CodeSharing: a simple API for disseminating our TEI encoding

Martin Holmes

1. Introduction

Although the TEI Guidelines are full of helpful examples, and other inititatives such as *TEI By Example* have made great progress in providing more access to samples of text-encoding to help beginners get started, there is no doubt that one of the biggest obstacles to encoders at many levels is finding out how other scholars and projects have chosen to encode a particular feature or use a specific tag or attribute. Burghart and Rehbein tell us that the majority of TEI users are "self-taught or learned by doing," and Dee (forthcoming) reports that users need "a source for a compendium of examples suitable for inductive learning." Many projects now share their XML code, but that in itself is only marginally helpful; it can take substantial time to sift through the XML code in a large project to find what you're looking for.

This talk presents a simple specification for an Application Programming Interface, along with a sample implementation written in XQuery and designed for the eXist XML database, providing straightforward access both for applications and end-users to sample code from any TEI project. The API is modeled on the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), a mechanism designed to allow archival search tools to ingest metadata from repositories. The CodeSharing protocol and the sample implementation were first presented at the Digital.Humanities@Oxford Summer School in July 2013, and a number of improvements

have been made to the code and specification based on feedback from that presentation.

2. Target audience

The CodeSharing proposal arises out of two separate but intersecting needs: those of novice encoders and project managers who are not really TEI experts, and those of people doing research into encoding practices on a large scale across multiple projects.

2.1. Novice encoders

At the time of writing, *The Map of Early Modern London (MoEML)* project, a typical grant-supported digital humanities project with a large encoding component, has a team of around seven or eight encoders working part-time, with frequent changes of personnel. The project provides regular TEI training, but there is usually a mix of experienced and rookie encoders, and project managers have to provide a lot of live help to supplement the documentation. One of the most difficult things for inexperienced encoders is to find examples of the usage of elements and attributes they haven't used before. There are of course lots of resources to help with this, including the *TEI Guidelines* examples, the *TEI by Example* project, and Marjorie Burghart's *Cheatsheets*, but it is unusual to find in such external resources enough examples of the use of a particular element or attribute which exactly match the current use-case. It is really more effective to search the codebase of your own project.

The *MoEML* encoders do have access to a lot of the existing codebase, but doing text searches of this is often ineffective. They are not normally familiar with XPath, XQuery, or regular expressions, and most will never learn them, so in searching for e.g. a **<birth>** element which has a **@notBefore-custom** attribute, they will search for "**<**birth notBefore-custom," and

therefore miss all the **birth** elements which happen to have their **@datingMethod** attribute first, or which have two spaces between the tag name and the attribute name instead of one. Searching the entire codebase may also retrieve examples of encoding in obsolete or unedited documents, which might provide bad examples. It is more effective, therefore, to enable them to search the online XML database which contains only documents which have actually been published.

To serve these needs, we began to think about writing a simple search interface which would form part of our *MoEML* web application, and which would provide access to lots of examples of individual tags and attributes.

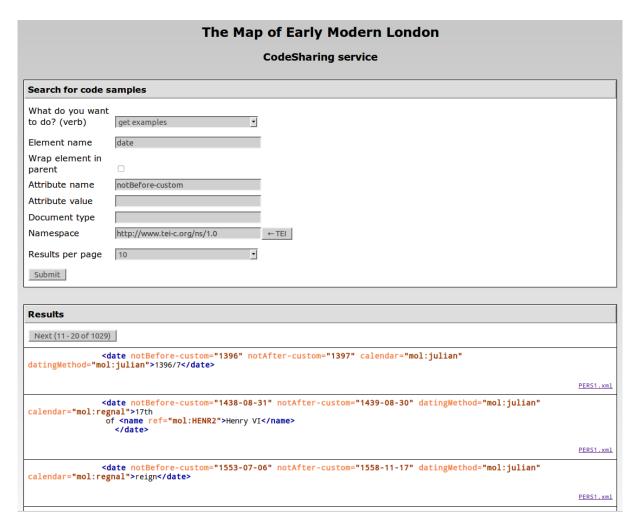


Figure 1. The Web interface of the CodeSharing service on the *MoEML* site.

This straightforward form-based interface enables our encoders to retrieve examples of encoding quickly and easily from across our text collection.

2.2. Research into encoding practices

As I worked on the interface above, I also began to think about broader possibilities. In our work on the TEI Council, we frequently find ourselves asking:

- Do people ever actually use this element or attribute?
- If so, how do they use it?

We typically resort to posting questions to the TEI-L mailing list to ask for examples from the community, but this is a rather slow, hit-and-miss method of gathering data. If we could retrieve large numbers of sample usages of particular elements or attributes and analyse them, this would be very helpful in development of the *Guidelines*, and in assessing the impact of any change we might make to the schemas or recommendations. We might also be able to supply fresh examples for the *Guidelines* text itself by selecting from among thousands of sample usages, rather than drawing only from our own projects or concocting examples. It would therefore be very helpful if there were an API that projects could implement to provide access to samples of their encoding practice in a standardized way, such that the same query could be run over multiple projects simultaneously and the results combined.

