(/)

An OS to build, deploy and securely manage billions of devices

Latest News:

Apache Mynewt 1.10.0, Apache NimBLE 1.5.0 (/download) released (May 6, 2022)

Search documentation

Version: latest

Newt Tool Guide

Introduction

Newt is a smart build and package management system for embedded contexts. It is a single tool that accomplishes both the following goals:

- source package management
- build, debug and install.

Rationale

In order for the Mynewt operating system to work well for constrained environments across the many different types of micro-controller applications (from doorbells to medical devices to power grids), a system is needed that lets you select which packages to install and which packages to build.

The build systems for embedded devices are often fairly complicated and not well served for this purpose. For example, autoconf is designed for detecting system compatibility issues but not well suited when it comes to tasks like:

- Building for multiple targets
- Deciding what to build in and what not to build in
- Managing dependencies between components

Fortunately, solutions addressing these very issues can be found in source package management systems in higher level languages such as Javascript (Node), Go, PHP and Ruby. We decided to fuse their source management systems with a make system built for embedded systems and create Newt.

Build System

A good build system must allow the user to take a few common steps while developing embedded applications:

- Generate full flash images
- Download debug images to a target board using a debugger

- Conditionally compile libraries & code based upon build settings

Newt can read a directory tree, build a dependency tree, and emit the right build artifacts. An example newt source tree is in mynewt-blinky/develop:

```
$ tree -L 3 .
├── DISCLAIMER
├── LICENSE
├── NOTICE
├── README.md
├── apps
│ └── blinky
│ ├── pkg.yml
│ └── src
├── project.yml
└── targets
    ├── my_blinky_sim
    │ ├── pkg.yml
    │ └── target.yml
    └── unittest
        ├── pkg.yml
        └── target.yml

6 directories, 10 files
```

When Newt sees a directory tree that contains a “project.yml” file, it is smart enough to recognize it as the base directory of a project, and automatically builds a package tree. It also recognizes two important package directories in the package tree - “apps” and “targets”. More on these directories in Theory of Operations (newt_operation.html).

When Newt builds a target, it recursively resolves all package dependencies, and generates artifacts that are placed in the bin/targets/<target-name>/app/apps/<app-name> directory, where the bin directory is under the project base directory, `target-name` is the name of the target, and `app-name` is the name of the application. For our example `my_blinky_sim` is the name of the target and `blinky` is the name of the application. The `blinky.elf` executable is stored in the bin/targets/my_blinky_sim/app/apps/blinky directory as shown in the source tree:

```
$ tree -L 6 bin/
bin/
└── targets
    ├── my_blinky_sim
    │   ├── app
    │   │   ├── apps
    │   │   │   └── blinky
    │   │   │       ├── apps
    │   │   │       ├── apps_blinky.a
    │   │   │       ├── apps_blinky.a.cmd
    │   │   │       ├── blinky.elf
    │   │   │       ├── blinky.elf.cmd
    │   │   │       ├── blinky.elf.dSYM
    │   │   │       ├── blinky.elf.lst
    │   │   │       └── manifest.json
    │   │   ├── hw
    │   │   │   ├── bsp
    │   │   │   │   └── native
    │   │   │   ├── drivers
    │   │   │   │   └── uart
    │   │   │   ├── hal
    │   │   │   │   ├── hw_hal.a
    │   │   │   │   ├── hw_hal.a.cmd
    │   │   │   │   └── repos

<snip>
```

More operations using Newt

Once a target has been built, Newt allows additional operations on the target.

- **load**: Download built target to board
- **debug**: Open debugger session to target
- **size**: Get size of target components
- **create-image**: Add image header to the binary image
- **run**: Build, create image, load, and finally open a debug session with the target
- **target**: Create, delete, configure, and query a target

For more details on how Newt works, go to Theory of Operations (newt_operation.html).

Source Management and Repositories

The other major element of the Newt tool is the ability to create reusable source distributions from a collection of code. **A project can be a reusable container of source code.** In other words, projects can be versioned and redistributed, not packages. A project bundles together packages that are typically needed to work together in a product e.g. RTOS core, filesystem APIs, and networking stack.

A project that has been made redistributable is known as a **repository**. Repositories can be added to your local project by adding them into your project.yml file. Here is an example of the blinky project's yml file which relies on apache-mynewt-core:

```
$ more project.yml
<snip>
project.repositories:
      - apache-mynewt-core

repository.apache-mynewt-core:
      type: github
      vers: 1-latest
      user: apache
      repo: incubator-mynewt-core
```

When you specify this repository in the blinky's project file, you can then use the Newt tool to install dependencies:

```
$ newt install
Downloading repository description for apache-mynewt-core... success!
Downloading repository incubator-mynewt-core (branch: develop) at https://github.com/apache/incubator-mynewt-core.git
Cloning into '/var/folders/7l/7b3w9m4n2mg3sqmgw2q1b9p80000gn/T/newt-repo814721459'...
remote: Counting objects: 17601, done.
remote: Compressing objects: 100% (300/300), done.
remote: Total 17601 (delta 142), reused 0 (delta 0), pack-reused 17284
Receiving objects: 100% (17601/17601), 6.09 MiB \| 3.17 MiB/s, done.
Resolving deltas: 100% (10347/10347), done.
Checking connectivity... done.
Repos successfully installed
```

Newt will install this repository in the <project>/repos directory. In the case of blinky, the directory structure ends up looking like:

```
$ tree -L 2
.
├── DISCLAIMER
├── LICENSE
├── NOTICE
├── README.md
├── apps
│   └── blinky
├── project.state
├── project.yml
├── repos
│   └── apache-mynewt-core
└── targets
     ├── my_blinky_sim
     └── unittest
```

In order to reference the installed repositories in packages, the “@” notation should be specified in the repository specifier. As an example, the apps/blinky application has the following dependencies in its pkg.yml file. This tells the build system to look in the base directory of repos/apache-mynewt-core for the `kernel/os`, `hw/hal`, and `sys/console/full` packages.

```
$ more apps/blinky/pkg.yml
pkg.deps:
  - "@apache-mynewt-core/kernel/os"
  - "@apache-mynewt-core/hw/hal"
  - "@apache-mynewt-core/sys/console/full"
```

Newt has the ability to autocomplete within `bash`. The following instructions allow MAC users to enable autocomplete within `bash`.

1. Install the autocomplete tools for bash via `brew install bash-completion`
2. Tell your shell to use newt for autocompletion of newt via `complete -C "newt complete" newt`. You can add this to your .bashrc or other init file to have it automatically set for all bash shells.

Notes:

1. Autocomplete will give you flag hints, but only if you type a ‘-‘.
2. Autocomplete will not give you completion hints for the flag arguments (those optional things after the flag like `-l DEBUG`)
3. Autocomplete uses newt to parse the project to find targets and libs.

Previous: Sample application (../network/mesh/sample.html) Next: Theory of Operations (newt_operation.html)