

Summer Student Report

# **Dependable Continuous Delivery for Machine Protection Software**

TE-MPE-MS European Organization for Nuclear Research  
(CERN) Geneva, Switzerland

*Submitted by:*

Francini Corrales Garro

*Supervisors:*

Marc-Antoine Galilee

Jean-Christophe Garnier

July - August, 2019

# **Abstract**

The present document shows how the migration process was made from CBNG to CBNG 3 of some software projects in the TE-MPE-MS Department. This report also, includes an overview about some CBNG and CBNG 3 features, how the workflow is, the dependencies resolution, and the multi project build process, necessary to understand some aspects in the migration itself. After that, it includes a real example of how the migration process was done.

# Contents

<b>Introduction</b>	<b>1</b>
<b>Overview</b>	<b>2</b>
1    CBNG . . . . .	2
1.1    CBNG Workflow . . . . .	2
1.2    CBNG Project Structure . . . . .	2
1.3    CBNG Tasks . . . . .	3
2    CBNG 3 . . . . .	3
2.1    Multi Project Builds . . . . .	3
2.2    Maven Scopes in CBNG 3 . . . . .	4
2.3    Multi Project Structure . . . . .	4
<b>Migration Process</b>	<b>5</b>
3    Projects to migrate . . . . .	5
4    Mpe-Commons Project Structure . . . . .	6
4.1    Root folder . . . . .	7
4.2    Settings.gradle . . . . .	7
4.3    Build.gradle . . . . .	8
4.4    Product.xml . . . . .	8
5    Resolving Dependable Issues . . . . .	8
5.1    Gradle 5.4.0 incompatibilities . . . . .	9
5.2    Bad definition of dependencies Scopes . . . . .	9
<b>Conclusion and What is Next</b>	<b>10</b>
<b>Acknowledgments</b>	<b>11</b>

# List of Figures

1	CBNG Project Structure. . . . .	3
2	Projects to migrate to CBNG 3. . . . .	5
3	CBNG 3 Multi Project Build Structure. . . . .	7
4	Settings.gradle File Structure. . . . .	7
5	Build.gradle file in CBNG 3. . . . .	8
6	Product.xml file in CBNG 3 . . . . .	8

# Introduction

The Large Hadron Collider (LHC) protection relies on a wide range of software components and frameworks. In order to create these software projects, the Machine Protection Software Section (MS) uses continuous integration and continuous delivery (CI/CD) which allows them to build, automate, and deliver software in a fast and reliable way. CBNG is the custom build tool used in the Accelerator Control Environment and that was used to implement this CI/CD. However, this tool is notably fragile causing frequently interruptions in the pipeline. For this reason, the migration of all the software projects to CBNG 3, a version closer to pure Gradle, is necessary in order to unify and improve the integration and delivery process in the section. That implies a successful migration for all the projects which requires the creation of complex Gradle configurations with hundreds of dependencies.

# Overview

## 1 CBNG

CBNG is a build management tool supporting the process of developing accelerator control software. This tool provides automatic tasks such as compiling, building, testing, releasing and deploying. CBNG is the successor of the CommonBuild build tool used widely around the Beams Department. [2]

### 1.1 CBNG Workflow

All of the the dependencies in CBNG are defined in the `product.xml` which is the primary project descriptor file in CBNG. These dependencies are downloaded from the Artifactory repository that contains the third party libraries and CERN products that are available for development [2]. CBNG parses the dependencies according to the version. In this file dependencies have defined a version that could take the value of **"PRO"**, **"CTX"**, **"NEXT"**, and also could be an empty value or have an specific version as **"1.0.0"**.

### 1.2 CBNG Project Structure

The CBNG projects should follow a defined structure supported by the Standard Directory Layout of Maven. **The Figure 1** shows how the CBNG project's structure should looks like.

