Kiln Documentation

Release 1.0

Miguel Vieira

Nov 27, 2019

Contents

Kiln is an open source multi-platform framework for building and deploying complex websites whose source content is primarily in XML. It brings together various independent software components, with Apache Cocoon at their centre, into an integrated whole that provides the infrastructure and base functionality for such sites.

Kiln is developed and maintained by a team at the Department of Digital Humanities (DDH), King's College London. Over the past years and versions, Kiln (formerly called xMod) has been used to generate more than 50 websites which have very different source materials and customised functionality. Since DDH has in-house guidelines for using TEI P5 to create websites, Kiln makes use of certain TEI markup conventions. However, it has been adapted to work on a variety of flavours of TEI and other XML vocabularies, and has been used to publish data held in relational databases.

CHAPTER 1

Further reading

Apache Cocoon is at the core of Kiln, please consult its documentation for more detailed information on how to configure its components.

CHAPTER 2

Requirements

Java 1.5+ is required to run the Kiln webapps. In order to use the built-in Jetty web server (for local, development use only), Java 1.7 is required.

CHAPTER 3

Support

See our issue tracker.

CHAPTER 4

Tutorial

Kiln comes with a *tutorial* that covers using the main components of Kiln, and is recommended for new users.

CHAPTER 5

Contents

5.1 Quickstart

1. Download or clone the Kiln code from the GitHub repository.
2. Open a terminal window and go to the directory where Kiln is installed (hereafter KILN_HOME).
3. Run the command `build.sh` (Mac OS X/Linux) or `build.bat` (Windows), and leave the Terminal window open.
4. Open a browser and got to http://localhost:9999/. It should display a welcome page together with some very basic *navigation*.
5. Store project XML content in the folders `KILN_HOME/webapps/ROOT/content/xml/tei` (TEI) and `KILN_HOME/webapps/ROOT/content/xml/epidoc` (EpiDoc).
6. View HTML versions of the XML at http://localhost:9999/en/text/<filename>.html (TEI) and http://localhost:9999/en/inscriptions/<filename>.html (EpiDoc) [filetype extensions not included in <filename>]
7. Customise the templates, transformations and site URL structure. The example project in `KILN_HOME/example` provides some guidance on how this can be done.

5.1.1 Principles

Overriding XSLT by using `xsl:import` - a Kiln XSLT is imported by a local XSLT, allowing for templates to be redefined for the project.

URL scheme: `_internal` for internal-to-Cocoon Kiln URLs, `admin` for viewable but not public local material, `internal` for local internal-to-Cocoon URLs.

The directory *structure* makes explicit the division between those parts of Kiln that are its core, and should not be changed in any project installation, and the project-specific material. Kiln material is always a descendant of a directory called kiln.

This division also carries through to the Cocoon pipelines and the XSLT they use. Some Kiln pipelines are designed to be flexible enough to suit multiple different uses by a project-specific pipeline (see the Schematron pipelines for an

example). Where a Kiln pipeline uses XSLT that might reasonably be customised by a project, it calls a proxy XSLT not within a kiln directory, that in turn imports the Kiln version. This allows for customisation without changing Kiln core files.

5.2 Tutorial

This tutorial walks you through the creation of a basic web site edition of some historical letters. It is designed to provide familiarity with how some of the pieces of Kiln work and can be used together. It is not a tutorial in the individual technologies; it does not try to teach XSLT programming, or RDF, or Solr. However, it also does not require any great knowledge of same.

5.2.1 Installation

Installation of Kiln itself is simple. You can download a ZIP file of all the code and unpack it, or use the Git version control system to clone the repository. Either way, the code is available at the Kiln repository, and you'll end up with that code somewhere on your filesystem.

You'll also need to have Java 1.7 installed. If it isn't on your system already, you can download it from https://www.java.com/.

5.2.2 The development server

Let's verify that the installation worked. From the command line, cd into the directory where you installed Kiln. There, run the `build.sh` script (if you are running GNU/Linux or Mac OS X) or the `build.bat` batch file (if you are running Windows). You'll see the following output on the command line:

```
Buildfile: <path to your Kiln>/local.build.xml
Development server is running at http://127.0.0.1:9999
Quit the server with CONTROL-C.
```

You've started Jetty, a lightweight web server, that is pre-configured to run all of the various Kiln components. Note that it may take a few seconds after it prints out the above for the server to become responsive.

Now that the server is running, visit http://127.0.0.1:9999/ with your web browser. You'll see a "Welcome to Kiln" page.

Note: Changing the port

By default, the `build` command starts the development server on the internal IP at port 9999.

If you want to change the server's port, pass it as a command-line argument. For instance, this command starts the server on port 8080:

```
./build.sh -Djetty.port=8080
```

To change the default, edit the value of `jetty.port` in the file `local.build.properties`.

The default values of the Solr and RDF4J servers, as specified in `webapps/ROOT/sitemaps/config.xmap`, use port 9999, so if you intend to use those under the dev server, that value will need to be changed.

5.2.3 Adding content

The main content of many Kiln sites is held in TEI XML files, so let's add some. Unzip `the accompanying set of TEI files` into `webapps/ROOT/content/xml/tei/`.

Now navigate to the text overview at http://127.0.0.1:9999/text/, available as the Texts menu option. This presents a table with various details of the texts in sortable columns. With only a homogenous collection of a few letters, this is not very useful, but it does provide links to the individual texts. Follow the link to the first letter.

5.2.4 Customising the TEI display

Given the enormous flexibility of the TEI to express various semantics, and the range of possible displays of a TEI document, there is no one size fits all solution to the problem of transforming a TEI document into HTML. Kiln comes with XSLT code that provides support for some types of markup, but it is expected for each project to either customise it or replace it altogether. Let's do the former.

Kiln uses the XSLT at `webapps/ROOT/stylesheets/tei/to-html.xsl` to convert TEI into HTML. Open that file in your preferred XML editor. As you can see, it is very short! All it does is import another XSLT, that lives at `webapps/ROOT/kiln/stylesheets/tei/to-html.xsl`. This illustrates one of the ways that Kiln provides a separation between Kiln's defaults and project-specific material. Rather than change the XSLT that forms part of Kiln (typically, files that live in `webapps/ROOT/kiln`), you change files that themselves import those files. This way, if you upgrade Kiln and those files have changed, you're not stuck trying to merge the changes you made back into the latest file. And if you don't want to make use of Kiln's XSLT, just remove the import.

Note: So how does Kiln know that we want to transform the TEI into HTML using this particular XSLT?

This is specified in a Cocoon sitemap file, which defines the URLs in your site, and what to do, and to what, for each of them. In this case any request for a URL starting `texts/` and ending in `.html` will result in the XML file with the same name being read from the filesystem, preprocessed, and then transformed using the `to-html.xsl` code.

Sitemap files are discussed later in the tutorial.

Let's change the rendering, in an admittedly trivial way, so that the names of people and places are italicised. This involves adding a template like the following:

```
<xsl:template match="tei:persName | tei:placeName">
  <i>
    <xsl:apply-templates />
  </i>
</xsl:template>
```

Add this after the `xsl:import` element. Now reload the page showing that text, and you'll see the text rerendered with italics sprinkled throughout.

Warning: Cocoon automatically caches the results of most requests, and invalidates that cache when it detects changes to the files used in creating the resource. Thus after making a change to `to-html.xsl` (the one in `stylesheets/tei`, not the one in `kiln/stylesheets/tei/`), reloading the text shows the effects of that change. However, Cocoon does not follow `xsl:import` and `xsl:include` references when checking for changed files. This means that if you change such an imported/included file, the cached version of the resource will be used.

To ensure that the cache is invalidated in such cases, update the timestamp of the including file, or the source document. This can be done by re-saving the file (add a space, remove it, and save).

Adding images

Images referenced within TEI files (using `tei:figure/tei:graphic`) are converted by the `kiln/stylesheets/tei/to-html.xsl` XSLT into HTML `img` elements. The `src` URL is typically to `/images/{/tei:TEI/@xml:id}/{@url}` and these URLs are resolved to look in `content/images/` for the file. So if you add the following to `content/xml/tei/Had1.xml`:

```
<figure>
  <graphic url="image-filename.jpg" />
  <figDesc>This becomes HTML alt text.</figDesc>
</figure>
```

and place `image-filename.jpg` (using whatever JPEG image file you wish) in `content/images/Had1/`, the image should appear in the HTML display.

Images that are part of the site design, rather than content, should be put in `assets/images/`, and the pipelines in `kiln/sitemaps/assets.xmap` used. In a template, for example:

```
<img src="{$kiln:assets-path}/images/logo.jpg" alt="My Favourite Sponsor" />
```

Kiln can support any image file type, since no processing is done to the files. The pipelines simply transmit the files with an appropriate MIME type. Pipelines exist for GIF, JPEG, and PNG images; others are easily added, to `sitemaps/main.xmap` and/or `kiln/sitemaps/assets.xmap`.

