How to be a Forrest developer

This How-To provides some tips and procedures for being a Forrest developer.

Table of contents

1 Intended Audience....2
2 Purpose....2
3 Prerequisites....2
4 Development techniques and scenarios....2
4.1 Development environment....2
4.2 Using Subversion....3
4.3 Editing content....4
4.4 Understanding the Cocoon sitemap....5
4.5 Debugging and profiling techniques....6
4.6 Finding the relevant sources....8
5 Frequently Asked Questions....8
5.1 1 General issues....8
5.2 2 Other issues....9
6 Tips....9
6.1 Explanations about howto topics on the mailing lists....9
7 References....9

1. Intended Audience

Warning:
This document is under initial development.

People who are ready to go beyond the basics of using Forrest. This might be to utilise Forrest for your advanced needs, debugging, creating a new plugin, enhancing an existing plugin, enhancing the core capabilities, contributing such enhancements back to the Apache Forrest project, etc. In all cases, this is what we mean by "developer".

Actually, users will also find that some parts of this document are useful. For example, the section about debugging and the section about editing content.

2. Purpose

This How-To provides some tips and procedures for being an Apache Forrest developer. Ideally a developer would also contribute back to the project, so these notes assume that. Various key development tasks are used as worked examples.

This document is intended to be an introduction to developing in Forrest, specifically, for those developers without a strong Cocoon background. Some key concepts in Forrest like sitemaps, pipelines, and locationmaps can be challenging enough to understand without also struggling with the fundamental development chores of debugging, extension methods, etc. The goal of this document is to reduce the steep learning curve by providing answers to some really practical Forrest development questions.

3. Prerequisites

- You have achieved the basic level of using Forrest. You have Forrest installed and can create a new site with 'forrest seed'. You have followed at least the first parts of the Using Forrest document.
- You will enventually see that understanding of the Cocoon sitemap is important. For the initial examples below, you can do without that. However please explore the sitemap soon.

4. Development techniques and scenarios

Various scenarios are utilised to describe aspects of development. Bear in mind that there are many ways to do things. Each developer has different tools and habits, and different operating systems are used. So you will need to glean the principles and apply them to your own situation.

This document assumes that you intend to contribute some parts of your work back to the project. Techniques for network-based collaborative development are encouraged.

4.1. Development environment

There is no *proper* dev environment. It really depends on your personal preferences. In an opensource project there is huge variety of environments. This makes it quite a challenge to keep

sources consistent (see discussion below).

Some people use vi or emacs, others use graphical editors, others use integrated development environments such as Eclipse or IDEA.

Ensure to configure your xml editor to take advantage of the local sets of DTDs provided by Forrest. This also explains how to use xml validators such as 'xmllint'. If your editor of choice doesn't validate XML, then most XML validation issues can be discovered using `forrest validate-xdocs`

There really isn't much Java code in Forrest, but a Java Development Environment such as Eclipse or any text editor of your choice will work just fine. If you point Eclipse at the Forrest build file it will make life easy for you.

4.2. Using Subversion

The Subversion source control system is used for all ASF projects. You can leverage this to ease your own development.

The "trunk" is where all new development and bugfixing happens. We aim to keep the trunk usable at all times.

Each version release is a "branch", such as "forrest_07_branch". Crucial bugfixes are also applied to the relevant release branch.

Branches are also used for developing complex new code which would otherwise disrupt the trunk. When the new work is suitable, then that branch is merged back to the trunk as soon as possible.

To get started, see the instructions for obtaining the Apache Forrest sources via SVN.

Whether you use the svn command-line client or a fancy client, then you still need to make sure you know how to operate SVN properly. http://svnbook.red-bean.com is a must.

4.2.1. Multiple working copies

Most developers would have a number of separate SVN working copies. Hopefully you are brave enough to use the trunk for all your sites. Sometimes that is not possible, for example when you are co-operativley managing a site with other people who are not so brave, so you need to use a specific release. However consider using the SVN release branch, rather than the release archive (tar or zip). This enables you to easily keep up with bugfixes. You can also easily see what local changes that you have made by using 'svn status; svn diff'.

Here is one layout ...

```
[localhost]$ ls /svn/asf
forrest_07_branch
forrest-trunk
```

4.2.2. Watch email notifications for svn differences

Either subscribe to the project's svn mailing list or monitor it via one of the mail archives. This enables you to be immediately up-to-date with changes to the repositories. The svn differences (diffs) are automatically sent whenever a committer makes some changes.

4.2.3. Updating svn backwards to find where something broke

Sometimes the addition of new features will break something. Often it is difficult to find where the break occurred and what caused it. Updating your svn backwards will enable this.

