The Plan

- Overview
- Elixir v1.3-dev
- Improving OTP
- R&D: GenStage & GenBroker
Overview
Elixir v1.0

- September/2014
- >180 contributors
- 3 books out
- First Elixirconf!
Elixir v1.1

- September/2015
- >295 contributors
- ~ 1000 packages on hex.pm
- ElixirConf US and ElixirConf EU
Elixir v1.2

- January/2016
- >384 contributors
- Since then...
Building Community With Top Talks & Helpful Hacking

Register Today!
Empire City Elixir Conference 2016

A one-day conference for curious programmers
Saturday, May 21st
New York City

BUY TICKETS
Learning resources

1 Learn Elixir
2 Screencasts
3 In-depth Resources

Our website provides a [Getting Started](#) guide to learn more about Elixir’s foundation and explore how to build projects with [Mix and OTP](#).

The Elixir Community has also produced plenty of resources to explore Elixir from different backgrounds and other perspectives. We are sure you will find a resource that follows your pace and interests.

Learn Elixir

Programming Elixir 1.2

You want to explore functional programming, but are put off by the academic feel (tell me about monads just one more time). You know you need concurrent applications, but also know these are almost impossible to get right. Meet Elixir, a functional, concurrent language built on the rock-solid Erlang VM.

Elixir's pragmatic syntax and built-in support for metaprogramming will make you productive and keep you interested for the long haul. And Programming Elixir is the introduction to Elixir. For experienced programmers, written by the creators of Elixir.

News: Elixir v1.2 released

Search...

JOIN THE COMMUNITY

- #elixir-lang on freenode IRC
- Elixir on Slack
- Elixir Forum
- elixir-talk mailing list
- @elixirlang on Twitter
- Meetups around the world
- Wiki with events, resources and talks organized by the community

IMPORTANT LINKS

- Source Code
- Issue tracker
- elixir-core mailing list (development)
- Crash course for Erlang developers

CODE EDITOR SUPPORT

- Emacs Mode
- Alchemist (Emacs Elixir Tooling)
- TextMate / Sublime Text Bundle
Elixir School

Lessons about the Elixir programming language, inspired by Twitter’s Scala School

Available in Việt ngữ, 汉语, Español, 日本語, Português and Bahasa Melayu

Your feedback and participation is encouraged!

About Elixir

“Elixir is a dynamic, functional language designed for building scalable and maintainable applications.” — elixir-lang.org

Elixir leverages the battle tested ErlangVM to build distributed and fault-tolerant systems with low-latency out of the box.

Features:

- Scalable
- Fault-tolerant
- Functional Programming
- Extensible

Share This Page
Welcome to the brand new Elixir Forum!

There's no time like the present to jump into Elixir - the functional language that's taking the world by storm 😊

As well as catering to the community's more general needs, we have a strong focus on learning, so if you’ve been curious about Elixir or are just starting out - join up - you'll be in great company!

This forum is also driven by the community. The more you participate here, the more trust you’ll earn and the more mod-type tools you’ll unlock - our way to reward regular members and your way to help give back to the community.

That's not all, we have lots of other features... so what are you waiting for? Take a sip of Elixir, sign up and join in the fun ❤️

<table>
<thead>
<tr>
<th>Topic</th>
<th>Category</th>
<th>Users</th>
<th>Replies</th>
<th>Views</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who is currently developing with Elixir at work?</td>
<td>Elixir Chat</td>
<td></td>
<td>33</td>
<td>1.1k</td>
<td>5h</td>
</tr>
<tr>
<td>Elixir Blog Posts</td>
<td>Elixir Chat</td>
<td></td>
<td>41</td>
<td>1.1k</td>
<td>6h</td>
</tr>
<tr>
<td>Elixir Koans</td>
<td>Learning Elixir</td>
<td></td>
<td>2</td>
<td>113</td>
<td>8h</td>
</tr>
<tr>
<td>The Little Elixir &amp; OTP Guidebook</td>
<td>Learning Elixir</td>
<td></td>
<td>13</td>
<td>286</td>
<td>8h</td>
</tr>
<tr>
<td>Blockchain</td>
<td>Elixir Chat</td>
<td></td>
<td>0</td>
<td>56</td>
<td>11h</td>
</tr>
<tr>
<td>Elixir regular expression editor &amp; tester</td>
<td>Elixir Chat</td>
<td></td>
<td>3</td>
<td>87</td>
<td>14h</td>
</tr>
<tr>
<td>Portland Meetup - 27 April 2016</td>
<td>Conf &amp; Meet Ups</td>
<td></td>
<td>3</td>
<td>79</td>
<td>15h</td>
</tr>
</tbody>
</table>
Using with Elixir

Simply specify your Mix dependencies as two-item tuples like `{:ecto, "~> 0.1.0"}` and Elixir will ask if you want to install Hex if you haven't already. After installed, you can run `$ mix local` to see all available Hex tasks and `$ mix help TASK` for more information about a specific task.

