The concept of an application profile effectively supersedes the OpenEmbedded concept of an image recipe. Therefore we only make use of one image recipe within Oryx and this is the `oryx-image` recipe. This recipe pulls in the packages needed by the chosen application and system profiles.

The `oryx-image` recipe also ensures that an extended `os-release` file is included in the image. This `os-release` file includes the usual information such as the distro name, version and home URL as well as Oryx-specific information such as the selected system profile, application profile and machine.

image-json-file

The `image-json-file` recipe creates a JSON formatted data file for the current image which is used by *oryxcmd* when downloading the image onto a host system.

oryx-publish

To simplify deployment of Oryx images we also have a top-level `oryx-publish` recipe. This recipe copies files specified by the chosen system profile from the OpenEmbedded `deploy/images` directory to a new `deploy/oryx` directory. This may seem trivial but it gives two benefits. As only those files required by the boot or installation method used with a given system profile are copied into the new directory, there is no clutter or confusion. Also, the `deploy/oryx` directory has sub-directories for the current version, selected system profile and selected application profile and this ensures that an image produced for one configuration is not accidentally overwritten by a subsequent build for a different configuration.

In normal usage, the top-level bitbake recipe used to build an Oryx image will therefore be `oryx-publish`.

1.4.4 Using Integrated Sources

The recommended way to build Oryx Linux images is to use the integrated source tree which combines the `meta-oryx` layer and a pre-configured build environment with the OpenEmbedded build system. This is the method which is used for Oryx Linux releases and is regularly tested as part of the Continuous Integration (CI) system.

The full contents of the integrated Oryx Linux sources is as follows:

- The base `openembedded-core` layer.
- The corresponding version of `bitbake`.
- Additional supporting layers: `meta-openembedded` and `meta-virtualisation`.
- Additional BSP layers: `meta-arduino` and `meta-raspberrypi`.
- The Oryx Linux distro layer: `meta-oryx`.
- Pre-configured build environment consisting of `build/conf/local.conf` and `build/conf/bblayers.conf` files which typically do not require further modification.
- The `build/conf/setenv` environment setup script.
- Build scripts and other supporting scripts under `build/scripts/`.

Fetching and Updating Sources

Integrated sources may be obtained either from a source release in `.tar.xz` format, or from git using the `repo` tool.

Using a Source Release

Each point release of Oryx Linux includes a source tarball alongside the compiled images. This integrated source release contains all OpenEmbedded layers needed to build Oryx Linux images and is essentially a point-in-time snapshot of the sources which may be obtained from git using the `repo` tool.

For the v0.2.0 release, this source release may be obtained from https://downloads.toganlabs.com/oryx/distro/0.2.0/oryx-0.2.0.tar.xz.

Once a source release has been downloaded, it simply needs to be extracted before following the steps in the *Preparing the Environment* section.

Using repo

The sources for Oryx Linux are split between several git repositories and the `repo` tool may be used to fetch an integrated source tree which combines these repositories. This method allows a formal Oryx Linux release to be obtained with similar results to *Using a Source Release* above. It also allows the latest commit from each repository on either the master branch or a stable branch to be obtained.

Firstly, the `repo` tool must be installed as follows:

- For Ubuntu 16.04, simply execute `apt install repo` as root. This may also work on more recent releases of Ubuntu and related distributions.
- For other distributions, see the installation instructions at https://source.android.com/source/downloading#installing-repo.

Then in a new, empty directory initialise `repo` as follows:

- To use the `master` branch of Oryx Linux:

```
repo init -u git@gitlab.com:oryx/oryx-manifest.git
```

  The `master` branch is the active development branch and so may incorporate breaking changes at any time. Follow the `master` branch at your own risk!

- To use a stable branch of Oryx Linux, such as the `pyro` branch:

```
repo init -u git@gitlab.com:oryx/oryx-manifest.git -b pyro
```

  Changes in the stable branches follow a strict stable branch policy and so should not introduce breakage. Stable branch names match those used in OpenEmbedded, for further details see the upstream list of stable branches and maintainers.

- To use a formal release of Oryx Linux, such as the v0.2.0 release:

```
repo init -u git@gitlab.com:oryx/oryx-manifest.git -b refs/tags/v0.2.0
```

  For other tagged releases, ensure that the `refs/tags/` prefix is used in the `repo init` command.

Once `repo` has been initialised, sources may be obtained by running `repo sync`. To update sources at a later date, simply re-run `repo sync`.

Preparing the Environment

Once the Oryx Linux source tree has been downloaded, simply source the `build/conf/setenv` script in a bash shell to prepare the environment for a build.

