Quick Start of SBT

`The **interactive** build tool`

`**Define** your tasks in Scala`

`Run them in **parallel** from sbt's interactive shell`

Batch Mode

```
$ sbt clean compile "testOnly TestA TestB"
```

Note: Running in batch mode requires JVM spinup and JIT each time, so **your build will run much slower**. For day-to-day coding, we recommend using the sbt shell or Continuous build and test feature described below.

Interactive Sample

- projects / project
- reload
- inspect / inspect tree
- < task > / show < task >
- testOnly / testQuick

Command line Reference

Task Engine - Task

The common commands you can run are almost all tasks!

compile , test , sources

```
sbt:subModule> inspect sources
[info] Task: scala.collection.Seq[java.io.File]
[info] Description:
[info]  All sources, both managed and unmanaged.
[info] Provided by:
[info]  ProjectRef(uri("file:/Users/yoga/Documents/workspace/subModule/"), "submodule") / Compile / sources
[info] Defined at:
[info]  (sbt.Defaults.sourceConfigPaths) Defaults.scala:390
[info] Dependencies:
[info]  Compile / managedSources
[info]  Compile / unmanagedSources
[info] Delegates:
[info]  Compile / sources
[info]  sources
[info]  ThisBuild / Compile / sources
[info]  ThisBuild / sources
[info]  Zero / Compile / sources
[info]  Global / sources
[info] Related:
[info]  mb / Test / sources
[info]  Test / sources
[info]  ma / Test / sources
[info]  ma / Compile / sources
[info]  mb / Compile / sources
sbt:subModule>
```

Task Engine -Setting

The common commands you can run are almost all tasks!

name, scalaSource, javaSource ....

```
sbt:subModule> inspect name
[info] Setting: java.lang.String = subModule
[info] Description:
[info]  Project name.
[info] Provided by:
[info]  ProjectRef(uri("file:/Users/yoga/Documents/workspace/subModule/"), "submodule") / name
[info] Defined at:
[info]  /Users/yoga/Documents/workspace/subModule/build.sbt:2
[info] Reverse dependencies:
[info]  onLoadMessage
[info]  normalizedName
[info]  projectInfo
[info]  description
[info] Delegates:
[info]  name
[info]  ThisBuild / name
[info]  Global / name
[info] Related:
[info]  mb / name
[info]  ma / name
```

TaskEngine - Setting VS Task

- Settings are evaluated at project load time
- Tasks are executed on demand, often in response to a command from the user
- Settings can only depend on Settings
- Tasks can depend on Settings and Tasks

Depend

.value

```
scalaSource in Compile := baseDirectory.value / "src",
javaSource in Compile := (scalaSource in Compile).value,
```