Using with Erlang

Download rebar3, put it in your PATH and give it executable permissions. Now you can specify Hex dependencies in your rebar.config like `{deps, [hackney]}.`

A productive web framework that does not compromise speed and maintainability.

Build APIs, HTML 5 apps & more
See our guides

HOW IS PHOENIX DIFFERENT?
Phoenix brings back the simplicity and joy in writing modern web applications by mixing tried and true technologies with a fresh breeze of functional ideas.
Get started with Phoenix

BUILDING THE NEW WEB
Create rich, interactive experiences across browsers, native mobile apps, and embedded devices with our real-time streaming technology called Channels.
Learn about channels

BATTLE-PROVEN TECHNOLOGY
Phoenix leverages the Erlang VM ability to handle millions of connections alongside Elixir's beautiful syntax and productive tooling for building fault-tolerant systems.
More about Elixir & the Erlang VM
nerves

craft bulletproof firmware in the stunningly productive elixir language

platform
Using a lean, custom cross-compiled linux, nerves boots directly to the battle hardened BEAM VM, starting your application in seconds.

framework
Most devices need to get on a network, get discovered, update firmware, and deal with I/O of various kinds. You’re not on your own.

tooling
Cross-compilation can be a total drag. Our tools make it smooth as silk. Go from "mix new" to running code on your device in minutes.

Nerves is young, but already powers rock-solid shipping industrial products! Check us out if you are a hearty experimenter and interested in a new way of creating embedded systems.

Nerves is fully open source. Nerves uses Buildroot to provide Linux, so some portions of the platform are licensed under the GPL.
Unlocking New Features in Moz Pro with a Database-Free Architecture

Moz Pro is undertaking a comprehensive overhaul of our backend architecture to improve the performance and speed of our application and to unlock significant new features in high demand by our customers. We are abandoning MySQL database storage—our current infrastructure’s decisive bottleneck—in favor of a database-free architecture and an Elixir-driven data indexing model. In this post, I’ll discuss our new architecture, the competitive and technical reasons we chose to invest in this project, and some interesting implementation challenges we faced.

Overview of Moz Pro

Moz Pro Campaigns

Rankings

275 Tracked Keywords
29 Moved up
20 Moved down

Keyword Rankings

Universal Results

Search Visibility

Export
Introducing new open-source tools for the Elixir community

The eighties aren’t typically remembered as a somber and serious decade. Yet, while the masses were listening to Wang Chung’s “Everybody have fun tonight” and hoping for a DeLorean based time machine, several engineers at a Swedish telecom company were accidentally inventing the future. They were solving a problem for telecommunications, but by a strange coincidence, the problem they were solving also turned out to be very similar to the problems faced by engineers that would see that DeLorean time machine after it completed its 30 year jump to the future.

Despite its many advantages, this magical system never really got traction, but spread like an urban legend: Spoken in hushed tones by seasoned engineers, who always heard about it third-hand. Sure, you knew a cousin who had a roommate that knew a gal that ran a couple million connections off of two servers, but you never actually believed that stuff anyway.

Well, I’m here to tell you that the stories were real. And there’s a sequel.

I am, of course, talking about Erlang, and while Erlang has had some recent
Elixir v1.3-dev
Calendar types

# Persist yesterday + 1 hour with Ecto
GoodTimes.Date.yesterday
>>> Calendar.DateTime.from_erl
>>> Calendar.DateTime.add(hours: 1)
>>> Calendar.DateTime.to_erl
>>> Ecto.DateTime.from_erl
Calendar types

- Time
- Date
- NaiveDateTime
- DateTime
Calendar types

- Common ground for interoperability
- Pending: basic conversion functions
- Support multiple calendars (ISO8601)
Calendar types

