Menus and Linking

$Revision: 1.13 $

1. Introduction

This document describes Forrest's internal URI space; how it is managed with `site.xml`, how menus are generated, and how various link schemes (site:, ext:) work. An overview of the implementation is also provided.

2. site.xml

`site.xml` is what we'd call a 'site map' if Cocoon hadn't already claimed that term. `site.xml` is a loosely structured XML file, acting as a map of the site's contents. It provides a unique identifier (an XPath address) for 'nodes' of information in the website. A 'node' of site information can be:

- A category of information, like 'the user guide'. A category may correspond to a directory, but that is not required.
- A specific page, e.g. 'the FAQ page'
- A specific section in a page, e.g. "the 'general' section of the FAQ page" (identified by `id="general"` attribute)

In addition to providing fine-grained addressing of site info, `site.xml` allows *metadata* to be associated with each node, with attributes or child elements. Most commonly, a `label` attribute is used to provide a text description of the node.

There are currently two applications of `site.xml`

Menu generation
`site.xml` is used to generate the menus for the HTML website.

Indirect linking
`site.xml` provides a basic aliasing mechanism for linking, e.g. one can write <link href="site:changes"> from anywhere in the site, and link to the 'changes' information node (translated to changes.html). More on this below.

Here is a sample site.xml, a stripped-down version from Forrest's own website:

```xml
<?xml version="1.0"?>
<site label="Forrest" href="" tab="home"
  xmlns="http://apache.org/forrest/linkmap/1.0">

  <about label="About">
    <index label="Index" href="index.html"/>
    <license label="License" href="license.html"/>
    <your-project label="Using Forrest" href="your-project.html">
      <new_content_type href="#adding_new_content_type"/>
    </your-project>
    <linking label="Linking" href="linking.html"/>
    <changes label="Changes" href="changes.html"/>
    <todo label="Todo" href="todo.html"/>
    <live-sites label="Live sites" href="live-sites.html"/>
  </about>

  <community label="Community" href="community/" tab="community">
    <index label="About" href="index.html"/>
    <howto-samples label="How-To Samples" href="howto/" tab="howto">
      <overview label="Overview" href="index.html"/>
      <single-page label="Single Page" href="v10/howto-v10.html"/>
      <multi-page label="Multi-Page" href="multi/">
        <intro label="Intro" href="howto-multi.html"/>
        <step1 label="Step 1" href="step1.html"/>
```

```
            <step2 label="Step 2" href="step2.html"/>
          </multi-page>
        </howto-samples>
      </community>

      <references label="References">
        <gump label="Apache Gump" href="http://jakarta.apache.org/gump/"/>
        <cocoon label="Apache Cocoon" href="http://xml.apache.org/cocoon/"/>
      </references>

      <external-refs>
        <mail-archive href="http://marc.theaimsgroup.com"/>
        <xml.apache.org href="http://xml.apache.org/">
          <cocoon href="cocoon/">
            <ml href="mail-lists.html"/>
            <actions href="userdocs/concepts/actions.html"/>
          </cocoon>
          <forrest href="forrest/"/>
          <xindice href="xindice/"/>
          <fop href="fop/"/>
        </xml.apache.org>

        <mail>
          <semantic-linking href="http://marc.theaimsgroup.com/?l=forrest-dev
            &amp;m=103097808318773&amp;w=2"/>
        </mail>
        <cool-uris href="www.w3.org/Provider/Style/URI.html"/>
        <uri-rfc href="http://zvon.org/tmRFC/RFC2396/Output/index.html"/>

      </external-refs>

    </site>
```

As you can see, things are pretty free-form. The rules are as follows:

- The root element must be 'site', and normal content should be in the namespace
  `http://apache.org/forrest/linkmap/1.0`. Feel free to mix in your own content (RDF, dublin core, etc) under new namespaces
- Element names are used as identifiers. The `foo` in `site:foo` must therefore be a valid NMTOKEN.
- Elements with `href` attributes can be used as identifiers in `site:` URIs
- Relative href attribute contents are 'accumulated' by prepending hrefs from ancestor nodes
- Elements without `label` attributes (and their children) are not displayed in the menu.
- Elements below `external-refs` are mapped to the `ext:` scheme. so `ext:cocoon/ml` becomes
  `http://xml.apache.org/cocoon/mail-lists.html`

3. Generating Menus

Two files are used to define a site's tabs and menu; `site.xml` and `tabs.xml`. Both files are located in `src/documentation/content/xdocs/`.