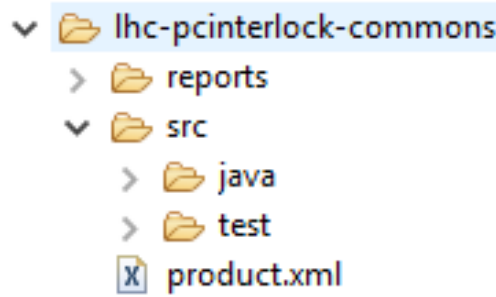


Figure 1: *CBNG Project Structure.*

### 1.3 CBNG Tasks

Once defined the `product.xml` file and following the recommended structure, CBNG can run several tasks included build, compile, devrelease and release.

- **Compile:** compiles all source files.
- **Build:** the build task is in charge of compiling the code, running the tests and creating the artifacts (like main, jar, sources, javadoc).
- **Test:** this task runs the tests.
- **Devrelease:** this task always builds the project from the local directory and uploads the artifacts to the beco-development-local repository or beco-autobuilds-local if run from CI [3].
- **Release:** Builds the project and uploads the artifacts to the beco-release-local repository. It is executed remotely on the release server [3].

## 2 CBNG 3

CBNG 3 is a tool based on a version closer to pure Gradle and is the successor tool of CBNG. This tool allows new features like multiproject builds, Gradle version 5 and Maven scopes support.

### 2.1 Multi Project Builds

Multi-project builds helps with modularization. It allows a person to concentrate on one area of work in a larger project, while Gradle takes care of dependencies from other parts of the project. [4]

The multi project builds are supported in CBNG 3 and offers multiples benefits in the software development process that include, for example, the possibility to build

and release projects together, use parallel task execution, and link projects through the IDE.

## 2.2 Maven Scopes in CBNG 3

In CBNG 3, the dependency attributes "local/excluded", defined in the product.xml file, are replaced to Maven scopes and could take the values of "**compile**", "**test**", "**runtime**", and "**provided**".

## 2.3 Multi Project Structure

The multi project build works with two different kind of projects which are the root project and subprojects or modules. With the migration to Gradle 5.4.0 new files appear inside the root project and subprojects. Settings.gradle, and build.gradle are some of them.

Even though the product.xml is not a new file, their dependencies suffer some changes in the process. Following, there is a little description about these files.

- **Settings.gradle file:** in this file are defined all of the subprojects that are going to be built together and also, includes the root project name.
- **Gradle.gradle file:** the version can be specified here that each subproject will inherit, otherwise, it can be specified per each subproject. Moreover, this file could include two important properties, *org.gradle.parallel=true* and *org.gradle.configureondemand=true*. The first property force Gradle to execute tasks in parallel as long as those tasks are in different projects [5] and the second one only configure necessary projects for the build.
- **Build.gradle file:** defined project dependencies among the subprojects.
- **Product.xml file:** defined all of the external dependencies of a project. The dependencies which were declared in the build.gradle file must not be declared in the product.xml file.

# Migration Process

The migration process from CBNG to CBNG 3 in all of the MPE-MS software projects, implies not only the multi projects building but, the conflicts resolution.

This process was done with a Python script that simplifies the migration process. To execute it, it was necessary to have a file with a list of subprojects and the name of the project root. The script is in charge of modifying some dependencies on the projects files and creating new documents like settings.gradle and build.gradle.

Even though the script is very helpful, it does not resolve all of the version and scopes dependencies successfully.

The next section explains step by step how to do this process successfully and how it is possible to resolve the dependencies conflicts.

## 3 Projects to migrate

The MPE-MS section required the migration for all of the following projects.

