

Plugin Infrastructure

Table of contents

1 Overview.....	2
2 What is a Forrest Plugin?.....	2
2.1 Types of Plugin.....	2
2.2 Naming Conventions.....	3
2.3 An Example Plugin.....	3
3 What Does a Forrest Plugin Look Like?.....	3
3.1 The IMS Manifest Plugin.....	4
4 How does Installation work?.....	5
5 Further Reading.....	5

2.1.3 Internal Plugins

Internal plugins are for advanced use only. They provide ways of extending or overriding Forrest's internal operations. For example, the IMSManifest plugin allows Forrest projects to use an IMS Manifest file instead of a site.xml and tabs.xml configuration files.

Internal plugins provide an `internal.xmap` file. This provides the infrastructure matchers (i.e. site.xml, faq.xml, issues.xml), and will be mounted before *any* of the Forrest matches. This sitemap can override any behaviour within Forrest and so developers of these plugins must be especially careful with the construction of their matchers, since they will be processed before any other matchers and consequently can easily break existing functionality. You must only do a `<map:generate ...>` if you are certain you are going to process the full result.

2.2 Naming Conventions

Technically you can name a plugin anything you like with one small restriction (see below). However, we do have some naming conventions that we recommend you follow. This is to minimise the chances of collision between plugins from different developers.

The name should be structured like a java package name, and should include a relevant reverse domain name. For example:

```
org.apache.forrest.plugin.PLUGIN_TYPE.PLUGIN_NAME
net.sf.forrestPlugins.PLUGIN_TYPE.PLUGIN_NAME
```

Where `PLUGIN_TYPE` is either "internal", "input" or "output" and `PLUGIN_NAME` is a suitable name chosen by yourself.

Warning:

Plugin names cannot have a '-' character in them. This character is used to indicate the start of a version number when defining a plugin to be used. See [Using Plugins](#) for more information.

2.3 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]
|-- plugin control files (xmap etc.)
|-- conf
|   |-- cocoon and component config files (e.g. *.xconf, jtidy)
|-- resources
|   |-- schema
|       |-- catalog.xcat
|       |-- dtd (DTDs etc.)
|   |-- stylesheets (XSLs 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:

```
org.apache.forrest.plugin.internal.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="{properties:content.xdocs}imsmanifest.xml">
      <map:generate src="{properties:content.xdocs}imsmanifest.xml" />
      <map:transform src="resources/stylesheets/imsmanifest2site.xsl" />
      <map:transform src="{forrest:forrest.stylesheets}/absolutize-linkmap.xsl" />
      <map:transform src="{forrest:forrest.stylesheets}/site2site-normalizetabs.xsl" />
      <map:serialize type="xml"/>
    </map:when>
    <map:when test="{properties:content.xdocs}site.xml">
      <map:generate src="{properties:content.xdocs}site.xml" />
      <map:transform src="{forrest:forrest.stylesheets}/absolutize-linkmap.xsl" />
      <map:transform src="{forrest:forrest.stylesheets}/site2site-normalizetabs.xsl" />
      <map:transform src="{forrest: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?

See the [Using Plugins](#) for an overview of how the plugin installation system works and the places and order that will be searched.

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 a temporary sitemap that manages plugins for your content object. For example, installing the IMSManifest plugin described above will result in the following entry being added to the this temporary sitemap:

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

To be more accurate, the entries are made in one of three temporary sitemaps, `input.xmap`, `output.xmap` and `resources.xmap`. These temporary sitemaps are rebuilt each time the content object is built or run, thus we are always guaranteed of having the right plugins installed.

5 Further Reading

If you want to build a plugin you might like to start with our [HowTo on Building Plugins](#).