5.2.5 Searching and indexing

Indexing

In order to provide any useful results, the search engine must index the TEI documents. This functionality is made available in the admin section of the site. You can either index each document individually, or index them all at once.

Note: If you started Kiln with a different port from the default, you must change the port in `solr-server` element in the file `webapps/ROOT/sitemaps/config.xmap` to match.

There are two possible parts of customising the indexing: changing the available fields that data can be indexed into, and changing the XSLT that specifies what information gets stored in which fields.

To change the fields in the index, modify the Solr schema document at `webapps/solr/conf/schema.xml`. Refer to the Solr documentation for extensive documentation on this and all other aspects of the Solr search platform.

It would be useful to index the recipient of each letter, so that this may be displayed as a facet in search results. In the `fields` element in `schema.xml`, define a recipient field:

```
<field indexed="true" multiValued="false" name="recipient"
       required="true" stored="true" type="string" />
```

After changing the schema, you will need to restart Jetty so that the new configuration is loaded. You can check the schema that Solr is using via the Solr admin interface at http://127.0.0.1:9999/solr/ (the specific URL is http://localhost:9999/solr/#/collection1/schema).

Changing the data that is indexed is done by modifying the XSLT `stylesheets/solr/tei-to-solr.xsl`. Just as with the TEI to HTML transformation, this XSLT imports a default Kiln XSLT that can be overridden. We need to modify this file (not the default Kiln XSLT) to add in the indexing of the recipient into the new schema. Looking at `kiln/stylesheets/solr/tei-to-solr.xsl`, the default indexing XSLT traverses through the

teiHeader's descendant elements in the mode `document-metadata`. It is a simple matter to add in a template to match on the appropriate element:

```
<xsl:template match="tei:profileDesc/tei:particDesc//tei:person[@role='recipient']"
              mode="document-metadata">
  <field name="recipient">
    <xsl:value-of select="normalize-space()" />
  </field>
</xsl:template>
```

You will also need to add a namespace declaration for the `tei` prefix to the root `xsl:stylesheet` element: `xmlns:tei="http://www.tei-c.org/ns/1.0"`.

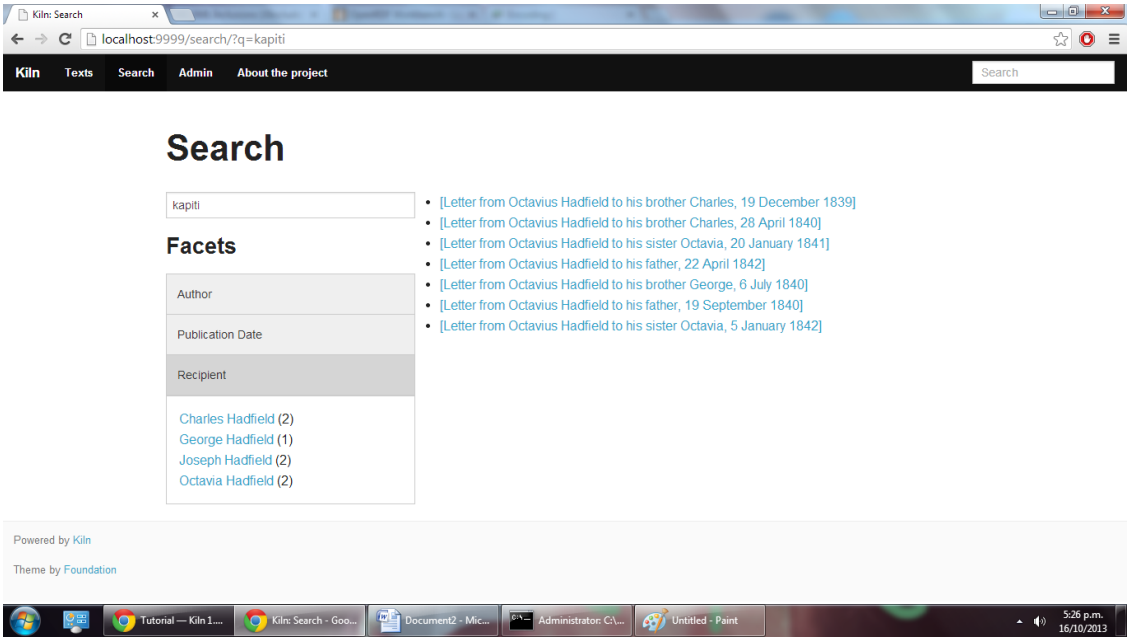
Now reindex the letters.

Warning: Omitting a namespace prefix that is used in an XPath expression in an XSLT document will cause incomprehensible and difficult to debug errors in the output, rather than a useful error message. If you get results that make no sense, check that all of the namespace prefixes that are used in the code are declared!

Facets

To customise the use of facets, modify the XML file `webapps/ROOT/assets/queries/solr/facet_query.xml`. This file defines the base query that a user's search terms are added to, and can also be used to customise all other parts of the query, such as how many search results are displayed per page. The format is straightforward; simply add elements with names matching the Solr query parameters. You can have multiple elements with the same name, and the query processor will construct it into the proper form for Solr to interpret.

Add in a facet for the recipient field and perform a search. The new facet is automatically displayed on the search results page, and should look something like this:



Results display

The default results display is defined in `stylesheets/solr/results-to-html.xsl` and gives only the title of the matching documents. Modify that XSLT to provide whatever format of search results best suits your needs.

5.2.6 Building static pages

Not all pages in a site need be generated dynamically from TEI documents. Let's add an "About the project" page with the following steps.

Note: This section introduces a lot of concepts that may be entirely new to you. Some of this is about the Cocoon software; if you need more information, try looking at the Overview of Apache Cocoon and Cocoon Concepts documentation on the Cocoon site.

Adding a URL handler

Each URL or set of URLs available in your web application is defined in a Cocoon sitemap that specifies the source document(s), a set of transformations to that document, and an output format for the result. Sitemaps are XML files, and are best edited in an XML editor. Open the file `webapps/ROOT/sitemaps/main.xmap`.

The bulk of this file is the contents of the `map:pipelines` element, which holds several `map:pipeline` elements. In turn, these hold the URL definitions that are the `map:match` elements. Each `map:match` has a `pattern` attribute that specifies the URL(s) that it defines. This pattern can include wildcards, `*` and `**`, that match on any sequence of characters except `/` and any sequence of characters, respectively.

The order of the `map:match` elements is important — when a request for a URL is handled by Kiln, it is processed using the first `map:match` whose pattern matches that URL. Then the child elements of the `map:match` are executed (the XML here is all interpreted as code) in order.

Go to the part of the document that defines the handler for the `search/` URL. Below that, add in a match for the URL `about.html`. Since we'll be putting the content of the page we want to return into the template (this is not the only way to do it!), our source document is just the menu, and the only transformation is applying the template. Your `map:match` should look something like the following (and very similar to the one for the home page):

```
<map:match id="local-about" pattern="*/about.html">
  <map:aggregate element="aggregation">
    <map:part src="cocoon://_internal/menu/main.xml?url={1}/about.html" />
  </map:aggregate>
  <map:transform src="cocoon://_internal/template/about.xsl">
    <map:parameter name="language" value="{1}" />
  </map:transform>
  <map:serialize />
</map:match>
```

Even in such a short fragment there is a lot going on. The `pattern="about.html"` attribute specifies that when a request is made for the URL `http://localhost:9999/about.html` (assuming we are running on the default Kiln development server), the response is defined by the contents of this `map:match` element. As mentioned above, each of these definitions consists of generating a source document, transforming it in some fashion, and serialising the result in some format (such as XML or HTML or PDF). Only one document can be generated, and it is serialised only once, but there can be any number of transformations that occur in between.

`map:aggregate` creates an XML document with a root element of `aggregation`, containing in this case one part (subelement). This part is the product of internally making a request for the URL `_internal/menu/main.xml?url=about.html`, which returns the menu structure. The use of URLs starting with `cocoon:/` is com-

mon, and allows a modular structure with lots of individual pieces that can be put together. If you want to see the `map:match` that handles this menu URL, open `webapps/ROOT/kiln/sitemaps/main.xmap` and look for the `kiln-menu` pipeline.

Note: A pipeline (a collection of `map:match` elements) may be marked as internal only (`map:pipeline internal-only="true"`), meaning that it is only available to requests from within Kiln (via a `cocoon:/` or `cocoon://` URL). If you request a URL that is matched by such an internal pipeline, such as via your browser, it will not match.

Kiln's generic pipelines are generally marked as internal only, and are grouped under the URL `_internal` (eg, `http://localhost:9999/_internal/menu/main.xml`). It also uses the convention of putting internal only pipelines that are project specific under the URL `internal` (without the initial underscore).