✓ lhc-hwc-acctesting-analysis-consistency	✓ lhc-hwc-acctesting-analysis-modules	✓ lhc-hwc-acctesting-client	✓ lhc-hwc-acctesting-constraints	✓ lhc-hwc-acctesting-constraints	✓ lhc-hwc-acctesting-core	✗ lhc-hwc-acctesting-proxy	✓ lhc-hwc-acctesting-rdserver	✗ lhc-hwc-acctesting-rdserver-rc
✓ lhc-hwc-acctesting-gui-ext	✓ lhc-hwc-acctesting-gui-ext-hwc	✗ lhc-hwc-acctesting-gui-rc	✓ lhc-hwc-acctesting-meta	✓ lhc-hwc-acctesting-proxy	✗ lhc-hwc-acctesting-proxy-rc	✓ lhc-hwc-acctesting-rdserver	✓ lhc-hwc-acctesting-rdserver-rc	✓ lhc-hwc-acctesting-testing
✓ lhc-hwc-acctesting-scheduling	✓ lhc-hwc-acctesting-server	✓ lhc-hwc-acctesting-server-ext	✓ lhc-hwc-acctesting-server-ext-hwc	✓ lhc-hwc-acctesting-server-ext-molr	✓ lhc-hwc-acctesting-server-gradle-tasks	✗ lhc-hwc-acctesting-server-rc	✓ lhc-hwc-acctesting-testing	✓ lhc-hwc-acctesting-testing
✓ lhc-hwc-acctesting-webserver-testing	✓ mpe-analysis-client	✓ mpe-analysis-core	✓ mpe-analysis-gui	✓ mpe-analysis-language	✓ mpe-analysis-meta	✓ mpe-analysis-processor	✓ mpe-analysis-providers	✓ mpe-analysis-providers
✓ mpe-analysis-server	✓ mpe-analysis-server-rc	✓ mpe-analysis-signals	✓ mpe-analysis-signals-generation	✓ mpe-analysis-tensorics-modules	✓ mpe-analysis-testing	✓ mpe-analysis-testmodules	✓ mpe-online-monitoring-service	✓ mpe-online-monitoring-service
✓ bic-config-client	✓ bis-common	✓ bis-db	✓ bis-decoders	✓ bis-domain	✓ bis-fesa-beans	✓ bis-server	✓ bis-streams	✓ bis-streams
✓ bis-streams-gui	✓ bis-streams-gui-rc	✓ bis-testing	✓ mpe-commons-akka	✓ mpe-commons-cals	✓ mpe-commons-dataproc	✓ mpe-commons-dataviewer	✓ mpe-commons-domain	✓ mpe-commons-domain
✓ mpe-commons-gui	✓ mpe-commons-guicomp	✓ mpe-commons-lsa	✓ mpe-commons-rda-server	✓ mpe-commons-rdaprox	✓ mpe-commons-reloading	✓ mpe-commons-reloading-ant	✓ mpe-commons-reloading-spring	✓ mpe-commons-reloading-spring
✓ mpe-commons-remote	✓ mpe-commons-testing	✓ mpe-commons-util	✓ CIBDS EDGE driver	✓ CIBG EDGE driver	✓ CIBM EDGE driver	✓ CISA EDGE driver	✓ CISC EDGE driver	✓ CISC EDGE driver
✓ CISO EDGE driver	✓ CISO EDGE driver	✓ CISO EDGE driver	✓ CISO EDGE driver	✗ CIB3	✓ CIBSM3	✓ CISOV	✓ CISOX	✓ CISOX
✓ DQAmxLS2	✓ DQAmxLS2 MOCK	✓ mpe-gradle-application-runner-plugin	✓ mpe-gradle-core	✓ mpe-gradle-deployment-plugin	✓ mpe-gradle-gui-plugin	✓ mpe-gradle-jnip-deployment-plugin	✗ mpe-gradle-migration-plugin	✗ mpe-gradle-migration-plugin
