Building embedded systems using Elixir

An introduction to Nerves and Phoenix Framework

Milton Mazzarri
milmazz @

Jul 17, 2019

Houston Functional Programmers Group

Introduction

Goals

- Introduction to Elixir and Nerves
- Boot a Raspberry Pi to an IEx prompt
- Boot a Raspberry Pi 3 B+ to an IEx prompt

Definitions

Nerves

Nerves defines a new way to build embedded systems using Elixir. It's specifically designed for embedded systems, not desktop or server systems. Nerves is composed by three parts:

Platform a customized, minimal Buildroot-derived Linux that boots directly to the BEAM VM

Framework ready-to-go library of Elixir modules to get you up and running quickly

Tooling powerful command-line tools to manage builds, update firmware, configure devices, and more

Nerves

Nerves defines a new way to build embedded systems using Elixir. It's specifically designed for embedded systems, not desktop or server systems. Nerves is composed by three parts:

Platform a customized, minimal Buildroot-derived Linux that boots directly to the BEAM VM

Framework ready-to-go library of Elixir modules to get you up and running quickly

Tooling powerful command-line tools to manage builds, update firmware, configure devices, and more

Nerves

Nerves defines a new way to build embedded systems using Elixir. It's specifically designed for embedded systems, not desktop or server systems. Nerves is composed by three parts:

Platform a customized, minimal Buildroot-derived Linux that boots directly to the BEAM VM

Framework ready-to-go library of Elixir modules to get you up and running quickly

Tooling powerful command-line tools to manage builds, update firmware, configure devices, and more

Host

The computer on which you are editing source code, compiling, and assembling firmware

Target

The platform for which your firmware is built (for example, Raspberry Pi, Raspberry Pi 2, or Beaglebone Black)