The templating transformation, which puts the content of the `aggregation` element into a template, also internally requests a URL. That URL returns the XML template file transformed into an XSLT document, which is then applied to the source document!

Finally, the document is serialised; in this case no serializer is specified, meaning that the default (HTML 5) is used.

Now that the `about.html` URL is defined, try requesting it at http://127.0.0.1:9999/about.html. Not surprisingly, an error occurred, because (as the first line of the stacktrace reveals) there is no `about.xml` template file. It's time to make one.

Adding a template

Template files live in `webapps/ROOT/assets/templates/`. They are XML files, and must end in `.xml`. In the `map:match` we just created, the template was referenced at the URL `cocoon://_internal/template/about.xsl` — there the `xsl` extension informally specifies the format of the document returned by a request to that URL, but it reads the source file `about.xml` in the templates directory. You can see how this works in the sitemap file `webapps/ROOT/kiln/sitemaps/main.xmap` in the `kiln-templating` pipeline.

Create a new file, `about.xml`, in the template directory. We could define everything we want output in this file, but it's much better to reuse the structure and style used by other pages on the site. Kiln templates use a system of inheritance in which a parent template defines arbitrary blocks of output that a child template can override or append to. Open the `base.xml` file in the templates directory to see the root template the default Kiln site uses. Mostly this is just a lot of HTML, but wrapped into chunks via `kiln:block` elements. Now look at the `tei.xml` template, which shows how a template can inherit from another and provide content only for those blocks that it needs to.

Go ahead and add to `about.xml` (using `tei.xml` as a guide) whatever content you want the "About the project" page to have. This should just be HTML markup and content, placed inside the appropriate `kiln:block` elements. Since there is no source document being transformed, there's no need to have the `xsl:import` that `tei.xml` has, and wherever it has `xsl:value-of` or `xsl:apply-templates`, you should just put in whatever text and HTML 5 markup you want directly.

Updating the menu

In the `map:match` you created in `main.xmap` above, the aggregated source document consisted only of a call to a URL (`cocoon://_internal/menu/main.xml?url={1}/about.htm`) to get a menu document. In that URL, `main.xml` specifies the name of the menu file to use, which lives in `webapps/ROOT/assets/menu/`. Let's edit that file to add in an entry for the new About page. This is easy to do by just inserting the following:

```
<menu match="local-about" label="About the project" />
```

Reload any of the pages of the site and you should now see the new menu item. Obviously this menu is still very simple, with no hierarchy. Read the *full menu documentation* for details on how to handle more complex setups.

5.2.7 Harvesting RDF

In order to make use of Kiln's RDF capabilities, some setup is required. Firstly create a repository in the RDF4J server using the "New repository" link at http://127.0.0.1:9999/rdf4j-workbench/, using the default options. The ID you provide should just contain letters.

Next set two variables in `webapps/ROOT/sitemaps/config.xmap`: `sesame-server-repository` to the name of the repository you just created, and `rdf-base-uri` to any absolute URI for your triples; we'll use http://www.example.org/.

With that setup done, it is time to create the XSLT that will generate RDF XML from the TEI documents. Place the provided `harvesting XSLT` at `webapps/ROOT/stylesheets/rdf/tei-to-rdf.xsl` (replacing the existing placeholder file). Now you can harvest the RDF data using the links in the admin. You can use the workbench link given above to examine the data in the repository.

Note: Both the ontology and the harvesting are primitive, and designed to be simple enough for the tutorial, without being entirely trivial. Harvesting the ontology from each TEI document is not good practice, nor is harvesting identifiers multiple times for the same entity.

5.2.8 Querying RDF

Having put RDF data into the repository, it is of course necessary to be able to get it back out. The simplest approach is to create an XML file in `webapps/ROOT/assets/queries/sparql/` that has a root `query` element containing the plain text of the SPARQL query.

For example, to retrieve just the triples giving the recipient of each letter, save the following to `webapps/ROOT/assets/queries/sparql/recipients.xml`:

```
<query>
PREFIX ex:&lt;http://www.example.org/>
PREFIX rdf:&lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?correspondence ex:has_recipient ?recipient ;
                            ex:has_document ?letter . }
WHERE { ?correspondence ex:has_recipient ?recipient ;
                        ex:has_document ?letter . }
</query>
```

Note: Within an XML SPARQL query document, XML rules apply, meaning that XML-significant characters (primarily <) need to be escaped (<).

To get the results from this query, use the URL `cocoon://admin/rdf/query/graph/recipients.xml` in a sitemap's `map:generate` or `map:part src` attribute. Remember that `map:generate` and `map:aggregate` (which contains `map:part` elements) are the way that Cocoon generates a source document.

Note: While the Sesame RDF server can return results in various formats, due to Kiln working best with XML documents it is set up to make Graph Queries (using the CONSTRUCT command) with results in RDF XML.

Let's use a similar set of query results to display a list of other letters to the same recipient on each letter's page. As it stands the query returns the letters for *all* recipients in the collection, not just those that match a particular recipient. Therefore we need a way to pass in the name of the current letter's recipient to the query and get back the filtered results. Remember that the query document is just an XML document, so we can modify it with XSLT to supply that value.

The `new query` should be saved at `webapps/ROOT/assets/queries/sparql/recipients.xml` (you don't need the old version). Take a look at how it has changed, through the addition of the `recipient` element placeholder and using a custom output that better matches the information we want.

The URL mentioned above for performing a query of the RDF server calls the URL `cocoon://admin/rdf/construct/graph/{1}.xml` (where "{1}" is whatever is matched by the "**" of the first URL). This URL is handled by a `map:match` in `webapps/ROOT/sitemaps/rdf.xmap`, by reading the specified file. It is this `map:match` that needs to be modified or added to in order to customise the query.

Since you may want to handle multiple SPARQL queries in different ways, we'll add another `map:match`, before the one with the id "local-rdf-query-from-file". Its pattern needs to match `/admin/rdf/construct/graph/**.xml`, but be more specific to catch only the recipient query. There also needs to be an element in the URL that specifies the particular recipient we want to include in the query. A pattern of `construct/graph/recipient/*.xml` is suitable, where * will be the recipient name. The path to the query file can be specified explicitly.

Note: The "/admin/rdf" part of the URL is common to all patterns specified in the `rdf.xmap` file. A sitemap file (`*.xmap`) includes another sitemap by mounting it at a particular URL, and can specify a URI prefix that is common to all URL patterns defined therein. See the `uri-prefix` attributes on the `map:mount` elements in `main.xmap` and `admin.xmap`.

The full `map:match` is as follows:

```
<map:match pattern="construct/graph/recipient/*.xml">
  <map:generate src="../assets/queries/sparql/recipients.xml" />
  <map:transform src="../stylesheets/rdf/add-recipient.xsl">
    <map:parameter name="recipient" value="{1}" />
  </map:transform>
  <map:serialize type="xml" />
</map:match>
```

Note how the name of the recipient (that will be matched by * in the pattern) is passed as a parameter to the XSLT. That XSLT, which is very simple, is as follows:

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="recipient" />

  <xsl:template match="recipient">
    <xsl:value-of select="$recipient" />
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

This should be saved to `webapps/ROOT/stylesheets/rdf/add-recipient.xsl`.

Now of course we need to call the query URL, including the specific recipient name to search on. This means the request for that URL must come at a point in the processing that has access to the TEI document being displayed. We'll use an XInclude to include the query results in our TEI document. This works by adding an XInclude element with an `href` attribute specifying the URL of the resource to be included, and then using Cocoon's XInclude processor to perform the actual inclusion.

To add the XInclude element, we of course use XSLT. In `webapps/ROOT/sitemaps/main.xmap`, modify the `map:match` for TEI display (its id is "local-tei-display-html") to add the line:

```
<map:transform src="../stylesheets/tei/add-recipient-query.xsl" />
```

before the existing `map:transform`. Then place the provided XSLT at `webapps/ROOT/stylesheets/tei/add-recipient-query.xsl`. This XSLT just copies the existing document and adds the XInclude element.

To actually process the XInclude element so that the resource at the URL it specifies is included into the document, add the following line to the `map:match`, immediately after the `map:transform` element you just added:

```
<map:transform type="xinclude" />
```

Now the document that is manipulated by the template consists of a top-level `aggregation` element that has three sub-elements: `tei:TEI` (the TEI document), `kiln:nav` (the site navigation), and `rdf:RDF`, the query results. It's now possible, after all this setup, to modify the template to transform the query results into the list of other letters to the same recipient. Edit `webapps/ROOT/assets/templates/tei.xml` and add the line:

```
<xsl:apply-templates mode="recipients" select="/aggregation/rdf:RDF" />
```

after the line that applies templates to the `teiHeader` element (and before the closing `div` tag). You will also need to add a namespace declaration to the `kiln:root` element:

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

Now edit `webapps/ROOT/stylesheets/tei/to-html.xsl` and add in the following (along with, again, the RDF namespace declaration and one binding the prefix `ex` to `http://www.example.org/`; you'll get some odd errors if you don't!):

```
<xsl:template match="rdf:RDF" mode="recipients">
  <xsl:if test="count(rdf:Description) &gt; 1">
    <div class="section-container accordion" data-section="accordion">
      <section>
        <h2 class="title" data-section-title="">
          <small><a href="#">Other Letters to this Recipient</a></small>
        </h2>
        <div class="content" data-section-content="">
          <ul class="no-bullet">
            <xsl:apply-templates mode="recipients" />
          </ul>
        </div>
      </section>
    </div>
  </xsl:if>
</xsl:template>

<xsl:template match="rdf:Description" mode="recipients">
  <xsl:variable name="tei_id" select="ex:has_identifier" />
  <xsl:if test="$tei_id != /aggregation/tei:TEI/@xml:id">
    <li>
      <a href="{$tei_id}.html">
```

(continues on next page)

(continued from previous page)

```
        <xsl:value-of select="ex:has_date" />
      </a>
    </li>
  </xsl:if>
</xsl:template>
```

And there it is! It's important to note that the above is not the only way to achieve this result. The XInclude step might have been incorporated into the TEI preprocessing pipeline; or the RDF query modified to use the TEI ID as the variable rather than the recipient's name; or the letter title harvested and used as the link title rather than the date. Much depends, in crafting the components that go into generating the resource for a URL, on whether and how those components are used by other parts of the system.

5.2.9 Development aids

The admin section provides a few useful tools for developers in addition to the processes that can be applied to texts. The Introspection section allows you to look at some of what Kiln is doing when it runs.

Match for URL takes a URL and shows you the full Cocoon `map:match` that processes that URL. It expands all references, and links to all XSLT, so that what can be scattered across multiple sitemap files, with many references to `*` and `**`, becomes a single annotated piece of XML. Mousing over various parts of the output will reveal details such as the sitemap file containing the line or the values of wildcards.

Much the same display is available for each `map:match` that has an ID, in *Match by ID*.

Finally, *Templates by filename* provides the expanded XSLT (all imported and included XSLT are recursively included) for each template, and how that template renders an empty document.

The level of detail in the error messages Kiln provides can be reduced by setting the `debug` element's value to 0 in the file `webapps/ROOT/sitemaps/config.xmap`. This should be done in production environments to avoid providing useless and/or system information revealing information to users.

5.3 Kiln structure and file layout

Kiln uses a very specific file structure, as outlined below. It creates a degree of separation between the Kiln files, that should not be modified, and the project-specific files, keeping each in their own directories.

- build.bat — Ant startup script for Windows.
- build.sh — Ant startup script for Mac OS X/Linux.
- buildfiles — Ant core build files. This should not need to be modified.
- local.build.properties — Local Ant properties file.
- local.build.xml — Local Ant build file to override core functionality.
- example - Example webapps.
- sw — Software used in building and running Kiln.
- **webapps**
  - **ROOT — Project webapp.**
    - **assets**
      - foundation - Foundation CSS/JS framework
      - images - Non-content images.

- menu — Navigation menu.
- **queries - XML containing base queries**

  solr - Solr query fragments
- **schema**

  menu — Kiln schema, do not change.

  tei — Schema for TEI files.

  test — Kiln schema for test-case files, do not change.
- scripts — JavaScript files and libraries.
- styles — CSS files and libraries.
- **templates — Templates for content display.**

  admin - Templates for the admin system.
- translations - Catalogue files containing translations of site content.

* **content**
  - images — Project/content images.
  - **xml**

    authority - Authority files.

    indices - Content index files.

    tei — TEI content files.

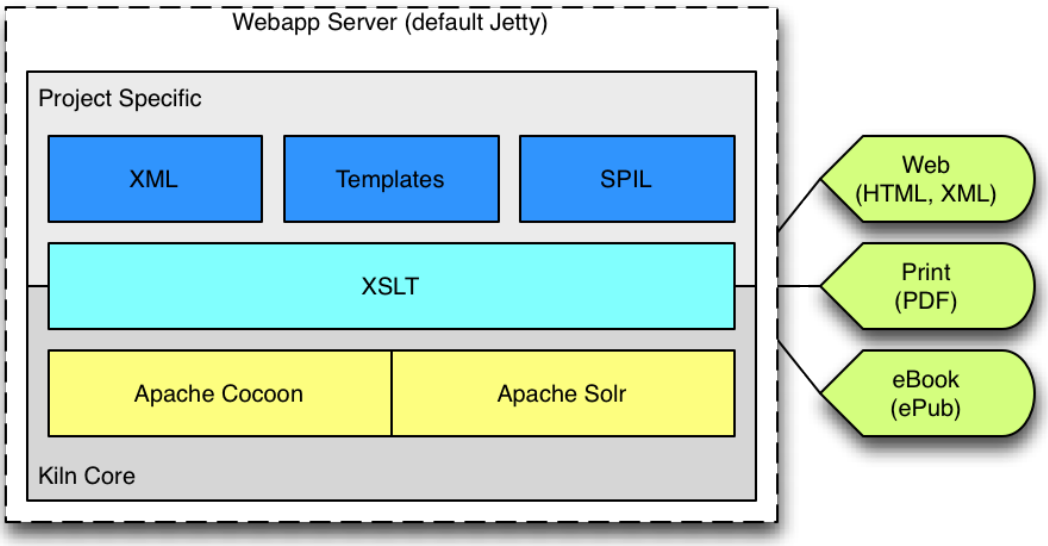
    epidoc - EpiDoc content files.
* kiln — Kiln core files, should not need to be modified.
* mount-table.xml — Cocoon's sitemap mount table (do not modify).
* not-found.xml — Default file to display when a resource is not found.
* resources — Cocoon resources (do not modify).
* sitemap.xmap — Cocoon default sitemap (do not modify).
* **sitemaps — Project's sitemaps.**
  - admin.xmap — Admin and editorial pipelines.
  - config.xmap — Configuration (global variables, etc).
  - internal.xmap - Internal (not exposed by URL) pipelines.
  - main.xmap — Main pipelines.
  - rdf.xmap — RDF pipelines.
  - solr.xmap — Search pipelines.
  - test.xmap — Automated testing pipelines.
* **stylesheets — Project's XSLT stylesheets.**
  - defaults.xsl — Defines default globals and reads parameters from the sitemaps.
  - escape-xml.xsl - Formats XML for literal display within HTML.
  - admin - Admin and editorial transformations.

        - epidoc - EpiDoc display.
        - error - Error handling.
        - introspection - Introspection of sitemaps and XSLT.
        - menu - Menu manipulation.
        - metadata - Extraction of metadata from files.
        - rdf - RDF harvesting and querying.
        - schematron — Schematron output.
        - solr — Searching and indexing.
        - system — Cocoon stylesheets (do not modify).
        - tei — TEI display.
        - test — Display of automated test results.
    - **test-suite — Project's test cases and data.**
        - cases — Test-case files.
        - data — Expected output data.
    - WEB—INF — Webapp configuration.
- openrdf-sesame and openrdf-workbench — RDF / Linked Open Data framework.
- **solr — Searching framework.**
    - **conf**
        - schema.xml - Definition of fields etc.

5.4 Components

- Apache Cocoon web development framework for XML processing, built with integrated eXist XML database that can be used for storage, indexing and searching using XPath expressions.
- Apache Solr searching platform for indexing, searching and browsing of contents.
- Sesame 2 for storage and querying of RDF data.
- Apache Ant build system that has tasks for running the built-in web application server, running the Solr web server, creating a static version of the website, generating a Solr index of all the desired content.
- Jetty web application server for immediate running of Cocoon with automatic refreshing of changes.
- An XSLT-based templating language that supports inheritance, similar to Django's template block system.
- Foundation, a set of HTML/CSS/JavaScript building blocks.

