Building SBT Plugins

Mads Hartmann Jensen
@mads_hartmann
http://mads379.github.com/

Code shown in presentation:
https://github.com/mads379/sbt-plugin-examples

Agenda

The Simple Steps

- Setting up your build definition
- Implementing your plugin
- Running it

In Your build.sbt file

```
sbtPlugin := true

name := "example-plugin"

organization := "org.example"
```

Two ways to do it

- "A plugin extends the build definition, most commonly by adding new settings"
- Will show
  - A plugin that provides a command
  - A plugin that provides some settings
  - A plugin with tab-completion

Command Plugin

- For when you don’t need customization

Implementing your plugin

Command Plugin

```
import sbt._
import Keys._

object CommandPlugin extends Plugin {

  override lazy val settings = Seq(commands += myCommand)

  lazy val myCommand =
    Command.command("hello") { (state: State) =>
      println("Hi there!")
      state
    }
}
```

Command Plugin

- In `~/.sbt/plugins/build.sbt` or
  `<project>/project/build.sbt`

```
addSbtPlugin("com.sidewayscoding" % "settings-plugin" % "0.1")
```

Command Plugin

- For local development (trail/error) create a project with the build definition:

  `<project>/project/project/build.scala`

```
import sbt._
import Keys._

object Playground extends Build {

  val commandPlugin = RootProject(file("../../command-plugin"))

  lazy val root = Project(id = "playground", base = file("."))
                    .dependsOn(commandPlugin)
}
```

Implementing your plugin

Command Plugin

Demo

Settings Plugin

- Useful when your plugin is customizable

Settings Plugin

```
import sbt._

object SettingsPlugin extends Plugin {

  val newTask = TaskKey[Unit]("new-task")
  val newSetting = SettingKey[String]("new-setting")

  val newSettings = Seq(
    newSetting := "test",
    newTask <<= newSetting map { str => println(str) }
  )

}
```

Settings: Using it

- In `~/.sbt/plugins/build.sbt` or
  `<project>/project/build.sbt`

```
addSbtPlugin("com.sidewayscoding" % "settings-plugin" % "0.1")
```

- In `<project>/build.sbt`

```
seq( SettingsPlugin.newSettings : _*)

newSetting := "light"
```

Settings: Using it

Demo

Implementing your plugin

Tab-completion

- Parsing input and providing tab-completions through Parser Combinators

Tab-completion

```
import sbt._
import Keys._
import Defaults._
import complete.DefaultParsers._
import complete.{ Parser }

object ParserPlugin extends Plugin {

  override lazy val settings = Seq(commands += cmd)

  lazy val cmd = Command("parserCmd")(_ => parser)(action _)

  type parseResult = ...

  lazy val parser: Parser[parseResult] = ...

  def action(st: State, parsed: parseResult): State = ...

}
```

Implementing your plugin

Tab-completion

Demo