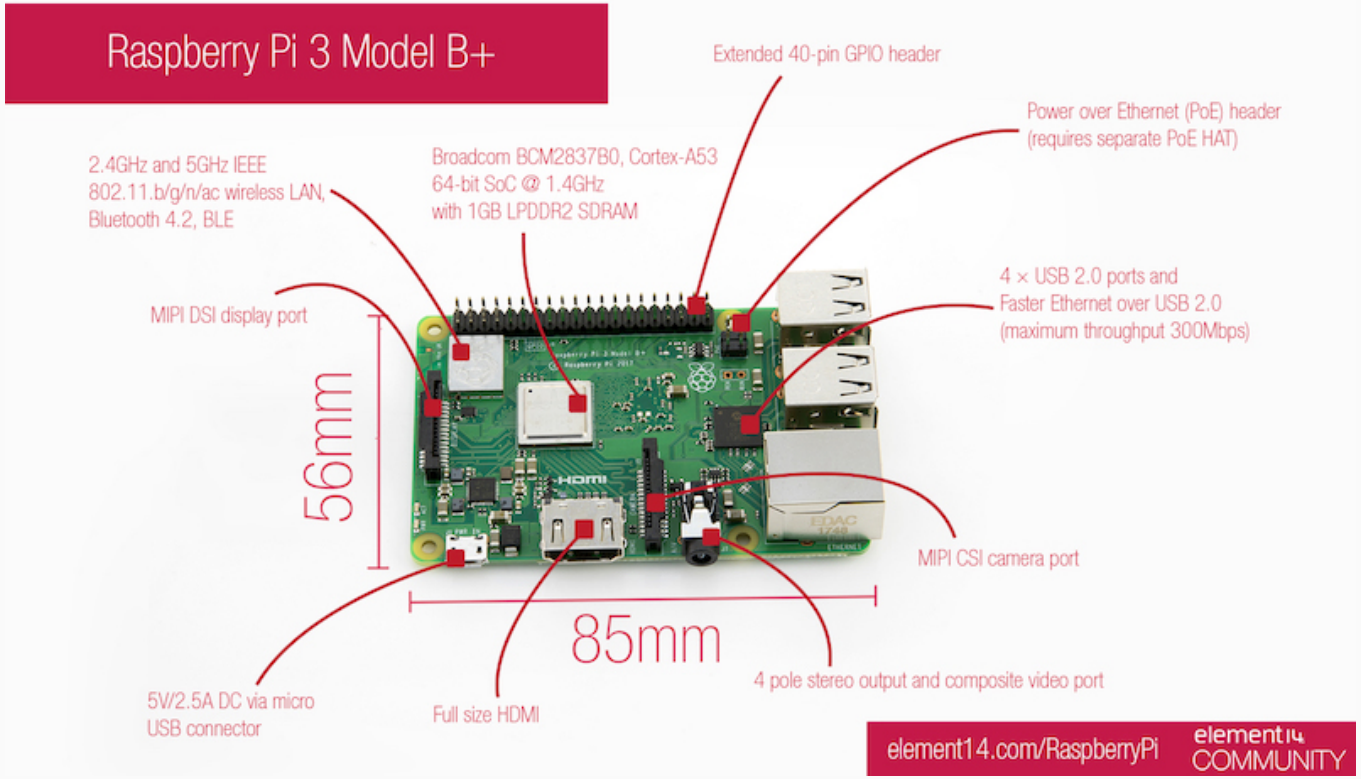


Figure 1: Raspberry Pi 3 Model B+

Source: https://www.element14.com

Toolchain

The tools required to build code for the target, such as compilers, linkers, binutils, and C runtime

System

A lean Buildroot-based Linux distribution that has been customized and cross-compiled for a particular target

Firmware bundle

A single file that contains an assembled version of everything needed to burn firmware

Firmware image

Built from a firmware bundle and contains the partition table, partitions, bootloader, etc.

