Libre QuickStart

This document is the current full documentation on the "libre" generator that was implanted into xml-forrest.

Table of contents

1 Intro

> **Warning:**
> This document is still relevant for ideas and potential solutions. However, the experimental code for Libre was removed from the scratchpad on 2003-04-18 during spring cleaning. If you want to resurrect it, then use the cvs tag "before_libre_departure".

The libre idea was born out of the cocoon book.xml itch. The actual need to start scratching was introduced by the higher volume of book.xml-editing-work that came along with the cocoon documentation and xml-forrest efforts.

The single idea behind it in fact is trying to automatically generate part of the navigation tree which is held now in the different book.xml 's. This automation effort however is held back by the lack of meta-data you can extract from the filesystem itself. This is why the libre approach still requires you to add this extra metadata using some libre.xml file. This libre.xml however has the following main advantages over the book.xml:

- The settings are 'inherited' down the directory tree, so you do not need a libre.xml on each directory level. You only need it to change the subdir traversing strategy from its parent dir.
- It combines some 'filesystem-introspection'-like declarations that are used in run-time filtering, sorting and attributing decisions. Introspection strategies are currently based on either (1) reading properties of the java.io.File object at hand, or (2) executing xpath expressions on the pointed at XML file.

2 Using Libre now (0.0 alfa)

> **Warning:**
> Disclaimer: most of what you read below is 'how it was intended' . To what extent that matches the actual execution process is largely dependent on my programming skills and thoroughness of testing.
> In other words: don't expect a thing unless you've seen it work. (at this time)

2.1 Generated Output

The XML output that comes out of the generator largely follows this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://outerx.org/yer/hierarchy/0.1">
  <collection label="content">
    <collection label="xdocs">
      <item label="dreams.xml"
               href="src/documentation/content/xdocs/dreams.xml"
               title="Forrest dream list"/>
      <item label="faq.xml"
               href="src/documentation/content/xdocs/faq.xml"/>
      <item label="book.xml"
               href="src/documentation/content/xdocs/book.xml"/>
      <item label="contrib.xml"
               href="src/documentation/content/xdocs/contrib.xml"
               title="Contribution to Forrest"/>
      <item label="mail-archives.xml"
               href="src/documentation/content/xdocs/mail-archives.xml"
               title="Mail Archives"/>
      <item label="mail-lists.xml"
               href="src/documentation/content/xdocs/mail-lists.xml"
               title="Mailing Lists"/>
      <item label="license.xml"
               href="src/documentation/content/xdocs/license.xml"
               title="The Apache Software License"/>
      <item label="index.xml"
               href="src/documentation/content/xdocs/index.xml"
```

```
               title="Welcome to Forrest"/>
      <item label="who.xml"
               href="src/documentation/content/xdocs/who.xml"
               title="Who we are"/>
    </collection>
  </collection>
</collection>
```

And it's not getting any harder in fact: only 2 elements, `<collection>` and `<item>` and that should do. The first maps to a menu-group in the navigation, guess what the second maps to?

The number and value (and its meaning) of the attributes on these elements are specified in the libre.xml file.

2.2 libre.xml Contents

That libre.xml file follows the src/resources/schema/dtd/libre-v10.dtd. In fact the current release allows for some extra elements (I'll explain where) and some unrestricted attribute CDATA types that cause some extensible xml output resp. some java-introspection to be triggered. So basically the DTD will be limiting you more than the runtime interpretation. (future versions will try to narrow this down seriously, main reason is that a more elaborate DTD allows for more XML-editor assistance in editing the files.)

The dtd:

```
<!ELEMENT libre (entry | auto)*>
<!ELEMENT entry (label?, href?)>
<!ATTLIST entry
  location CDATA #REQUIRED
>
<!ELEMENT auto (filter?, sorter?, label?, href?)>
<!ELEMENT label (xpath | property)>
<!ELEMENT href (xpath | property)>
<!ELEMENT filter (xpath | property)>
<!ATTLIST filter
  logic (inverse | normal) "normal"
  clear (yes | no) "no"
>
<!ELEMENT sorter (xpath | property)>
<!ATTLIST sorter
  order (ascending | descending) "ascending"
  clear (yes | no) "no"
>
<!ELEMENT xpath EMPTY>
<!ATTLIST xpath
  expression CDATA #REQUIRED
>
<!ELEMENT property EMPTY>
<!ATTLIST property
  name CDATA #REQUIRED
  mask CDATA #IMPLIED
  regex CDATA #IMPLIED
  substitute CDATA #IMPLIED
>
```

2.2.1 Building Blocks

The following elements get the following meaning when interpreted by the LibreConfigBuilder

```
<libre xmlns="http://outerx.org/libre/config/0.1">
```

- This is one of those libre.xml files, that will configure how items are filteres, sorted and attributed

```
<entry location="[relative location path]" />
```

- Allows to manually sort out specific files or directories.

- Comparable to standard book.xml behaviour, except for the fact that
  - libre doesn't yet support external hrefs (should be easy though)
  - there is no difference between `<menu>` and `<menu-item>`, there just is `<entry>`. It will become a `<collection>` or `<item>` in the output based on the fact if the location points to a directory resp. a file.
  - For locations that point to a filter it doesn't make sense, but when it points to a directory it is nested `<filter>` and `<sort>` elements get inherited down to the next level.

FIXME (mpo):

This last remarks actually means (1) I need to update the DTD to reflect this and (2) check the code for actually doing this.

```
<auto>
```

- Automatically generates more `<collection>` and `<item>` elements in the output, based on the specifications of the nested elements: `<filter>` (which resources?) and `<sort>` (in which order?).

```
<filter logic="inverse" clear="no">
```

- This element wraps a so-called AttributeReader (there are currently two of them: `<xpath>` and `<property>`)
- The AttributeReader is going to specify which information-element is going to be retrieved from the file or directory it is pointing at. Depending on which one is used this wrapping filter will test for presence or regex match of the resource being read. Based on the outcome of this test (true or false) the passed file will be accepted or not in the list.
- This wrapping filter element allows to inverse the acceptance-logic (accept what normally should be rejected and vice versa).
- Using the `clear="yes"` attribute stops the inheritance of the used filter strategy from the parent directory. Instead the default filter strategy (which is to accept all files) is slided in at this level.

```
<sort order="descending" clear="no">
```

- This element wraps a so called AttributeReader (there are currently two of them: `<xpath>` and `<property>`).
- The AttributeReader is going to specify which information-element is going to be retrieved from the file or directory it is pointing at. This information element will be considered to be a simple Key-String and `<collection>` and `<item>` elements in the output will appear in the order defined by the alphabetic sorting of these keys.
- This wrapping sort element allows to reverse the order. (z->a instead of a->z)
- Using the `clear="yes"` attribute stops the inheritance of the used sort strategy from the parent directory. Instead the default sort strategy (which is to use default filesystem sorting, alphabetic on filename) is slided in at this level.

```
<label>, <href>, <YOURTAG>.... {AttributeDefinitions}
```

- The remainder of the elements inside the `<auto>` tag specify the attributes that need to be applied to the generated `<collection>` and `<item>` elements in the output: `<item label=".." href=".." YOURTAG=".." />`
- There is currently only support for adding attributes, not nested elements.
- These elements all wrap a so called AttributeReader (there are currently two of them: <xpath> and <property>)

- In these cases the wrapped AttributeReader is going to specify which information-element is going to be retrieved from the file or directory it is pointing at. This information element will be considered to be a simple String-value that gets slided in as the corresponding output attribute value.

```
<xpath expression="/document/header/title/text()">
```

- This element specifies an xpath AttributeReader to use inside `<filter>`, `<sort>` or {AttributeDefinitions}.
- It allows to specify an xpath expression that should result in one single text node to be returned when applied to the root node of the xml file at the location of any given entry. The contents of this text-node is the string value to sort (`<sort>` usage) or to fill in the specified attribute (`<label>`, `<href>`... use). When inside a `<filter>`: the presence of the node results in passing the test.

> **Warning:**
> This currently breaks for non xml (`*.gif`) files, so get your filter right please, and in the mean time: sorry for not being able to use it in the filter yet `:-(`.

```
<property name="path" regex="(\.[\\/])*(.*)" substitute="$2"/>
<property name="name"  mask="CVS"/>
```

- This element specifies an xpath AttributeReader to use inside `<filter>`, `<sort>` or {AttributeDefinitions}.
- It allows to specify a JavaBean-like property to read (this introspection behavior will probably not survive the future release) on the file at the 'location' of any given entry. The (object-)value of this property is automatically converted to a String (toString()) that becomes the value to sort (`<sort>` usage) or to fill in the specified attribute (`<label>`, `<href>`... use). When inside a `<filter>`, the test passes if the read property is not null or "".
- Furthermore this element allows to express more elaborate true-false tests (filter use) or regex substitution (other use) attributes:
  - combination of @regex with @substitute accounts for a s/{regex}/{substitute}/ kind of operation on the string property.
  - while @mask or @regex by their own (filter use) allow for a glob-mask or regex test to be applied on the read property.

2.3 Important Side Effects

There are some things that libre is doing that you should be aware of.

2.3.1 No libre.xml

When using an `<auto>` section, the filter will NEVER accept the `libre.xml` file to be in the generated output. You can however include a manual `<entry>` to point to the `libre.xml` file if needed.

2.3.2 No Duplicates

You can combine multiple `<entry>` and `<auto>` elements after each other. The system will make sure that the resulting list of `<collection>` and `<item>` will not contain duplicates. So the filters in `<auto>` sections lower down the `libre.xml` file can include already accepted files or directories, they will only show up once in the output.

2.4 Example Constructs

Adding sorting and filtering to the filesystem with libre becomes a subtle play with editable filesystem properties, smart XML content and `libre.xml` configs. This should be considered as the 'extended' contract between the following roles in the documentation system: the one choosing (or creating) the DTDs, the one applying those to create content and give the resulting files a name, the one that sets up the directories to store those files and writes the `libre.xml` files.

2.4.1 Sorting your files or your menu entries?

In every case the very pragmatic approach can become something like this:

```
+ content
  + xdocs
    + 010Topic
      + 010Foo
      + 111Bar
    + 050Aspect
    + NotInList
```

In combination with something that lives by the introduced alphabetic order, but yet hides the ugly number-prefixes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libre PUBLIC "-//Outerthought//DTD Libre Configuration V0.1//EN" "libre-v01.dtd" >
<libre xmlns="http://outerx.org/libre/config/0.1">
  <auto>
    <filter logic="normal">
      <property name="name" regex="\d{3}(.*)"/>
    </filter>
    <label>
      <property name="name" regex="\d{3}(.*)" substitute="$1"/>
    </label>
  </auto>
</libre>
```

Will produce an automatic list of entries (collections and items in the output) that

- `<filter>`: only resources which name starts with a 3-digit pattern
- No `<sort>`: in their natural filesystem order assured by the digit-prefix
- `<label>`: hold a label attribute that strips of the ugly number prefix

Of course the advantage over book.xml only comes when more menu items should be easily slided in later on, and/or deeply nested directory structures can all benefit from this same filenaming/sorting strategy.

2.4.2 Naming your files or asking them their name?

Given the poor expressiveness of the filesystem, the labels that need to show up in the menu can hardly remain the filenames they are now (specially if we're adding these ugly number prefixes). Instead we can sign a contract with the content writer to also provide the navigation system with a sensible name for his entry using XML metadata that the system will pick up using an xpath expression.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE libre PUBLIC "-//Outerthought//DTD Libre Configuration V0.1//EN" "libre-v01.dtd" >
<libre xmlns="http://outerx.org/libre/config/0.1">
  <entry location="dreams.xml" >
      <label>
         <xpath expression="/document/header/title/text()"/>
      </label>
  </entry>
  <auto>
```

```
    <filter>
      <property name="name" regex="\.xml$" />
    </filter>
    <sorter>
        <xpath expression="/document/header/title/text()"/>
    </sorter>
      <label>
          <xpath expression="/document/header/title/text()"/>
      </label>
  </auto>
</libre>
```

3 Next Libre (0.1)

> **Note:**
> Next libre is in fact largely in your hands... just drop the forrest-dev mail list a line, and see what happens...

3.1 Itches

There is quite a number of fast code patches that can/need to happen

- package renaming and restructuring (ideas welcome, but not top of mind)
- on same level: possible xmlns and/or elms/atts renaming on the generated output and the libre.xml file
- when compiling you currently get 4 stupid deprecation warnings that should be removed, in fact:
- LibreConfigHelper has a silly test in it to switch to own parser and resolver if there is no avalon component manager in the neighborhoud (historical reason is the testing outside cocoon with the command line util, which should become some kind of avalon based junit task: if you have a clue how to start this, throw it at us please.)
- xpath property reader needs to survive working on a non-xml document (by returning nothing rather then breaking on the exception)
- general robustness and resilience towards mis-configurations
- filestreams need to get closed and avalon resources need to be released properly
- caching at the level of the generator needs to be set up
- in fact general performance has not been subject to loads of early optimizations :-P

3.2 Upcoming Features

More importantly however there is a major set of new features that is waiting to get in there. It all boils down in fact to having a more expressive libre.xml file... some of the thoughts:

3.2.1 Combinations of filter logic

Some itching stuff:

- logic="inverse" on the <filter> element seems a bit awkward
- *n*th degree of slickness in the regexes will only bring us so far, combinatory filter logic seems to be the way to go...:

```
<!ELEMENT filter (xpath | property | and | or | not)>
<!ELEMENT not    (xpath | property | and | or | not)>
<!ELEMENT and    (xpath | property | and | or | not)+>
<!ELEMENT or     (xpath | property | and | or | not)+>
```

So we can make up some richer:

```

<filter>
```

```
  <not>
      <and>
      <xpath .../>
      <not><property ..../></not>
      <or>
         ...
      </or>
    </and>
  </not>
</filter>
```

3.2.2 Separating property-retrieval from formatting and testing

Playing around with the attributes in `<property>`:

- poses hard to explain combinatory effects (@regex with @substitute vs without, @regex can't be combined with @mask, different behaviour inside <filter>== test or <sort>==formatting)
- which in fact are hard (if not impossible) to rule out by modifying the DTD
- makes you wonder why it's not available on the <xpath> ?

So maybe an example more down the lines of the following would be easier to use:

```
<label><!-- same applies for the sort context -->
  <regexformatter exp="..." substitute="....">
    <property name="absoluteLocation" />
  </regexformatter>
</label>
```

Allowing the formatter to be used around the xpath reader as well. And opening up the possibility to maybe format other stuff than Strings: `<dateformat format="dd/mmm/yy">`

It would also clearly distinguish the semantical difference of applying a test in the `<filter>` context:

```
<filter>
  <regextest match="...">
    <property ... />
  </regextest>
</filter>
```

And more logically introduce other tests like `<globtest match="...">` or `<availabletest>` or...

3.2.3 Replace the introspection with semantically richer named properties to read.

Currently the `<property name="someJavaBeanProp">` is applied in a java introspection for the `getSomeJavaBeanProp()` on the `java.io.File` object that is actually representing the node in the hierarchy at any given time. The DTD declares the attribute as of type CDATA. These decisions however:

- lead to a lesser user guidance for the libre.xml writer using an XML (and DTD) savvy editor
- leads to assuming the `libre.xml` editor has access to and knows how to interpret jdk javadoc
- leads to poor semantical support and thus more possible RUNTIME errors for those just filling in some valid CDATA value that is not mapping any getter.
- leads to confusion for all, since who actually knows the subtle difference between all the get*Path methods on java.io.File?

So the big idea here would be to go for an upfront declared list of sensible and clearly defined properties that we would like to read... Today's ideas about that list:

- name
- isDirectory (isCollection?)

- abs and relPath (or abs/rel Location? why would we need abs?)
- canRead
- canWrite
- lastModified
- length

The DTD would then list the possible attributeValues.

3.3 Avalonising

There are a number of perceived opportunities in taking up a stronger dependecy towards Avalon. Some of the possibilities become clear when looking into the current design...

- Currently the EntryFactory is a abstract factory, the factory part could be done by an Avalon Component manager. Which would also allow the EntryFactory to become a cleaner component interface then it is now.
- Some investigation/feedback on the current hacker-way of using the Composables could be nice
- The current cli part in the package is only there for testing (avoiding the cocoon webapp cycle when developing/testing) it should be replaced by a more formal test class that actually would take up the role (probably delegate to ECM or the like) of the componentmanager to give the HierarchyReader the (avalon) environment he needs.

3.4 Unresolved Discussions

- do we need support for nested elements inside `<item>` output (retrieved by e.g. xpath expressions)?
- do we need an extra `<constant>` like attributereader that would allow like book.xml to add fixed values for expressed attributes
- clear set out inheritance rules, just doing 'something' now :-(
- votes on needed file properties to replace the current (limiting and semantically poor) Java-introspection

4 Libre Design

So why is that silly 'yer' package name in there? Yer originally was some all-hierarchy-structures to SAX event thing, and since some of that is in here as well, we kind of picked that idea up out of the dustbin.

So reflecting the current packagenames we kind of have these sets of responsibilities

- **.yer.hierarchy*: describe in a formal way how hierarchies should be built up in order to have them dumped to XML using the HierarchyReader.
- **.yer.use.cocoon*:house of the generator. It basically just gets a reader and subscribes the next ContentHandler in the cocoon pipeline to the HierarchyReader that it is using.
- **.yer.impl*: hold the different implementations of the *.yer.hierarchy API
- **.yer.impl.fs*: (only current impl) Build the described filesystem oriented implementation of the hierarchy. It is using the libre configuration strategy.
- **.yer.libre*: provide a generic strategy for adding filtering, sorting and attributing information to a hierarchy through the use of XML config files (in an XML configuration/declarative manner)

... hope this somewhat clarifies how things have been setup for now.

4.1 Dependencies

- The regex stuff inside libre adds the dependency upon the oro package. Basically I failed to find substitution support inside the regex package (which is already in cocoon) in a timeframe comparable to just get on with this using oro.
- The HierarchyGenerator is the first one in the chain (and the last in fact) that actually needs the cocoon package (at least it was intended this way, could be that there are some glitches on this statement)
- There is a sort of false dependency on Avalon right now (some Composables in there, no real container stuff though). As expressed higher there are some plans to stronger benefit from this dependency.