Assume that our `tabs.xml` looked as follows:

```
<tabs ...>
    <tab id="home" label="Home" dir=""/>
    <tab id="community" label="Community" dir="community" indexfile="mailLists.html"/>
    <tab id="howto" label="How-Tos" dir="community/howto"/>
</tabs>
```

Using the `site.xml` listed above, we would get these menus:

Menu generated from site.xml

Community menu generated from site.xml

Howto menu generated from site.xml

When using the `dir` attribute as above the value of the `indexfile` parameter is appended to the value of the `dir` attribute

(together with a preceding '/'). For example, the link for the community tab above is `community/mailLists.html`. Note that `indexfile` defaults to `index.html` if no value is supplied. Therefore the link for the howto tab is `community/howto/index.html`.

3.1. Tabs for External Resources

You can make a tab point to an external resource by using the `href` attribute instead of the `dir` attribute. The value of `href` should be the URI of the resource you wish to link to. For example:

```
<tab id="apache" label="XML Apache" href="http://xml.apache.org/"/>
```

Unlike the `dir` attribute, the value of `href` is left unmodified by Forrest unless it is root-relative and obviously specifies a directory (ends in '/'). In which case /index.html will be added

3.2. Selecting menu entries

Forrest decides which menu entries to display by examining the `tab` attributes in `site.xml`. All `site.xml` entries with a `tab` equal to that of the current page are put in the menu.

Consider our `site.xml` example:

```
<site label="Forrest" href="" tab="home"
  xmlns="http://apache.org/forrest/linkmap/1.0">

  <about label="About">
    <index label="Index" href="index.html"/>
    <license label="License" href="license.html"/>
    <your-project label="Using Forrest" href="your-project.html">
      <new_content_type href="#adding_new_content_type"/>
    </your-project>
    <linking label="Linking" href="linking.html"/>
    ....
  </about>

  <community label="Community" href="community/" tab="community">
    <index label="About" href="index.html"/>
    <howto-samples label="How-To Samples" href="howto/" tab="howto">
      <overview label="Overview" href="index.html"/>
      <single-page label="Single Page" href="v10/howto-v10.html"/>
      <multi label="Multi-Page" href="multi/">
        <intro label="Intro" href="howto-multi.html"/>
        <step1 label="Step 1" href="step1.html"/>
      ...
```

Every `site.xml` node can potentially have a `tab` attribute. If unspecified, nodes inherit the `tab` of their parent. Thus everything in the **<about>** section has an implicit `tab="home"`.

> **Note:**
> You can see this by viewing your site's [abs-menulinks](#) pipeline in a browser.

Say that the user is viewing the `linking.html` page. The **<linking>** node has an implicit tab value of `home`. Forrest will select *all nodes with tab="home"*, and put them in the menu.

