Build and Source Control - Feature #4645

Migrate Ant build logic into Gradle

05/15/2020 07:30 AM - Hynek Cihlar

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Build and Source Control - Feature #2699: convert the ant build en...	**Closed**

History

#1 - 05/15/2020 07:32 AM - Hynek Cihlar

Currently our FWD build logic is split in two build files, Gradle and Ant. Gradle manages the dependencies and drivers the Ant tasks. Ans contains the core build logic. All the Ant logic must be ported and the whole build must be handled by Gradle.

#2 - 07/11/2022 11:40 AM - Greg Shah

- Related to Feature #2699: convert the ant build environment to gradle added

#3 - 07/11/2022 11:42 AM - Greg Shah

Marcin has done some exploration on how to accomplish this migration.

Marcin: Please post a diff with any useful results you have. It would be best to build on top of branch 3821c which is effectively our "trunk" at the current time. Post any issues you see and we can discuss it.

#4 - 07/12/2022 06:02 AM - Marcin Jaskolski

At this point I was attempting to move Ant tasks into Gradle to learn the build process, get familiar with code, this sort of thing.

Successes: IDE (IntelliJ) correctly recognizes Java sources location and Java dependencies.

The fundamental issue: src directory contains not only Java sources, but also other files (Antlr grammars, Antlr generated code, UI graphical files, etc) which leads to large number of manual task definitions (cleanup, copying, manual VSC configs, etc). It's more difficult to take advantage of tooling (Gradle, IDEs, etc)

I do not know how much wiggle room is allowed here.
For example, I'd probably start by moving ANTLR grammars to a separate dir, the same goes for UI elements.

Greg, if you want me to work on this issue - great - can we define the success criteria and smaller steps (versioning - probably no one wants to end up with huge merge) ?

#5 - 07/12/2022 06:17 AM - Greg Shah

In regard to project import for an IDE, we have had good results in Eclipse by importing from the ant build.xml. I have not tried from gradle, but since there is so much in the ant build, the import would have to traverse that anyway.

We have not planned to work on this task right now, but if you have some safe changes that move us ahead we can consider merging them into a branch.

At a minimum, I want to capture the list of issues to be resolved.

> The fundamental issue: src directory contains not only Java sources, but also other files (Antlr grammars, Antlr generated code, UI graphical files, etc) which leads to large number of manual task definitions (cleanup, copying, manual VSC configs, etc). It's more difficult to take advantage of tooling (Gradle, IDEs, etc)

I do a fair amount of the ANTLR work in the project. I do not use Eclipse for that. Nor do I use Eclipse for any UI resources. So I admit that I have not considered this aspect of our project structure. On the other hand, I am somewhat resistent to the idea that we must separate our project into top level directories based on the type of resource. I prefer grouping things by meaning/purpose instead of source language. When the same people work on related code, the source language doesn't matter too much IMO. This requirement is something I dislike about the "standard" maven project structure.

Anyway, let's discuss it here to inform our future work.

#6 - 07/12/2022 06:39 AM - Marcin Jaskolski

If you do not plan any work on this task now, I do not have anything safe to deploy. It was exploratory.

The questions of build scripts and sources layout are rather philosophical, not sure if you are interested in discussing them here ?

#7 - 07/12/2022 07:10 AM - Greg Shah

> The questions of build scripts and sources layout are rather philosophical, not sure if you are interested in discussing them here ?

The advantages of moving to a standard Maven project structure:

- People are familiar with it.
- Some tools may depend upon it. Maven does, though we have no interest or plans to use Maven since we use Gradle for dependency processing.

Disadvantages:

- It is extra work for little benefit.
- Some people (i.e. me) consider the Maven standard project structure to be poorly designed.

Unless I've missed something important, we can probably leave the discussion there. :)

#8 - 07/12/2022 07:41 AM - Hynek Cihlar

I agree with Greg that if you don't work in an IDE, having the standard, more complex and deeper (Maven, Gradle, etc.) directory structure is a burden. In an IDE you typically don't traverse directories as this is either automated or hidden to you. Also the majority of the code is kept in a single project, which also doesn't require the more complex directory structures.

#9 - 07/12/2022 07:49 AM - Marcin Jaskolski

Tooling is familiar with standard structure.

Take automated tests - the standard Maven way really boils down to "add Surefire plugin, add JUnit dependency" - boom, you're done. Maven knows what to do, IDE knows what to do - hey, you can get even code coverage report with a click of a button. CI/CD is happy.

Automation - each manual task is error prone. People are horrible at remembering small menial things. I know I am horrible at them ;-)
Take VCS - with "standard" approach - "Hey, VCS - just take care of the src directory and don't touch target, thank you". Pretty much done.
Take JAR builds - again - std tools just know "compile java srcs, copy resources, maybe add deps"

I'd say these are powerful tools at developer's hands, but I have to agree - it would be a major effort.

#10 - 07/12/2022 08:37 AM - Marcin Jaskolski

Hynek Cihlar wrote:

> I agree with Greg that if you don't work in an IDE, having the standard, more complex and deeper (Maven, Gradle, etc.) directory structure is a burden. In an IDE you typically don't traverse directories as this is either automated or hidden to you. Also the majority of the code is kept in a single project, which also doesn't require the more complex directory structures.

Right, well, if you don't use IDE this might be the case.

#11 - 07/12/2022 08:58 AM - Hynek Cihlar

Marcin Jaskolski wrote:

> Tooling is familiar with standard structure.
>
> Take automated tests - the standard Maven way really boils down to "add Surefire plugin, add JUnit dependency" - boom, you're done. Maven knows what to do, IDE knows what to do - hey, you can get even code coverage report with a click of a button. CI/CD is happy.
>
> Automation - each manual task is error prone. People are horrible at remembering small menial things. I know I am horrible at them ;-)
> Take VCS - with "standard" approach - "Hey, VCS - just take care of the src directory and don't touch target, thank you". Pretty much done.
> Take JAR builds - again - std tools just know "compile java srcs, copy resources, maybe add deps"
>
> I'd say these are powerful tools at developer's hands, but I have to agree - it would be a major effort.

To be fair all the benefits you mention above are one-time only. While setting them up in non-standard project layout does take time, it is only the

matter of making it work, once. On the other hand interacting with the project on daily basis is a continuous kind of effort. Thus having a project layout that is easy to work with for all the members of the team has a higher priority IMO.

Btw I use IntelliJ IDEA myself and I also used to be a happy user of Eclipse.

#12 - 07/12/2022 09:12 AM - Greg Shah

Thanks for the thoughts, Guys. I agree that the project has a ways to go to integrate nicely into an IDE environment. We'll have to consider the improvements that can be made when we get back to this task.

#13 - 07/12/2022 09:58 AM - Marcin Jaskolski

OK, I see. What about the unit tests ? Shouldn't we integrate them into the build ?

#14 - 07/12/2022 10:51 AM - Greg Shah

Our unit tests are limited. We intend to add more, but the Testing Process for 4GL support is really more important and is getting our investment of time. Regardless, we intend to implement CI/CD for running these nightly. For our main testing, there are too many tests and too much conversion/setup to run integrated with the build.