Nerves overview

Batteries included

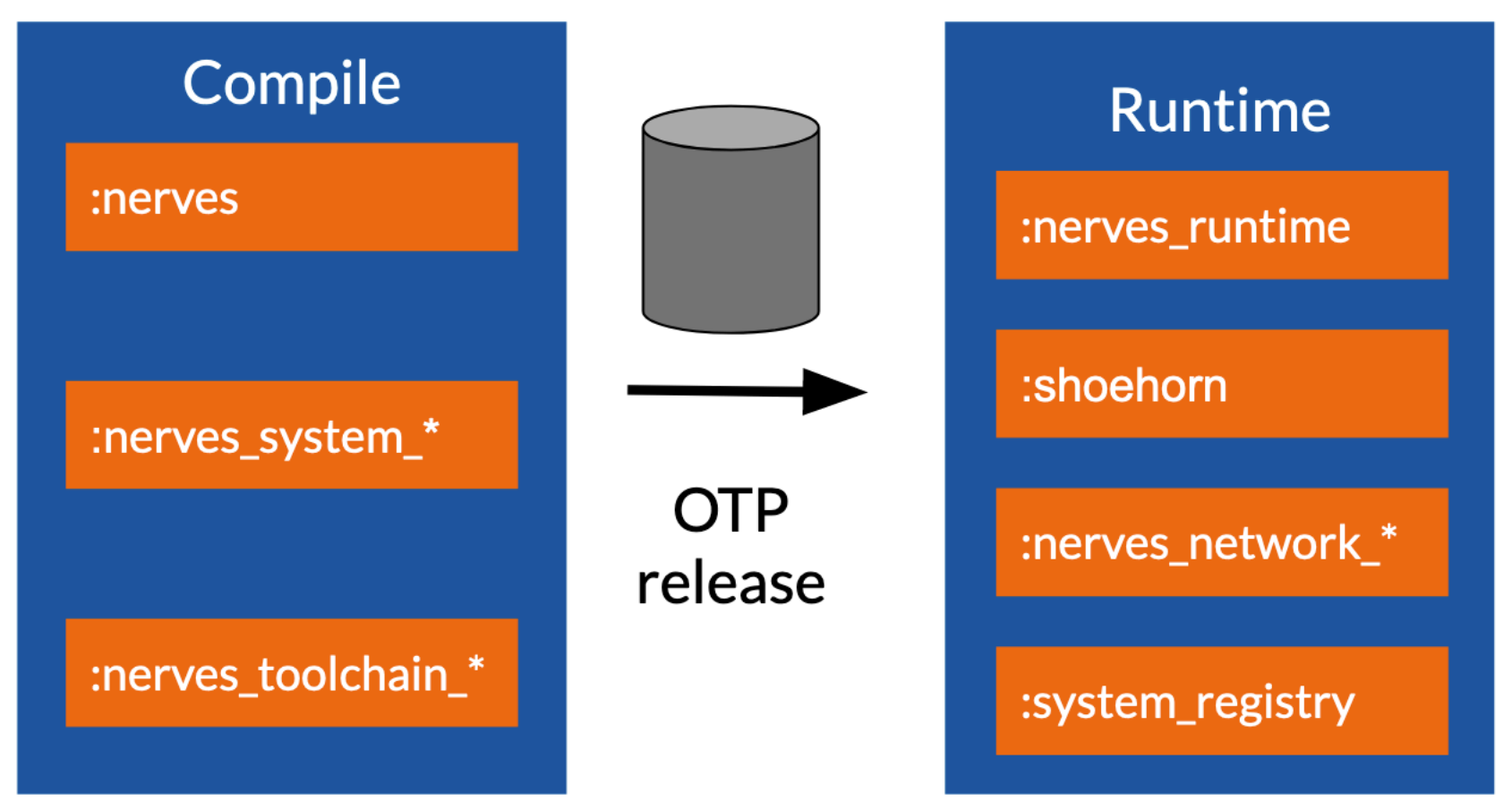


Figure 2: Nerves Overview

Supported targets and systems

Target	System	Tag
Raspberry Pi A+, B, B+	`nerves_system_rpi`	`rpi`
Raspberry Pi Zero and Zero W	`nerves_system_rpi0`	`rpi0`
Raspberry Pi 2	`nerves_system_rpi2`	`rpi2`
Raspberry Pi 3 B, B+	`nerves_system_rpi3`	`rpi3`
BeagleBone Black, BeagleBone Green, BeagleBone Green Wireless, PocketBeagle	`nerves_system_bbb`	`bbb`
Generic x86_64	`nerves_system_x86_64`	`x86_64`

Artifacts

- application name

Example

`nerves_system_rpi-portable-1.7.2-ED1F541.tar.gz`

- host
- version
- checksum

Artifacts

- application name

Example

`nerves_system_rpi-portable-1.7.2-ED1F541.tar.gz`

- host
- version
- checksum

Artifacts

- application name

Example

`nerves_system_rpi-portable-1.7.2-ED1F541.tar.gz`

- host
- version
- checksum

Artifacts

- application name

Example

`nerves_system_rpi-portable-1.7.2-ED1F541.tar.gz`

- host
- version
- checksum

tar files `~/.nerves/dl`

uncompressed files `~/.nerves/artifacts`

Getting started

mix help nerves.new

Creates a new Nerves project

```
mix nerves.new PATH [--module MODULE] [--app APP]
                    [--target TARGET] [--cookie STRING]
```

The project will be created at PATH. The application name and module name will be inferred from PATH unless --module or --app is given.

An --app option can be given in order to name the OTP application for the project.

A --module option can be given in order to name the modules in the generated code skeleton.

A --target option can be given to limit support to one or more of the officially Nerves systems. For a list of supported targets visit https://hexdocs.pm/nerves/targets.html#supported-targets-and-systems

A --cookie options can be given to set the Erlang distribution cookie in vm.args. This defaults to a randomly generated string.

Generate a project without nerves_init_gadget support by passing --no-init-gadget.

mix help nerves.new

Creates a new Nerves project

```
$ mix nerves.new blinky
$ # Is equivalent to:
$ mix nerves.new blinky --module Blinky
```

Generate a project that only supports Raspberry Pi 3

```
$ mix nerves.new blinky --target rpi3
```

Generate a project that supports Raspberry Pi 3 and Raspberry Pi Zero

```
$ mix nerves.new blinky --target rpi3 --target rpi0
```

Generate a project without `nerves_init_gadget`

```
$ mix nerves.new blinky --no-init-gadget
```

Demo

Recap

Our first Nerves project

Example

```
$ mix nerves.new say --target rpi --cookie hello
* creating say/config/config.exs
* creating say/lib/say.ex
* creating say/lib/say/application.ex
* creating say/test/test_helper.exs
* creating say/test/say_test.exs
* creating say/rel/vm.args
* creating say/rootfs_overlay/etc/iex.exs
* creating say/.gitignore
* creating say/.formatter.exs
* creating say/mix.exs
* creating say/README.md

Fetch and install dependencies? [Yn] Y
* running mix deps.get
Your Nerves project was created successfully.
```

