

|  |
| --- |
|  |

**SBT (Scala Build Tool)**

**Guidelines and Best Practices**

# Table of Contents

[Table of Contents 2](#_Toc423524654)

[1. Introduction 3](#_Toc423524655)

[2. COnfiguration Guidelines FOr Maven 3](#_Toc423524656)

[3. USer Guidelines FOr user for maven 3](#_Toc423524657)

[4. Gragle guidelnes i3](#_Toc423524658)

# 

# Introduction

SBT stands for 'Scala Build Tool' is an open source tool for building Scala and Java projects, similar to Java's Maven or Ant.

Its main features are:

* Native support for compiling Scala code and integrating with many Scala test/code coverage/static code analysis etc. frameworks
* Dependency management using Ivy (which supports Maven-format repositories)
* Continuous compilation, testing, and deployment
* Integration with the Scala interpreter for rapid iteration and debugging
* Support for mixed Java/Scala projects

When continuous compilation mode is entered, the Scala compiler is only once instantiated which eliminates subsequent startup costs, and source file changes are tracked so that only affected dependencies are recompiled.

The interactive console allows modifying build settings on the fly and entering the Scala REPL along with all class files of the project.

# Configuration Files

This section lists various configuration files require for project build.

**build.sbt** : In SBT, project-specific properties, such as library dependencies, Scala version and so on, which are required for a successful build are declared in the build definition. The .sbt file should be located in the base directory and is generally named build.sbt.

Each key-value pair is a build property. So, we could say that a build definition is a list of properties.



**plugins.sbt :** A plugin extends the build definition, most commonly by adding new settings. The new settings could be new tasks. For example, a plugin could add a codeCoverage task which would generate a test coverage report. There are various plugins are configured in following plugins.sbt to enable the integration with different test, test, database migration, automation testing etc. frameworks.



**build.properties :** You can force a particular version of sbt by creating a file build.properties.



# Multiproject build

A build definition that consists of multiple project configurations is termed as a multiproject build. These are extremely useful when your project is a combination of two or more modules. If they depend on one another, you could also specify the dependencies so that whenever a change is made to one project, it is reflected in the projects that depend on it.

Let’s understand how to configure and work with multiproject build.

Create following project structure in your Eclipse:



At the end of project structure creation you’ll have following list of directory and package structure in your IDE :

MultiProjectSBT – Name of our Base Project

subproject1 – Name of sub-module 1

subproject2 – Name of sub-module 2

Source packages :

src/main/scala (under MultiProjectSBT)

subproject1/src/main/scala (under subproject1)

subproject2/src/main/scala (under subproject2)

Packages :

com.main.test (under MultiProjectSBT)

com.sp1.test (under subproject1)

com.sp2.test (under subproject2)

Base project’s build.sbt is available directly under MultiProjectSBT directory whereas every submodule have their independent build.sbt.

Source code :



Copy HelloMain.scala into com.main.test package of MultiProjectSBT

Copy Human.scala, Friend.scala, Animal.scala into com.sp1.test package of subproject1

Copy FriendTraitClient.scala into com.sp2.test package of subproject2



SBT configuration files :

Copy and paste following build.sbt as seen in below image:





We have configured two module i.e. sp1 and sp2 into our parent build.sbt (i.e. the one under MultiProjectSBT directory) . Here ‘dependsOn’ configuration describe that sp2 module is dependent on sp1, It means you can’t build sp2 without building sp1 module.



Finally we should have following build.properties into MultiProject’s project folder:





All the source codes and configuration files are now at it place, Let’s begin building the project.

**General build procedure**

Open command prompt and change to directory (e.g C:\Scala\_SBT\MultiProjectSBT), Type following commands :

*C:\Scala\_SBT\MultiProjectSBT> sbt*

*>clean*

*>compile*

*>run*



To list all the projects in given build, You can type following command on sbt interactive shell

>projects



If you can able to execute above command without any error that means your project is configured properly.

Let’s package the build by typing clean, compile, package in sequence at the sbt interactive shell as can be seen in following image.



After successful packaging, Check the project structure in the eclipse IDE(see below screen). You should have three project artifacts (jar files) created under your project structure.



subproject1\_2.11-1.0.jar (under subproject1 directory)

subproject2\_2.11-1.0.jar (under subproject2 directory)

multiprojectsbt\_2.10-0.1-SNAPSHOT.jar (under project base directory)

In order to understand the dependency of sp2 module on sp1, Delete ‘dependsOn(sp1)’ and try compiling the project, You should get error as seen below.



It shows, sp2 must have an access to all the scala class dependency in order to build correctly.

# USer Guidelines FOr user for maven

Following are the guidelines for Lead:

1. Details ………….

# Gragle guidelines

Following are the guidelines for Confluence User and Administrator: