# Kiln Documentation

***Release 1.0***

**Miguel Vieira**

**Sep 27, 2017**

# 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

## 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 (TEI) in the folder `KILN_HOME/webapps/ROOT/content/xml/tei`.
6. View HTML versions of the TEI XML at http://localhost:9999/text/<TEI filename>.html
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.

### 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.

## 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.
      - scripts — JavaScript files and libraries.
      - styles — CSS files and libraries.
      - **templates — Templates for content display.**
        admin - Templates for the admin system.
    - **content**
      - images — Project/content images.
      - **xml**
        tei — TEI 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).

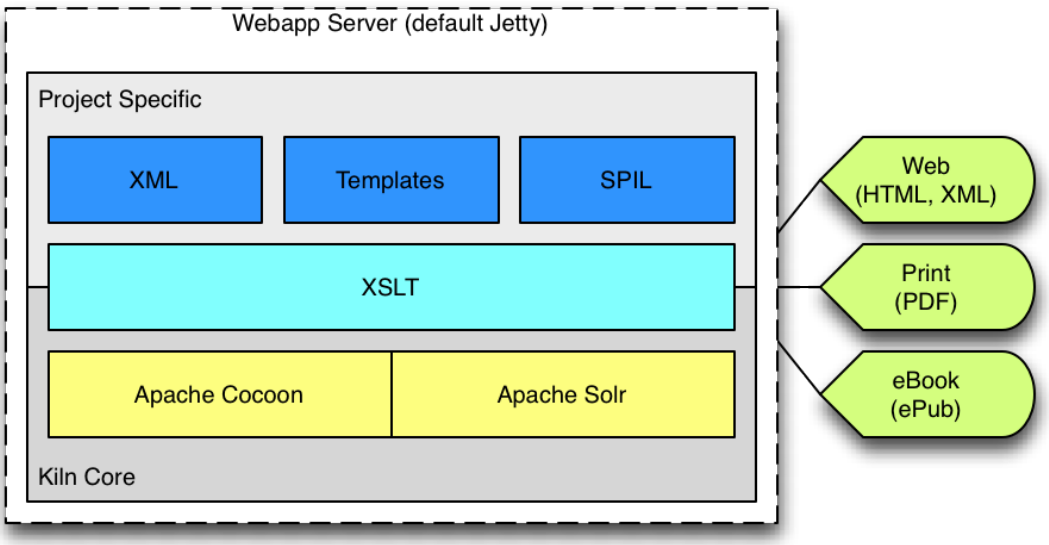
    * sitemap.xmap — Cocoon default sitemap (do not modify).
    * **sitemaps — Project's sitemaps.**
        - admin.xmap — Admin and editorial pipelines.
        - config.xmap — Configuration (global variables, etc).
        - internal.xmap - Internal (not exposed by URL) pipelines.
        - main.xmap — Main pipelines.
        - rdf.xmap - RDF pipelines.
        - solr.xmap - Search pipelines.
    * **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.
        - 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.
    * WEB—INF — Webapp configuration.
  - openrdf-sesame and openrdf-workbench — RDF / Linked Open Data framework.
  - **solr — Searching framework.**
    * **conf**
        - schema.xml - Definition of fields etc.