Our first Nerves project

```
You should now pick a target. See
https://hexdocs.pm/nerves/targets.html#content
for supported targets. If your target is on the list, set `MIX_TARGET`
to its tag name:

For example, for the Raspberry Pi 3 you can either
  $ export MIX_TARGET=rpi3
Or prefix `mix` commands like the following:
  $ MIX_TARGET=rpi3 mix firmware

If you will be using a custom system, update the `mix.exs`
dependencies to point to desired system's package.

Now download the dependencies and build a firmware archive:
  $ cd say
  $ mix deps.get
  $ mix firmware

If your target boots up using an SDCard (like the Raspberry Pi 3),
then insert an SDCard into a reader on your computer and run:
  $ mix firmware.burn

Plug the SDCard into the target and power it up. See target documentation
above for more information and other targets.
```

Our first Nerves project

Example

```
$ cd say
$ set -x MIX_TARGET rpi
$ mix deps.get
```

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

Project structure

```
.
├── README.md
├── _build
├── config
│   ├── config.exs  4
│   └── target.exs
├── deps  6
├── lib
│   ├── say
│   │   └── application.ex  2
│   └── say.ex  3
├── mix.exs  1
├── mix.lock
├── rel  7
│   └── vm.args.eex
├── rootfs_overlay
│   └── etc
│       └── iex.exs  8
└── test  5
    ├── say_test.exs
    └── test_helper.exs
```

1. Mix Project definition
2. Application definition
3. Main module
4. Configuration for all targets
5. Unit test suite
6. Dependencies archives
7. Releases configuration
8. The code loaded from here will be evaluated in the IEx session context of the target.

mix.exs - Mix Project definition

```elixir
defmodule Say.MixProject do
  use Mix.Project

  @app :say
  @version "0.1.0"
  @all_targets [:rpi]

  def project do
    [
      app: @app,
      version: @version,
      elixir: "~> 1.9",
      archives: [nerves_bootstrap: "~> 1.6"],
      start_permanent: Mix.env() == :prod,
      build_embedded: true,
      aliases: [loadconfig: [&bootstrap/1]],
      deps: deps(),
      releases: [{@app, release()}],
      preferred_cli_target: [run: :host, test: :host]
    ]
  end

  def bootstrap(args) do
    Application.start(:nerves_bootstrap)
    Mix.Task.run("loadconfig", args)
  end
```

mix.exs - Mix Project definition

```
  def application do
    [
      mod: {Say.Application, []},
      extra_applications: [:logger, :runtime_tools]
    ]
  end

  defp deps do
    [
      # Dependencies for all targets
      {:nerves, "~> 1.5.0", runtime: false},
      {:shoehorn, "~> 0.6"},
      {:ring_logger, "~> 0.6"},
      {:toolshed, "~> 0.2"},
      # Dependencies for all targets except :host
      {:nerves_runtime, "~> 0.6", targets: @all_targets},
      {:nerves_init_gadget, "~> 0.4", targets: @all_targets},
      # Dependencies for specific targets
      {:nerves_system_rpi, "~> 1.8", runtime: false, targets: :rpi},
    ]
  end
end
```

Unit tests

Example

```
# test/say_test.exs
defmodule SayTest do
  use ExUnit.Case
  doctest Say

  test "greets the world" do
    assert Say.hello() == :world
  end
end
```

Example

```
# test/test_helper.exs
ExUnit.start()
```

Example

```
# lib/say.ex
defmodule Say do
  @moduledoc """
  Documentation for Say.
  """

  @doc """
  Hello world.

  ## Examples

      iex> Say.hello
      :world

  """
  def hello do
    :world
  end
end
```

Running unit tests on host

Example

```
$ env MIX_TARGET=host mix test
==> nerves
Compiling 37 files (.ex)
Generated nerves app
==> toolshed
Compiling 9 files (.ex)
Generated toolshed app
==> ring_logger
Compiling 4 files (.ex)
Generated ring_logger app
==> shoehorn
Compiling 8 files (.ex)
Generated shoehorn app
==> say
Compiling 2 files (.ex)
Generated say app

..

Finished in 0.03 seconds
1 doctest, 1 test, 0 failures
```

Mix Tasks

Mix Task	Description
`mix firmware`	Build a firmware image for the selected target platform
`mix firmware.burn`	This task calls `mix firmware` and `mix burn` to burn a new firmware to a SDCard
`mix nerves.info`	Prints nerves system information
`mix burn`	Writes the generated firmware image to an attached SDCard or file
`mix firmware.image`	Create a firmware image file that can be copied byte-for-byte to an SDCard or other memory device

`nerves_init_gadget`

The `nerves_init_gadget` project provides the basics for getting started with Nerves. This includes bringing up networking, over-the-air firmware updates and many other little things that make using Nerves a little better.

