CA Gen Integration

Building CA Gen code through Jenkins

Christian Kersters

Broadcom Limited
Web: www.broadcom.com
Corporate Headquarters: San Jose, CA

Revision History

Revision	Date	Change Description
v1.0	2019/09/30	Initial version
v1.1	2019/10/15	Minor corrections (copyright notices)

.

Contents

.

Introduction

Jenkins is a widely used Continuous Integration platform, many organizations rely on to integrate their developments with.

By lack of in-depth knowledge of CA Gen, however, many customers haven't tried to push the use of Jenkins in CA Gen territory. This results in a loss of major opportunities:

- ***Communication***: Although CA Gen is very different from other development environments, from a CI perspective, the difference stops when CA Gen toolset or encyclopedia has generated the source files corresponding to the code (C, Cobol, …). This helps communication between DevOps teams, reducing the barrier generally separating CA Gen from other development teams
- ***Integration***: Once created, a CA Gen build / split / transfer step can easily be integrated into a Jenkins pipeline, to achieve more complex integration processes and DevOps workflows
- ***Workload reduction***: Using only one tool, DevOps teams only need to check at one place, to identify any issue occurring in the integration process
- ***Configuration management***: Jenkins can easily ensure the presence of the right / latest versions of software (like EAB libraries, bitmaps, …) the CA Gen application requires, increasing the quality of the CA Gen builds
- ***3rd-party products usage***: with the easiness with which Jenkins can work across environments, CA Gen build steps can be concentrated on a single computer only, significantly reducing the required licenses of 3rd-party products. Also, concentration on one environment makes 3rd-party products upgrades much easier and safer

Whatever the target environment, such integration can easily be achieved with CA Gen.

Although integration with Jenkins could be achieved in all types of environments, this document will focus on generation with the Client/Server Encyclopedia, as:

- It contains a shared, official snapshot of CA Gen applications, as opposed to Developer workstations
- Its generators apply to all supported target environments.

In this document, we will see:

1. How to create a Jenkins job to process CA Gen code and what CA Gen-specific features can be processed with Jenkins
2. What main possibilities are available to submit Jenkins jobs for build of CA Gen applications using Jenkins
3. Some possibilities to discover RMT files that need to be built.

.

The Jenkins job

The Jenkins job receives the location and name of the *remote (RMT)* or *installation control module (ICM)* file to process as a minimum.
There are normally 3 possible actions (not necessarily exclusive) the Jenkins job can perform:

1. ***Transfer***: the RMT file can be transferred to its target destination, using, for instance, (S)FTP or, for mainframe code, CA Brightside / Zowe[1]
   The next step will then consist in the split.
   If file transfer is needed, there will then be 2 options:
   a. Using a Jenkins agent to process the RMT
   b. Using the CA Gen Build tool to perform a distributed build

2. ***Split***: A RMT file is easy to split into its individual components. So, here also 2 options are possible:
   a. Use custom tools to split the RMT file. This option can be preferred if CA Gen Build Tool is not used to process the file
   b. Use the Build Tool to split the RMT file. This is safer, as the build tool will also check the integrity of the files that are extracted. When this option is selected, split and build processing are normally done together
3. ***Build***: Here again, 2 options are possible:
   a. Build the load module / RI Trigger library using the CA Gen Build Tool
   b. Build the load module / RI Trigger library using a custom build procedure (make, ant, Endevor, …).

As this document relies on real-world implementation, not all steps or options will be detailed in the current version. Based on feedback / experience, future versions will incorporate documentation of more options.

Splitting RMT files

CA Gen-based Jenkins split job

Using CA Gen Build tool to split a RMT file is very easy.

Parameters

I suggest having 2 parameters:

1. **LoadModule**: load module name (or name of RMT file without extension)

[1] Although transfer of individual files making up the RMT could be considered, it’s strongly advised (because much easier) to transfer the RMT as a whole and split it in the target location

.