Tasks

Tasks/Settins

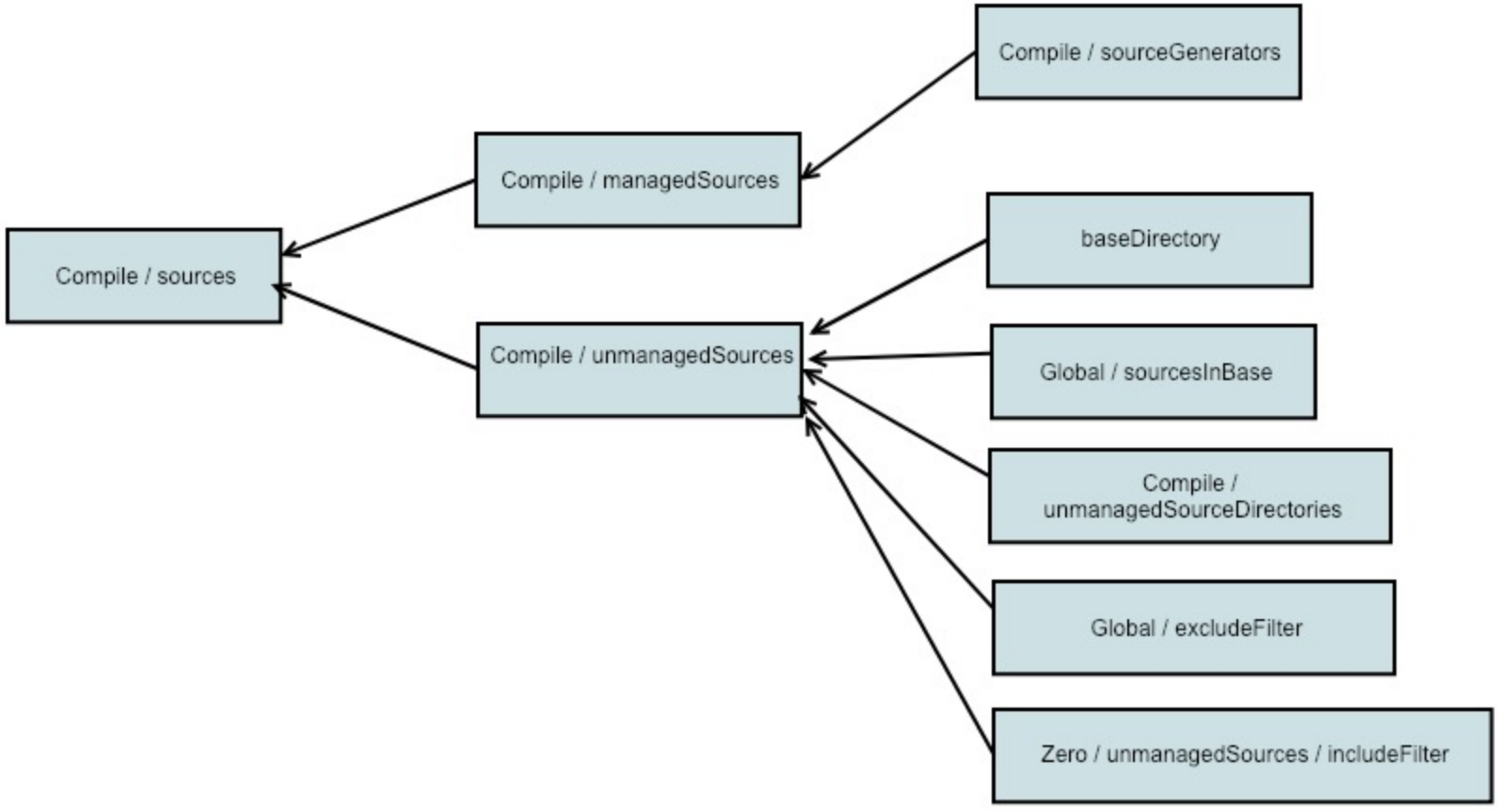
TaskEngine - DAG

Inspect tree sources

```
sbt:subModule> inspect tree sources
[info] Compile / sources = Task[scala.collection.Seq[java.io.File]]
[info]   +-Compile / managedSources = Task[scala.collection.Seq[java.io.File]]
[info]   | +-Compile / sourceGenerators = List()
[info]   |
[info]   +-Compile / unmanagedSources = Task[scala.collection.Seq[java.io.File]]
[info]     +-baseDirectory =
[info]     +-Global / sourcesInBase = true
[info]     +-Compile / unmanagedSourceDirectories = List(/Users/yoga/Documents/workspace/subModule/src/main/scala-2.11, /Users/..
[info]     | +-Global / crossPaths = true
[info]     | +-Compile / javaSource = src/main/java
[info]     | | +-Compile / sourceDirectory = src/main
[info]     | |   +-Compile / configuration = compile
[info]     | |   +-sourceDirectory = src
[info]     | |     +-baseDirectory =
[info]     | |       +-thisProject = Project(id submodule, base: /Users/yoga/Documents/workspace/subModule, aggregate: List(Pro..
[info]     | |
[info]     | +-pluginCrossBuild / sbtBinaryVersion = 1.0
[info]     | +-Global / sbtPlugin = false
[info]     | +-scalaBinaryVersion = 2.11
[info]     | +-Compile / scalaSource = src/main/scala
[info]     |   +-Compile / sourceDirectory = src/main
[info]     |     +-Compile / configuration = compile
[info]     |     +-sourceDirectory = src
[info]     |       +-baseDirectory =
[info]     |         +-thisProject = Project(id submodule, base: /Users/yoga/Documents/workspace/subModule, aggregate: List(Pro..
[info]     |
[info]     +-Global / excludeFilter = sbt.io.HiddenFileFilter$@6c167296
[info]     +-Zero / unmanagedSources / includeFilter = SimpleFileFilter(SimpleFilter(PatternFilter(.*\Q.java\E) | PatternFilter..
[info]
sbt:subModule>
```

TaskEngine - DAG

Inspect tree sources -> Change a view to see



TaksEngine - Scope

a full scope in sbt is formed by a **tuple** of a subproject, a configuration, and a task value:

```
Provided by:
ProjectRef(uri("file:/Users/yoga/Documents/workspace/subM
Compile /
sources
```

```
projA / Compile / console / scalacOptions
```