Look at the svn@ mail list to guess which change might be the culprit, e.g. svn revision 406862.

```
Update backwards to just before the upgrade:
svn update -r 406861
... do 'build clean; build' and test.

Go back further if it still doesn't work.
After success, start upgrading forward.

Looking at the svn@ mailing list shows that
could jump forward past minor changes such as doc edits.
e.g. just after the upgrade:
svn update -r 406863
... do 'build clean; build' and test.

If it still works, move a bit further on:
svn update -r 407260
... do 'build clean; build' and test.

After finding the break, move back to head of trunk
svn update -r HEAD.
```

4.2.4. Creating patches

See instructions for creating and contributing patches. Make sure to do three things before creating the patch:

- Do 'svn update' to be in sync with the repository in case someone else changed your files in SVN.
- Do 'svn status' to ensure that you are not including unnecessary changes.
- Do 'svn diff' to ensure that changes are what you expect.

After creating the patch, open it with a text editor and review it.

4.2.5. Tips

- Keep a copy of this book, or the online version, close at hand: Version Control with Subversion - the opensource SVN book.
- See all available branches and other repositories: http://svn.apache.org/repos/asf/forrest/
- Use online repository browsers to quickly see past activity for the files that you are working on: http://svn.apache.org/viewcvs.cgi/forrest/trunk/
- Use 'svn log foo.xsl' for a summary of recent activity and to see dates and revision numbers for changes.

4.3. Editing content

See the FAQ. Basically any editor can be used, because Forrest treats the editing of content as a separate concern. Be sure to configure the editor to find local copies of DTDs.

4.3.1. Code style guidelines

Consistent code makes everyone's life easier. See the Apache Cocoon tips. We don't get too hung up on style, but those few basic things are important. However we know that coding style is mixed, so just

follow the same style as that which exists in the file you are editing. We can occasionally clean up later.

Don't change anything that is not necessary. Remember that people need to be able to read the differences on the svn@forrest mailing list.

4.3.2. Whitespace

For new files, use spaces instead of tabs (java files have four-space indentation, xml files and other text files have two-space indentation).

Don't let your editor automatically change the whitespace for existing files.

We know that many files in SVN do not have consistent whitespace. This issue is continually being addressed. Please don't attempt to rectify whitespace mixed up with other changes. This makes the important changes difficult to see. Occasionally committers will rectify whitespace for a set of files, when they know that no-one else is working on that set.

FIXME (open):

The issues of whitespace and line endings needs to be very clearly described. See some mail discussion references.

4.3.3. Line length

If each paragraph of an xml source document is one enourmous long line, then it is extremely difficult to know the changes with the SVN diffs. Developers and especially committers, need to be able to efficiently review the changes. Fold long lines to a sensible line-length (normally 80-characters wide but not more than 120 characters).

4.3.4. Use 'forrest run' for immediate gratification

Edit documentation content and immediately view it in the browser. When you are satisifed, then do 'forrest site' to ensure that the whole documentation set hangs together and there are no broken references.

In the dynamic 'forrest run' mode, you will get some feedback about some xml validation errors. However, it is better to treat validation as a separate concern. Use an xml editor or command-line tools such as "xmllint". As a last resort, you can use 'forrest validate-xdocs'.

4.3.5. Tips

- ####

4.4. Understanding the Cocoon sitemap

The Cocoon sitemap is essential for Forrest developers. See some introductions ....

- Forrest sitemap reference.
- Introduction to Pipelines in this How-to.
- About Forrest project sitemaps.
- Cocoon concepts.
- Cocoon sitemap.
- Cocoon protocols, i.e. cocoon:/ and cocoon:// and context:// and resource:// etc. and the file://

4.5. Debugging and profiling techniques

4.5.1. View logfiles

The log files in WEB-INF/logs are indispensible when it comes to debugging sitemaps. While ERRORs will generally always print in the log files, while you're debugging something you may find it useful to also customize log output in the "logkit.xconf" in webapp/WEB-INF and raise the level of some loggers to WARN.

This FAQ describes the location of the Cocoon logfiles and their configuration.

4.5.2. View intermediate processing

Perhaps the easiest way to "debug" Forrest is to be aware of all the processing steps between taking in the source and outputting the end result. When you know these you can do 'forrest run' and request the intermediate documents, such as "http://localhost:8888/body-index.xml" which will return the intermediate processing of the body of the document.

The techniques described below help you to be aware of the intermediate processing steps.

4.5.3. Using Cocoon sitemap profiler

Cocoon provides a simple profiler to analyse itself. This enables us to list the various sitemap pipelines and components that are being used, how much time was used by each, whether each component uses the Cocoon cache, and show the actual xml data.

Note that the profiler is not used by default. To switch it on, edit `main/webapp/sitemap.xmap` and search for "profiler". Follow the instructions there to replace the standard "map:pipe" components with the profiling pipes.

Now start your application as normal using 'forrest run' and request localhost:8888/index.html three or four times to populate the profiler with data.

Now request the special uri localhost:8888/cprofile.html to see the results. Start at the "index.html" request, then follow the processing. (If the table is empty, then you either forgot to do some requests before looking for results, or forgot to switch on the profiler in sitemap.)

NOTE: Do not forget to turn off the profiler in `main/webapp/sitemap.xmap` when finished.

