ReleaseCandidates

This page is an appendix to the overall Derby release instructions found here: DerbySnapshotOrRelease

Table of Contents

For each release candidate:

1. Prepare the code and documentation.
    - drive bug list to zero
    - add a new section to testing wiki pages
    - arrange for appropriate version numbers in JIRA
    - ensure all new (and old) files have correct copyright & license
    - update `releaseSummary.xml`
    - ensure all release notes are current
    - generate `RELEASE_NOTES.html` in the branch and check it into the svn repository. Please consult the instructions for generating release notes.
2. If you are building a 10.7 or later release, build the release distributions by executing the "release" target in the top level build.xml file. Skip the step which follows--the master build target preforms all of the work described in the following step.
3. If, however, you are building a 10.6 or earlier release, perform the following steps:
    - svn update source and doc trees
    - build all, then:
    - copy `rrefexcept71493.dita` from `<source>/classes/doc` to `<docsource>/src/ref`
    - clean doc & source trees, then build release artifacts
    - sign your release artifacts
    - bump version to prepare for a possible next build
4. After building the release distributions:
    - (optional🙂 build eclipse plugin or get a volunteer to do it for you
    - copy the release artifacts to `people.apache.org` web site (which actually means updating your `public_html directory` on `home.apache.org` using `sftp`)
    - verify a downloaded lib, bin and src distribution (build in the src distribution, preferably create a release)
    - update release wiki page
    - call for vote

For each release candidate

Verify that:

- `beta` property is false (unless you are creating a *beta* release candidate). If this is the first release candidate off this branch the `beta` property will have been set to false.
- Version number is correct. On the 10.6 and earlier branches, the version number should have been bumped after spinning the previous release candidate.

You may need to worry about the following arcana if you are building a release on a 10.6 or earlier branch: The third and fourth parts of the version number are combined into a single property, maint, where maint = (third digit * 1000000) + fourth digit. Note that removing the beta flag will not have an effect unless the 3rd digit (fixpack) is greater than 0, since version numbers with fixpack=0 always are considered alpha. Fixpack (3rd digit) will normally be set to 1 when the branch is cut, but if it isn't, it must be incremented before the release candidate can be created.

Managing version numbers in JIRA

Careful management of the JIRA release versions enables us to track the particular issues that were fixed in each particular release, even release *candidates*.

It's perhaps easiest to explain with an example:

1. RENAME 10.13.0.0 to be 10.13.1.0. This will automagically associate the release with all bugs which have been fixed for 10.13.
2. Create a new 10.13.1.1 release id to represent the head of the 10.13 branch after the release goes GA.

Note that you have to adjust the 10.13 ids every time you create a new release candidate. But the idea is that, when the release is finished, its id will be

10.13.1.x

and the head of the 10.13 branch will be advanced to

10.13.1.(x+1)

Check-ins just before generating release artifacts

You may skip this section if you are building a 10.7 or later release. The work described in this section is performed by the master "release" target in the top level 10.7 build.xml file. If you are running behind a firewall and/or using a proxy server, you may need to set your ANT_OPTS environment variable as described on https://issues.apache.org/jira/browse/DERBY-6640.

1. For releases on the 10.6 or earlier branches, Adjust version numbers in documentation. Make sure the copyright notices has the correct years.
2. For versions < 10.3; finalize `CHANGES`. For later releases the `CHANGES` file has been deprecated.
   The tool `java/tools/ChangesFileGenerator.java` can help you generate this file. It can be invoked by running `ant genchanges` in `tools/release`.
3. Check in the latest SQLState documentation. (This step can be done in advance, but make sure that no SQLState has been modified before creating the release).
   Build the source tree. The SQLStates are documented in the Reference Guide on the following page: `rrefexcept71493.dita`. This file is generated by the Derby build and placed in `classes/doc`. Take the version of this file generated by building the code branch and check it into the doc branch at `src/ref/rrefexcept71493.dita`. While you're at it, merge that change to the trunk so that the nightly build will generate an up-to-date Reference Guide too.
4. Sync the source and doc repository.
   'svn up' in your subversion view to bring all files in your view up to the latest revision. Otherwise, the version output by svn which is captured for the build number will be a range (e.g. 290275:320938).

Verify your machine configuration

1. Check you have all required pieces: - doc tree

- with DITA library
- with latest SQLState - `KEYS` checked in
- `RELEASE_NOTES.html` (and, *only for version < 10.3*, CHANGES) checked in.
- md5 & pgp and docs info set correctly in `~/ant.properties` or in `tools/ant/properties/packaging.properties` and available (PATH)
- (*for version < 10.13*) `~/ant.properties` set correctly for: jdk15, jdk16, jsr169
- sane not set in `ant.properties`
- non-source files for building source available: `junit.jar`, `felix.jar`, (with 10.3 and earlier: `osgi.jar`), dita library (again).

Build the release artifacts

You may skip this section too if you are building a 10.7 or later release.

The work described in this section is performed by the master "release" target in the top level 10.7 build.xml file. for 10.7 or later, all you have to do is invoke the "release" target and answer the questions it poses.

1. Build the documentation.
   The documentation needs to be included in the -bin distribution and src, so you will need to access the doc branch when running the ant release target. The doc build is not controlled by the source tree build, and thus needs to be completed beforehand.
   Information on building the docs is located at http://db.apache.org/derby/manuals/dita.html.
2. Create the distributions for release by running:

```
svn up
ant prepareforrelease
cd tools/release
ant release
ant sign
```

💡 Note: ant prepareforrelease does a few basic checks, and then:

```
ant clobber
rm -rf jars javadoc snapshot  # really clean
rm tools/release/*.zip tools/release/*.tar.gz  # really,
rm tools/release/*.md5 tools/release/*.asc     # really clean
ant sane ; ant all ; ant buildjars   # for lib-debug
ant clobber
ant insane
ant -Dsane=false snapshot
ant publishedapi
```

You will need to enter your PGP passphrase several times as the release distributions are signed.
You can save yourself some typing by using gpg's `--passphrase-fd <fd>` option. This instructs gpg to read the passphrase from the specified file descriptor.
This will create the following files in your tools/release directory:
*db-derby-[version]-bin.tar.gz
*db-derby-[version]-bin.zip
*db-derby-[version]-lib.tar.gz
*db-derby-[version]-lib.zip
*db-derby-[version]-lib-debug.tar.gz
*db-derby-[version]-lib-debug.zip
*db-derby-[version]-src.tar.gz
*db-derby-[version]-src.zip
The Eclipse core plugin is generated in the snapshot directory at the top level by the snapshot target. You should also create the Eclipse UI plugins (see `plugins/eclipse/readme.txt`, except use the core plugin created in the snapshot directory), but this requires Eclipse. If you don't want to do it yourself, those interested in the Eclipse plugins will likely volunteer to generate them for you.
Note that ui-plugin can be built as a *beta*. If you don't build the ui-plugin yourself, make sure that its *beta* status matches that of the release candidate you are building.
You should also create checksums and signatures for these files with:

```
gpg --armor --detach-sign derby_ui_plugin_[version].zip
gpg --armor --detach-sign derby_core_plugin_[version].zip
md5 -q derby_ui_plugin_[version].zip > derby_ui_plugin_[version].zip.md5
md5 -q derby_core_plugin_[version].zip > derby_core_plugin_[version].zip.md5
```

There is a problem with the `ant sign` target on Cygwin that may occur elsewhere. If for some reason the `ant sign` target hangs, it may be prompting and waiting for input that you cannot see.
In that case, you can also use this simple script to automate signing the release archives:

```
#!/bin/sh signone() {
  gpg --detach-sign --armor $1
  md5 -q $1 > $1.md5
}

signone $1-bin.tar.gz
signone $1-bin.zip
signone $1-lib.tar.gz
signone $1-lib.zip
signone $1-lib-debug.tar.gz
signone $1-lib-debug.zip
signone $1-src.tar.gz
signone $1-src.zip
```

Invoking this 'sign.sh db-derby-10.1.1.0' would sign all of the release archives for Derby 10.1.1.0, for example.
Be sure to replace the commands for gpg and md5 with the correct commands for your system. Note that on cygwin, the md5 switch is "-n" rather than "-q".

3. Verify the signatures and checksums.
As an example, the Derby 10.1 archives would be verified with GPG as follows:

```
gpg --verify derby_ui_plugin_1.1.0.zip.asc derby_ui_plugin_1.1.0.zip
gpg --verify derby_core_plugin_10.1.1.zip.asc derby_core_plugin_10.1.1.zip
gpg --verify db-derby-10.1.1.0-src.zip.asc db-derby-10.1.1.0-src.zip
gpg --verify db-derby-10.1.1.0-src.tar.gz.asc db-derby-10.1.1.0-src.tar.gz
gpg --verify db-derby-10.1.1.0-lib.zip.asc db-derby-10.1.1.0-lib.zip
gpg --verify db-derby-10.1.1.0-lib.tar.gz.asc db-derby-10.1.1.0-lib.tar.gz
gpg --verify db-derby-10.1.1.0-lib-debug.zip.asc db-derby-10.1.1.0-lib-debug.zip
gpg --verify db-derby-10.1.1.0-lib-debug.tar.gz.asc db-derby-10.1.1.0-lib-debug.tar.gz
gpg --verify db-derby-10.1.1.0-bin.zip.asc db-derby-10.1.1.0-bin.zip
gpg --verify db-derby-10.1.1.0-bin.tar.gz.asc db-derby-10.1.1.0-bin.tar.gz
```

The md5 checksums can be verified by generating them via another method. For example, using openssl:

```
openssl md5 < db-derby-10.1.1.0-src.zip
```

And comparing the output of openssl to the output from ant in `db-derby-10.1.1.0-src.zip.md5`

4. Bump the fourth digit of the source in preparation for a possible next build. I don't think this step does anything except update the last digit of the release id in release.properties. The bumped release id represents the head of the branch.

```
cd tools/release
ant bumplastdigit
```

Commit the changed version number.

Post the release artifacts

1. Keep the `jars/insane/*.jar` and `jars/insane/derby.war` files available. You will need them for maven deployment after the vote is complete.
   It is advisable to copy the jars to another directory and include the version number of the release candidate in the directory name. This way you avoid having to extract the jars from the release artifacts if you inadvertently overwrite the jars (through a rebuild) before deploying to the maven repository.
2. Add the next release id to JIRA. This is the release id on the branch. For 10.6 and earlier releases, you will have bumped this id by hand. For 10.7 releases and later, the id will have been bumped by the master "release" target. This allows bugs to be marked as fixed at the head of the branch. See the following email thread for our latest thinking on this topic: http://old.nabble.com/Re%3A-Plans-for-a-10.5.3.1--tt26533770.html#a26577283
3. Create a new db-derby-x.y.z.w directory under https://dist.apache.org/repos/dist/dev/db/derby, copy all of the release artifacts (distros, checksums, signatures) to that subdirectory, and commit these changes.
4. Vote on the distributions
   Call for a vote on derby-dev to have the distributions posted on your `public_html` accepted for the release.
   ℹ It is not always easy to decipher all the voting rules which govern the acceptance/rejection of a release candidate. First there are the Apache rules for releases, the DB decision guidelines, and finally the DB PMC Bylaws.
   The existing practice is: A vote needs to be running for at least 7 days, so, give at least that much time before closing the vote to give adequate time for others to inspect and test the distributions. If no-one has responded after a week, prod gently until you get a response. A majority of votes, and at least one binding +1 vote are required for acceptance.
   Forward or bcc a copy of the call for vote to private@db.apache.org so the DB PMC is aware that a vote is in progress. Also forward the results post to private@db.apache.org. (⚠ Note: do not **cc** the PMC; **bcc** or forward a copy of the post.)

During the vote

1. Address items on the ReleaseVettingChecklist
   Make sure that the community addresses relevant items on the ReleaseVettingChecklist.

After an unsuccessful vote

1. If vote does not pass and go back to targeting bugs in Jira.
   If the vote does not pass and additional release candidates need to be made, then presumably it won't have passed because of a showstopper-type bug or similar issue, so you need to go back to the bug-fixing section.