Usage : you want different task result in different configuration, e.g different source files for Compile and Test

Look up: Delegates

TaksEngine - Scope

Let's define a task / setting of ourselves

```
//define a task Key
val taskSample = taskKey[String]("task sample")
// give the task an implemention, we can call it an entry
taskSample := {
  println("this in taskSample, can be executed multiple t
  "Content for TaskSample"
}

// define a setting Key
val settingSample = settingKey[String]("setting sample")
// give the task an implemention
settingSample := {
  println("this is in settingSample, should be executed o
  "Content for SettingSample"
}
```

TaksEngine - Scope

another task with name "**taskSample**" but **in different scope**

```
ThisBuild / settingSample := "ThisBuild: Content for Sett

val ma = (project in file("Ma")).settings(
  scalaSource in Compile := baseDirectory.value / "src",
  javaSource in Compile := (scalaSource in Compile).value
  taskSample in Compile := {
    println("MA: this in taskSample, can be executed mult
    "MA: Content for TaskSample"
  },
  name := "MA"
)

val mb = (project in file("Mb")).settings(
  scalaSource in Compile := baseDirectory.value / "src",
  javaSource in Compile := (scalaSource in Compile).value
  settingSample in Compile := {
    println("MB: this is in settingSample, should be execu
    "MB: Content for SettingSample"
  },
  name := "MB"
)
```

SBT- DSL

It's not just Scala! It's built on Scala.

Three things I had to learn about Scala before it made sense

How do we define dependencies in mvn?

```
<dependency>
    <groupId>net.vz.mongodb.jackson</groupId>
    <artifactId>mongo-jackson-mapper</artifactId>
    <version>1.4.1</version>
</dependency>
```

SBT- DSL

How can we describe a dependency in a scala way?

```
def groupId(groupId: String) = new GroupId(groupId)

class GroupId(val groupId: String) {
  def artifact(artifactId: String) = new Artifact(groupId
}

class Artifact(val groupId: String, val artifactId: Strin
  def version(version: String) = new VersionedArtifact(gr
}

class VersionedArtifact(val groupId: String, val artifact
}
```

Then we got

```
groupId("net.vz.mongodb.jackson")
    .artifact("mongo-jackson-mapper")
    .version("1.4.1")
```

SBT- DSL

Scala methods can be made of **operator characters**

```
def groupId(groupId: String) = new GroupId(groupId)

class GroupId(val groupId: String) {
  def %(artifactId: String) = new Artifact(groupId, artif
}

class Artifact(val groupId: String, val artifactId: Strin
  def %(version: String) = new VersionedArtifact(groupId,
}
class VersionedArtifact(val groupId: String, val artifact
```

Then

```
groupId("net.vz.mongodb.jackson").%("mongo-jackson-mapper"
```

Omit the `.`

```
groupId("net.vz.mongodb.jackson") % ("mongo-jackson-mappe
```

SBT- DSL

Scala lets you **implicitly convert** any type to any other type

```
implicit def groupId(groupId: String) = new GroupId(group
```

Then when we write down below

```
"net.vz.mongodb.jackson" % "mongo-jackson-mapper" % "1.4.
```

It's equal to

```
"net.vz.mongodb.jackson".artifact("mongo-jackson-mapper")
```

SBT- DSL

implicitly convert

```
"net.vz.mongodb.jackson".artifact("mongo-jackson-mapper")
```

> What it looks like is that the `String` class has a method called `artifact`.
> But we know it doesn't, and the Scala compiler knows it doesn't.
> When it sees that, it says "are there any implicit methods available that accept a `String` as an argument, and return a type that has an `artifact` method?"
> And so it finds the `groupId` method, which matches that criteria, and converts the code to this:

```
groupId("net.vz.mongodb.jackson").artifact("mongo-jackson-
```

SBT- DSL

Theres are all **functions** with return type are **Def.Setting[T]** or **Seq[Def.Setting[T]]**