3.3. Alternative menu selection mechanisms.

The `tab` attribute-based scheme for selecting a menu's entries is not the only one, although it is the most flexible. Here we describe a few alternatives.
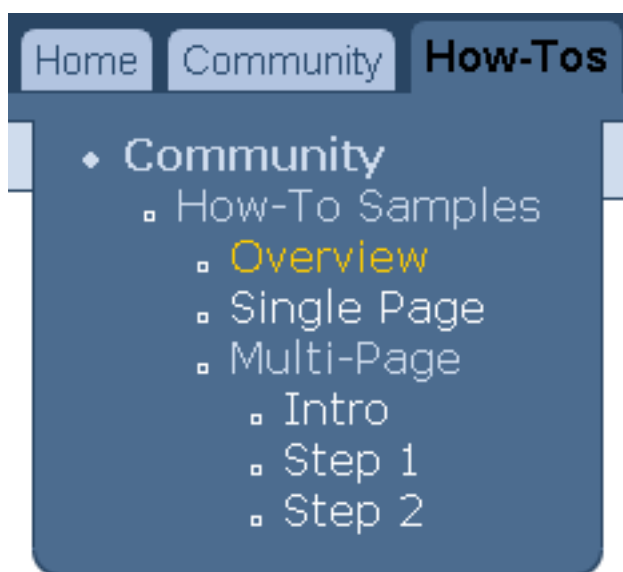
3.3.1. Directory-based selection

In this scheme, each tab corresponds to a directory within the site. All content below that directory is included in the menu.
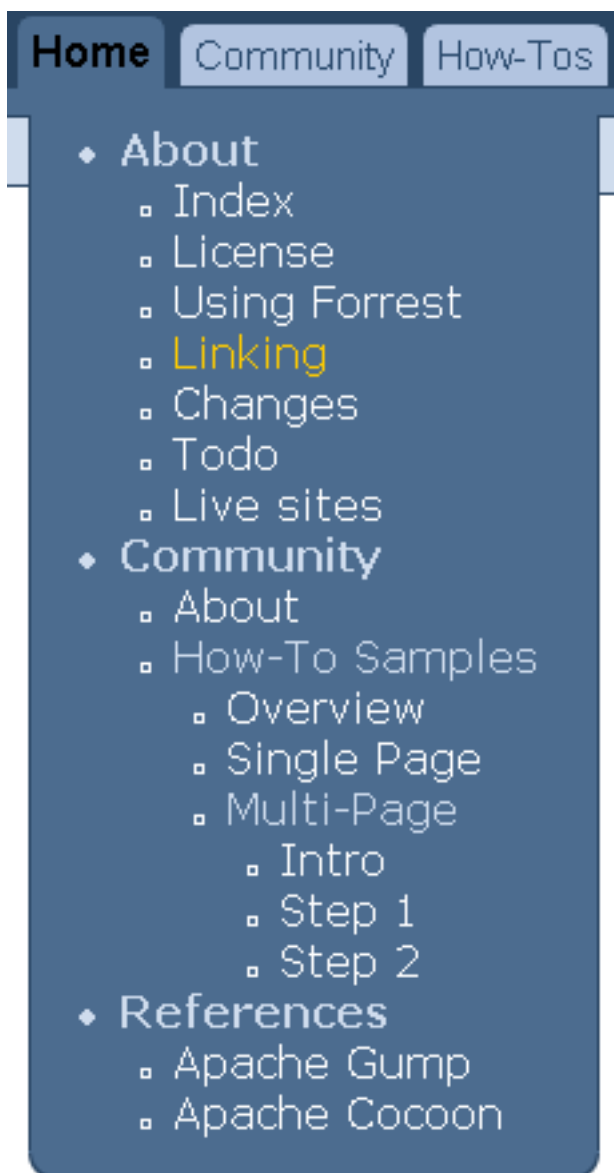
Directory-based site menu
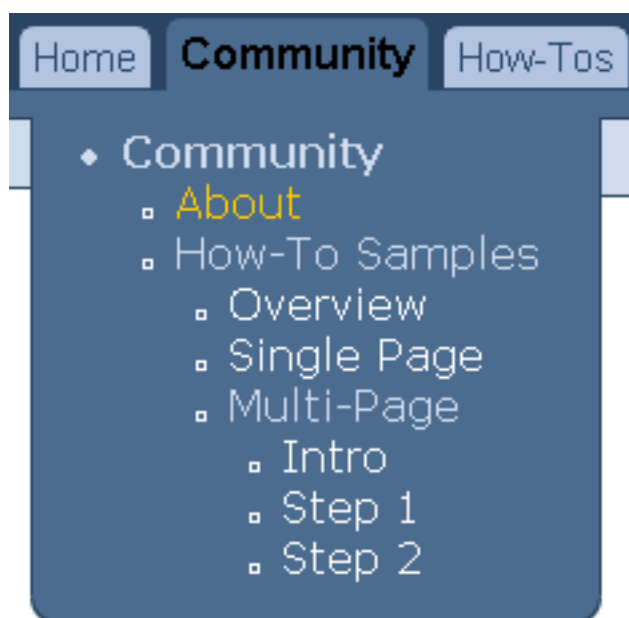
community/ directory menu

