

|  |
| --- |
|  |

**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.

# Installation

Please refer SBT official site ( <http://www.scala-sbt.org/0.13/tutorial/Setup.html> ) for details about how to install SBT on Windows/Mac/Linux etc. OS.

# 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.



# CoMMON COMMANDS

|  |  |
| --- | --- |
| COMMANDS | DESCRIPTION |
| clean | Deletes all generated files (in the target directory). |
| compile | Compiles the main sources (in src/main/scala andsrc/main/java directories). |
| test | Compiles and runs all tests. |
| run | Runs the main class for the project in the same virtual machine as sbt |
| package | Creates a jar file containing the files in src/main/resources and the classes compiled from src/main/scala and src/main/java. |
| publish | It will publish your generated artifacts to configured repository e.g. Ivy or Maven. |

# case study : hellosBT

This section guides you how to create SBT build of simple scala project. We’ll also see various command described in above section e.g. clean, compile, test, run, package, publish etc.

**Prerequisites :**

* It is also assumed that you’ve installed Scala as well, Please refer (<http://www.scala-lang.org/download/>) to download and install latest version of Scala.
* Please make sure you have SCALA\_HOME is created as user variable in System’s Environment Variable entries.
* It is assumed that you have installed SBT on your windows system. Please refer ( <http://www.scala-sbt.org/0.13/tutorial/Setup.html> ) for more details about how to install SBT on Windows.
* Please make sure you’ve SBT\_HOME is created automatically inside ‘system variable’ of your system’s Environment Variables after installation of SBT.
* You should have two directories i.e. ‘.sbt’ and ‘.Ivy’ created automatically under C:\Users\<USER\_HOME>

**Assumptions :**

This case study assumes that you’ve installed following version of SBT and Scala.

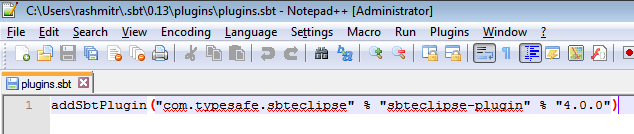
* Scala (version 2.11.4)
* SBT (version 0.13.8)

Let’s start preparing the required configuration and project structure for HelloSBT project.

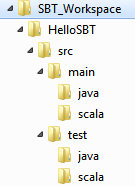
1. **Eclipse plugin :** Let’s begin our case study by adding Eclipse plugin for sbt. This plugin is used for converting our sbt project into eclipse project so we can import and work with our project using Eclipse IDE.

Create plugins.sbt under C:\Users\rashmitr\.sbt\0.13\plugins directory. Copy and paste following content into plugins.sbt

*addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "4.0.0")*



1. **Project structure creation :** 
   1. Create following project structure in your C:\



* 1. Create a file build.sbt in C:\SBT\_Workspace\HelloSBT and copy/paste following content into it.

*//Organization*

*organization := "com.cybage.hellosbt"*

*//Version of your project*

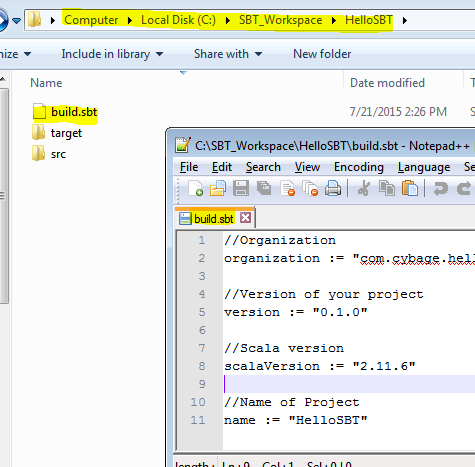
*version := "0.1.0"*

*//Scala version*

*scalaVersion := "2.11.6"*

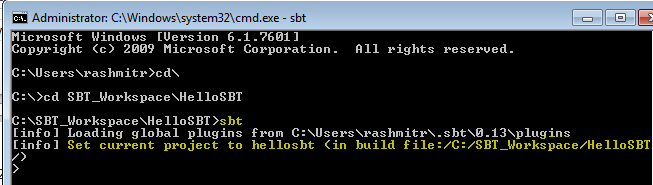
*//Name of Project*

*name := "HelloSBT"*

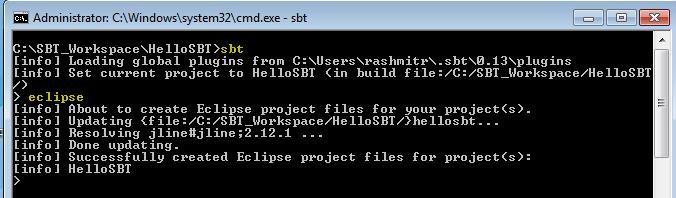


* 1. Open command prompt, Change to HelloSBT directory and type command ‘sbt’ as shown below. At the end of command, HelloSBT will become a SBT project and gives you a shell prompt (i.e. > ) where you can type various build commands.

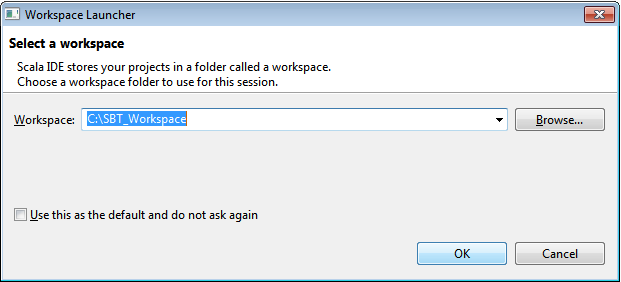
By making sbt project, It’ll allows you to start interactive command execution, incremental compilation and continuous compilation etc feature in you project which we’ll be looking into subsequent sections.

