IBM Z DevOps Acceleration Program

Setup your Groovy Development Environment in IDZ v15 to develop IBM Dependency Based Build groovy scripts



brice@ibm.com

Jean-Yves Baudy

baudy.jy@fr.ibm.com

Dennis Behm

dennis.behm@de.ibm.com

Baljeet Singh

baljeet.s@ibm.com

Abstract

A step-by-step guide to setup your IDE to leverage DBBs JavaDoc



Table of Contents

1	- 1	ntroduct	ion	3
2			on / Preparation	
	2.1		Groovy Development Tools	
	2.2		ve JavaDoc and dbb-core toolkit	
3	Use groovy tooling			
	3.1	Config	ure your zAppBuild project	6
	3	3.1.1	Set Eclipse project nature in .project	б
	3	3.1.2	Configure JavaBuild path	
	3.2	Levera	ge the Groovy Tooling	9
	3	3.2.1	Script navigation	9
	3	3.2.2	Displaying JavaDoc within the editor	. 10
4	9	Summary	′	12
5	A	Appendix	a – Alternative IDE options for Groovy Programming languages	13
	5.1	Eclipse	IDE for Java Developers	. 13

1 Introduction

While IBM Developer for z/OS (IDZ) Version 15 is built upon the IBM Aqua platform 3.2 which is based on Eclipse 4.8, you can extend your installation with the Groovy Development Tools¹ to increase your comfort in developing and maintaining your IBM Dependency Based Build groovy script.

One aspect is to have the JavaDoc next to your code instead of accessing it just via the browser at IBMs knowledge center.²

The purpose of this document is to outline the steps to configure your IDE to leverage the standard Eclipse features like Code completion, syntax highlighting and JavaDoc.

The prerequisites to enable IDZ to support groovy are:

- IBM Developer for z/OS version 15.0.x
- IBM Dependency Based Build 1.0.8 or later, which ships the JavaDoc as part of the DBB toolkit installation

If you are developing / maintaining groovy scripts in your company, this is what you are looking for.

<u>Please note:</u> Groovy Development Tools cannot be installed in **IDZ 16**, because the open-source Groovy Development Environment project decided to switch to a different installation procedure using a JDT patch method, that relies on the exact match of the Eclipse IDE build. The way to get it to work, requires to rebuild the GDE.

An alternative way is described in the Appendix of this document!

¹ https://marketplace.eclipse.org/content/groovy-development-tools

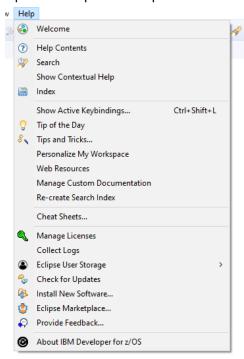
² https://www.ibm.com/support/knowledgecenter/SS6T76 1.0.9/pr intro.html

2 Installation / Preparation

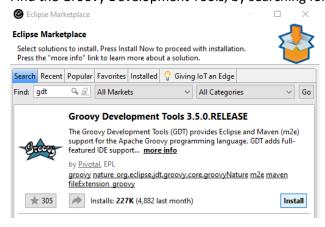
This section walks you through the steps to install the necessary software parts as well how to obtain the Dependency Based Build JavaDoc and toolkit API.

2.1 Install Groovy Development Tools

• Open the Eclipse Marketplace



Find the Groovy Development Tools, by searching for 'GDT'



Accept the license agreements and Install the plugin into your IDZ environment.

2.2 Retrieve JavaDoc and dbb-core toolkit

 Please use IDZ to connect to your mainframe environment and navigate to the location of the dbb toolkit in z/OS UNIX System Services. Within the folder archive, you will find the java doc in dbbztoolkit-javadoc-<version>.zip



Use the Remote Systems view to copy the dbb-ztoolkit-javadoc-1.1.0.zip as well as the lib/dbb-core_x.x.x.jar to your local drive.

Let's use C:/Users/ibmuser/dbb/groovy-development-environment/ as the target directory for it.