5.4.1 Architecture



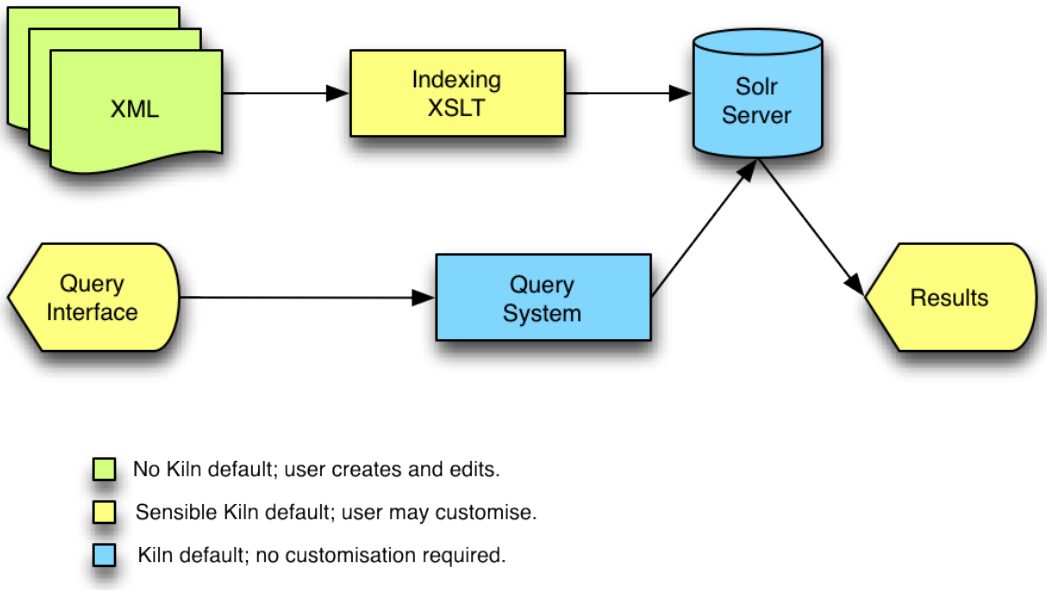
In a production web server context, Kiln integrates with other web publishing tools to support images (IIPimage/Djatoka), maps (GeoServer; MapServer; OpenLayers) and other data sources, like relational data (MySQL or other RDBMS).

5.4.2 Customisation

Kiln has been developed around the concept of the separation of roles, allowing people with different backgrounds, knowledge and skills to work simultaneously on the same project without overriding each other's work. The parts of the system used by developers, designers and content editors are distinct; further, the use of a version control system makes it simpler and safer for multiple people with the same role to work independently and cooperatively.

Since it is impossible to predict every eventuality with regards to a project's specific XML markup, Kiln offers basic output options which cover the functionality and formats (HTML, PDF, etc) common to all websites, together with an extensible framework supporting the development of any custom functionality that is needed. The system provides for a high-level of customisation, beyond the usable and useful defaults, in the following other areas:

- Schematron validation based on, and linked to, encoding guidelines published in ODD.
- Editorial workflow validation. Kiln provides web-based management pages that allow XML files to be checked for inconsistencies and errors.
- Templates for common types of pages, such as search and search results, indices, and bibliographies.
- XSLT for indexing contents for Solr. By default it indexes the full text and all the references to marked up entities.



Kiln provides native support for multilingual websites, RSS feeds, form processing, and automated navigation such as sitemaps and indexes, but with some customisation can support the publishing of more complex materials with much deeper markup, such as medieval charters, musicological bibliographies, classical inscriptions, biographies, glossaries and so forth.

5.4.3 Templates

Kiln provides a templating mechanism that provides full access to XSLT for creating the output, and an inheritance mechanism. Templates use XSLT as the coding language to create any dynamic content. Template inheritance allows for a final template to be built up of a base skeleton (containing the common structure of the output) and 'descendant' templates that fill in the gaps. In addition to supplying its own content, a block may include the content of the block it is inheriting from.

- Example of basic structure:

```
<kiln:root>
    <kiln:parent>
        <!-- Extend another template by including it. -->
        <xi:include href="base.xml" />
    </kiln:parent>
    <kiln:child>
        <!-- Override a block defined in an ancestor template. -->
        <kiln:block name="title">
            <h1>Title here</h1>
        </kiln:block>
    </kiln:child>
</kiln:root>
```

- Example of inheriting content:

```
<kiln:block name="title">
    <!-- Include the parent template's content for this block. -->
    <kiln:super />
```

(continues on next page)

(continued from previous page)

```
    <!-- Add in new content. -->
    <h2>Smaller title here</h2>
</kiln:block>
```

Fuller documentation is available.

5.5 Running the webapp

5.5.1 Development

For development work, Kiln provides the Jetty web application server, pre-configured, to minimise setup. To use it:

- Run the build script found in the KILN_HOME directory. On Windows double-clicking the BAT file is sufficient.
- Open the URL http://localhost:9999/ (for the root of the site).
- To stop the web server, press Ctrl-C in the shell window.

5.5.2 Production

The built-in Jetty web server is not suitable for production deployments, and a more robust solution such as Apache Tomcat should be used. Kiln uses the standard webapp structure, so deployment is a matter of copying the files in `webapps` into the server's existing webapps directory. Under Tomcat, at least, the Solr webapp requires an extra step: adding a `solr.xml` file specifying a Tomcat Context to `TOMCAT_HOME/conf/Catalina/localhost/`. An example of such a file is provided at `webapps/solr/conf/solr.xml`.

Further, there are good reasons to run a proxying web server in front of Tomcat, for caching, load balancing, and/or URL rewriting. It is advised that Solr and RDF4J not be exposed to any clients (including users) that are not local (eg Cocoon). If this is required, in the case of Solr, *a more recent version of Solr should be used* than that which comes with Kiln.

5.5.3 Static Build

Kiln includes a task that allows to create a static version of the website. To execute it:

- Run the build script as described above to start the web application.
- Re-run the build script supplying `static` as argument.

5.5.4 WAR Build (Web Application Archive)

Kiln includes a task that allows to create a Web Application Archive (for use with Apache Tomcat, e.g.). To execute it:

- Run the build script supplying `war` as argument.

5.6 Using Kiln with the latest version of Solr

- Download and set up Solr
- Create a new Solr core for the Kiln project
- Copy the default Solr settings from `configsets/_default/` into the new core
- Set XML as the default Solr response in `solrconfig.xml`:

```
<queryResponseWriter name="xml" default="true" class="solr.XMLResponseWriter" />
```

- **Copy the schema fields from the Kiln Solr to the new Solr:**
  - Note that some field types have been deprecated or renamed in the latest versions of Solr, make sure to update those accordingly
  - For example the field types `int` and `float` are now called `pint` and `pfloat`
- Edit the Kiln `config.xmap` sitemap, and update `solr-server` to the new Solr URL. The Solr URL should point to the core, and not just to the root Solr URL, for example `http://localhost:8983/solr/core_name/`.

5.7 Navigation

Navigation structures (menu hierarchies and breadcrumbs) are defined in XML files that accord to the Kiln menu schema (see `webapps/ROOT/assets/schema/menu/menu.rng`). These specify a hierarchy of labels and their associated URLs (which may be external to the Kiln site).

By default Kiln's navigation is configured in the file `KILN_HOME/webapps/ROOT/assets/menu/main.xml`. It is possible to have multiple menu files used in a single site, and even within a single `map:match`.

The structure of a menu is that of a list of items, with each item being able to hold another list of items. However, it makes a distinction between a `kiln:menu` and a `kiln:item`. The former may or may not reference a resource, while the latter must. A `kiln:menu` may have children, while a `kiln:item` may not.

It is important to understand that there is no necessary connection between the position of a `kiln:menu` or `kiln:item` in the hierarchy, and the URL it references. Specifically, the `@href` of an element, if it is not an absolute or root relative URL, is appended to the 'path' of `@root` values from its ancestors.

For example, given the following menu:

```
<root xmlns="http://www.kcl.ac.uk/artshums/depts/ddh/kiln/ns/1.0">
  <menu href="text/" label="Texts">
    <item href="intro.html" label="Introduction" />
    <menu label="Reports" root="text/reports">
      <item href="1.html" label="Report #1" />
    </menu>
  </menu>
</root>
```

The "Texts" item links to `text/`, "Introduction" links to `intro.html` (*not* `text/intro.html`), and "Report #1" links to `text/reports/1.html`. "Reports" is not a link at all.

Requiring explicit `@root` values in order to provide a link hierarchy allows for a hierarchical menu to map on to a less or differently organised underlying URL structure.

The simpler form of the above example menu, with everything in a nice hierarchy, is:

```
<root xmlns="http://www.kcl.ac.uk/artshums/depts/ddh/kiln/ns/1.0">
  <menu href="" label="Texts" root="text">
    <item href="intro.html" label="Introduction" />
    <menu label="Reports" root="reports">
      <item href="1.html" label="Report #1" />
    </menu>
  </menu>
</root>
```

Note that there is no requirement to pin any `@href` or `@root` to a root URL (`/`). This will be handled automatically by the menu processor. That includes, whether or not the references are root relative or not, adjusting the URLs if Kiln is mounted at a URL other than `/`.

If a constructed URL is meant to reference the same server, but to a path outside of Kiln's context, start the `@href` or `@root` with "//".