nerves-project/nerves_init_gadget

Configuring nerves_init_gadget

Example

```
config :nerves_init_gadget,
ifname: "eth0",
address_method: :dhcpd,
mdns_domain: "nerves.local",
node_name: node_name,
node_host: :mdns_domain
```

Listing 1: config/target.exs

Our first Nerves project

Burn firmware

Example

```
$ mix firmware
$ # Insert SD Card
$ mix firmware.burn
$ ssh nerves.local
```

Our first Nerves project

Start a SSH session

```
$ ssh nerves.local
Interactive Elixir (1.9.0) - press Ctrl+C to exit (type h() ENTER for help)
Toolshed imported. Run h(Toolshed) for more info
RingLogger is collecting log messages from Elixir and Linux. To see the
messages, either attach the current IEx session to the logger:

  RingLogger.attach

or print the next messages in the log:

  RingLogger.next

iex(say@nerves.local)1>
```

Our first Nerves project

Trying out some things on the target:

Example

```
iex(say@nerves.local)1> Say.hello()
:world
iex(say@nerves.local)2> cmd("date")
Thu Jan  1 00:04:21 UTC 1970
0
iex(say@nerves.local)4> exit
Connection to nerves.local closed.
```

To close our SSH session we use the escape sequence `~.` or `exit/0` (provided by the Toolshed package)

Toolshed adds a number of commands to the IEx prompt to make working at the console more enjoyable. It's an experiment in aggregating code snippets from projects into one place.

fhunleth/toolshed

Some commands provided by Toolshed

command	description
`cmd`	run a command and print out the output
`top`	get a list of the top processes and their OTP applications based on CPU and memory
`exit`	exit an IEx session (useful over ssh)
`tree`	list directory contents as a tree
`save_term/load_term`	save and load Elixir terms to files
`tping`	check if a remote host is up (like ping, but uses TCP)
`ifconfig`	list network interfaces
`lsusb`	list USB devices

To get the complete list of commands execute: `iex> h Toolshed`

nerves_time

`Nerves.Time` **keeps the system clock on Nerves devices in sync when connected to the network and close to in sync when disconnected. It's especially useful for devices lacking a Battery-backed real-time clock and will advance the clock at startup to a reasonable guess.**

fhunleth/nerves_time

Introducing `nerves_time`

Example

```
$ mix hex.info nerves_time
Keep time in sync on Nerves devices

Config: {:nerves_time, "~> 0.2.1"}
Releases: 0.2.1, 0.2.0, 0.1.0

Licenses: Apache-2.0
Links:
  GitHub: https://github.com/fhunleth/nerves_time
```

Add `nerves_time` as dependency

```
# Dependencies for all targets except :host
{:nerves_runtime, "~> 0.6", targets: @all_targets},
{:nerves_init_gadget, "~> 0.4", targets: @all_targets},
{:nerves_time, "~> 0.2", targets: @all_targets},
```

Update dependencies and push changes back

Example

```
$ # Updated dependencies
$ mix deps.get
$ mix firmware.gen.script
Writing upload.sh...
$ # Upload new firmware to nerves.local
$ ./upload.sh
$ ssh nerves.local
```

Running IEx session on target

Example

```
iex(say@nerves.local)1> cmd("date")
Mon Jul  8 18:11:56 UTC 2019
0
iex(say@nerves.local)2> exit
Connection to nerves.local closed.
```

Introducing a matching error

Example

```
def start(_type, _args) do
  true = false

  # See https://hexdocs.pm/elixir/Supervisor.html
  # for other strategies and supported options
  opts = [strategy: :one_for_one, name: Say.Supervisor]
  children =
    [
      # Children for all targets
      # Starts a worker by calling: Say.Worker.start_link(arg)
      # {Say.Worker, arg},
    ] ++ children(target())

  Supervisor.start_link(children, opts)
end
```

Listing 2: `lib/say/application.ex`

Running on host

Example

```
$ env MIX_TARGET=host iex -S mix
Erlang/OTP 22 [erts-10.4.3] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-
threads:1] [hipe]

Compiling 1 file (.ex)
warning: no clause will ever match
  lib/say/application.ex:9

Generated say app
**     (Mix)     Could     not     start     application     say:     ex-
ited in: Say.Application.start(:normal, [])
    ** (EXIT) an exception was raised:
        ** (MatchError) no match of right hand side value: false
            (say) lib/say/application.ex:9: Say.Application.start/2
                        (kernel)  application_master.erl:277:  :applica-
tion_master.start_it_old/4
```