Extract the dbb-ztoolkit-javadoc-1.1.0.zip using your preferred archive manager

3 Use groovy tooling

We are assuming, that you have already cloned the git project with the build scripts.

3.1 Configure your zAppBuild project

This section can also be applied for your own dbb build implementation project.

3.1.1 Set Eclipse project nature in .project

• Locate the .project file of zAppBuild and add the following two new project natures to it.

```
<nature>org.eclipse.jdt.core.javanature</nature>
<nature>org.eclipse.jdt.groovy.core.groovyNature</nature>
```

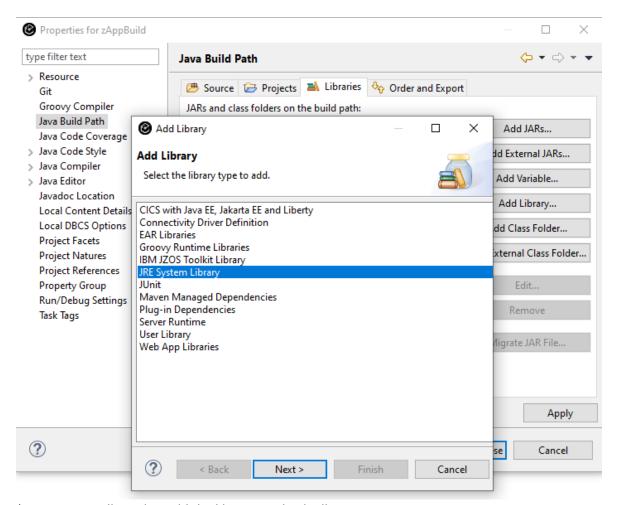
It should look similar to:

```
📄 .project 🛭
  <?xml version="1.0" encoding="UTF-8"?>
  ctDescription>
      <name>zAppBuild</name>
      <comment></comment>
      cts>
      </projects>
      <buildSpec>
      </buildSpec>
      <natures>
          <nature>com.ibm.ftt.ui.views.project.navigator.local</nature>
          <nature>com.ibm.ftt.dbbz.integration.dbbzprojectnature
          <nature>org.eclipse.jdt.core.javanature</nature>
          <nature>org.eclipse.jdt.groovy.core.groovyNature</nature>
      </natures>
  </projectDescription>
```

3.1.2 Configure JavaBuild path

From the Project Explorer, open the properties for the zAppBuild project and select "Java Build
Options" from the left-hand pane. In the right-hand pane, select the "Libraries" tab and click the Add
Library button.

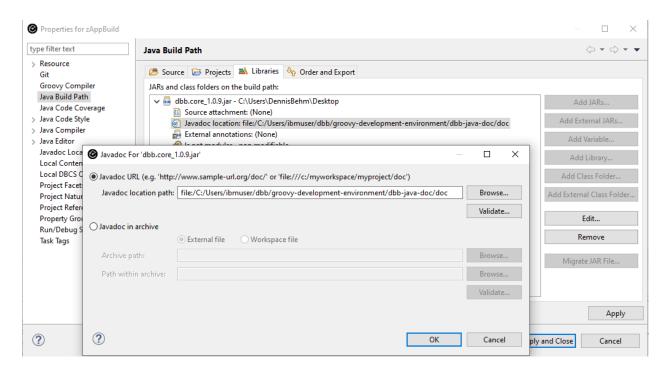
Add the following libraries to your Java Build Path configuration: **Groovy Runtime Library, IBM JZOS Toolkit Library** and the **JRE System library**. Take the defaults on any dialog boxes.



*Note - You will need to add the libraries individually.

Additionally, add the dbb-core.x.x.x.jar to the build path via "Add External JARs...", select the dbb-core package which you have retrieved to the local directory.

• Expand the dbb.core.x.x.x.jar entry that was just added, select the "Javadoc location" entry and click the **Edit** button to set the path for the JavaDoc. Point to the path where you extracted the java doc which you have obtained from the mainframe.