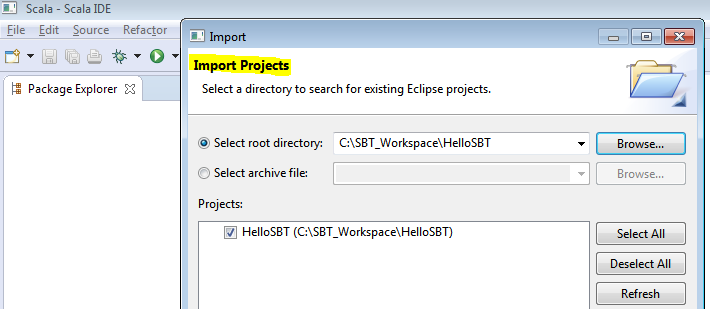


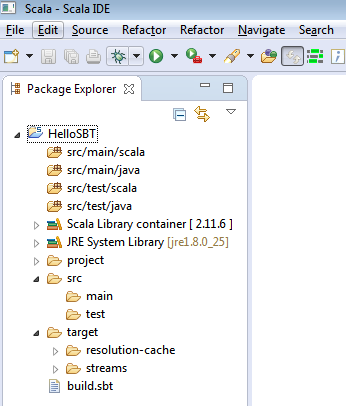
* 1. Type *eclipse* on command prompt as show in image. This will make your sbt project compatible to be imported in Eclipse IDE.



* 1. Now, Import the HelloSBT project into Scala Eclipse IDE as shown below.





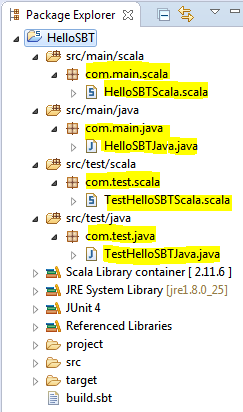


At the time of writing this document, Unlike Maven/Ant, there is no in-built sbt plugin available in Eclipse. However sbt allowed us to build Eclipse project structure which we just built and imported here. So far so good, Scala developer can do the development of source code now on Scala Eclipse IDE.

Let move our focus on writing some sample java and scala programs as well as java and scala test cases which we’ll use for building our project.

**Package Creation :**

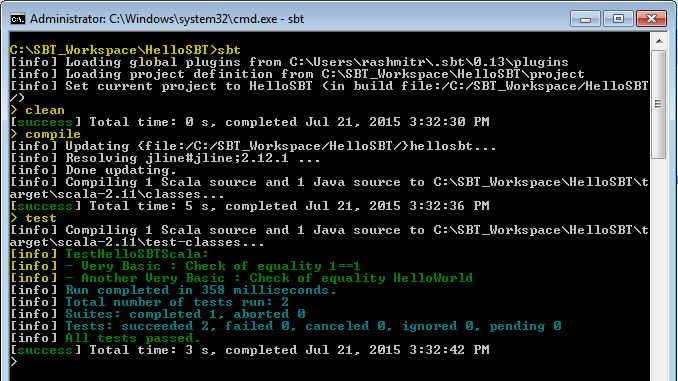
Using Eclipse, create the packages e.g. com.main.scala, com.main.java, com.test.scala, com.test.scala under their respective src and test folder. Please refer following screen to understand the location of each package(highlighted). At the end of package creation, Your HelloSBT project structure should look exactly the same as in below screen.



**Source Code (Scala, Java and their test cases) :**

Now we’ll create list of scala and java source code and test cases (highlighted) and put it in the respective folder as visible in above screen.

|  |  |  |
| --- | --- | --- |
| ClassType | ClassName | SourceCode |
| Scala main class | HelloSBTScala.scala | **package** com.main.scala  **class** HelloSBTScala(**val** name:*String*){  **def** hello() : *String* = "Hello "+name  }  **object** HelloSBTScala {  **def** main(args: Array[*String*]): Unit = {  **val** obj = **new** HelloSBTScala("World")  println(obj.hello())  }  } |
| Java main class | HelloSBTJava.java | **package** com.main.java;  **public** **class** HelloSBTJava {  String name;  **public** HelloSBTJava() {  }  **public** HelloSBTJava(String name){  **this**.name =name;  }  **public** String hello(){  **return** "Hello "+name;  }  **public** **static** **void** main(String args[]){  HelloSBTJava obj = **new** HelloSBTJava("Cybage");  String msg = obj.hello();  System.***out***.println(msg);  }  } |
| Scala test case | TestHelloSBTScala.scala | **package** scalatest  **import** org.scalatest.FunSuite  **class** TestHelloSBTScala **extends** FunSuite {  test("Very Basic : Check of equality 1==1") {  assert(1 == 1)  }  test("Another Very Basic : Check of equality HelloWorld") {  assert("Hello World" == "Hello World")  }  } |
| Java test case | TestHelloSBTJava.java | **package** com.test.java;  **import** **static** org.junit.Assert.\*;  **import** org.junit.Test;  **import** com.main.java.HelloSBTJava;  **public** **class** TestHelloSBTJava {  @Test  **public** **void** testHello() {  HelloSBTJava obj = **new** HelloSBTJava("Cybage");  *assertEquals*("Hello Cybage", obj.hello());  }  } |



# Configuration Files

# Configuration Files

# Configuration Files

# Configuration Files

# Configuration Files

# Configuration Files

# Configuration Files

# 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: