SmartFrog Release Process

31 Oct 2011

Release Process for SmartFrog

31 Oct 2011

1 Introduction

This project, `core/release`, contains the Ant and Ivy files to generate binary and source releases of SmartFrog as self-contained JAR file installers or as RPM files.

2 How to make a release

To make a public release, you need to be running Java1.6 on an Linux system with the RPM tools installed. That is, the package `rpm-build` must be installed, otherwise the RPMs will not be created. Test: type `rpmbuild` on the command line, if you get an error message about no spec file, the tool is installed.

You should build on a clean check out of the SVN repository, with a retail Ant release.

We have a special virtualized system image that can be used for this; it guarantees there is no development contamination of the build. This is the only way you should do a public release.

The details on how to set all this up are covered at the end of this document.

2.1.Get the machines ready first

Getting the host machine and VM ready can take time -and if you are trying to do it while the release is being made, it will make everything unstable and the release will not be as planned

Do not try and upgrade the host OS the week before a release -VMWare will not be reliable, and you will not get a release out.

Do not try and upgrade VMWare the day of the release -again, the VM will be unstable.

If you are using this VM, then before you release (days before, if need be)

- Start the VM.
- Bring it up to date with all its libraries, Java version, etc.
- Clean up any directories of old files, so that the (small, virtual) disk has space
- Reboot it
- Make sure the clock is synchronised.
- Make sure there is a few hundred MB of spare disk space. If there is not, clean up outdated Linux kernel versions.
- In the SmartFrog root directory `core` run `svn update`

A two week break can be enough to stop the image from working -updating the VMWare host will often break it.; this can trigger a need to switch back to the VESA display driver, reinstall the VMWare tools, etc., which can only waste time. Also: make sure that `gcj` hasn't snuck in; RedHat/CentOS images seem to do this sometimes, which breaks tests. A `java -version` call can verify the Java version is Sun's and current.

2.2.Release on a Thursday

This is to give the option of slipping a day without rushing stuff out. If do not start to build the release until Friday, you can end up rushing something broken out on Friday afternoon without enough testing. A release should be unhurried.

> *Do not try and release in a rush, as a last minute activity. It takes about an hour of elapsed time to do a release, most of which is the VM image running rmic, javadocs and the tests.*

2.3.Freeze the code

Do not mix cutting a release with last minute changes to the code. Those last minute changes will be inadequately tested, incompletely thought through, undocumented, or some other source of trouble. If you pause the build while a change/fix goes in, then more changes will pile up, you won't cut a release until the evening and you will rush it out without enough testing. Then, when it doesn't work, it will be whoever made the release that gets the blame, not whoever made the last minute change.

> *Do not allow last minute changes, either to source or to build files/process.*

If last minute changes must go in before the release is made, halt the release to the following day. Take in the changes, add the tests, make sure everything is working and then cut a release with the code. Give whoever made the changes the responsibility of adding the tests and the documentation, before it will be accepted in the release.

> *Don't mix changes with cutting a release, it only leads to disaster.*

Requiring a slip of day whenever a last minute change is made has another consequence: it discourages last-minute changes. It is much better to release regularly, every two weeks, than try and get everything in for a build.

2.3.1 Freeze the batch files/scripts earlier

The launcher scripts turn out to be extremely prone to causing surprises when changed. If the batch files or shell scripts have been changed *in any way*, you can't release on the same day. They need to be tested by our internal users before shipping.

2.4.Verify that the automated build is happy

If the build or tests are failing, you can't release. So check the Hudson servers to make sure the build is passing all its tests.

If tests are failing, halt the release process, fix the problems, and return to step 2.1. Do not try and release while tests are failing.

> *Exception: tests that everyone agrees are acceptable to ship with broken.*

Broken tests must be turned off as follows

2.4.1 How to turn off a test

The process for turning off a test is documented in the wiki.

1. Note the name of the test.
2. Open a bug report, record that the test has failed. Give it priority *major*, file it against the forthcoming release, and that it should be fixed in the current branch.
3. Assign the bug to whoever maintains that code, or find someone to own the problem.
4. Add the bug to the failing-tests pattern in the `testharness/build.xml` file. Navigate to the `"init-tests"`

target to find this pattern. Exclude the specific failing test class unless `run.failing.tests` is set. Add the JIRA bug ID of the bug filed about this test failing.

Do not neglect tests that fail. Someone needs to be fixing them. There should be no more than one or two failing tests, and they must be fixed by the next release.

2.5.Check out the repository

Check everything out (`svn update`) and record the SVN revision number.

2.6. Create a new JIRA task for releasing the build.

You can use this for any commits you make during the release process (new release notes, version counters, etc.).

2.7.Set the release version

Select a project release version, split into categories

value	*meaning*	*example*
`sf.majorRelease`	Major version number	3
`sf.minorRelease`	Minor version number	12
`sf.build`	build counter	003
`sf.status`	release status	beta

These versions are defined in `core/smartfrog/sf-build-version.properties`, and are used to set the SmartFrog version inside `smartfrog.jar`, and it is reused when publishing the artifacts to the local Ivy and Maven repositories. To set the status to nothing, just set the line to

```
sf.status=
```

The current policy is have a separate number for the release than for the working version. So you can develop on 3.12.003dev, but the release version should be 3.12.004 *purely for the release*. As soon as the release is cut, the version number would go to 3.12.005dev.

2.8. If you you need to, rebuild the PDFs

There is a target in `core/build.xml` to turn `doc/*.sxw` in every project into PDFs,

```
ant pdf-doc-folder
```

For it to work, the `.vbs` file in `components/documentation/doc` must be added to an OpenOffice library called SmartFrog, as the module `Utils`. It takes a couple of minutes, but the host's desktop is unusable as every document gets opened and turned into PDF.

You can run the same Ant target in a specific module, to update only that module's PDFs.

After creating the PDFs, check them in (use your “create a new release” JIRA task number), then do another `svn update`, to give a new SVN revision number.

2.9.Create the release

In the `core` directory, run:

```
ant release
```

This will create a clean release of everything, using the new version number.

2.10. Test the release

In the `core` directory, run:

```
ant test
```

This will test everything. Any test failure must halt the release process until the cause is identified. Any code changes trigger a restart of the process.

2.11.Create the release notes

In the core/release directory, run something like (changing the revision number):

```
ant create-release-notes -Dsvn.revision=4800
```

This will create a set of release using the new SmartFrog release number (which is inferred from the last compiled SmartFrog version), and the `svn.revision` number from the `svn update` command.

Edit the newly-created release notes in `/release/doc/announcements/` to provide information on what has changed.

2.12. Test that the RPMs work

Run the `core/release` RPM remote test against a configured RHEL target, usually a VMWare image:

```
ant rpm-remote-test -Dserver=skye
```

2.13.Test that the izpack JAR works

In `core/release`, run the `run-no-rebuild` target to check that the izpack installer works. This will bring up the install GUI for the just-created JAR

```
ant run-no-rebuild
```

Check the version strings are correct, that the dates in the license are current, and that the package list includes everything you wish to bundle.

2.14.Tag the repository

In `core/release`, tag the repository with the target revision number

```
ant svn-tag -Dsvn.revision=4800
```

Important: this will only work if you already have SVN authentication set up on this machine from the command line. Do an svn commit of some updated file -if it prompts you for a password, enter it. Only once you have done that will the ant-executed SVN operations work. And you need the svn tools on the command line.

2.15.Upload the new release

To upload the Maven artifacts to SourceForge for serving, and FTP of the main redistributables for building a release, run the core/release targets:

```
ant upload -Dserver=sourceforge-steve
```

This must be followed by the (manual) release process described later. A different server definition is needed when someone else is making the release, obviously.

Release the artifacts through SourceForge

See the *Uploading to SourceForge* section below. You need to go to

`https://sourceforge.net/project/admin/editpackages.php?group_id=87384`

3 Maven Repository Artifacts

Most artifacts are created by the individual components, and published to `core/build/m2/repository`.

The release target `maven-artifacts` will build a Zip file from those artifacts explicitly selected in the package-`maven-artifacts` target; this is a subset of those artifacts that the build creates. The zip file is then expanded into release/build/m2/repository, so that the behaviour of the tasks can be checked.

The file `http://maven.apache.org/guides/mini/guide-central-repository-upload.html` covers the long term plan: rsync integration with the ibiblio repository, For now, the zip file should be expanded into `shell.sf.net` under `smartfrog/htdocs/repository`, which can be viewed as `http://smartfrog.sourceforge.net/repository/`

4 The release VM

As mentioned, we have a VMWare image which can be used for cutting a release. This host can be obtained from Steve or Julio. It can be used to build the system, and to run the tests.

To build the system, the VM must have its proxy settings correct for Ant. Rather than fiddle with all the proxy settings, we have a special version of Ant, `ant-hp-proxy`, which patches in the proxy settings for HPLB (`web-proxy:8088`). You can use this directly, and its sibling, `ant-no-proxy`, both of which reside under `/home/smartfrog/Java/Apps/ant/bin`. Run `ant -diagnostics` to see what the proxy settings are, if you have any doubts.

You must get the networking settings for the machine right; it must be able to look up its own hostname. If it cannot connect to itself, then RMI will not work properly and the tests will fail.

5 Building the release

In the core directory, go

```
ant release
```

Then wait. Remember, if you are coding at the same time, those changes go into the following release!

6 Tagging the release

There is a target `svn-tag` in `core/release` that will create a subversion tag for the (nominated) subversion revision. This requires the executable `svn` on the command line.

1. Check in all the code.
2. Make sure you have the SVN command line client installed, and that command line commits work (i.e. server is authenticated and commits work)
3. Identify the revision to be used for the release, e.g. revision 4328
4. run `ant svn-tag -Dsvn.revision=4328`

5. Check that the revision is public by browsing to `http://smartfrog.svn.sourceforge.net/viewvc/smartfrog/tags/`.
6. The default tag, `revision${smartfrog.version}` can be changed by setting the `svn.tag` property
7. The default message can be changed by setting the `svn.message` property.

7 Reviewing the packaged artifacts

Do not be afraid to open up the `.zip` and `.tar` files to make sure everything looks correct. Check the `lib` and `dist/lib` directories of `core/smartfrog` to see that they do not contain outdated artifacts from previous builds. If they do, file bug reports (or reopen SFOS-162), and then fix the build process to ensure valid artifacts get pulled in.

The 3.11.000beta lacked the `.bat` files in the `.zip` and `.tar` redistributables; again, this is something that should be checked for. If we could have automated tests, then even better.

8 Update Jira

A Jira project administrator should

1. Review all open bugs against the release, and close any that are fixed, mark as WONTFIX any that you have no plans to work on (i.e. inactive components or tools).
2. Create the planned successor release, if it is not already present.
3. Mark the project as released; and move all open defects to the successor release.
4. You can generate a change log report which can be included in the release notes.

9 Release Announcement

Create a release note that can be included with the release. These are kept in `release/doc/announcements`; Use the "`create-release-notes`" target in `core/release` to create a text announcement for use with SourceForge release and in emails.

Make sure the release begins with a product description, as a release announcement on a site such as SourceForge or freshmeat may be the first place that people hear of the product. Check the spelling and punctuation, so as not to create a bad impression up-front.

Include the SVN revision number and branch label in the announcement. This is useful for the team's own use.

Jira can be used to create a report of all changes since the last release, which can be appended to the release notes

1. Browse to http://jira.smartfrog.org/jira/secure/ConfigureReleaseNote.jspa?projectId=10000
2. Select the version being released, and the style (text for text announcements, HTML for any HTML announcements)
3. Strip out any issues that are wontfix; they aren't changes.
4. Strip out issues that aren't related to components/artifacts that are not in the release.