4.5.4. Using Cocoon sitemap execution logger

In main/webapp/WEB-INF/xconf/forrest-core.xconf search for "sitemap execution" and uncomment the component. For each sitemap component that is executed, a message will go to WEB-INF/logs/debug.log

4.5.5. Using Cocoon LogTransformer

LogTransformers can be inserted in the sitemaps. This will write the sax events at that point into a named log file. Here is an example (the logfile will be written relative to this particular sitemap) ...

```
<map:match pattern="*.html">
  <map:generate src="sources/{1}.xml"/>
```

```
  <map:transform type="log">
    <map:parameter name="logfile" value="my-1.log"/>
    <map:parameter name="append" value="no"/>
  </map:transform>
  <map:transform src="stylesheets/source-to-table.xsl"/>
  <map:transform src="stylesheets/table-to-page.xsl"/>
  <map:transform type="log">
    <map:parameter name="logfile" value="my-2.log"/>
    <map:parameter name="append" value="no"/>
  </map:transform>
  <map:transform src="stylesheets/page-to-html.xsl"/>
  <map:serialize type="html"/>
</map:match>
```

Another use for this technique is when you are not sure which path is being taken through the sitemap. Add various LogTransformers with different filenames and see which one gets triggered.

4.5.6. Using Cocoon Validation Transformers

Validating Transformers can be inserted in the sitemaps to validate the xml stream at that stage. Enables RELAX NG validation and W3C XML Schema validation using Jing and Xerces.

The Validation Block is already added to Forrest and configured. To use it simply add entries to your sitemap like this:

```
...
<map:transform type="validation-report"
  src="{forrest:context}/resources/schema/relaxng/unstable/any.rng"/>
...
```

See Pier's note to cocoon-dev and Cocoon documentation: ValidatingTransformer and ValidationReportTransformer.

4.5.7. Java code

There are two ways to debug your java code. The first is through embedded logging, the second is by tracing the codes execution.

You may find `getLogger().debug()` useful for understanding the processing that goes on. You can then open the page that will use the selected code and use the log files mentioned above to look for the information message.

To step through the java code you need to run forrest with Java debugging turned on. The `forrest.jvmargs`code> property in the `forrest.properties` file can be used to start forrest in debug mode on a specific port. For example:

```
forrest.jvmargs=-Xdebug
-Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
```

4.5.8. Finding broken internal links

Do 'forrest site' to produce the whole documentation set. Cocoon will report its progress and reveal any problems. This FAQ explains the messages that Cocoon sends to standard output. Broken links are also reported to a special file, which also shows the source file containing the break. The location of this file is reported when Cocoon starts.

Broken links are also reported in 'forrest run' mode. Use your mouse to point to each link. The browser status bar will show "error:..." instead of the actual URL.

The most common cause is that the entry is missing in the site.xml configuration file or the link in your

source document is not using the correct name for the "site:..." value.

4.5.9. Tips

- Doing 'forrest -v' will provide verbose build messages to the standard output.

4.6. Finding the relevant sources

You will need to be able to find which sources, sitemaps, stylesheets are responsible for certain processing.

4.6.1. Scenario: How does i18n work

We will do a search for "i18n" to start with, then refine that after exploring some of the sources.

The UNIX tools find, grep, and sed are very powerful. We need a helper script, otherwise 'find' is going to report matches for the hidden .svn files and also files in /build/ directories.

```
echo "sed '/\.svn/d;/\/build\//d;/\/work\//d'" > ~/bin/exclude-find-svn
chmod +x ~/bin/exclude-find-svn
```

Now we will run find, use grep to search for the pattern in each file and list the filenames. However, there is a stack of forrest.properties files from the plugins, and there is i18n:text being used in the viewHelper plugin, and some DTDs. So weed them out ...

```
cd /svn/asf/forrest-trunk
find . -type f | xargs grep -l "i18n" | ~/bin/exclude-find-svn \
 | grep -v "forrest.properties" | grep -v viewHelper | grep -v "\/schema\/"
```

The list of files shows that there is an FAQ about i18n, there are various sitemaps in main/webapp/, some stylesheets in main/webapp/skins/common/ and pelt, some other stylesheets in main/webapp/resources/stylesheets/ ... we will look at the sitemaps first. Use grep to list the actual matches and the filenames.

```
cd main/webapp
grep i18n *.*
```

Shows that five sitemaps are involved in some way. Always start with sitemap.xamp and forrest.xmap as they do initial processing and then delegate to other sitemaps. Open each file in your editor, and search within for each "i18n" match. See that the xslt transformer is declared to use i18n, then further down the page the "skinit" pipeline uses the i18n transformer only if i18n is switched on.

4.6.2. Tips

- ####

5. Frequently Asked Questions

5.1. 1 General issues

5.1.1. 1.1 FAQ 1

####

5.2. 2 Other issues

5.2.1. 2.1 FAQ 2.1

###

6. Tips

This is a collection of general tips that do not fit in the sections above.

6.1. Explanations about howto topics on the mailing lists

Often there are useful discussions on the mailing lists which explain how to do certain tasks. If you don't have time to summarise that and add to this howto document, then please consider just adding a tip which links to the email discussion. Later someone else can summarise.

7. References

These are some other documents that are useful for developers.

- ###