community/howto/ directory menu

To use this scheme:

- Edit `forrest.properties` and set `project.menu-scheme=directories`
- Remove the `id` attributes from `tabs.xml` entries.

3.3.2. Specifying menus with book.xml

Historically, menus in Forrest have been generated from a `book.xml` file, one per directory. This mechanism is still available, and if a `book.xml` is found, it will be used in preference to the `site.xml`-generated menu. `book.xml` files can use `site:` URIs to ease the maintenance burden that led to book.xml's obsolescence. In general, however, we recommend that users avoid `book.xml`.

3.4. Selecting the current tab

The tab selection algorithm is quite simple: the tab with the `id` which matches that of the current `site.xml` node is "selected". If you experience any problems, try to add a `<foo label="Foo" href="index.html"/>` to `site.xml`

4. Table of Contents Generation

Each page can have an automatically generated table of contents. This is created from the titles of each section in your xdoc. By default only sections up to two levels deep are included and the table of contents is displayed at the top of the page. However, you can configure this behaviour in `src/documentation/skinconf.xml` using the `toc` element.

```
<toc level="2" location="page"/>
```

- **`level`** - is the depth to which you want your table of contents to go. Setting it to 3 will therefore include sections nested to 3 levels. Setting this to 0 will result in no table of contents being generated.
- **`location`** - indicates where you want the table of contents to be rendered. It can be set to one of three values:
  - `page` - the table of contents will be rendered at the top of the body of your page
  - `menu` - the table of contents will be rendered in the menu to the left of the body of the page
  - `menu, page` - table of contents will be rendered in both the page and the menu positions

5. Linking systems

5.1. Direct linking

In earlier versions of Forrest (and in similar systems), there has been only one URI space: that of the generated site. If `index.xml` wants to link to `todo.xml` then `index.xml` would use

```
<link href="todo.html">to-do list<link>
```

The theoretical problem with this is that the content producer should not know or care how Forrest is going to render the source. A URI should only *identify* a resource, not specify it's type [mail ref] [cool URIs]. In fact, as Forrest typically renders to multiple output formats (HTML and PDF), links in one of them (here, the PDF) are likely to break.

5.2. Indirect linking

Forrest's solution is simple: instead of <link href="todo.html">, write <link href="site:todo">, where:

site
is a URI 'scheme'; a namespace that restricts the syntax and semantics of the rest of the URI [rfc2396]. The semantics of 'site' are "this identifier locates something in the site's XML sources".
todo
identifies the content in `todo.xml`, by reference to a 'node' of content declared in `site.xml`.

We call this indirect, or *semantic* linking because instead of linking to a physical representation (todo.html), we've linked to the 'idea' of "the todo file". It doesn't matter where it physically lives; that will be sorted out by Forrest.

5.2.1. Resolving site: URIs

How exactly does `site:todo` get resolved? A full answer is provided in the implementation section. Essentially, the `todo` part has `/site//` prepended, and `/@href` appended, to form string `/site//todo/@href`. This is then used as an XPath expression in `site.xml` identifying the string replacement, in this case `todo.html`.

Thus by modifying the XPath prefix and suffix, just about any XML format can be accommodated.

> **Note:**
> Actually, the XPath is applied to XML generated dynamically from `site.xml`. The generated XML has `@href`'s fully expanded ('absolutized') and ..'s added ('relativized') as

needed.