Include the SHA1 checksums of the files in the release announcement. These are things to check when verifying that the mirrors are serving files correctly, and if the announcement is sent in a signed email, can be used to create a chain of trust.

10 Uploading to SourceForge

There's two aspects of SourceForge uploads: Maven artifacts and upload artifacts.

10.1.Automated upload

The release build file has targets to upload files, and to test that a connection can be made. Get the connection ready in advance of trying to cut a release, as debugging SSH problems can be time consuming.

Use the `ssh-test` target to test the connection without uploading anything.

```
ant ssh-test -Dserver=sourceforge-steve
```

You must have the key configured in `metadata/servers/server.properties` added to the `.ssh/authorized_keys` list in the remote server, otherwise you will see an error messages such as "`JschException : Auth Cancel`". A successful test run will look like the following

```
ant ssh-test -Dserver=sourceforge-steve
Buildfile: build.xml


release.load-server-settings:
     [echo]
     [echo]       SCP target is shell.sourceforge.net
     [echo]       User is steve_l
     [echo]       trust=true
     [echo]       keyfile=/home/slo/.ssh/sourceforge.private
     [echo]

release.ssh-ls:
[ssh-remote] Connecting to shell.sourceforge.net:22
[ssh-remote] antbook
[ssh-remote] build.xml
[ssh-remote] deployment
[ssh-remote] smartfrog

release.ssh-test:
     [echo] ssh appears to be working

BUILD SUCCESSFUL
Total time: 8 seconds
```

Once the connection is live, you can do an upload.

Run

```
ant upload -Dserver=sourceforge-steve
```

The `upload` target **does not rebuild the application**. You can do as many uploads as you like, to different places; the artifacts remain the same.

As the artifacts are big, a full upload can take time. Allow 30 minutes to do a release from work, and don't attempt it at home or on a WLAN.

The upload process uses SFTP, you must have the secret password set up in a build.properties file:

```
secret.sftp.password=UseYourSourceForgePassPhraseHere
```

If upload fails with an auth error, this is the first likely cause to check up on.

11 Uploading the release notes

The release notes can be uploaded into the same directory via the command

```
ant upload-release-notes
```

11.1.Manual Uploads

We have hit problems with sftp not finishing.

To upload by hand, here are the commands issued to push up the release 3.18.010, where sftp had already created the destination directory

```
cd build/upload
sftp steve_l@frs.sourceforge.net:/home/frs/project/s/sm/smartfrog/development/smartfrog-
3.18.010
put *.jar
put *.gz
put *.zip
quit
```

This is a bit lower level, but seems more reliable.

11.2.Maven artifacts

The `upload` target will automatically upload the artifacts to our SourceForge-hosted repository.

The artifacts can also be deployed manually by

1. copying the Maven distribution package (`dist/smartfrog-maven-artifacts.zip`) to `shell.sf.net` using `scp`.
2. ssh-ing in to that server and using `unzip` to expand the file into `smartfrog/htdocs/repository`
3. checking that they are present by browsing to `http://smartfrog.sf.net/repository`

Do not delete old artifacts as they may still be imported by external projects.

To upload only the Maven artifacts, run `ant upload-maven-artifacts` .The settings in `metadata/servers/sourceforge-stevel.properties` show what settings are needed. They work because in the ssh servers in SourceForge, the user has set up a symlink `smartfrog` that points to the project's own web site. This was done by issuing the following command

```
ln -s /home/project-web/smartfrog/ smartfrog
```

11.3.Redistributable artifacts

To upload files for download using the SourceForge mirrors, you must first SCP them to the servers, then publish a new release on the web. This is the most hands-on part of the deployment.

To upload from the command line, in the `core/release` directory, run `ant sftp-upload` with the property server set to a specific destination:

```
ant sftp-upload -Dserver=sourceforge-steve
```

The `upload` target depends on `sftp-upload`, so if you run `ant upload` you get this for free.

