Managing Parliament™ Dependencies

Ian Emmons

October 17, 2022

1 Introduction

This document lists the steps to create a new release of Parliament™.

As a preliminary step, create a directory to store the release artifacts so that you will always have them available locally. The rest of this document will refer to this location as `release`. For example, the author uses this folder:

```
~/Desktop/Parliament/releases/«version-number»
```

2 Consider Upgrading Boost

Check on the latest available version of the Boost libraries[1] and consider upgrading to that version. This step is not strictly necessary, but it is nice to stay relatively current, and Boost updates every quarter. Instructions for this process can be found in the Parliament User Guide.

3 Prepare the Release on Each Plaform

Parliament currently supports these platforms:

- Macintosh
- Ubuntu Linux
- Red Hat Enterprise Linux (RHEL)
- Windows (both 64- and 32-bit)

[1] `https://www.boost.org`

For each platform above, start the appropriate machine or virtual machine and perform the following steps:

1. Run the operating system's update process. On Windows, also run the Visual Studio updater.
2. If the compiler has been updated, consider rebuilding the Boost and Berkeley DB libraries. (See the User Guide for details.)
3. Create a clean build of Parliament itself.
4. Check the JUnit report (`target/reports/junit-noframes.html`) to be sure that the only outstanding test issues are the usual 12 geospatial index failures.
5. Copy the distribution archive from `target/distro` to `release`.

Finally, on Ubuntu only, build the Docker image:

```
cd jena/docker
ant -Ddistro=../../target/distro/
      parliament-2.8.2-ubuntu20-64.zip
cp ../../target/distro/
      parliament-2.8.2-ubuntu-docker.tar.bz2 release
```

4 Test on Each Plaform

For each supported platforms, run through the following steps to verify that things are working well.

Step 1: Set a short query timeout of 10 or 15 seconds and start Parliament.

Step 2: Start Parliament and create the indexes.

Step 3: Load `deft-data-load.nt` and `geo-example.ttl` from the dependencies directory.

Step 4: Verify that there are 4 instances in the geospatial index.

Step 5: Explore a bit to ensure that functionality works as expected.

Step 6: Run this query:

```
   select distinct ?cls ?p ?value where {
      values ?pred { owl:maxQualifiedCardinality
            owl:qualifiedCardinality }
      ?cls rdfs:subClassOf ?r .
      ?r a owl:Restriction ;
         owl:onProperty ?p ;
         ?pred ?value .
   } order by ?cls ?p
```

Step 7: Run this query:

```
   prefix acore: <http://adept-kb.bbn.com/adept-core#>
   select distinct ?x ?xType ?xDirectType where {
      ?x a acore:Role .
      {
         {
            ?x a ?xType .
         } union {
            ?x par:directType ?xDirectType .
         }
      }
   } order by ?x ?xType ?xDirectType
```

Step 8: Run this query:

```
   prefix acore: <http://adept-kb.bbn.com/adept-core#>
   select distinct ?super ?directSuper where {
      {
         acore:Divorce rdfs:subClassOf ?super .
      } union {
         acore:Divorce par:directSubClassOf ?directSuper .
      }
   } order by ?super ?directSuper
```

Step 9: Run this query:

```
select ?feature1 ?feature2 ?distance where {
   ?feature1 geo:hasGeometry ?geometry1 .
   ?geometry1 a sf:Point ;
      geo:asWKT ?wkt1 .
   ?feature2 geo:hasGeometry ?geometry2 .
   ?geometry2 a sf:Point ;
      geo:asWKT ?wkt2 .
   filter (str(?feature1) <= str(?feature2))
   bind (geof:distance(?wkt1, ?wkt2, uom:metre) as ?distance)
} order by ?feature1 ?feature2
```

Step 10: Run this query, which contains a Cartesian product in the result set. Switch to the admin page and watch it time out.

```
select distinct ?x1 ?y1 ?x2 ?y2 where {
   ?x1 rdfs:subClassOf ?y1 .
   ?x2 rdfs:subClassOf ?y2 .
}
```

Step 11: On the Insert Data page, attempt to insert a reserved predicate and verify that this fails:

```
owl:Thing par:directType owl:Class .
```

Step 12: Export the entire repository as N-Triples, and then export the default graph as Turtle.

Step 13: Shut down the server.

Step 14: From the `kb-data` directory, run these commands:

On Linux:

```
env LD_LIBRARY_PATH=../bin ../bin/ParliamentAdmin -s
env LD_LIBRARY_PATH=../bin ../bin/ParliamentAdmin -t -e foo.nt
```

On Mac:

```
../bin/ParliamentAdmin -s
../bin/ParliamentAdmin -t -e foo.nt
```

On Windows:

```
..\bin\ParliamentAdmin /s
..\bin\ParliamentAdmin /t /e foo.nt
```

Step 15: On Windows and Linux, install as service (see Section 2.2.1 in the User Guide) and start it running. On Mac, start as a daemon.

Step 16: Run through the test steps above.

Step 17: Shut down the service or daemon. On Windows and Linux, uninstall the service.

Step 18: Start the Docker container, run through the test steps above, and stop the container.

5 Prepare the Dependencies Archive

From the root of your Parliament working copy, run these commands:

```
find dependencies -name .DS_Store -delete
zip -9 -r release/parliament-dependencies-2022-04-25.zip
      dependencies/
git tag -a -m "Dependencies archive released on
      2022-04-25 prior to releasing version 2.8.2"
      dependencies-2022-04-25
```

6 Update README.md

Update the What's New section of `README.md`.

7 Tag and Publish the Release on GitHub

Tag the release:

```
git tag -a -m "Release version 2.8.2, created 2022-04-27"
      release-2.8.2
```

Upload artifacts, including dependencies

8 Publish the Docker Image

```
docker tag parliament-2.8.2-ubuntu idemmons/parliament:2.8.2
docker push idemmons/parliament:2.8.2
```

9 Bump the Version Number

In preparation for the next version, increment the version number in this header file:

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

Be sure to set both the string and numeric forms to the same values.

Add a new (empty) section to the top of the readme file for changes in the next version, and commit the changes.