Updating the target anyway

Example

```
$ mix firmware
$ ./upload.sh
```

Starting a SSH session on target

```
Interactive Elixir (1.9.0) - press Ctrl+C to exit (type h() ENTER for help)
Toolshed imported. Run h(Toolshed) for more info
RingLogger is collecting log messages from Elixir and Linux. To see the
messages, either attach the current IEx session to the logger:

  RingLogger.attach

or print the next messages in the log:

  RingLogger.next

iex(say@nerves.local)1> Say.hello()
:world
```

Did the failure cause a crash?

The failure indeed caused a crash, but still the application was able to recover. How that internally works?

```
iex(say@nerves.local)2> RingLogger.grep("MatchError")
18:57:09.126 [info]  Application say exited: exited in: Say.Application.start(:normal, [])
    ** (EXIT) an exception was raised:
        ** (MatchError) no match of right hand side value: true
            (say) lib/say/application.ex:9: Say.Application.start/2
            (kernel) application_master.erl:277: :application_master.start_it_old/4
:ok
```

Shoehorn

Shoehorn **acts as a shim to the initialization sequence for your application's VM. Using** Shoehorn**, you can ensure that the VM will always pass initialization.**

This is especially useful when running Nerves.

nerves-project/shoehorn

Shoehorn configuration

Example

```
config :shoehorn,
init: [:nerves_runtime, :nerves_init_gadget],
app: Mix.Project.config()[:app]
```

Listing 3: `config/config.exs`

Manually reverting firmware updates

```
iex(say@nerves.local)1> Nerves.Runtime.KV.get_all()
%{
  "a.nerves_fw_application_part0_devpath" => "/dev/mmcblk0p3",
  "a.nerves_fw_application_part0_fstype" => "ext4",
  "a.nerves_fw_application_part0_target" => "/root",
  "a.nerves_fw_architecture" => "arm",
  "a.nerves_fw_author" => "The Nerves Team",
  "a.nerves_fw_description" => "",
  "a.nerves_fw_misc" => "",
  "a.nerves_fw_platform" => "rpi",
  "a.nerves_fw_product" => "say",
  "a.nerves_fw_uuid" => "c92646d3-5ed9-5725-f475-6d1d2f9fee61",
  "a.nerves_fw_vcs_identifier" => "",
  "a.nerves_fw_version" => "0.1.0",
  "nerves_fw_active" => "a",
  "nerves_fw_devpath" => "/dev/mmcblk0",
  "nerves_serial_number" => ""
}
```

Uploading changes

```
$ ./upload.sh
Path: ./_build/rpi_dev/nerves/images/say.fw
Product: say 0.1.1
UUID: 22de7e22-2313-5eb4-8786-fe1e2a7e296d
Platform: rpi

Uploading to nerves.local...
Running fwup...
fwup: Upgrading partition B
|=====================================| 100% (23.69 / 23.69) MB
Success!
Elapsed time: 16.138 s
Rebooting...
```