- Lau Taarnskov (Calendar)
- Paul Schoenfelder (Timex)
with (recap)

with :ok <- validate_strategy(strategy),
    :ok <- validate_restarts(max_restarts),
    :ok <- validate_seconds(max_seconds) do
  {:ok, %DynamicSupervisor{...}}
end
with (recap)

```elixir
defp validate_restarts(restart) when is_integer(restart),
  do: :ok
defp validate_restarts(_),
  do: {:error, "max_restarts must be an integer"}

defp validate_seconds(seconds) when is_integer(seconds),
  do: :ok
defp validate_seconds(_),
  do: {:error, "max_seconds must be an integer"}
```
with (recap)

with :ok <- validate_strategy(strategy),
     :ok <- validate_restarts(max_restarts),
     :ok <- validate_seconds(max_seconds) do
{:ok, state}
end

# {:ok, state} OR {:error, message}
with (recap)

res =
  with :ok <- validate_strategy(strategy),
       :ok <- validate_restarts(max_restarts),
       :ok <- validate_seconds(max_seconds)
      do
    {:ok, state}
  end

case res do
  {:ok, state} ->
    {:ok, state}
  {:error, message} ->
    {:error, String.upcase(message)}
end
with-else

with  :ok  <-  validate_strategy(strategy),
      :ok  <-  validate_restarts(max_restarts),
      :ok  <-  validate_seconds(max_seconds)  do
    {:ok,  state}
else
  {:error,  message}  ->
    {:error,  String.upcase(message)}
end
10) test maps; mixed diff (Difference)

lib/ex_unit/examples/difference.exs:58

Assertion with == failed

code: map1 == map2

lhs: %{11 => 11, 39 => 39, 34 => 34, 26 => 26, 15 => 15, 20 => 20, 17 => 17, 25 => 25,
       13 => 13, 8 => 8, 36 => 36, 7 => 7, 1 => 1, 32 => 32, 37 => 37, 35 => 35, 3 => 3,
       6 => 6, 2 => 2, 10 => 10, 9 => 9, 19 => 19, 14 => 14, 5 => 5, 18 => 18, 31 => 31,
       22 => 22, 29 => 29, 21 => 21, 27 => 27, 24 => 24, 40 => 40, 30 => 30, 23 => 23,
       28 => 28, 16 => 16, 38 => 38, 4 => 4, 12 => 12}

rhs: %{11 => 11, 39 => 39, 34 => 34, 26 => 26, 15 => 15, 20 => 20, 17 => 17, 25 => 25,
       13 => 13, 36 => 36, 1 => 1, 32 => 32, 37 => 37, 35 => 35, 3 => 3, 2 => 2, 33 => 33,
       19 => 19, 14 => 14, 18 => 18, 31 => 31, 22 => 22, 29 => 29, 21 => 21, 27 => 27,
       24 => 24, 40 => 40, 30 => 30, 23 => 32, 28 => 28, 16 => 16, 38 => 38, 4 => 4,
       12 => 12}

diff: %{23 => 232 (off by +9), 8 => 8, 7 => 7, 6 => 6, 10 => 10, 9 => 9, 5 => 5, 33 => 33, ...}

stacktrace:
lib/ex_unit/examples/difference.exs:61
Mix Archives (recap)

$ mix archive.install
   phoenixframework.org/phoenix.ez

$ mix phoenix.new new_app
Mix Escripts

$ mix escript.install
   phoenixframework.org/phoenix

$ phoenix new_app
Mix Escripts

- Added as an alternative to archives
- Run apart from Mix
- Do not depend on the Elixir version installed locally
MIX_DEBUG=1

~/OSS/mime[master]$
MIX_DEBUG=1 mix compile
** Running mix loadconfig (inside MIME.Mixfile)
** Running mix compile (inside MIME.Mixfile)
** Running mix loadpaths (inside MIME.Mixfile)
** Running mix depts.check (inside MIME.Mixfile)
** Running mix archive.check (inside MIME.Mixfile)
** Running mix compile.all (inside MIME.Mixfile)
** Running mix compile.yecc (inside MIME.Mixfile)
** Running mix compile.leex (inside MIME.Mixfile)
** Running mix compile.erlang (inside MIME.Mixfile)
** Running mix compile.elixir (inside MIME.Mixfile)
Compiled lib/mime.ex
** Running mix compile.app (inside MIME.Mixfile)
Generated mime app
** Running mix compile.protocols (inside MIME.Mixfile)
Consolidated List.Cchars
Consolidated Collectable
Consolidated String.Cchars
Consolidated Enumerable
Consolidated IEx.Info
mix *.tree

$ mix app.tree
$ mix deps.tree
Contributions?