2. **RMTFolder**: folder containing remote files generated by CA Gen[2]

Build Step

This step will consist in a Windows batch command (or similar for another OS). The build step can call the ***bldtool.bat*** file (located in the Gen folder of Developer workstations), or, more generically and with less overhead, call the ***bt.ui.jar*** file (Gen\bt folder of Developer workstations).
With the second solution, a typical build step to split a RMT file would be:

```
cd /D "%RMTFolder%"
call java -jar "C:\Program Files
(x86)\CA\Gen86\Gen\bt\bt.ui.jar" -c command -a SPLIT -l . -n
%LoadModule%.rmt
if "%errorlevel%" == "1" type %LoadModule%.out & exit
```

Building CA Gen executable artifacts

CA Gen-based Jenkins build job

Parameters

Same parameters can be used for Build as for Split step, with a few additions:

1. **LoadModule**: load module name (or name of RMT/ICM file without extension)
2. **SourceFolder**: folder containing source files generated by CA Gen[3]
3. **BuildProfile**: optionally, if you need specific build profile to build your load modules, you can add a BuildProfile parameter, with an adequate default value
4. **ProfilesFolder**: location of the build profiles definitions (with default, for easy use / change)
5. **BuildProfile**: name of build profile (optionally with default for most common profile)

Build Step

This step will consist in a Windows batch command (or similar for another OS). The build step can call the ***bldtool.bat*** file (located in the Gen folder of Developer workstations), or, more

[2] We will here consider that the folder is not where the RMT file was generated, because it would then contain all the necessary files, so a split would not be needed

[3] If directly generated from the CSE, it consists of the *Source Code* / *Installation Control* / *Remote Installation* path, specified in the CSE configuration, followed by *Operating System* and *language*, as always added by the CA Gen Construction server. If you want to build it directly from the RMT, no need for a distinct split step, everything can be done at once, selecting a SourceFolder equal to the RMTFolder of the previous (Split) step.

.

generically and with less overhead, call the ***bt.ui.jar*** file (Gen\bt folder of Developer workstations).
With the second solution, a typical build step for an ICM file would be:

```
cd /D "%SourceFolder%"
call java -Duser.home="%ProfilesFolder%" -jar "C:\Program Files
(x86)\CA\Gen86\Gen\bt\bt.ui.jar" -c command -a BUILD -l . -n
%LoadModule%.icm -f %BuildProfile% >%loadModule%.java.out
type %loadModule%.java.out
for /F "tokens=1-9" %%f in (%LoadModule%.java.out) do call
:process %%f %%g %%h %%i %%j %%k %%l %%m %%n
if "%errorlevel%" == "1" type %LoadModule%.out & exit
%errorlevel%
goto :EOF
:process
if "%1" == "Build-FAILED" set errorlevel=1& goto :EOF
shift /1
if not "%1" == "" goto :process
goto :EOF
```

(Note the special logic to detect a build failure and report it to Jenkins).
To process RMT files, simply replace %LoadModule%.icm with %LoadModule%.rmt[4]

External tool-based build job

As previously mentioned, build is not a specific CA Gen activity: it only uses the Build Tool scripts to drive execution of the relevant 3rd-party products on the source files mentioned in the ICM.
Based on the environment and the generated pieces of code, this can however be more or less complicated.

Java Proxy build with Ant

Building a generated Java Proxy with Anit is very simple.

Parameters

Again, you need the similar 2 parameters:

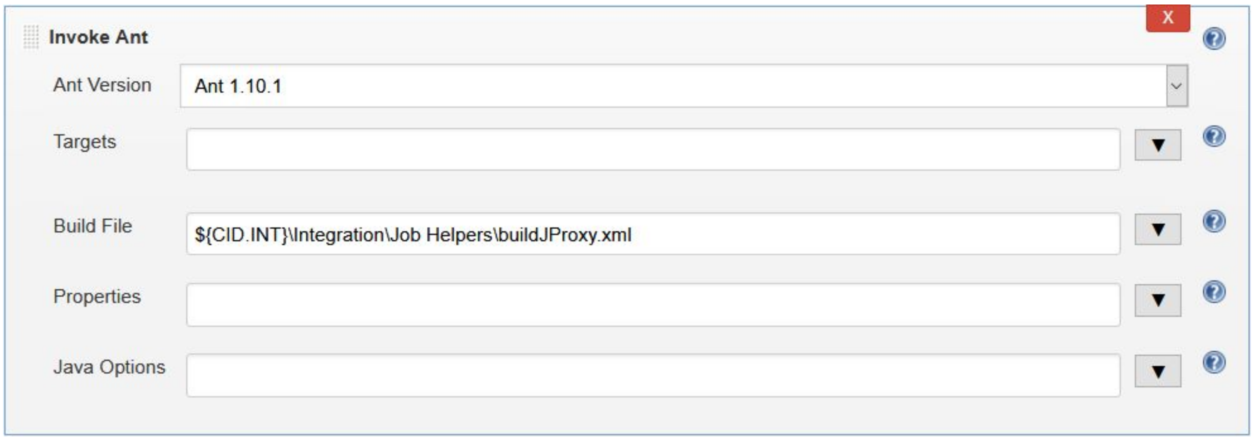
1. **Proxy**: name of the proxy (or name of RMT/ICM file without extension)
2. **SourceFolder**: folder containing source files generated for the proxy by CA Gen[5]

[4] Or add an icm/rmt parameter to the job
[5] If directly generated from the CSE, it consists of the *Source Code* / *Installation Control* / *Remote Installation* path, specified in the CSE configuration, followed by */proxy/java*, as always added by the CA Gen Construction server. If you want to build it directly from the RMT, no need for a

.

Build Step

This time, the build step is an Ant step. There, you need to specify a version of Apache Ant installed in Jenkins, together with a pointer to the buid script that will be executed, as shown below (in this specific case, using parameters)


Invoke Ant

Ant Version: Ant 1.10.1

Targets:

Build File: ${CID.INT}\Integration\Job Helpers\buildJProxy.xml

Properties:

Java Options:

The ant script would then look like:

```
<project name='JProxy' default='all'>
  <property environment="env"/>
  <path id='classpath.base'>
    <pathelement location='C:\Program Files
(x86)\CA\Gen86\Gen\classes\Gen86.jar'/>
  </path>
  <target name='all' depends='compile,jar'/>
  <target name='compile'>
    <property environment="env"/>
    <echo message='... Compiling Java code'/>
    <mkdir dir='${env.SourceFolder}/classes/${env.Proxy}'/>
    <javac fork="yes"
executable="${env.JAVA_HOME}/bin/javac.exe"
srcdir='${env.SourceFolder}/src/${env.Proxy}'
destdir='${env.SourceFolder}/classes/${env.Proxy}'
         includes='com/**'
         debug='on' target='1.6' source='1.6'
classpathref='classpath.base'/>
  </target>
  <target name='jar'>
    <echo message='... Building JAR file'/>
```

distinct split step, everything can be done at once, selecting a SourceFolder equal to the RMTFolder of the previous (Split) step.

.

```
    <mkdir dir='${env.SourceFolder}/deploy'/>
    <jar destfile='${env.SourceFolder}/deploy/${env.Proxy}.jar'
basedir='${env.SourceFolder}/classes/${Proxy}' update='false'
includes='**'/>
  </target>
</project>
```

Submitting Jenkins builds

Submitting Jenkins builds for CA Gen code is fairly easy. There is however no automated way to trigger such builds just after generation of CA Gen code from the CSE.

The 3 easiest possibilities are:

- Build submission using Jenkins console
- Use of a build pipeline
- Execution of individual jobs, based upon some form of RMT discovery

The first approach is trivial, for Jenkins users, and won't be detailed here.

Use of a Build Pipeline

Static Jenkins Pipeline

Use of a static Build Pipeline is certainly the most powerful approach, as you specify:

- Initialization step
- Termination step
- All intermediary build steps, with relevant parameters for each.

Jenkins pipelines are very powerful, and support many requirements. The disadvantage is that you need to manually keep it in sync with your application architecture. Whenever it changes (like addition of a load module), you need to update it (or you need to create a repetitive build step, with loss of modularity and flexibility).

