Managing Parliament™ Dependencies*

Ian Emmons

June 9, 2015

1 Introduction

Parliament™ has the following two targets in its main build file:

- `publish-integration`
- `publish-release`

The first of these targets is used by the nightly build of the continuous build machine, `odin.bbn.com`, to publish build artifacts to the BBN Ivy repository, which is located on `tools.ke.bbn.com`. The version number for these targets is defined here:

```
Parliament/KbCore/parliament/Version.h
```

2 Recommended Practice

The recommended practice is to "release" each version of Parliament using the "publish-release" target. This requires a manual invocation of the target. Then you immediately increment `Version.h` and check it into the repository. The next nightly build will call the target "publish-integration", which adds "-SNAPSHOT" to the end of version numbers. Every incremental build going forward will be working toward the *next* version of Parliament. Once it is time to create a new release, follow this procedure:

*The following is adapted from email messages written by Troy Self.

1. Manually call "publish-release." This target simply uploads whatever binaries are present in your working copy to the Ivy server and then updates the metadata. It does so for whatever version is in `Version.h` and whatever binaries happen to be in the build directory.
2. After publishing, increment the version number in `Version.h` and commit. Let the continuous build complete, thereby publishing the first snapshot under the new version number. Note that you may need to poke Cruise Control[1] to start a nightly build if you don't want to wait until the next day for this.
3. Commit changes towards the release of the new version number.

3 What If I Commit Before Releasing?

Suppose that someone commits major, disruptive changes to Parliament without first releasing the previous version and incrementing the version number. In such cases, any dependents on version "latest.integration" will automatically pick up these changes and, most likely, cease working or even building. Here's how to recover from this situation:

1. First, the binaries from the last Subversion revision before the disruptive commit need to be published to the Ivy repository as a release of the existing version number. Recovering or recreating these files may be a challenge — you will need to be resourceful. Once you have the binary files, copy them to where Ant expects them to be, and then call "ant publish-release". In the future, the easiest way to do this is to invoke "ant publish-release" as soon as the build completes.
2. Next, update the `Version.h` file in your working copy to reflect the next version number, and then commit this change to the Parliament repository. After this is done, all changes to Parliament going forward will show in Ivy as «new-version»-SNAPSHOT, so that the old version number will always refer to a static set of files.
3. Dependents should consider updating their Ivy files to be more specific in their dependency on Parliament components:

[1] http://odin.bbn.com:8080/cruisecontrol/

- Parliament
- JenaParliament
- JosekiParliamentClient
- ParliamentDB-win32
- ParliamentDB-win64

If it is set to "latest.integration" then it will still try to grab the latest, potentially incompatible version. Harden that to the older of the two version numbers discussed in the steps above. This will require searching through all the dependent's ivy.xml files. There might be a way to specify an ant property, though we have not yet figured out how to do so.

4. Ensure that Parliament coders know they cannot make disruptive changes to the trunk of Parliament without considering versioning and the impacts to dependent projects.

4 Write Access to the Ivy Repository

Invoking either of the publish targets requires write access to the Ivy repository on `tools.ke.bbn.com`. For this, you need a login on that machine. You will also want to set up your home directory on `tools.ke.bbn.com` with your SSH key. And, as a final step, you need to instruct Ivy where to find your SSH key on your development machine. Ivy expects to find this key here:

```
${ivy.user.dir}/loki.keyfile
```

The Ivy user directory is typically called `.ivy2` and is located in your home directory. On a UNIX-like development machine (including Macintosh and Linux), this requirement is easy to satisfy by simply creating a symbolic link in your Ivy user directory called `loki.keyfile` that points to your SSH key.