3.2 Leverage the Groovy Tooling

You can now leverage the different views in Eclipse.

3.2.1 Script navigation

Already, with the simple Project Explorer view, you can already see all the methods.

```
👜 z/OS Proj... 📔 Project Ex... 🛭 🔝 Git Reposi... 🗀 📋 😰 Cobol.groovy 🖂
                                   //@groovy.transform.BaseScript com.ibm.dbb.groovy.ScriptLoader baseScript
  > 🖶 > (default package) [dbb-zappbuild development]
  🗸 🏭 > languages
                                                          ⊕ import com.ibm.dbb.repository.*[.]
     > 🗿 Assembler.groovy
      BMS.groovy
                                                            // define script properties
     @Field BuildProperties props = BuildProperties.getInstance()

√ G > Cobol

                                                            @Field def buildUtils= loadScript(new File("${props.zAppBuildDir}/utilities/BuildUtiliti
@Field def impactUtils= loadScript(new File("${props.zAppBuildDir}/utilities/ImpactUtili
            S main(String...): void
            Cobol()
                                                             @Field def bindUtils= loadScript(new File("${props.zAppBuildDir}/utilities/BindUtilities
                                                            @Field RepositoryClient repositoryClient
            Cobol(Binding)

    createCobolParms(String, LogicalFile) : Obj

                                                            println("** Building files mapped to ${this.class.getName()}.groovy script")

    createCompileCommand(String, LogicalFil

    createLinkEditCommand(String, LogicalFile

                                                            // verify required build properties
                                                            buildUtils.assertBuildProperties(props.cobol_requiredBuildProperties)
            getRepositoryClient() : Object
            🔈 run() : Object
                                                            def langQualifier = "cobol"
     > 🗿 DBDgen.groovy
                                                            buildUtils.createLanguageDatasets(langQualifier)
     > A LinkEdit.groovy
                                                            if (props.runzTests == "True") {
    langQualifier = "cobol_test"
     > A MFS.groovy
     > PLI.groovy
                                                                 buildUtils.createLanguageDatasets(langQualifier)
     > 🗿 PSBgen.groovy
     > ZunitConfig.groovy
       README.md
                                                            // sort the build list based on build file rank if provided
   🗸 🚓 samples
                                                            List<String> sortedList = buildUtils.sortBuildList(argMap.buildList, 'cobol_fileBuildRank
     > 済 application-conf
                                                            // iterate through build list
   > 🖶 samples.MortgageApplication
                                                          SortedList.each { buildFile ->
    println "*** Building file $buildFile"
   > 🖶 samples.MortgageApplication.bms
   > 🖶 samples.MortgageApplication.cobol
```