✓ lhc-app-fmcm	✓ pic-db	✓ wic-db	✓ Clean TestBed PM Storage	✗ mpe-pm-api-server	✓ mpe-pm-api-server-docker	✓ mpe-pm-api-server-docker-rc	✓ pm-backend	✓ pm-backend
✓ pm-cpp-SLC6	✓ pm-dc-lib-cpp-rda2	✓ pm-dc-lib-cpp-rda3-to-rda2	✓ pm-dc-rda3-lib-cpp	✓ pm-watchdog	✓ mpe-preop-bis	✓ mpe-preop-core	✓ mpe-preop-gui	✓ mpe-preop-gui
✓ mpe-preop-gui-rc	✓ mpe-preop-smp	✓ mpe-properties-public-service	✓ mpe-properties-public-service-rc	✓ lhc-dqamx	✓ lhc-dqamx-rc	✓ mpe-ucap	✓ qps-benchmark	✓ qps-benchmark
✓ qps-commons	✓ qps-fair-parameters	✓ qps-fesa-beans	✓ qps-lhc-board-specifications	✓ qps-lhc-commons	✓ qps-lhc-core	✓ qps-lhc-db	✓ qps-lhc-dbprovider	✓ qps-lhc-dbprovider
✓ qps-lhc-guilauncher	✓ qps-lhc-guilauncher-rc	✓ qps-lhc-hdsstatus	✓ qps-lhc-lhcstatus	✓ qps-lhc-status-server	✓ qps-lhc-swisstool	✓ qps-lhc-swisstool-acceptance-tests	✓ qps-lhc-swisstool-rc	✓ qps-lhc-swisstool-rc
✓ qps-lhc-voltagetoground	✓ seq-task-qps	✓ mpe-research-kubernetes-groups-memory-leak-fix	✓ mpe-research-scarlet	✓ mpe-systems-blm	✓ mpe-systems-client	✓ mpe-systems-coll	✓ mpe-systems-controls	✓ mpe-systems-controls
✓ mpe-systems-core	✓ mpe-systems-domain-mips	✓ mpe-systems-gui	✓ mpe-systems-lbds	✓ mpe-systems-machinecheckout	✓ mpe-systems-provider-blm	✓ mpe-systems-provider-circuit	✓ mpe-systems-provider-coll	✓ mpe-systems-provider-coll
✓ mpe-systems-provider-controls	✓ mpe-systems-provider-lbds	✓ mpe-systems-provider-machinecheckout	✓ mpe-systems-provider-mips	✓ mpe-systems-qps	✓ mpe-systems-server	✓ mpe-systems-server-rc	✓ seq-task-smp	✓ seq-task-smp
✓ smp-common	✓ smp-gui	✓ smp-gui-rc	✓ smp-logger	✓ streamingpool-ext-bis	✗ pm-aper-analysis	✓ pm-beans	✗ pm-bic-ipoc-analysis	✗ pm-bic-ipoc-analysis
✗ pm-bim-analysis	✗ pm-collimator-hierarchy	✓ pm-data	✓ pm-data-gui	✓ pm-dataconv-fgc	✓ pm-event	✓ pm-event-gui	✗ pm-fill-statistics	✗ pm-fill-statistics
✗ pm-fmcm-analysis	✓ pm-gui-tools	✗ pm-qps13kA-analysis	✓ pm-qps600A-analysis	✗ pm-sce-analysis	✓ pm-smp-analysis	✗ pm-smp-ipoc-analysis	✓ lhc-pcinterlock-commons	✓ lhc-pcinterlock-commons
✓ lhc-pcinterlock-core	✓ lhc-pcinterlock-gui	✓ lhc-pcinterlock-gui-rc	✓ lhc-pcinterlock-server	✓ lhc-pcinterlock-server-rc	✓ lhc-pcinterlock-state	✓ seq-task-bis		

