Extending Forrest with Plugins

Table of contents

1. Overview

Forrest provides the core functionality for generating documentation in various output formats from a range of input formats. However, it does not end there. Forrest can be extended through the addition of plugins. This document serves as an introduction to the Forrest plugin mechanism.

1.1. What plugins are available?

You can run the command `forrest available-plugins` to get a list of the known plugins for Forrest.

If you would like to have your own plugin added to this list then contact the developer mailing list.

1.2. How is a Plugin Installed?

If a site requires one or more plugins then the site designer will have named them in the `project.required.plugins` property in the projects `forrest.properties` file. When Forrest builds the site it will automatically discover the plugin and install it. In otherwords, the user need do nothing. For example, `project.required.plugins=OpenOffice.org,simplified-docbook` will cause Forrest to load the plugins called "OpenOffice.org" and "simplified-docbook".

1.2.1. Upgrading from a Version of Forrest Without Plugins

The plugin functionality was introduced in version 0.7 of Forrest. At this time some of the functionality previously in Forrest was extracted into a plugin. However, we have not broken backward compatability with earlier versions. In the absence of a `project.required.plugins` property in the projects `forrest.properties` file all plugins that contain functionality previously part of Forrest itself will be loaded automatically. Unless you intend to use new functionality provided by a plugin you will not need to make any changes top your project.

If you do require additional plugin functionality, be sure to include all required plugins in the `project.required.plugins` property in the projects `forrest.properties`. You can view `forrestcore/src/core/context/default.forrest.properties` to see the names of plugins that provide previously core functionality.

It is also worth noting that there is a small performance improvement if you remove plugins that are not in use. Therefore, if you do not use one or more of the plugins named in the

`project.required.plugins` property of `forrestcore/src/core/context/default.forrest.properties` it is recomended that you override this value in your projects `forrest.properties` file.

2. What is a Forrest Plugin?

A Forrest plugin is a set of resources and configuration files that extend the functionality of Forrest. They will typically consist of a sitemap, zero or more stylesheets and zero or more schema's.

The plugins sitemap is mounted by Forrest's sitemap after the project specific sitemap but before the Forrest default matchers. This allows individual projects to override/extend functionality provided in either a plugin or Forrest whilst plugins are only able to override/extend the default Forrest behaviour.

Forrest is easily extensible through the existing sitemap.xmap files, however the more features we add, the more complex the sitemap becomes. It is already quite difficult to understand the default sitemap.xmap files, and this will only get worse as new features find their way into the core.

By adopting a plugin model we can keep the core of Forrest tightly focused on the basic functionality, whilst still facilitating extensions to suit individual projects needs.

2.1. An Example Plugin

In order to fully understand the applicability of Forrest Plugins we will consider an extension to the way in which Forrest defines the structure of the site. By default Forrest uses a site.xml file to define navigation through the site and a tabs.xml file to define the tabs across the top of the page. But what if we want to use a different file to describe site structure? For example, what if we want to use an IMS Manifest file from the SCORM content package standards (http://www.adlnet.org/).

An IMS Manifest file describes the structure of a site. It is also possible to define a set of rules for extracting tab information from such a file. Consequently, it is possible to use an IMSManifest file to create Forrest's site.xml and tabs.xml files. The advantage would be that we can then use SCORM compliant content objects within Forrest.

Unfortunately, IMS Manifests are much more complex than site.xml and tabs.xml files. Therefore, not all users will want to use them. Adding the functionality as an optional plugin seems to be the ideal solution.

3. What Does a Forrest Plugin Look Like?

Plugins will need to conform to a specified directory structure. This mirrors the default forrest directory structure:

```
[plugin_name]
  |
  |-- config files (xmap, skinconf etc.)
  |
  |-- resources
      |
      |-- schema
      |   |
      |   |-- catalog.xcat
      |   |
      |   |--DTD (dtd's etc.)
      |
      |-- stylesheets (xsl's etc.)
```

3.1. The IMS Manifest Plugin

If we consider the IMS Manifest Plugin described above we see that we will need the following files and directory structure:

```
IMSManifest
  |
  |-- sitemap.xmap
  |
  |-- resources
      |
      |-- stylesheets
                    |
                    |- imsmanifest2site.xsl
                    |- imsmanifest2tabs.xsl
                    |- pathutils.xsl
                    |- repositoryUtils.xsl
```

The sitemap.xmap file will override the default behaviour for the navigation generation matchers in Forrest, for example, it contains a matcher as follows:

```
<map:match pattern="abs-menulinks">
  <map:select type="exists">
    <map:when test="{project:content.xdocs}imsmanifest.xml">
       <map:generate src="{project:content.xdocs}imsmanifest.xml" />
       <map:transform
src="{forrest:plugins}/IMSManifest/resources/stylesheets/imsmanifest2site.xsl"/>
       <map:transform src="{forrest:stylesheets}/absolutize-linkmap.xsl" />
       <map:transform
src="{forrest:stylesheets}/site2site-normalizetabs.xsl" />
     <map:serialize type="xml"/>
   </map:when>
   <map:when test="{project:content.xdocs}site.xml">
      <map:generate src="{project:content.xdocs}site.xml" />
```

```
      <map:transform src="{forrest:stylesheets}/absolutize-linkmap.xsl" />
      <map:transform
src="{forrest:stylesheets}/site2site-normalizetabs.xsl" />
      <map:transform src="{forrest:stylesheets}/normalizehrefs.xsl"/>
    <map:serialize type="xml"/>
  </map:when>
  </map:select>
</map:match>
```

Note:

Note that this matcher will default to the behaviour provided by Forrest if there is no imsmanifest.xml file present in the project. At present it is necessary to copy this default behaviour from the original Forrest *.xmap files. We hope to improve on this in the future.

4. How does Installation work?

When Forrest installs a plugin it downloads a zip of the plugin code and extracts it into the `plugins` directory of Forrest and an entry is made in `src/plugins/sitemap.xmap`. For example, installing the IMSManifest plugin described above will result in the following entry being added to the plugin sitemap:

```
<map:select type="exists">
  <map:when test="{forrest:plugins}/IMSManifest/sitemap.xmap">
    <map:mount uri-prefix=""
      src="{forrest:plugins}/IMSManifest/sitemap.xmap"
      check-reload="yes"
      pass-through="true"/>
  </map:when>
</map:select>
```

Installed plugins are managed by the FORREST_INSTALL_DIR/plugins/sitemap.xmap file. This file is mounted by the main Forrest sitemap with the following code:

```
       <map:pipeline internal-only="false">
          <map:mount uri-prefix=""
                     src="{forrest:plugins}/sitemap.xmap"
                     check-reload="yes"
                     pass-through="true"/>
       </map:pipeline>
```

Note:

The plugin sitemap.xmap file is automatically managed by Forrest, the end user need never edit this file.