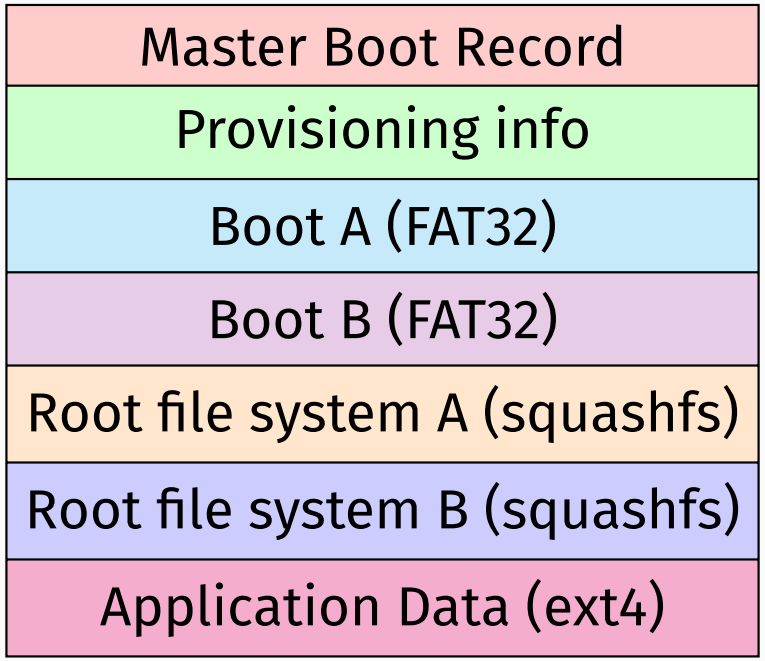
Manually reverting firmware updates

```
iex(say@nerves.local)1> Nerves.Runtime.KV.get_all()
%{
  "a.nerves_fw_application_part0_devpath" => "/dev/mmcblk0p3",
  # ...
  "b.nerves_fw_application_part0_devpath" => "/dev/mmcblk0p3",
  "b.nerves_fw_application_part0_fstype" => "ext4",
  "b.nerves_fw_application_part0_target" => "/root",
  "b.nerves_fw_architecture" => "arm",
  "b.nerves_fw_author" => "The Nerves Team",
  "b.nerves_fw_description" => "",
  "b.nerves_fw_misc" => "",
  "b.nerves_fw_platform" => "rpi",
  "b.nerves_fw_product" => "say",
  "b.nerves_fw_uuid" => "22de7e22-2313-5eb4-8786-fe1e2a7e296d",
  "b.nerves_fw_vcs_identifier" => "",
  "b.nerves_fw_version" => "0.1.1",
  "nerves_fw_active" => "b",
  "nerves_fw_devpath" => "/dev/mmcblk0",
  "nerves_serial_number" => ""
}
iex(say@nerves.local)2> Nerves.Runtime.revert()
Received disconnect from 172.31.129.157 port 22:11:
Terminated (shutdown) by supervisor
```

Manually reverting firmware updates

```
iex(say@nerves.local)1> Nerves.Runtime.KV.get_all()
%{
  "a.nerves_fw_application_part0_devpath" => "/dev/mmcblk0p3",
  "a.nerves_fw_application_part0_fstype" => "ext4",
  "a.nerves_fw_application_part0_target" => "/root",
  "a.nerves_fw_architecture" => "arm",
  "a.nerves_fw_author" => "The Nerves Team",
  "a.nerves_fw_description" => "",
  "a.nerves_fw_misc" => "",
  "a.nerves_fw_platform" => "rpi",
  "a.nerves_fw_product" => "say",
  "a.nerves_fw_uuid" => "c92646d3-5ed9-5725-f475-6d1d2f9fee61",
  "a.nerves_fw_vcs_identifier" => "",
  "a.nerves_fw_version" => "0.1.0",
  # ...
  "nerves_fw_active" => "a",
  "nerves_fw_devpath" => "/dev/mmcblk0",
  "nerves_serial_number" => ""
}
```

SD Card partitions



ring_logger

RingLogger is an in-memory ring buffer backend for the Elixir Logger with convenience methods for accessing the logs from the IEx prompt.

nerves-project/ring_logger

RingLogger use cases

- Get log messages in real-time over remote IEx sessions
- Grep and tail through log messages without setting up anything else
- Keep logs in limited resource environments
- Capture recent log events for error reports

RingLogger usage sample

Example

```
# env MIX\_TARGET=host iex --sname host1 --cookie hello -S mix
iex(host1@heimdallr)1> Logger.add_backend(:console)
{:ok, #PID<0.214.0>}
iex(host1@heimdallr)2> Logger.remove_backend(RingLogger)
:ok
iex(host1@heimdallr)3> require Logger
Logger
iex(host1@heimdallr)4> Logger.info("hello")
15:51:39.866 [info]  hello
:ok
iex(host1@heimdallr)5> Logger.add_backend(RingLogger)
{:ok, #PID<0.221.0>}
iex(host1@heimdallr)6> Logger.info("hello")
15:52:01.412 [info]  hello

:ok
```

Example

```
# env MIX\_TARGET=host iex --sname host2 --cookie hello --remsh host1
iex(host1@heimdallr)1> RingLogger.attach()
:ok

15:52:01.412 [info]  hello
```

Phoenix Demo

mobileoverlord/training_kiosk

Questions?

More information

- nerves-project.org
- embedded-elixir.com
- hexdocs.pm/nerves
- github.com/nerves-project
- github.com/nerves-hub
- elixirforum.com/tags/nerves
- elixir-slackin.herokuapp.com
- github.com/nerves-project/nerves_examples

Thanks!

Slides: speakerdeck.com/milmazz