The Outline View shows all the defined methods and can be used for navigating within your build script:

```
🕮 z/OS Proj... 🚹 Project Ex... 🛭 🔝 Git Reposi... 🗀 📋 📳 *Cobol.groovy
                                                                                                           @groovy.transform.BaseScript com.ibm.dbb.groovy.ScriptLoader baseScript
                                                    □ $ 6 * 

⊕ import com.ibm.dbb.repository.*

> 🚔 > zAppBuild [dbb-zappbuild development]
                                                                                        // define script properties
                                                                                        @Field BuildProperties props = BuildProperties.getInstance()
                                                                                       @Field def gitUtils= loadScript(new File("utilities/GitUtilities.groovy"))
@Field def buildUtils= loadScript(new File("utilities/BuildUtilities.groovy"))
@Field def impactUtils= loadScript(new File("utilities/ImpactUtilities.groovy"))
@Field String hashPrefix = ':githash:'
@Field RepositoryClient repositoryClient
                                                                  - -
Properties 📴 Outline 🛭
                                                                                       // start time message
def startTime = new Date()
props.startTime = startTime.format("yyyyMMdd.hhmmss.mmm")
println("\n** Build start at $props.startTime")
    △ baseScript : ScriptLoader
     △ props : BuildProperties
     △ gitUtils : Object
     △ buildUtils : Object
                                                                                        // initialize build
     △ impactUtils : Object
                                                                                       initializeBuildProcess(args)
     △ hashPrefix : String
     △ repositoryClient : RepositoryClient
                                                                                        // create build list
     △ startTime : def
                                                                                       List<String> buildList = createBuildList()

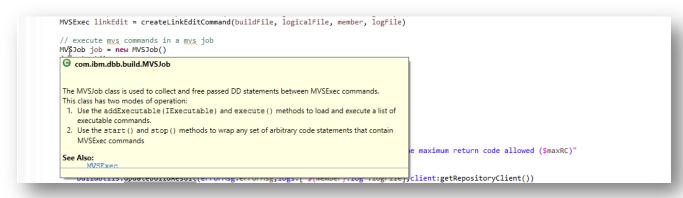
▲ buildList: List<String>
                                                                                        // build programs in the build list

△ processCounter: def

                                                                                       def processCounter = 0
def scriptPath = ""
     △ scriptPath : def
     △ buildOrderList : String[]
                                                                                       if (buildList.size() == 0)
    println("*! No files in build list. Nothing to do.")
     △ testOrderList : String[]
     initializeBuildProcess(String...): Object
                                                                                             e {
   if (!props.scanOnly) {
      println("** Invoking build scripts according to build order: ${props.buildOrder}")
      String[] buildOrderList = props.buildOrder.split(',')
      String[] testOrderList;
   if (props.runzTests == "True") {
      println("** Invoking test scripts according to test order: ${props.testOrder}")
     parseArgs(String...) : Object
     populateBuildProperties(String...): Object
     createBuildList(): Object
     finalizeBuildProcess(Map): Object
```

3.2.2 Displaying JavaDoc within the editor

Hovering over a DBB class, you can see the JavaDoc:



It also provides the details on certain methods:

```
compile.dd(new DDStatement().name("SYSPRINT").options(props.cobol_printTempOptions))
compile.dg()

compile.
(1..17).

AbstractExec com.ibm.dbb.build.AbstractExec.dd(DDStatement dd) throws BuildException
comp
}

// Write
String d
String d
String l
String l
String l
String l
Add a DD statement that will be allocated prior to executing the program and freed once program
execution has completed.

Parameters:
} else i
Compile.dd(new DDStatement().name("SYSPRINT").options(props.cobol_printTempOptions))

AbstractExec com.ibm.dbb.build.AbstractExec.dd(DDStatement dd) throws BuildException
('shr').output(true))

Press 'F2 for focus'
('shr').output(true))
```

It also helps you to format the code or to identify if you have an issue with quotes.

However, please note that the Groovy editor also has some limitations (also due to the language itself), like identifying methods which don't exist.

4 Summary

Groovy Development Tools help you in understanding, developing and maintaining your build framework written in Groovy which leverages the IBM Dependency Based Build toolkit API.

5 Appendix – Alternative IDE options for Groovy Programming languages

If you are already using IBM Developer for z/OS (IDZ) Version 16.x and want to set up groovy development, we have many popular IDEs and text editors that support Groovy programming language. However, most of them lack standard features such as

- Syntax highlighting
- Code completion
- Refactoring & so on ...

Our recommendation will be to install and configure **Eclipse IDE for Java developers²** as it's based on Eclipse and provides a similar interface to IBM Developer for z/OS (IDZ) Version 15.x.

Other alternatives¹ are IntelliJ IDEA and NetBeans.

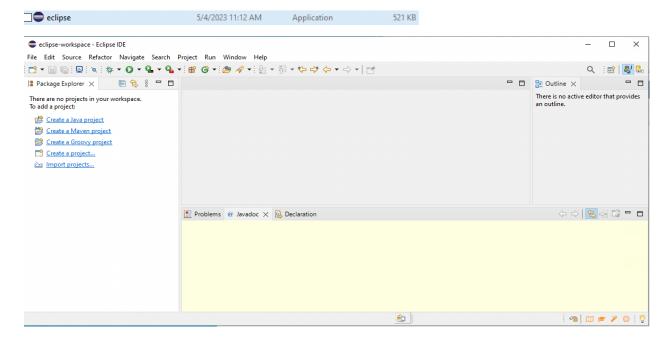
5.1 Eclipse IDE for Java Developers

We can download **Eclipse IDE for Java developers**² from their official website based on the version and operating system of the developer's machine. You can download the archive file and unarchive/untar it in a preferred location of your system.