Notice that the '//' allows us any degree of specificity when linking. In the sample `site.xml` above, both `site:new_content_type` and `site:about/your-project/new_content_type` identify the same node. It is up to you to decide how specific to make links. One nice benefit of link 'ambiguity' is that `site.xml` can be reorganized without breaking links. For example, 'new_content_type' currently identifies a node in 'your-project'. By leaving that fact unspecified in `site:new_content_type`, we are free to make 'new_content_type' its own XML file, or a node in another file, in another category.

5.2.2. ext: URIs: linking to external URLs

The `ext:` scheme was created partly to demonstrate the ease with which new schemes can be defined, and partly for practical use. `ext:` URIs identify nodes in `site.xml` below the <external-refs> node. By convention, nodes here link to URLs outside the website, and are not listed in the menu generated from `site.xml`.

Here is a `site.xml` snippet illustrating `external-refs`:

```
<site>
  ...
  <external-refs>
    <mail-archive href="http://marc.theaimsgroup.com"/>
    <xml.apache.org href="http://xml.apache.org/">
      <cocoon href="cocoon/">
        <ml href="mail-lists.html"/>
        <actions href="userdocs/concepts/actions.html"/>
      </cocoon>
    </xml.apache.org>
    <forrest href="forrest/"/>
    <xindice href="xindice/"/>
    <fop href="fop/"/>

    ...
  </external-refs>
</site>
```

As an example, <link href="ext:cocoon/ml"> generates the link http://xml.apache.org/cocoon/mail-lists.html

The general rules of `site.xml` and `site:` linking apply. Specifically, the @href aggregation makes defining large numbers of related URLs easy.

5.2.3. Theory: source URIs

`site:` URIs like `site:todo` are examples of *source* URIs, in contrast to the more usual `foo.html`-style URIs, which we here call *destination* URIs. This introduces an important concept: that the *source* URI space exists and is independent of that of the generated site. Furthermore, URIs (ie, links) are first-class objects, on par with XML documents, in that just as XML content is transformed, so are the links. Within the source URI space, we can have all sorts of interesting schemes (person:, mail:, google:, java:, etc). These will all be translated into plain old `http:` or relative URIs in the destination URI space, just like exotic XML source formats are translated into plain old HTML in the output.

5.2.4. Future schemes

So far, `site:` and `ext:` schemes are defined. To give you some ideas on other things we'd like to implement (and we'd welcome help implementing), here are a few possibilities.

Scheme	Example 'From'	Example 'To'	Description
java	java:org.apache.proj.SomeClass	/apidocs/org/apache/proj/SomeClass.html	Links to documentation for a Java class

			(typically generated by `javadoc`).
mail	mail::<Message-Id>	`http://marc.theaimsgroup.com?m=103461678`	Links to an email identified by its `Message-Id` header. Any mail archive website could be used.
search	search:<searchterm>	`http://www.google.com/search?q=searchterm`	Links to a set of results from a search engine
person	person:JT, person:JT/blog etc	`mailto:jefft<at>apache.org`, `http://www.webweavertech.com/jefft/weblog/`, etc:	A person: scheme could be used, say, to insert an automatically obfuscated email address, or link to a URI in some way associated with that person.

There are even more possibilities in specific environments. In an intranet, a `project:XYZ` scheme could identify company project pages. In a project like Apache Ant, each Task could be identified with `task:<taskname>`, eg `task:pathconvert`.

6. Concept

The `site:` scheme and associated ideas for `site.xml` were originally described in the 'linkmap' RT email to the forrest-dev list (RT means 'random thought'; a Cocoon invention). Only section 2 has been implemented, and there is still significant work required to implement the full system described. In particular, there is much scope for automating the creation of `site.xml` (section 4). However, what is currently implemented gains most of the advantages of the system.

7. Implementation

Full details on the implementation of link rewriting and menu generation are available in the Sitemap Reference