Build Script

Once you have sourced the `setenv` script, you can use run-build:

```
scripts/run-build.py [-C] [-L] [-V VERSION] [-M MACHINE] [-S SYSTEM_PROFILE] [-A␣
↪APPLICATION_PROFILE]
```

This script uses bitbake to build the recipe specified by *oryx-publish*.

Output files from run-build are saved in the pub directory, which is divided into subdirectories by, respectively: version, machine, system profile, and application profile. As well as the build output, this contains the *logs* file if you have chosen -L, and a *FAILED* file if the build itself has failed.

Customising a build

There are a number of ways available to customise your build.

- `-V VERSION`: Sets the ORYX_VERSION variable.
  - Allows you to specify the version string used to identify this build.
  - The default value is “dev”.
- `-S SYSTEM_PROFILE`: System profile selection.
  - This sets the ORYX_SYSTEM_PROFILE variable.
  - See the *System Profiles* section for details on how system profiles work, and what options are available.
  - The default value is “native”.
- `-A APPLICATION_PROFILE`: Application profile selection.
  - This sets the ORYX_APPLICATION_PROFILE variable.
  - See the *Application Profiles* section for details on application profiles, as well as the options available.

  - The default value is “minimal”.
- `-M MACHINE`: Machine selection.
  - This sets the MACHINE variable.
  - Supported machines are: `qemux86`, `raspberrypi`, `raspberrypi3` and `arduino-yun`.
  - The default value is “qemux86”.
- `-C`: Performs a clean build.
  - Removes the contents of the tmp directory before running bitbake.
  - The default is not to perform a clean build, leaving the previous content of the tmp directory intact.
- `-L`, `--logs`: Captures and archives log files.
  - Log files are copied from the tmp directory into a *logs.tar.gz* file located in: `pub/${ORYX_VERSION}/${MACHINE}/${ORYX_SYSTEM_PROFILE}/${ORYX_APPLICATION_PROFILE}`.
  - The default is not to capture log files.

For example:

```
scripts/run-build.py -S native -A host -C
```

Performs a clean build using the `native` system profile and the `host` application profile.

1.4.5 Using meta-oryx as a Standalone Layer

Although the above method of *Using Integrated Sources* is preferred as this is the tested and supported method, it’s also possible to use the `meta-oryx` layer as a traditional OpenEmbedded layer. This layer may be obtained from the git repository at https://gitlab.com/oryx/meta-oryx and added into an OpenEmbedded build environment as normal.

Once the `meta-oryx` layer has been added to the OpenEmbedded build environment, the following variables should be set in `conf/local.conf` or another appropriate location to fully configure the Oryx Linux distribution:

- Set the distro: `DISTRO = "oryx"`.
- Set the Oryx Linux version: `ORYX_VERSION = "custom"`. Using a unique version string here will help identify this build.
- Choose a *System Profile*: `ORYX_SYSTEM_PROFILE = "native"`.
- Choose an *Application Profile*: `ORYX_APPLICATION_PROFILE = "minimal"`.

Once these variables are set appropriately, `bitbake` may be executed as normal. As discussed in the section on *OpenEmbedded Recipes*, the top-level command to build an Oryx Linux image is typically `bitbake oryx-publish`.

CHAPTER

TWO

ORYX LITE

2.1 Oryx Lite

Oryx Lite is a stripped down configuration of Oryx Linux for platforms which do not support `runc` or some other component of the full Oryx Linux distro. The *oryxcmd* tool and other containerisation features of the full Oryx Linux distro are therefore not available in Oryx Lite. Instead, a selection of application-focused native images are made available in Oryx Lite.

2.1.1 Supported Platforms

This release supports Oryx Lite on the following demonstration platform:

- Arduino Yún: This is a 32-bit MIPS device.

2.1.2 Installation

Download one of the following Oryx Lite rootfs images for the Arduino Yún:

- minimal rootfs image
- full-cmdline rootfs image

Additionally, download the following:

- Kernel image

Once the appropriate files have been downloaded, decompress the rootfs image:

```
unxz oryx-native-minimal-arduino-yun.ext2.xz
```

The uncompressed rootfs image should then be written to the first partition on an appropriate SD card (in this example the target partition appears in the system at `/dev/sdb1` but this should be replaced by the correct path for the system in use):

```
dd if=oryx-native-minimal-arduino-yun.ext2 of=/dev/sdb1 bs=1M
```

The kernel image must be made available on a local TFTP server. The bring-up instructions in the meta-arduino readme file may then be followed to boot Oryx Lite on the Arduino Yún.

2.1.3 Building Oryx Lite Images

Oryx Lite images are built in the same way as full Oryx Linux images and so all the information in *Building Oryx Linux Images* applies. The only difference in Oryx Lite is that the `guest` system profile and the `host` application profile are not used. Instead, the supported application images are all built using the `native` system profile.

CHAPTER

THREE

INDICES AND TABLES

- search