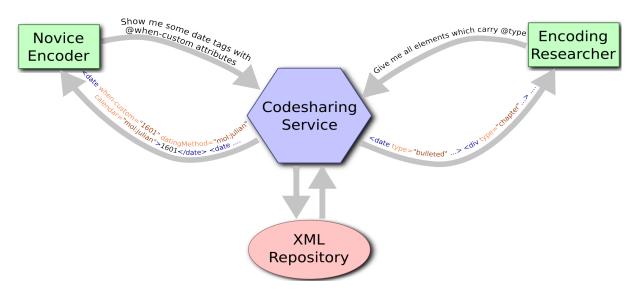


Figure 2. A CodeSharing service can serve the needs of novice encoders as well as encoding researchers.

3. The model: OAI-PMH

To answer these needs, I designed a web service that could be provided by large- and medium-scale encoding projects, enabling anyone to gather examples of their encoding practice

directly from their data. I modelled my protocol on an existing, well-tested system: the Open Archives Initiative Protocol for Metadata Harvesting, or OAI-PMH¹, which I had previously implemented for another project.

OAI-PMH is commendably simple and well designed. A participating repository may be a data provider, which exposes structured metadata through a web service implementing the OAI-PMH API; or a service provider, which gathers that metadata through requests to the data providers. The service providers can then act as meta-repositories, or federated archives, providing search functionality which encompasses the collections of all the data providers who have exposed their metadata for harvesting. An example is OCLC's WorldCat², which aggregates data from a large number of repositories and makes them searchable from a single interface. The OAI-PMH API is based on HTTP requests using GET or POST. It is designed to allow a harvester to find out what kinds of resources a repository has, and to gather full metadata records. All responses are in XML, and conform to a standard schema.

OAI-PMH is based around six core "verbs:"

- Identify
- ListMetadataFormats
- ListIdentifiers
- ListSets
- ListRecords
- GetRecord

A request like this³: http://bcgenesis.uvic.ca/oai.xq?verb=Identify will result in a response in

^{1.} http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

^{2. &}lt;a href="http://www.worldcat.org/">http://www.worldcat.org/

^{3.} This request is made to the OAI-PMH interface provided by the *Colonial Correspondence of BC and Vancouver Island* project, at http://bcgenesis.uvic.ca.

XML which provides identifying information about the repository:

```
<OAI-PMH xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
 <responseDate>2013-05-08T18:09:31.047Z/responseDate>
 <request verb="Identify">http://bcgenesis.uvic.ca/oai.xq</request>
 <Identify>
  <repositoryName>The colonial despatches of Vancouver Island and British Columbia
  1846-1871</repositoryName>
  <br/>
baseURL>http://bcgenesis.uvic.ca/oai.xq</br/>
baseURL>
  orotocolVersion>
  <adminEmail>mholmes@uvic.ca</adminEmail>
  <adminEmail>cpetter@uvic.ca</adminEmail>
  <earliestDatestamp>2012-11-19T12:00:00Z</earliestDatestamp>
  <deletedRecord>no</deletedRecord>
  <granularity>YYYY-MM-DD</granularity>
  <description>
  <oai-identifier
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai-identifier
http://www.openarchives.org/OAI/2.0/oai-identifier.xsd">
   <scheme>oai</scheme>
   <repositoryIdentifier>bcgenesis.uvic.ca</repositoryIdentifier>
   <delimiter>:</delimiter>
   <sampleIdentifier>oai:bcgenesis.uvic.ca:B63030SP.scx</sampleIdentifier>
  </oai-identifier>
  </description>
 </Identify>
 </OAI-PMH>
```

The harvester can use the **ListRecords** verb to retrieve individual records, which are typically in the form of Dublin Core elements embedded in a larger structure in the OAI namespace:

```
<record>
<header>
<identifier>oai:bcgenesis.uvic.ca:B585TE13.scx</identifier>
<datestamp>2013-12-17T14:29:44.553-08:00</datestamp>
<setSpec>1858</setSpec>
<setSpec>publicOffices</setSpec>
<setSpec>B.C.</setSpec>
```

```
</header>
  <metadata>
  <oai_dc:dc xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/oai_dc/
http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
        <dc:title>The colonial despatches of Vancouver Island and British Columbia 1846-1871:
11566, CO 60/2, p. 291; received 13 November. Trevelyan to Merivale (Permanent Under-Secretary)</dc:title>
        <dc:date>1858-11-12</dc:date>
[...]
        </oai_dc:dc>
        </metadata>
        </record>
```

Most repositories will have thousands of records, and retrieving them all at once would place an unacceptable burden on the server and network infrastructure, so OAI-PMH has a built-in staging system. Records are supplied in batches, and each batch ends with a *resumption token* which acts as a flow-control device:

```
<resumptionToken completeListSize="7154">
from:2010-01-01;until:2020-01-01;set:;next:21
</resumptionToken>
```

The format of the resumption token is not defined by the specification; the data provider may use any suitable format, and the harvester simply has to echo it back to the data provider to get the next set of records. This gives the data provider complete control over how rapidly it provides the data, and even in the course of a large transfer, the provider can throttle or accelerate its provision of records in response to local conditions such as server load or network bandwidth.

