XML Validation

DTDs, catalogs and whatnot

$Revision: 1.19 $

1. XML validation

By default, Forrest will try to validate your XML before generating HTML or a webapp from it, and fail if any XML files are not valid. Validation can be performed manually by typing '`forrest validate`' in the project root.

For an XML file to be valid, it *must* have a DOCTYPE declaration at the top, indicating its content type. Hence by default, any Forrest-processed XML file that lacks a DOCTYPE declaration will cause the build to break.

Despite the strict default behavior, Forrest is quite flexible about validation. Validation can be switched off for certain sections of a project. In validated sections, it is possible for projects to specify exactly what files they want (and don't want) validated. Forrest validation is controlled through a set of properties in `forrest.properties`:

```
###############
# validation properties

# These props determine if validation is performed at all
# Values are inherited unless overridden.
# Eg, if forrest.validate=false, then all others are false unless set to true.
#forrest.validate=true
#forrest.validate.xdocs=${forrest.validate}
#forrest.validate.skinconf=${forrest.validate}
#forrest.validate.sitemap=${forrest.validate}
#forrest.validate.stylesheets=${forrest.validate}
#forrest.validate.skins=${forrest.validate}
#forrest.validate.skins.stylesheets=${forrest.validate.skins}


# Key:
# *.failonerror=(true|false)    stop when an XML file is invalid
# *.includes=(pattern)         Comma-separated list of path patterns to validate
# *.excludes=(pattern)         Comma-separated list of path patterns to not validate

#forrest.validate.failonerror=true
#forrest.validate.includes=**/*
#forrest.validate.excludes=
#
#forrest.validate.xdocs.failonerror=${forrest.validate.failonerror}
#
#forrest.validate.xdocs.includes=**/*.x*
#forrest.validate.xdocs.excludes=
#
#forrest.validate.skinconf.includes=${skinconf-file}
#forrest.validate.skinconf.excludes=
#forrest.validate.skinconf.failonerror=${forrest.validate.failonerror}
#
#forrest.validate.sitemap.includes=${sitemap-file}
#forrest.validate.sitemap.excludes=
#forrest.validate.sitemap.failonerror=${forrest.validate.failonerror}
#
#forrest.validate.stylesheets.includes=**/*.xsl
#forrest.validate.stylesheets.excludes=
#forrest.validate.stylesheets.failonerror=${forrest.validate.failonerror}
#
#forrest.validate.skins.includes=**/*
```

```
#forrest.validate.skins.excludes=**/*.xsl
#forrest.validate.skins.failonerror=${forrest.validate.failonerror}
#
#forrest.validate.skins.stylesheets.includes=**/*.xsl
#forrest.validate.skins.stylesheets.excludes=
#forrest.validate.skins.stylesheets.failonerror=${forrest.validate.skins.failonerror}
```

For example, to avoid validating `${project.xdocs-dir}`/slides.xml and everything inside the `${project.xdocs-dir}/manual/` directory, add this to `forrest.properties`:

```
forrest.validate.excludes=slides.xml, manual/**
```

Note:
The `failonerror` properties only work for files validated with <xmlvalidate>, not (yet) for those validated with <jing>, where `failonerror` defaults to `true`.

2. Validating new XML types

Forrest provides an OASIS Catalog [see tutorial] `forrest/src/core/context/resources/schema/catalog.xcat` as a means of associating public identifiers (e.g. `-//APACHE//DTD Documentation V1.1//EN` above) with DTDs. If you add a new content type, you should add the DTD to `${project.schema-dir}/dtd/` and add an entry to the `${project.schema-dir}/catalog.xcat` file. This section describes the details of this process.

2.1. Creating or extending a DTD

The main Forrest DTDs are designed to be modular and extensible, so it is fairly easy to create a new document type that is a superset of one from Forrest. This is what we'll demonstrate here, using our earlier download format as an example. Our download format adds a group of new elements to the standard 'documentv11' format. Our new elements are described by the following DTD:

```
<!ELEMENT release (downloads)>
<!ATTLIST release
version CDATA #REQUIRED
date CDATA #REQUIRED>

<!ELEMENT downloads (file*)>

<!ELEMENT file EMPTY>
<!ATTLIST file
url CDATA #REQUIRED
name CDATA #REQUIRED
size CDATA #IMPLIED>
```

The document-v11 entities are defined in a reusable 'module': `forrest/src/core/context/resources/schema/dtd/document-v11.mod` The `forrest/src/core/context/resources/schema/dtd/document-v11.dtd` file provides a full description and basic example of how to pull in modules. In our example, our DTD reuses modules `common-charents-v10.mod` and `document-v11.mod`. Here is the full DTD, with explanation to follow.

```
<!-- ============================================================

Download Doc format

PURPOSE:
This DTD provides simple extensions on the Apache DocumentV11 format to link
to a set of downloadable files.

TYPICAL INVOCATION:
```

```
<!DOCTYPE document PUBLIC "-//Acme//DTD Download Documentation V1.0//EN"
"download-v11.dtd">


AUTHORS:
Jeff Turner <jefft@apache.org>

COPYRIGHT:
  Copyright 2002-2004 The Apache Software Foundation

  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.

================================================================== -->


<!-- ================================================================== -->
<!-- Include the Common ISO Character Entity Sets -->
<!-- ================================================================== -->

<!ENTITY % common-charents PUBLIC
"-//APACHE//ENTITIES Common Character Entity Sets V1.0//EN"
"common-charents-v10.mod">
%common-charents;

<!-- ================================================================== -->
<!-- Document -->
<!-- ================================================================== -->

<!ENTITY % document PUBLIC
"-//APACHE//ENTITIES Documentation V1.1//EN"
"document-v11.mod">

<!-- Override this entity so that 'release' is allowed below 'section' -->
<!ENTITY % local.sections "|release">

%document;

<!ELEMENT release (downloads)>
<!ATTLIST release
version CDATA #REQUIRED
date CDATA #REQUIRED>

<!ELEMENT downloads (file*)>

<!ELEMENT file EMPTY>
<!ATTLIST file
url CDATA #REQUIRED
name CDATA #REQUIRED
size CDATA #IMPLIED>

<!-- ================================================================== -->
<!-- End of DTD -->
<!-- ================================================================== -->
```

This custom DTD should be placed in `src/documentation/resources/schema/dtd/`

The <!ENTITY % ... > blocks are so-called parameter entities. They are like macros, whose content will be inserted when a parameter-entity reference, like `%common-charents;` or `%document;` is inserted.

In our DTD, we first pull in the 'common-charents' entity, which defines character symbol sets. We then define the 'document' entity. However, before the `%document;` PE reference, we first override the 'local.section' entity. This is a hook into document-v11.mod. By setting its value to '|release', we declare that our <release> element is to be allowed wherever "local sections" are used. There are five or so such hooks for different areas of the document; see document-v11.dtd for more details. We then import the %document; contents, and declare the rest of our DTD elements.

We now have a DTD for the 'download' document type.

Note:

Chapter 5: Customizing DocBook of Norman Walsh's "DocBook: The Definitive Guide" gives a complete overview of the process of customizing a DTD.

2.2. Associating DTDs with document types

Recall that our DOCTYPE declaration for our download document type is:

```
<!DOCTYPE document PUBLIC "-//Acme//DTD Download Documentation V1.0//EN"
          "download-v11.dtd">
```

We only care about the quoted section after `PUBLIC`, called the "public identifier", which globally identifies our document type. We cannot rely on the subsequent "system identifier" part ("download-v11.dtd"), because as a relative reference it is liable to break. The solution Forrest uses is to ignore the system id, and rely on a mapping from the public ID to a stable DTD location, via a Catalog file.

Note:

See this article for a good introduction to catalogs and the Cocoon documentation Entity resolution with catalogs.

Forrest provides a standard catalog file at `forrest/src/core/context/resources/schema/catalog.xcat` for the document types that Forrest provides. Projects can augment this with their own catalog file located in `${project.schema-dir}/catalog.xcat`. Use the "project.catalog" property in `forrest.properties` if you need a different pathname. Remember to raise the "verbosity" level if you suspect problems with your catalog.

Forrest uses the XML Catalog syntax by default, although the SGML format can also be used. Here is what the XML Catalog format looks like:

```
<?xml version="1.0"?>
<!-- OASIS XML Catalog for Forrest -->
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <public publicId="-//Acme//DTD Download Documentation V1.0//EN"
    uri="dtd/download.dtd"/>
</catalog>
```

The format is described in the spec, and is fairly simple and very powerful. The "`public`" elements map a public identifier to a DTD (relative to the catalog file).

We now have a custom DTD and a catalog mapping which lets Forrest locate the DTD. Now if we were to run `'forrest validate'` our download file would validate along with all the others. If something goes wrong, try running `'forrest -v validate'` to see the error in more detail. Remember to raise the "verbosity" level in `forrest.properties` if you suspect problems with your catalog.

3. Referring to entities

Look at the source of this document (`xdocs/validation.xml`) and see how the entity set `"Numeric and Special Graphic"` is declared in the document type declaration.

ISOnum.pen	½	½

4. Validating in an XML editor

If you have an XML editor that understands SGML or XML catalogs, let it know where the Forrest catalog file is, and you will be able to validate any Forrest XML file, regardless of location, as you edit your files. See the configuration notes your favourite editor.

5. Validation using RELAX NG

Other validation is also conducted during build-time using RELAX NG. This validates all of the important configuration files, both in Forrest itself and in your project. At the moment it processes all skinconf.xml files, all sitemap.xmap files, and all XSLT stylesheets.

The RNG grammars to do this are located in the `src/core/context/resources/schema/relaxng` directory. If you want to know more about this, and perhaps extend it for your own use, then see `src/core/context/resources/schema/relaxng/README.txt` and the Ant targets in the various build.xml files.