5.7.1 Using kiln:url-for-match

Rather than explicitly specifying URLs via the `@root` and `@href` attributes, the `kiln:url-for-match` function can be indirectly used in a menu. Specify the id of the `map:match` used to process the URL in a `kiln:menu` or `kiln:item`'s `@match` attribute, and if any parameters are required they can be specified (whitespace delimited) in the `@params` attribute. For example:

```
<root xmlns="http://www.kcl.ac.uk/artshums/depts/ddh/kiln/ns/1.0">
  <menu label="Texts" match="texts-home">
    <item match="texts-intro" label="Introduction" />
    <menu label="Reports">
      <item label="Report #1" match="texts-report" params="1" />
    </menu>
  </menu>
</root>
```

This method is greatly preferable to using `@href` for links within Kiln.

5.7.2 Marking the active item

The menu XML that is returned by `cocoon://_internal/menu/**.xml` can have a menu item matching a supplied URL marked as active. This is useful for display a menu with the item that matches the current position in the navigation structure highlighted.

To achieve this, supply a querystring to the Cocoon URL with a `url` parameter containing the URL to mark as active:

```
<map:match pattern="*/text/**/*.html">
  <map:aggregate element="aggregation">
    <map:part label="tei" src="cocoon://internal/tei/preprocess/{2}.xml" />
    <map:part src="cocoon://_internal/menu/{1}/main.xml?url={1}/text/{2}/{3}.html" />
  </map:aggregate>
  ...
</map:match>
```

The appropriate `li` element will be annotated with a class attribute with the value "active".

The supplied URL should be root relative, but *without* the initial "/", as in the example above.

Note that it does not matter what the URL of the request is; the menu uses only the URL explicitly passed to it. This allows for the same menu item to be marked as active for a multitude of URLs.

5.7.3 Translation and switching languages

To supply a key for looking up the translation of a menu item's label, add an `i18n:key` attribute to the item; translated strings are supplied as usual in `assets/translations/messages_<language code>.xml`.

Menu items, by default, link within the current language context. In order to have menu items that switch language context (ie, to point to a URL with a different language code as the first part of the path), supply a `language` attribute to the menu item, whose value is the language code the link should point to:

```
<item label="Recherche" language="fr" match="local-search" />
```

To link to the same URL only with a different language, use the `language_switch` attribute to specify the new language code:

```
<item label="English" switch_language="en" />
```

For linking to non-public URLs that do not have a language parameter (such as admin URLs), supply an empty language attribute.

For menus that are used in no language context (such as the admin's menu), the `map:match` should reference the menu pipeline that takes no language code:

```
<map:match pattern="admin/*.html">
  <map:aggregate element="aggregation">
    <map:part src="cocoon://_internal/menu/admin.xml?url=admin/{1}.html" />
  </map:aggregate>
  ...
</map:match>
```

5.8 URL matching

Kiln provides a mechanism for generating full URLs to Kiln resources based on the ID of a sitemap's `map:match` element. Since the actual URL is specified only in the `map:match/@pattern`, it can be changed without breaking any generated links, provided the number and order of wildcards in the `@pattern` are not changed. If they are, then at least it is easy to find all references to that `map:match` by searching the XSLT for its `@id`.

To use this functionality, include the XSLT `cocoon://_internal/url/reverse.xsl` and call the `kiln:url-for-match` function, passing the ID of the `map:match` to generate a URL for, a sequence containing any wildcard parameters for that URL, and a Boolean indicating whether to force the URL to be a `cocoon://` URL. For example:

```
<a href="{kiln:url-for-match('local-tei-display-html', ($language, 'Had1.xml'), 0)}">
  <xsl:text>Link title</xsl:text>
</a>
```

This generates a root-relative URL based on the `@pattern` value of the `map:match` with the ID "local-tei-display-html". If the identified `map:match` is part of a pipeline that is marked as `internal-only="true"`, the generated URL is a `cocoon://` URL.

If no wildcard parameters are required, pass an empty sequence:

```
<a href="{kiln:url-for-match('local-search', (), 0)}">Search</a>
```

If the third argument is true (eg, 1), then regardless of whether the pipeline the match belongs to is internal or not, the generated URL will be a `cocoon://` URL. This should be used when the generated URL will be used for situations in which the URL is evaluated without a webserver context, such as XIncludes.

Be sure to declare the kiln namespace (`http://www.kcl.ac.uk/artshums/depts/ddh/kiln/ns/1.0`), or else the call will be treated as a plain string.

Warning: Neither Kiln nor Cocoon performs any checks that the `id` values you assign to `map:match` elements are unique, either within a single sitemap file or across multiple sitemaps. If the same ID is used more than once, the first one (in sitemap order) will be used by the `url-for-match` template.

Warning: When developing in Kiln, be aware that all of the sitemap files must be well-formed XML, or this XSLT will not produce any results. This may lead to odd problems throughout the site that have no connection with the invalid sitemap.

5.9 Templating

Kiln provides a templating mechanism that provides full access to XSLT for creating the output, and an inheritance mechanism.

Template inheritance allows for a final template to be built up of a base skeleton (containing the common structure of the output) and 'descendant' templates that fill in the gaps.

This inheritance system works in much the same way as Django's template inheritance.

5.9.1 Using a template

To apply a template to a content source, a Cocoon `map:transform` is used, as follows:

```
<map:transform src="cocoon://_internal/template/path/to/template.xsl"/>
```

The matching template looks for the file `assets/templates/path/to/template.xml`. Note that the extension of the template file is **xml**.

5.9.2 Writing a template

The basic structure of a template is as follows:

```
<kiln:root xmlns:kiln="http://www.kcl.ac.uk/artshums/depts/ddh/kiln/ns/1.0"
           xmlns:xi="http://www.w3.org/2001/XInclude">
    <kiln:parent>
        <xi:include href="base.xml"/>
    </kiln:parent>
    <kiln:child>
        <kiln:block name="head-title">
        </kiln:block>
    </kiln:child>
</kiln:root>
```

In a base template (that is, one that does not extend another template), there are no `kiln:parent` or `kiln:child` elements, only `kiln:root` and `kiln:block` elements (and whatever content the template may hold, of course).

In a template that extends another, the template being extended is referenced by an `xi:include`, the only allowed content of `kiln:parent`.

5.9.3 Blocks

Block names must be unique within a template.

Attribute blocks

In order to create a block to cover the contents of an attribute, define a block immediately after the element holding the attribute, specifying the name of the attribute it is a block for in the `attribute` attribute.

```
<ul>
    <kiln:block name="ul-class" attribute="class">
        <xsl:text>block default</xsl:text>
    </kiln:block>
</ul>
```

Multiple attribute blocks for the same element should simply be defined in series. It is only necessary that they occur before any non-attribute blocks within that element.

Inheriting content

In addition to supplying its own content, a block may include the content of the block it is inheriting from. To do so, an empty `kiln:super` element should be added wherever the inherited content is wanted (multiple `kiln:super` elements may occur within a single block.

```
<kiln:block name="nav">
    <kiln:super/>
    <p>Extra navigation here.</p>
</kiln:block>
```

If the block being inherited also contains an `kiln:super` element, then that referenced content will also be included.

5.9.4 Dynamic content

Templates use XSLT as the coding language to create any dynamic content. `xsl:stylesheet` and `xsl:template` elements must not be used; the process that compiles the template into an XSLT wraps the template's XSL statements in a single `xsl:template` element matching on "/"; all processing occurs within this template, or imported/included XSLT.

`xsl:import` and `xsl:include` elements may be used (and the compiler will move them to the beginning of the XSLT), as may `xsl:apply-templates` and `xsl:call-template`.

5.9.5 Referencing assets

To reference assets (such as images, CSS and JavaScript files), ensure that the default variables XSLT is imported, and append the path to the file (relative to the assets directory) after the variable reference {$kiln:assets-path}.

```
<xsl:import href="cocoon://_internal/template/xsl/stylesheets/defaults.xsl" />

<kiln:block name="css">
  <link href="{$kiln:assets-path}/foundation/css/normalize.css"
              rel="stylesheet" type="text/css" />
</kiln:block>
```

This ensures that nothing that might change between installations (such as development serving the files files through Kiln, and production serving them off another dedicated server) is hard-coded into your templates.

See the settings in `stylesheets/defaults.xsl` for possible configurations.

5.10 Searching and Browsing

5.10.1 Introduction

Kiln has built-in support for using Solr to provide searching and browsing functionality. Kiln comes with:

- a Cocoon transformer to communicate with a Solr server as part of a pipeline match;
- customisable XSLT for generating Solr index documents from TEI files; and
- an Ant build task to index the entire site, by recursively crawl a Cocoon-generated page linking to the index documents to be added to the server.

5.10.2 Configuration

A global variable called `solr-server` should be specified in the project's `sitemaps/config.xmap`, with the value being the full URL of the Solr server (including a trailing slash).

The Solr schema may need to be modified to accomodate extra fields, or to enable various faceting approaches. It is less likely, but still possible, that the Solr configuration will need to be modified. Both of these files (`schema.xml` and `solrconfig.xml`), and other configuration files, are found in `webapps/solr/conf`.

Search and browse pages are so project-specific that the appropriate XSLT and Cocoon matches need to be created for them from scratch.

5.10.3 Built-ins

Kiln comes with a default indexing XSLT for TEI documents, using `kiln/stylesheets/solr/tei-to-solr.xsl`. This may be customised via changes to the importing XSLT `stylesheets/solr/tei-to-solr.xsl`. Other XSLT for indexing non-TEI documents may also be written; the Cocoon matches for these should follow the pattern of the TEI indexing match (in `solr.xmap#local-solr`).

Kiln also includes an XSLT, `kiln/stylehssets/solr/generate-query.xsl`, for adding XInclude elements that reference search results to an XML document, based on supplied query parameters in the source XML. See the documentation in the XSLT itself for details on the format of the supplied XML. When used in a pipeline, it must be followed by an XInclude transformation step.

Additionally, the XSLT `kiln/stylesheets/solr/merge-parameters.xsl` adds appropriate elements to the end of a query XML document, as above, from request data. `sitemaps/internal.xmap` provides a generic way to generate a search results document using this method.

This approach is not as redundant as it might seem, with a Solr query string being transformed into XML and then back into a query string that is run. It allows both for common query elements to be specified in an XML file, and also does not require that the query string passed to `merge-parameters.xsl` be formatted as Solr requires. Parameters can be repeated that `generate-query.xsl` will join together in the correct fashion. This frees whatever process generates the XSLT parameter value from knowing anything about Solr's details (eg, it can be a simple form with no processing).

5.10.4 Query files

Solr query files in `assets/queries/solr/` are a good way to provide static elements for a given search. They can specify what fields to facet on, default query strings, sorting, and so on.

Attributes on the root `query` element may specify which fields should be appended to the general query (`@q_fields`) and which fields should be appended to the general query as a range (`@range_fields`).

Child elements of `query` may specify a `default` attribute with the value `true`; the value of this element will only be used if no querystring parameter has the same name.

Facet fields (`facet.field`) may specify a `join` attribute with the value `or`; that facet is treated as a multi-select facet, with the selected values being ORed together. The default behaviour is for selected values to be ANDed together.

By default, the contents of fields are automatically escaped. When this is not desired, add an `escape` attribute with the value `false` to the element.

5.10.5 Indexing non-TEI documents

Kiln currently only has code for indexing TEI documents, and its display of search results assumes that those results come from TEI documents (in the `content/xml/tei` directory). To support the indexing and results display of documents in other directories, two things must be done:

- Add an XSLT in `stylesheets/solr` to perform the indexing. The filename should be `<dir>-to-solr.xsl`, where <dir> is the name of the directory under `content/xml` containing the documents.
- Add an appropriate condition path added to the `xsl:choose` in `stylesheets/solr/results-to-html.xsl` in the template for `result/doc` in the `search-results` mode.

5.10.6 Upgrading Solr

There are two parts to upgrading Solr separately from a Kiln upgrade (if Kiln has not yet incorporated that version of Solr): upgrading the solr webapp, and upgrading the Solr Cocoon transformer.

Upgrading the Solr webapp is straightforward, unless there are local modifications to the files under `webapps/solr` (except in `conf` and `data`). If there are, either these changes can be merged back in to the new versions (either manually or through whatever tools are available), or left in place.

First, delete all content in `webapps/solr` except for the `conf` and `data` directories. Next upack the contents of the solr WAR file distributed with Solr into `webapps/solr`. This can be done with the command `jar -xvf <filename>.war`. It is possible that there are incompatibilities between the new version of Solr and the existing configuration files in `webapps/solr/conf`, in which case these will need to be resolved manually. The Solr admin web page can be helpful in finding problems.

Upgrading the transformer is more complicated. After fetching a copy of the transformer source code, the JAR files in the `lib` directory must be replaced with those from the Solr distribution. Unless there has been a change to Solr's API, the transformer code does not need to be modified. The transformer can be rebuilt using Apache Ant with the command `ant dist`. The newly created `solr.transformer.jar` file in the `dist` directory must then be copied to `webapps/kiln/WEB-INF/lib/`, along with the JARs in the lib directory. These must be put in place of their equivalents (the filenames will differ, since the Solr JARs have the version number as part of the filename), and all in the `webapps/kiln/WEB-INF/lib/` and not a subdirectory.

5.11 Schematron validation

Kiln comes with pipelines and XSLT to perform Schematron validation of XML files.

The base transformation is the `validate/*/*/**.xml` match in `kiln/sitemaps/schematron.xmap`. The first parameter is the schema filename (with no extension), expected to exist as `assets/schematron/<schema>.sch`. The second parameter is the phase, which is a named subset of rules within the schema. If all phases should be run, use `#ALL` as the value. The third parameter is the path to the XML file to be validated, relative to `content/xml/`.

If the first parameter begins with a hash character (#), the remainder of the value is used as the basename of an ODD file at `assets/schema/<basename>.xml`, from which all Schematron rules are extracted.

`sitemaps/admin.xmap#local-schematron` contains an example match that is meant to automatically extract the schema from the project's encoding guidelines embedded in an ODD file. This functionality is not currently available.

5.12 RDF / Linked Open Data

Kiln includes the RDF4J framework for handling RDF data. RDF4J operates as a pair of Java webapps (`rdf4j-server` and `rdf4j-workbench`), and work no differently within Kiln as they do standalone. This includes extending it, for example by using OWLIM as the repository, and using its web interfaces.

5.12.1 Setting up a repository

The RDF4J documentation covers how to create a repository. Once created, the name of the repository must be set as the value of the `sesame-server-repository` variable in `webapps/ROOT/sitemaps/config.xmap`, along with a base URI (`rdf-base-uri`).

(Note that RDF4J used to be called Sesame, and that the former name is still used in some parts of Kiln.)

5.12.2 Integration with Cocoon

Kiln integrates RDF4J into its Cocoon processing via a transformer and a set of pipelines. The local sitemap `sitemaps/rdf.xmap` can be extended with matches to generate RDF/XML and SPARQL graph queries. The basic operations are defined and described in the internal kiln sitemap `kiln/sitemaps/sesame.xmap`.

5.12.3 Upgrading RDF4J

There are two parts to upgrading RDF4J separately from a Kiln upgrade (if Kiln has not yet incorporated that version of RDF4J): upgrading the rdf4j-server and rdf4j-workbench webapps, and upgrading the Sesame Cocoon transformer.

Upgrading the two RDF4J webapps is straightforward. First delete everything in `webapps/rdf4j-workbench`, and everything except `app_dir` (where the project data is kept) in `webapps/rdf4j-server`. Next unpack the contents of the two WAR files distributed with RDF4J into the appropriate directories. This can be done with the command `jar -xvf <filename>.war`.

Upgrading the transformer is more complicated. After fetching a copy of the transformer source code, the JAR files in `lib/sesame-lib` must be replaced with those from the RDF4J download. It is unlikely that all of the JARs that come with RDF4J are required. Unless there has been a change to RDF4J's API, the transformer code does not need to be modified. The transformer can be rebuilt using Apache Ant with the command `ant dist`. All of the newly created JAR files in the `dist` directory must then be copied to `webapps/kiln/WEB-INF/lib/` in place of their equivalents (the filenames will differ, since the RDF4J JARs have the version number as part of the filename).

5.13 Fedora Repository

Kiln includes pipelines for fetching data from a Fedora Commons repository. The generic matches for fetching XML and binary data are in `webapps/ROOT/kiln/sitemaps/fedora.xmap`, which should be accessed via matches in `webapps/ROOT/sitemaps/fedora.xmap`. The latter likely need to be customised to match the way the specific Fedora repository has been set up.

It is also necessary to set the global variable `fedora-url` in `webapps/ROOT/sitemaps/config.xmap`.

5.14 Multilingual sites

Kiln supports sites in which some or all of the content is available in more than one language. It does this by having a language code as the first element in all public URLs (not internal, not admin).

For example, if the site consisted of resources at index.html, about/index.html, and about/process.html, and these resources were available in both English and Telugu, the URLs for these resources would be:

- /en/index.html
- /en/about/index.html
- /en/about/process.html
- /te/index.html
- /te/about/index.html
- /te/about/process.html

5.14.1 Implementation

All of the matches in `sitemaps/main.xmap` incorporate a language code prefix that is used in the various transformations. The language code is passed as a parameter to XSLT that include `stylesheets/defaults.xsl`. This value is used in XSLT when creating links to textual content within Kiln.