Figure 2: Projects to migrate to CBNG 3.

Each project has a group of submodules (or sub projects) that are going to be built together. For instance, the mpe-commons, has the next subprojects inside:

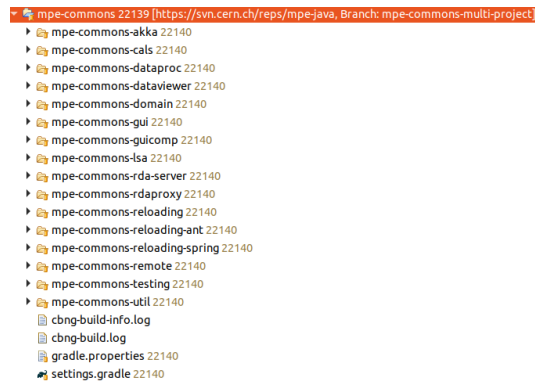
- mpe-commons-akka
- mpe-commons-cals
- mpe-commons-dataproc
- mpe-commons-dataviewer
- mpe-commons-domain
- mpe-commons-gui
- mpe-commons-guicomp
- mpe-commons-lsa
- mpe-commons-rda-server
- mpe-commons-rdaproxxy
- mpe-commons-reloading
- mpe-commons-reloading-ant
- mpe-commons-reloading-spring
- mpe-commons-remote
- mpe-commons-testing
- mpe-commons-util

Those projects are wrapped in the "mpe-commons project" and are compiled and built together.

## 4 Mpe-Commons Project Structure

In order to build a multi project is better if the project has a Maven Support Structure. In this case, the project mpe-commons follows the same structure in all of their sub-projects. The Figure 3. shows the mpe-commons project structure.

This structure includes a root folder (mpe-commons) and a list of subprojects. Also, includes the *settings.gradle*, *build.gradle*, and *gradle.properties* files which are explained in the following sections.



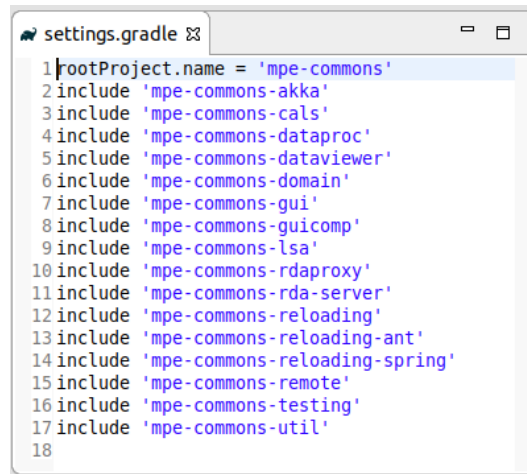
**Figure 3:** *CBNG 3 Multi Project Build Structure.*

## 4.1 Root folder

In this case, the root project is mpe-commons. This folder is in charge of wrapping all of the sub projects that will be built together. Moreover, this folder includes important files as settings.gradle and properties.gradle.

## 4.2 Settings.gradle

In this file are defined the mpe-commons sub projects and the root project name, in this case mpe-commons, necessary to build the projects together. The Figure 4 shows the structure of how to create this file.



**Figure 4:** *Settings.gradle File Structure.*

## 4.3 Build.gradle

Unlike CBNG, in CBNG 3 the subprojects are defined in the *build.gradle* file instead of product.xml file. This file is defined in each subproject. The Figure 5 shows how the build.gradle looks like for the mpe-commons-akka sub project.

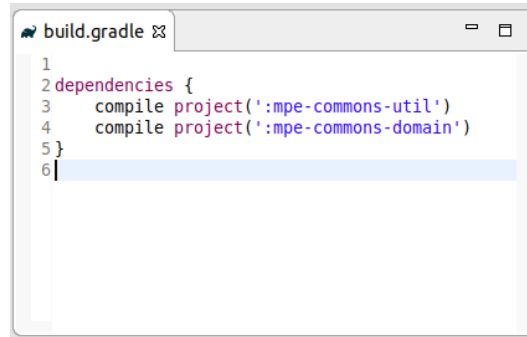


Figure 5: *Build.gradle* file in CBNG 3.

## 4.4 Product.xml

With CBNG 3 the dependencies start using Maven Scopes and the exclude and local attributes are not used anymore. The next figure shows an comparison of the implemented changes in the build.gradle file of one of the subprojects in mpe-commons where the local and excluded properties are replaced for maven scopes. Also, the subproject dependencies defined in the product.xml are deleted from this file and passed to the build.gradle file.