This is a good time to prepare those release notes, if you have not already. Note that the upload targets do not rebuild the application; this is to allow you to upload the same files. to as many places as you want.

12 September 2011: updated install process

After uploading the artifacts, you need to cut a release on SourceForge

1. Go to https://sourceforge.net/projects/smartfrog/ and then log in

2. Go to the files area

https://sourceforge.net/projects/smartfrog/files/development/

3. Find the new release subdir
   (e.g https://sourceforge.net/projects/smartfrog/files/development/smartfrog-3.18.012/)

4. go through the new artifacts and hit the "i" button to change their attributes

5. mark which are the defaults for which OS

Windows: zip

OS/X, BSD, solaris : tar.gz

Linux: rpms

12.1.Verifying the artifacts

After the upload has gone through, wait 24 hours for the changes to propagate through the mirrors. Then go to SourceForge and:

1. Take note of the link to the newly released artifacts; these can go into the announcement email.
2. Download all the artifacts from a mirror.
3. Use `sha1sum` to create the checksums, and verify (visually) that they match that of the published files

13 Public Announcements

The email announcement can use the same document as a template, but should be delayed until the artifacts are up on a SourceForge mirror, and their download has been verified.

The URL to the specific release must be inserted into the email announcement, so that the recipients have immediate access to the repository

14 Maintenance: adding new Artifacts to the release process

It is simple, albeit tedious, to add new artifacts to the packages. What is important is that you must (a) include every dependent JAR that you need to redistribute and (b) avoid accidentally claiming ownership of a JAR that another package manages. If you depend on other components, declare these dependencies in the izpack and RPM dependency files.

Do not do this as a last minute activity; it should be done the day before a release to stabilise.

The CI servers on Linux do build these artifacts, but do not test the installation; this is something that the release process does -so test it out before making a release.

14.1.Adding a package to izpack

Do not do this as a last minute activity; it should be done the day before a release to stabilise.

1. Open a new JIRA issue.
2. Add the new package to `ivy.xml`.
3. Verify (using the `ivy-report` target) that the libraries are pulled in. The `ivy-retrieve` target will pull all the JARs into `build/ivy/lib/${package}`, so you can check there too.
4. Verify that no dependent JARs that belong to other projects are being included in your lib/ directory. This creates confusion as to what is managing a specific package, confusion that breaks the RPM manager and izpack.
5. Edit the izpack file to add a new package. The configuration file is in `release/izpack/smartfrog-`

`install.xml`; add a new “pack” for the component, declaring all JARs that need to be included.

6. run the `izpack` target to create the new JAR.
7. run the `run` target to check that the izpack works.
8. Commit the changes against the JIRA issue
9. Mark the issue as fixed.

14.2. Adding a new package to the RPM

This is covered in the *creating and editing RPMs* document.

15 Releasing different Branches

If you wish to release different versions of SmartFrog, there are multiple ways to do this

1. Use `svn switch` to change to a different branch.
   This can leave files around, and get confused if what is a file in one branch is a directory in another.
2. Delete the current source, then check out, the different branch.
   This is slower but cleaner. It is strongly recommended.
3. Set up a separate VMWare image for the different branch
   This is best for maintaining ongoing branches, as it reduces the cost of switching from one version or another. It even allows you to have a different Java version for one branch than another.

When switching between branches, purge all SmartFrog artifacts from the Ivy cache, `~/.ivy`, by running the `ivy-purge` ant target in `core/`.

16 Troubleshooting

16.1. Upload fails when unzipping the Maven artifacts

SourceForge maintain a quota of filespace for a project, unless we purge old artifacts from our `htdocs/repository` space, they will eventually block the unzip process.