$ mix app.tree --dot
$ mix deps.tree --dot
What's coming in Elixir 1.3

I recently gave a talk about Elixir 1.3 in Tokyo, and spoke about the changes, new features, improvements and all the awesome stuff coming in Elixir 1.3. I decided to write this as a blog post, with a little more details, and some links for those who want to check in more details.

Here is a short table of contents:

- Deprecation of imperative assignment
- with on steroids
- Calendar datatypes
- escript installation related tasks
- ExUnit diff
- make compiler addition
- Changes in defdelegate
- Process.sleep addition
- Support of OTP optionalcallback

Photo and illustration by Ai Miyuki
Improving OTP
Improving OTP

- Simple one for one supervisors
- GenEvent
Supervision strategies

- one_for_one
- one_for_all
- rest_for_all
- simple_one_for_one
Simple one for one supervisor

- Used for spawning children dynamically
- init/1 must return a single child
- Many APIs expect different values
- etc
DynamicSupervisor

- Implements simple_one_for_one (simply called “one_for_one”)
- Push: limits max children?
- Pull: better back-pressure?
GenEvent

Manager

Handler 1
Handler 2
Handler 3
Handler 4
GenEvent

- Manager broadcasts events to handlers
- Handlers do not exploit concurrency
- Awkward error semantics:
  Handlers are removed and not re-added
Gen???

- Handlers are different processes
- Supports multiple “strategies"
- Provides back-pressure
R&D: GenStage & GenBroker
Collections

widgets

| > Enum.filter(fn b -> b.color == :red end)
| > Enum.map(fn b -> {b.title, b.height} end)
| > Enum.into(%{})

enumerable

collectable
**Collections**

```
CSV.parse(path)
|> Enum.filter(fn b -> b.color == :red end)
|> Enum.map(fn b -> {b.title, b.height} end)
|> Enum.into(IO.stream(:stdio, :inspect))
```
Collections + Laziness

CSV.parse(path)

|> Stream.filter(fn b -> b.color == :red end)
|> Stream.map(fn b -> {b.title, b.height} end)
|> Stream.into(IO.stream(:stdio, :inspect))
|> Stream.run()
Pipeline parallelism

```javascript
CSV.parse(path)
 |> Stream.async()
 |> Stream.filter(fn b -> b.color == :red end)
 |> Stream.map(fn b -> {b.title, b.height} end)
 |> Stream.into(IO.stream(:stdio, :inspect))
 |> Stream.run()
```
CSV.parse(path)
| ▸ ...  
| ▸ Stream.async()  
| ▸ ...  
| ▸ Stream.async()  
| ▸ ...  
| ▸ Stream.async()  
| ▸ ...  
| ▸ Stream.run()
Pipeline parallelism

CSV → async → async → async → run
GenRouter
GenRouter

CSV → router → router → router → run
GenRouter

Router
GenRouter

- Couples transformations with event balancing
- Event balancing incurs extra copying
GenStage

CSV → stage → stage → stage → run
GenStage: Demand-driven

1. consumer subscribes to producer
2. consumer sends demand
3. producer sends events
GenStage: Demand-driven

A → B
Asks 10
Sends max 10

B → C
Asks 10
Sends max 10
• It is a message contract
• It pushes back-pressure to the boundary
• GenStage is one impl of this contract
• Inspired by Akka Streams
GenBroker
Broker strategy: broadcast
Broker strategy: broadcast

Diagram:
- A node labeled 'prod'
- Arrows pointing to four other nodes labeled '1'

Broker strategy: broadcast
Broker strategy: broadcast
Broker strategy: demand size

1, 2, 3, 4, 5

prod

1
2, 5
3
4
GenStage & GenBroker

- Solves the issues from GenRouter
- Stages are separate processes
- Provides back-pressure
- Supports multiple broker strategies
Examples

RabbitMQ → Filter → Dynamic Supervisor

1 2 ... n
Examples
Improving OTP

- Demand-driven supervisors
  Children are spawned on demand
- GenEvent -> GenBroker
  Solves all issues we had with GenEvent
The path forward

- Finish GenStage
- Finish GenBroker
- Finish DynamicSupervisor
- https://github.com/elixir-lang/gen_broker
plataformatec
consulting and software engineering