Note: Below demonstration leverages installation on a Windows 10 machine & Eclipse IDE for Java Developers Version: 2023-03 (4.27.0); Build id: 20230309-1520



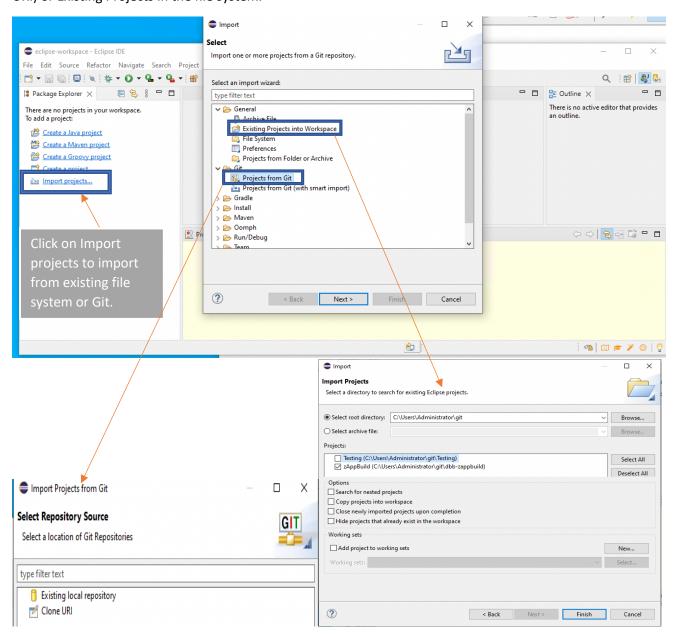
Open the eclipse IDE (Double Click on the eclipse icon). It will open in Java perspective.



¹https://groovy-lang.org/ides.html

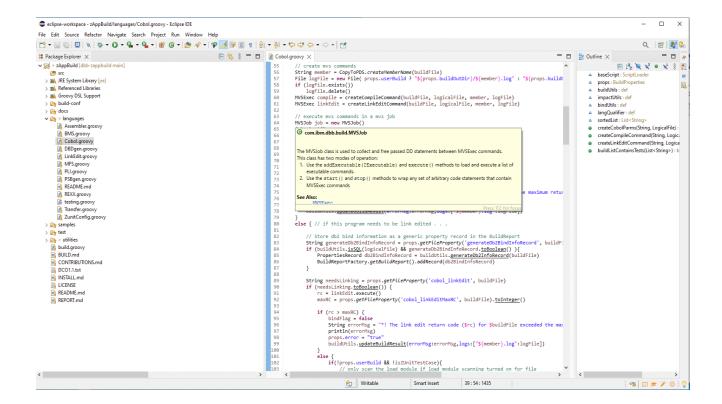
² https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers

Click on **Import projects** which will open a new window to select from various options such as Git (locally, URI) or Existing Projects in the file system.



Please refer to configuration of <u>Groovy Installation / Preparation section</u> for Groovy development plugin installation, java docs and leveraging groovy tooling.

Once set up is completed, you can start maintaining and developing your own groovy code. Below screenshot shows a sample zAppBuild project with Javadoc's displaying contents and an outline view giving additional information.



Title:

Setup your Groovy Development Environment in IDZ to write DBB scripts

Download URL:

https://www.ibm.com/support/pages/node/6366801

Change log:

```
November 2020 - 1.0 - Initial Version
```

February 2021 - 1.1 - Update to obtain dbb-ztoolkit-javadoc-<version>.zip from archive subfolder

May 2023 - 1.1.2 - Update to include alternative IDE options