16.2. Upload Fails on SFTP upload

If you cannot connect to the upload service, it is because Ant doesn't know the password for logging in to the frs server:

```
sftp-upload:
     [echo] SFTP target is frs.sourceforge.net
      [scp] Connecting to frs.sourceforge.net:22

BUILD FAILED - at 20/06/08 14:38
com.jcraft.jsch.JSchException: Auth fail
        at com.jcraft.jsch.Session.connect(Unknown Source)
```

A key problem here is that this server does not pick up the private keys from the rest of the SF infrastructure, so Ant needs to know the password. This should be done via `build.properties` files that are not checked in. In our (checked in) `metadata/servers/sourceforge-steve-properties` file, the password is set to

```
sftp.password=${secret.sftp.password}
```

Unless the login to SourceForge is actually set to `${secret.sftp.password}`; this will fail. Define the true value for this password in an uncommitted properties file.

16.3.Upload Fails on Maven upload

We've seen errors failing to create a directory because it exists:

```
release.upload-maven-artifacts:
[ssh-remote] Connecting to shell.sourceforge.net:22
[ssh-remote] cmd : mkdir -p smartfrog/htdocs/repository
[ssh-remote] mkdir: cannot create directory `smartfrog': File exists
```

This is odd, as mkdir -p explicitly says "don't fail if the parents aren't there, but instead create them"

17 Setting up a machine for making releases

We build releases on a dedicated Virtual Machine. Here are the notes to create/recreate such a machine

1. Build up a specific Linux VM. Give it a large hard disk -10GB+, as you do not wish to run out of disk-space mid build.
2. If you wish to upload artifacts to SourceForge from the VM, give it external network access -either bridged or direct. Alternatively: do the final upload from the host machine. Test: ssh in to shell.sf.net as the target user,project; ssh in to frs.sourceforge.net as the target user,project.
3. Install the RPM build tools to create RPMs. Test: type `rpmbuild` on the command line and expect to see the error message
   `rpmbuild: no spec files given for build`
   Install the `rpm-build` RPM/deb package to get this tool
4. Install subversion; to test, type `svn`.
   Warning: you may want a later version than ships with the Linux distribution, as they tend to lag and be less good at handling moves and copies over branches.
5. Set up SVN proxies. If you have a machine set up already, the .subversion/servers file does this
6. Install the relevant Java runtime and JDK. Test: `javac`, `rmic` and `javadoc` are all recognised as commands. Typing `javac -version` should print the version of Java you were expecting.
7. Point `JAVA_HOME` at where the JVM lives, add `$JAVA_HOME/bin` to the path.
8. Install Ant:
   1 Download the latest release from `http://ant.apache.org/`
   2 Untar it into a location (here `/home/smartfrog/ant`)
   3 Edit `~/.bash_profile` to add

```
export PROXYHOST="web-proxy"
export PROXYPORT=8088

#ant settings
export JVM_PROXY_SETTINGS="-Dhttp.proxyHost=$PROXYHOST -Dhttp.proxyPort=$PROXYPORT
-Dhttps.proxyHost=$PROXYHOST -Dhttps.proxyPort=$PROXYPORT"
export ANT_OPTS="$JVM_PROXY_SETTINGS"
export ANT_ARGS="-logger org.apache.tools.ant.listener.BigProjectLogger"
export ANT_HOME=/home/smartfrog/ant
```

   4 Test with a `bash -login` and then type `ant -diagnostics`; you should see a diagnostics listing showing the many tasks are missing
   5 Pull down the missing artifacts

```
cd $ANT_HOME
ant -f fetch.xml -Ddest=system
```
(if you are behind a firewall, you will have to set up the proxy first in `ANT_OPTS`) ; you need this anyway

6 Rerun `ant -diagnostics`; you should see a diagnostics listing showing less optional tasks are unavailable

9. Create the directory below which you wish to host the SmartFrog source tree.
10. Check out SmartFrog from the repository
```
svn co https://smartfrog.svn.sourceforge.net/svnroot/smartfrog/trunk/core
```
11. In the directory `core`, run `ant release`, as discussed previously in this document.
12. To do releases, you need to get the right ssh keys into `~/.ssh`. Obviously.