## 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.

## Architecture



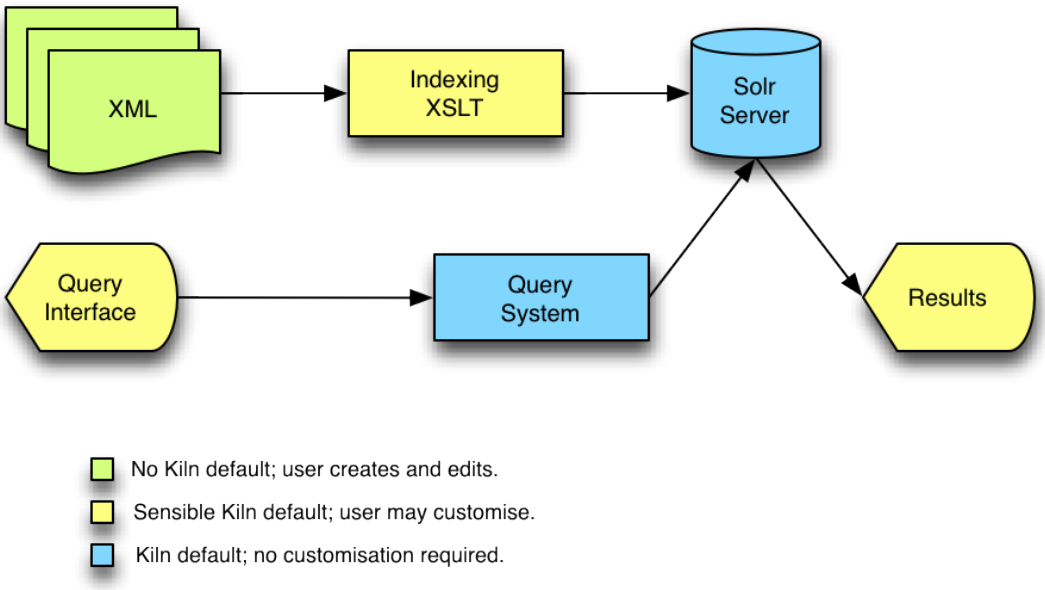

In a production web server context, Kiln integrates with other web publishing tools to support images (IIPimage/Djatoka), maps (GeoServer; MapServer; OpenLayers) and other data sources, like relational data (MySQL or other RDBMS).

## 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.

## 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 />
    <!-- Add in new content. -->
```

```
    <h2>Smaller title here</h2>
</kiln:block>
```

*Fuller documentation* is available.

# Running the webapp

## 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.

## 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`.

## 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.

## 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.

# 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 "//".

## 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>
```

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

## 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', ('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.

# 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.

## 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**.

## 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`.

## 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.

## 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`.

## 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.

# Searching and Browsing

## 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.

## 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.

## 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 data supplied in a parameter. This data must be in the form of a query string (without a leading "?"; eg: `fq=widget&facet=off`). `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).

### 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.

### 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.

## 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.

# RDF / Linked Open Data

Kiln includes the Sesame framework for handling RDF data. Sesame operates as a pair of Java webapps (`openrdf-sesame` and `openrdf-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.

## Setting up a repository

The Sesame 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`).

## Integration with Cocoon

Kiln integrates Sesame into its Cocoon processing via a Sesame 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`.

## Upgrading Sesame

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

Upgrading the two Sesame webapps is straightforward. First delete everything in `webapps/openrdf-workbench`, and everything except `app_dir` (where the project data is kept) in `webapps/openrdf-sesame`. Next unpack the contents of the two WAR files distributed with Sesame 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 Sesame download. It is unlikely that all of the JARs that come with Sesame are required. Unless there has been a change to Sesame'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 Sesame JARs have the version number as part of the filename).

# 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`.

# Multilingual sites

Kiln supports sites in which some or all of the content is available in more than one language. It does this by allowing for a language code as the first element in a URL, with the actual content URL hierarchy repeated for each language

after it.

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

Kiln treats the language code prefix similarly to the mount point of the webapp, so creating links between pages in the same language is no different from how it is in a monolingual site. Linking between languages is also simple.

### Implementation

`sitemaps/main.xmap` provides an example match incorporating 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`, which incorporates that code into the `xmg:context-path` variable (as well as being available separately as `xmg:language`. This variable is used in XSLT when creating links to textual content within Kiln.

What further use is made of this parameter/variable is site-dependent.

### HTTP content negotiation

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

## 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.

### 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.

### 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.
...
```

### 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);
```

## 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" />
  <!-- 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>
```

## 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.

### 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.

## 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).

## 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`.

## 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.
```

### 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.

## 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