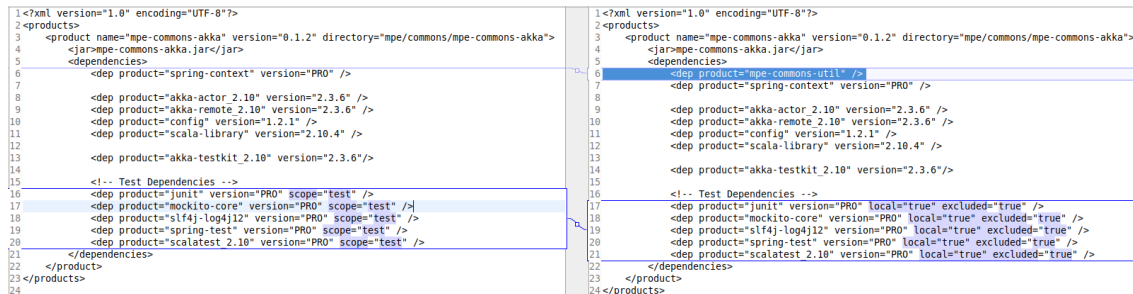


Figure 6: *Product.xml* file in CBNG 3

## 5 Resolving Dependable Issues

The migration process of these software presented some issues during the multi projects build . Those issues included Gradle incompatibilities, tests failures, wrong scopes and

version problems. In the next section are defined some issues.

## 5.1 Gradle 5.4.0 incompatibilities

This issue appears in some projects such as `pcinterlocks`. The "leftshift" operator used in task declarations was removed in Gradle 5. It must be replaced with `doLast`. [1]  
For example, this is how one of the `PcInterlocks` task looked in the beginning:

```
task startPcInterlockGui() << {  
    ...  
}
```

To solve this issue it is necessary to replace the "leftshift" symbol for *doLast*.

```
task startPcInterlockGui() {  
    doLast {  
        ...  
    }  
}
```

## 5.2 Bad definition of dependencies Scopes

The Python script tries to set the scopes according to some rules. However not all the time, the selected scope is the correct one. That implies to verify manually all of the dependencies inside the project and look if the scope is correct or if it is wrong.

Sometimes when a scope is declared in a wrong way, compilation errors can occur because one or more packages do not exist. The solution here could be to change the scope of the dependency to `compile` (default) or remove usages of this dependency in the main source code [6].

# Conclusion and What is Next

With the migration process were converted some projects to CBNG 3 through the use of multi projects builds and Gradle 5 which try to unify and improve the integration and delivery processes in the TE-MPE-MS section.

Actually the projects are located in SVN repository but in order to achieve all of the department goals and unify the migration software process, it will be necessary to migrate all of the projects to Gitlab repository and Bamboo.

# Acknowledgments

I would like to thank my supervisors Jean-Christophe Garnier and Marc-Antoine Galile, for their valuable guidance during the development project process. It is worth mentioning the help that Anita Stanisz gave me in the project process as well. Also I want to express my gratitude to all of the work team for making my stay at CERN an amazing experience and for giving me the opportunity to improve my technical skills and learn a lot of new things during the training.

Francini Corrales Garro, 2019

# Bibliography

- [1] Backward incompatible changes in cbng 3 and gradle 5. <https://wikis.cern.ch/display/DVTLS/Backward+incompatible+changes+in+CBNG+3+and+Gradle+5>, June 2019.
- [2] Cbng. <https://wikis.cern.ch/display/DVTLS/CBNG>, February 2019.
- [3] Cbng tasks. <https://wikis.cern.ch/display/DVTLS/CBNG+Tasks>, June 2019.
- [4] Creating multi-project builds. <https://guides.gradle.org/creating-multi-project-builds/>, June 2019.
- [5] Improving the performance of gradle builds. <https://guides.gradle.org/performance/>, June 2019.
- [6] Maven scope support. <https://wikis.cern.ch/display/DVTLS/Maven+scope+support>, July 2019.