Dynamic Jenkins Pipeline

A dynamic Jenkins pipeline can also be created. Based upon some form of RMT discovery, a Jenkins Pipeline is created to trigger build of all [new] remote files
This approach is fine if you build all your RMT files with the same settings (like build profile), but becomes much more complex if some flexibility is needed.
Also, a Dynamic Pipeline requires usage of a Source Control System.That dynamic pipeline could be created from a Jenkins build, or externally, then committed to the selected Source Control System.

.

Build triggering

Whatever the solution, the job can run periodically (including discovery for the second possibility), or upon demand, by use of the Jenkins console or any external trigger.

Individual Build jobs

Rather than using Build Pipelines, individual build jobs can be triggered, separately or based upon a discovery mechanism, be it from Jenkins itself or from an external utility.
Of course, such a solution is not suitable / advisable for workflows, but can be used as a point solution.

Submission of Jenkins build jobs

Depending on the type of submission desired, different approaches need to be taken for the submission of Jenkins build jobs for CA Gen:

- For ***static pipelines***, the best approach is scheduling. If not feasible or suitable, on-demand submission is the alternative
- For ***dynamic pipelines***, it is recommended to chain the discovery step with the execution of the pipeline in Jenkins. Once again, scheduling or on-demand triggering are possible. The first step would then:
  - Specify the dynamic pipeline using the selected RMT discovery mechanism
  - Commit the pipeline to the Source Control System
  - Chain to the execution of the pipeline
- For ***individual jobs***, as they are point solutions:
  - If there are very few jobs to submit, the Jenkins console is the right tool
  - Otherwise, best is to couple the discovery activity with a utility that posts build requests to Jenkins, through its REST interface.

RMT discovery

2 approaches are easy to implement, for discovery of remote files:

- Scanning of CSE log files
- Directory scanning

Scanning of CSE log files

The ***iefmd<nnn>.log*** file contains information issued by the Construction Server, in the form of lines like:

```
Command Line:  "C:\Program Files
(x86)\CA\Gen86\CSE\bin\rfg.exe" "HLL" "WINDOWS"
```

.

```
"C:\temp\gentest\ENCYADMN\remote.ctl"
"C:\temp\gentest\ENCYADMN\rfg.txt" "NODELETE" "*" "Y"
Packaging of C:\temp\gentest\mvs\cobol\P900.icm is complete
[...]
1684 UTLGENCD End Time: 2019-09-30 10:34:23
```

From there, it should be possible to determine the RMT files that need to be processed and build them.

Scanning of folders

Another possibility consists in scanning folders to discover the files that need to be built.
As an example, here is some windows command code for unconditional discovery of remote files. (This could be enhanced with conditional discovery, based, for instance, on remote file creation date).

```
@echo off
for /R %%f in (*.rmt) do call :process "%%f"
goto :EOF

:process
set rmt=%1
set rmt=%rmt:"=%
echo Submitting build job for %rmt%
for /F "delims=\ tokens=1-7" %%f in ("%rmt%") do call :submit
"%%f//%%g//%%h//%%i//%%j//%%k" %%l
goto :EOF

:submit
set path=%1
set path=%path:"=%
for /F "delims=. tokens=1" %%f in ("%2") do call SubmitJenkins
-p "BaseFolder=%path%" -p LoadModule=%%f
```

In this example, the command file automatically pushes build requests through its REST interface. Although, in this specific case, the SubmitJenkins code is in Java, many ways are available to achieve the same result. Basically, the call is a HTTP POST to an URL like:

```
http://<jenkinsHost>:<jenkinsPort>/job/<jobName>/build
```

or

```
http://<jenkinsHost>:<jenkinsPort>/job/<jobName>/buildWithParam
eters?parm1=val1&parm2=val2
```

and basic authorization.

.

If successful, the reply will contain a ***location*** header field, with value similar to:

```
      http://<jenkinsHost>:<jenkinsPort>/queue/item/1/
```

.