:= ++ ++= in

Belows are **expressions**! Not **statements**!

```
name := "subModule"
```

This is equal to

```
name.setEntry("subModule")
```

SBT- DSL

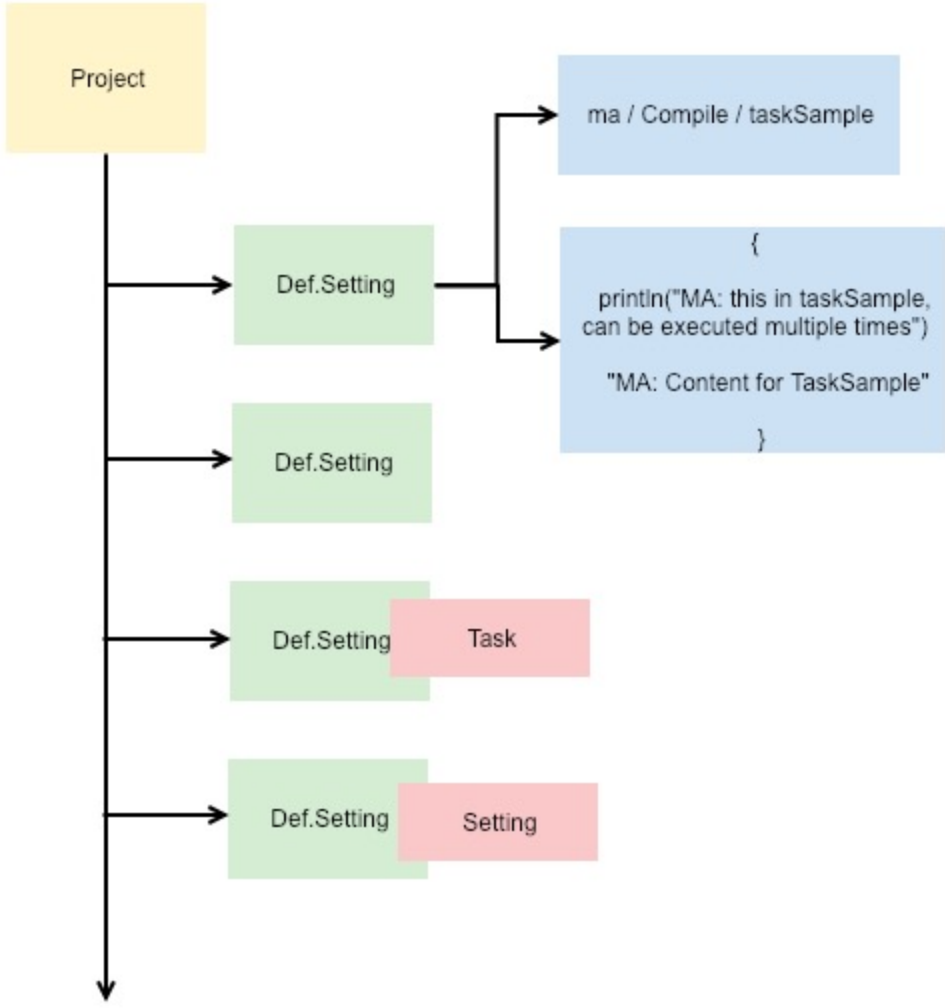
```
name := "subModule"
```

1. name is an instance of Type **SettingKey**
2. calls the **function :=**
3. with **String parameter** "submodule"
4. returns a **Setting**

Q: Look back on the build.sbt, try to explain what happened for the build.sbt you defined?

```
//this is a expression!! not a statement
//this := returns Setting[Task], and the return setting i
//OH MY GOD!!!!ddddddddddddde
```

SBT-Project


Project
Def.Setting
ma / Compile / taskSample
{
println("MA: this in taskSample, can be executed multiple times")
"MA: Content for TaskSample"
}
Def.Setting
Def.Setting
Task
Def.Setting
Setting

SBT- User Defined

Project is defined by multiple Def.Settings, this settings defined

```
1. what tasks(including setting-task) do we support in th
2. what's unique name for the tasks - the scopes
3. what's the dependencies between the tasks
4. how will the tasks be implemented
```

Statement VS Expression

```
val settingSample
```

```
:= ++ ++= ...
should return  Def.Setting[T] or Seq[Def.Setting[T]]
```

Aggerate VS Depend

aggregate:

- trigger all the tasks of the project you aggregate with

depend:

- trigger the compile of the project you depend on
- add the compile classes to your class path

Why does it fail?

In Project/CustomeSettings.scala

```
import sbt.{Setting, taskKey, settingKey}

object CustomeSettings {

  lazy val currentOs = settingKey[String]("The version of

  //choose the protobuf path according to the os
  lazy val printCurrentOS = taskKey[String]("print the cu

  currentOs := sys.props("os.name")

  printCurrentOS := {
    val os = currentOs.value
    println(s"current build on os:${os}")
    os
  }

}
```