4. The CodeSharing service

The complete specification for the CodeSharing API is available at

http://mapoflondon.uvic.ca/codesharing_protocol.xhtml, as part of the sample implementation on

the *MoEML* site; in what follows I cover only some key aspects of it.

4.1. The request API

CodeSharing is an XML-based API provided over HTTP, just like OAI-PMH. On the model of OAI-PMH, it's also based on a "verb" parameter, with five possible values:

- identify
- listElements
- listAttributes
- listNamespaces
- getExamples

The first four are designed to discover the nature of the repository, and what elements, attributes and namespaces occur in its document collections. The final value is a request for actual example encodings; when that value is used, other key-value pairs provide details about what has been requested:

- elementName
- attributeName
- attributeValue

Each parameter is used to specify what examples are being requested. These can obviously be combined, so a request for: **elementName=hi&attributeName=rend** will retrieve only **hi** elements which have **@rend** attributes; **attributeName=rend&value=italic** will retrieve any elements which have **@rend="italic"**.

Two further parameters are available:

- **namespace** (the namespace for requested elements, defaulting to the TEI namespace)
- **wrapped** (whether or not to return the parent containing the target element)

So a request for: **elementName=hi&wrapped=true** will retrieve **<hi>** elements in the context of their parent element. Encoders find it helpful to see not just the target element, but the surrounding context too.

Finally, we have to consider flow control, as in the case of OAI-PMH. It would be disastrous to attempt to honour a request for all of the <TEI> elements in a large collection; we need to negotiate a reasonable chunk size for the harvester and the server. In the case of OAI-PMH, the data provider always dictates the number of records it is prepared to supply. In the case of CodeSharing, I wanted to allow a little more flexibility, so there is a further parameter the harvester can provide:

• maxItemsPerPage (a positive integer)

The provider service is not required to honour this request; it may decide to send fewer items. But it may be sensitive enough to know, for instance, that when the request is for a relatively small element such as <hi>i, it might be practical to send 100 examples in one response, whereas it might send only 20 in the case of requests for or <div> elements.

4.2. The response

What form should the server's response take? The obvious answer is that it should be XML, and in fact that it should be TEI P5 XML. The exact format of the response document is only loosely specified, although some parts of it must follow certain rules. If the value of the **verb** parameter is **listElements**, for instance, then the body of the document must contain the list of all elements appearing in the collection as a list:

```
list rend="bulleted">
<item>
```

```
<gi>author</gi>
</item>
<item>
<gi>availability</gi>
</item>
<item>
<gi>back</gi>
</item>
[...]
```

Similar structures are used to list attributes and namespaces.

For returning actual examples, CodeSharing makes use of the <egXML> element which is also used for example code in the *TEI Guidelines*. The <egXML> element is in its own special namespace, http://www.tei-c.org/ns/Examples, and all the elements which are children of it, in the example code, are also by default in that namespace. This is useful, because it means that we can easily distinguish example code from other parts of the TEI file. (It also means we can use the API to retrieve examples of code *which themselves are intended as examples in their original context*.)

In addition to the results of the query, the protocol specification also requires that the parameters of the original request be returned to the requestor; this means that the result document is a complete and self-contained record of the query and results. Full details are available in the protocol documentation.

5. A sample implementation

A sample implementation of the CodeSharing protocol, including an HTML front-end, as shown in <u>Figure 1</u>, is available at http://mapoflondon.uvic.ca/codesharing.htm. It is written in XQuery

3.0 and runs in the eXist XML database which hosts the *MoEML* web application. The open-source code is available on SourceForge at https://sourceforge.net/projects/codesharing/, and includes these files:

- **codesharing.xql** (the XQuery implementation providing responses to queries in XML)
- codesharing_config.xql (a simple settings file that tailors the service to your own project)
- **codesharing.xsl** (a transformation which produces the HTML search page you see on the *MoEML* site)
- **codesharing_protocol.xhtml** (a semi-formal description of the API)
- **codesharing.odd** (an ODD file from which a schema can be generated to validate CodeSharing API responses)

I would welcome any contributions in the form of improvements, suggestions, and implementations in other languages.

Bibliography

- Burghart, Marjorie, and Malte Rehbein. 2012. "The Present and Future of the TEI Community for Manuscript Encoding." *Journal of the Text Encoding Initiative* 2. http://jtei.revues.org/372. doi:10.4000/jtei.372.
- Dee, Stella. Forthcoming. "Learning the TEI in a Digital Environment." *Journal of the Text Encoding Initiative* 7.
- *Open Archives Protocol for Metadata Harvesting (OAI-PMH)* Version 2.0. 2008. http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm.
- Van den Branden, Ron, Melissa Terras, and Edward Vanhoutte. 2010. *TEI by Example*. http://www.teibyexample.org/.