A default language code must be set in `sitemaps/config.xmap` in the global variable `default-display-language`.

Kiln also follows the approach for internationalisation (i18n) as documented in the Cocoon documentation. A sample catalogue of translations is available in `assets/translations/messages_en.xml`; translations into other languages should use the same filename convention and content format. Note that of the provided templates and menus, only `assets/templates/home.xml` and `assets/menu/main.xml` have even partial i18n markup; this must be added where appropriate to templates and XSLT.

5.14.2 HTTP content negotiation

Kiln does not include any code for handling automatic content negotiation based on the HTTP Accept-Language header.

5.14.3 Nice URLs for monolingual sites

Monolingual sites also use the same URL scheme with a language code prefix, which may not be desired in production. This problem can be solved by using, for example, `mod_proxy` and `mod_proxy_html` with Apache HTTPD to rewrite headers and links so that users see the URLs without the language code prefix.

5.15 Using Kiln as an independent backend

While Kiln provides facilities for creating and serving resources to user clients, it can equally well be used solely as a backend to another system or systems operating as the frontend. There is nothing special or different about such operation, and the same principles and approaches apply. Rather than a user client, such as a browser, making a request to a URL and getting a resource, the frontend system makes the request, perhaps using AJAX, and then operates on the resource.

5.15.1 Providing content for incorporation into HTML

A common use case is to have Kiln provide content that the frontend then incorporates into its own templating system for display as HTML. Kiln provides an example template and pipeline showing how this might be achieved. The template transforms TEI into HTML, and provides that, plus any metadata, CSS, and JavaScript that might be required for correct display. All of these are made available in individual elements within the XML document that Kiln returns.

This allows the frontend system to collate the various pieces as it wishes without having to perform any complicated transformations.

5.15.2 Requesting a resource with Python

Sample code for requesting a resource from Kiln using Python:

```
import requests

r = requests.get(kiln_resource_url)
xml = r.text
# Process XML.
...
```

5.15.3 Requesting a resource with JavaScript

Sample code for requesting a resource from Kiln using JavaScript (with jQuery):

```
function xml_content_handler (xml) {
  // Process XML.
}

$.get({
  url: kiln_resource_url,
  dataType: 'xml'
}).done(xml_content_handler);
```

5.16 Using web services

Kiln can interact with external web services that return XML by making HTTP requests with POSTed data. This is done using Cocoon's CInclude Transformer. There are three parts to the process, as follows:

1. Create a base query file (in `assets/queries`) that contains all of the static data for the request, and placeholders for dynamic data. For example:

```
<?xml version="1.0"?>
<data xmlns:cinclude="http://apache.org/cocoon/include/1.0"
      xmlns:xi="http://www.w3.org/2001/XInclude">
  <!-- POST a request to a GATE Named Entity Recognition service,
       supplying the contents of a TEI file to be annotated. -->
  <cinclude:includexml ignoreErrors="true">
    <cinclude:src>http://www.example.org/gate/ner-service/</cinclude:src>
    <cinclude:configuration>
      <cinclude:parameter>
        <cinclude:name>method</cinclude:name>
        <cinclude:value>POST</cinclude:value>
      </cinclude:parameter>
    </cinclude:configuration>
    <cinclude:parameters>
      <cinclude:parameter>
        <cinclude:name>source</cinclude:name>
        <!-- Create a placeholder XInclude where the TEI XML is
             wanted. -->
        <cinclude:value><xi:include href="source-placeholder" /></cinclude:value>
      </cinclude:parameter>
    </cinclude:parameters>
  </cinclude:includexml>
</data>
```

2. Write an XSLT to provide any dynamic data to the query. For example:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0"
                xmlns:cinclude="http://apache.org/cocoon/include/1.0"
                xmlns:xi="http://www.w3.org/2001/XInclude"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="tei-filename" />

  <!-- Replace the XInclude's placeholder href attribute with the
       path to the supplied TEI file. -->
  <xsl:template match="xi:include[@href='source-placeholder']">
    <xsl:copy>
      <xsl:attribute name="href">
        <xsl:text>path/to/tei/directory/</xsl:text>
        <xsl:value-of select="$tei-filename" />
      </xsl:attribute>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

3. Create a pipeline for making the request and display the result. For example:

```
<map:match pattern="gate-ner/**.xml">
  <map:generate src="../assets/queries/gate/ner.xml" />
```

(continues on next page)

(continued from previous page)

```
  <!-- Add an XInclude of the TEI file. -->
  <map:transform src="../stylesheets/gate/ner.xsl">
    <map:parameter name="tei-filename" value="{1}.xml" />
  </map:transform>
  <!-- Perform the XInclude of the TEI file. -->
  <map:transform type="xinclude" />
  <!-- Make the request. -->
  <map:transform type="cinclude" />
  <map:serialize type="xml" />
</map:match>
```

5.17 Kiln's admin and editorial area

Kiln provides a URL space, under `/admin/`, for administrative and editorial tasks and resources. These may include indexing documents into the Solr search engine, generating OAI-PMH records, and creating and harvesting RDF triples into Sesame.

`sitemaps/admin.xmap` specifies the applicable pipelines.

5.17.1 Adding authentication

The simplest approach to password protecting the admin area is to make use of HTTP Authentication. The configuration for this depends on which web server is sitting in front of Kiln — it may be Apache Tomcat, Apache HTTPD, NGinix or something else — and its documentation should be consulted.

5.18 PDF and ePub generation

Kiln provides sample pipeline matches for PDF and ePub generation from TEI source files in `sitemaps/main.xmap`. These make use of the same templating system used for HTML generation.

Unlike with HTML generation, however, Kiln does not provide XSLT to generate PDF and ePub output, as the variation in source markup and desired output between projects is too great. Instead either XSLT can be written from scratch, or existing stylesheets from other projects can be adopted and adapted. The suite of stylesheets available from the TEI website provide both XSL FO and ePub outputs; the former can be turned into PDF via Cocoon's fo2pdf serialiser (as per the example pipeline match).

5.19 Testing framework

Kiln provides a basic framework to run automated tests that check the XML output of a `map:match` with expected output. Tests are stored in the `webapps/ROOT/test-suite` directory, with the expected data being stored under `data` and the definition of the tests in files under `cases`.

The format for test-case files are specified in `assets/schema/test/test_case.rng`. Each test within a test-case file specifies the path to the expected data XML file (relative to `test-suite/data`) and the id and parameters of the `map:match` to be tested.

The admin menu links to the HTML report for running all of the tests in all of the test-cases under `test-suite/cases`, noting for each test whether it passed or failed, and if it failed, giving a rough difference between the actual and expected output.

5.20 Error handling

Kiln uses Cocoon's built-in error handling mechanism. It provides two levels of handling. The first, defined in `sitemaps/main.xmap`, makes use of the templating system to provide error messages that display in the style of the site. The second, defined in `sitemaps/config.xmap`, is untemplated, and used only if there is a problem with the code used in the first level error-handling.

Both levels will display all of the technical details Cocoon makes available when the global variable `debug` is set to "1", and hidden when it is set to "0".

Error messages can be overridden if desired; this is already done for 404 errors, for which see `sitemaps/main.xmap`.

5.21 License

Kiln is available under the Apache License. Unless otherwise specified, the following statement applies:

```
Copyright 2012 Department of Digital Humanities, King's College London

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.
```

5.21.1 Components' licenses

- Apache Cocoon, Apache Solr and Apache Ant are available under the Apache License.
- Sesame 2 is available under the BSD License.
- Jetty is available dual licensed under the Apache License and the Eclipse Public License.

5.22 Projects using Kiln

Over the past years and versions, Kiln has been used to generate more than 50 websites which have very different source materials and customised functionality. Since DDH has in-house guidelines for using TEI P5 to create websites, Kiln makes use of certain TEI markup conventions. However, it has been adapted to work on a variety of flavours of TEI and other XML vocabularies, and has been used to publish data held in relational databases.

- Ancient Inscriptions of the Northern Black Sea
- Corpus of Romanesque Sculpture in Britain and Ireland
- Centre for the History and Analysis of Recorded Music
- The Complete Works of Ben Jonson: Online Edition

- Digital Du Chemin
- Electronic Sawyer
- The Gascon Rolls Project
- Greek Bible in Byzantine Judaism
- Henry III Fine Rolls
- Hofmeister XIX
- Inscriptions of Roman Cyrenaica
- Inscriptions of Roman Tripolitania
- Jane Austen's Fiction Manuscripts
- Jonathan Swift Archive
- Language of Landscape
- Digital Edition of Hermann Burger's Lokalbericht
- Mapping Medieval Chester
- Nineteenth-Century Serials Edition
- Records of Early English Drama (REED)
- Schenker Documents Online
- Sharing Ancient Wisdoms