Logger

Version 0.1

Contents

[1. Introduction 3](#_Toc377634896)

[1.1 Overview 3](#_Toc377634897)

[1.2 Optional XLIFF validation 3](#_Toc377634898)

[1.3 How it works 3](#_Toc377634899)

[1.4 Resource Identification 3](#_Toc377634900)

[2. Technologies Used 4](#_Toc377634901)

[2.1 Triple Store 4](#_Toc377634902)

[2.2 XSLT 4](#_Toc377634903)

[3. How-Tos 4](#_Toc377634904)

[3.1 Provenance logging Algorithm 4](#_Toc377634905)

[3.2 Updating the XLIFF to RDF transformation 5](#_Toc377634906)

[3.3 Deployment 5](#_Toc377634907)

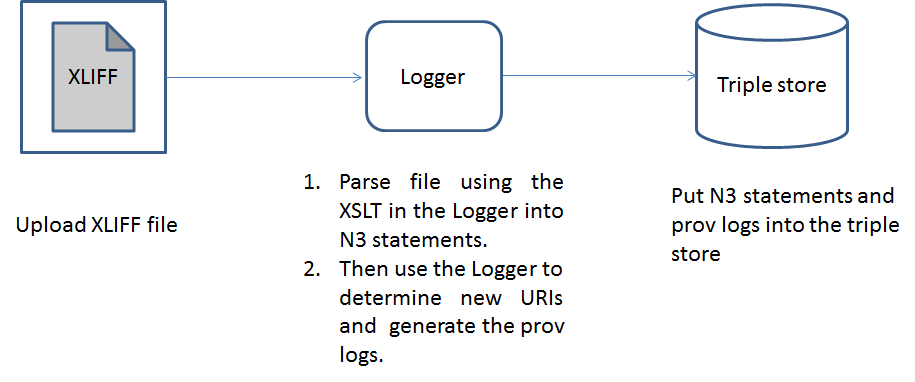
[3.3.1 Configuring for triple store server 5](#_Toc377634908)

[3.3.2 Configuring the Logger for various triple stores 5](#_Toc377634909)

[3.4 Using Java class 5](#_Toc377634910)

# 

# 1. Introduction­­



**Figure 1: Logger Overview**

# 1.1 Overview

The Logger component is a servlet which receives successive versions of XLIFF files (provided by SOLAS upon completion of each process), transforms this XLIFF input into RDF statements and merges these statements into the triple store. It also generates extra RDF statements to track provenance information.

# 1.2 Optional XLIFF validation

Logger optionally validates input against the XLIFF schema. If validation is enabled and XLIFF input is found to be non-conformant, processing will be aborted with the relevant validation error message in the response. Note that this feature requires Internet connectivity as the OASIS XLIFF schema references external elements on the WWW, and proxy settings may therefore need to be configured in Logger (in file config.properties) if XLIFF validation functionality is enabled.

# 1.3 How it works

Logger performs the following tasks for each input file it receives:

1. Transform the input XLIFF file into a set of RDF statements. This is done through an XSL transformation which processes the XLIFF document to produce N3 statements.
2. Merge these RDF statements into the triple store.
3. Add provenance information. A new "log" URI is created with associated statements, such as a timestamp and an input "job", and is linked using a provenance statement to each of the newly added URIs that have resulted from the merge.

# 1.4 Resource Identification

The process of logging iteratively into the triple store relies on the use of unique identifiers, which make it possible to determine when statements found in successive input files refer to identical resources. This is a challenge for XLIFF documents, which do not provide intrinsic strong identifiers. For instance, the job-id tag is optional (as per the OASIS XLIFF schema) and applies to each file element; there is no equivalent document-level. The XSLT constructs URIs based on the job id (assuming it is present and unique, which is true in the sample files but may not hold in some cases) to which names are appended in a manner that matches the structure of the XLF document e.g. <http://www.cngl.ie/jobs/b043a10ead/content.txt/#2/MT-Leverage/google>. This can of course be tweaked by changing the XSLT document. If necessary, an external parameter (a unique id from another system for example) can be passed to XSLT processor as an input parameter and used in the XSLT document.

# 2. Technologies Used

# 2.1 Triple Store

Lastly, an off-the-shelf triple store and it associated web service provides data persistence and allows the other components to add triples to the store and retrieve triples through SPARQL queries. The application should work with any SPARQL endpoint (note: has been tested with Jena and Seasame).

# 2.2 XSLT

The conversion of XLIFF files to RDF statements is done using XSLT. This makes it easy to modify and tweak the conversion process outside of the application code. We use the N3 format as the XSLT conversion output as it syntactically simple and easy to read. XSLT conversion is followed by post-processing to add provenance information.

# 3. How-Tos

# 3.1 Provenance logging Algorithm

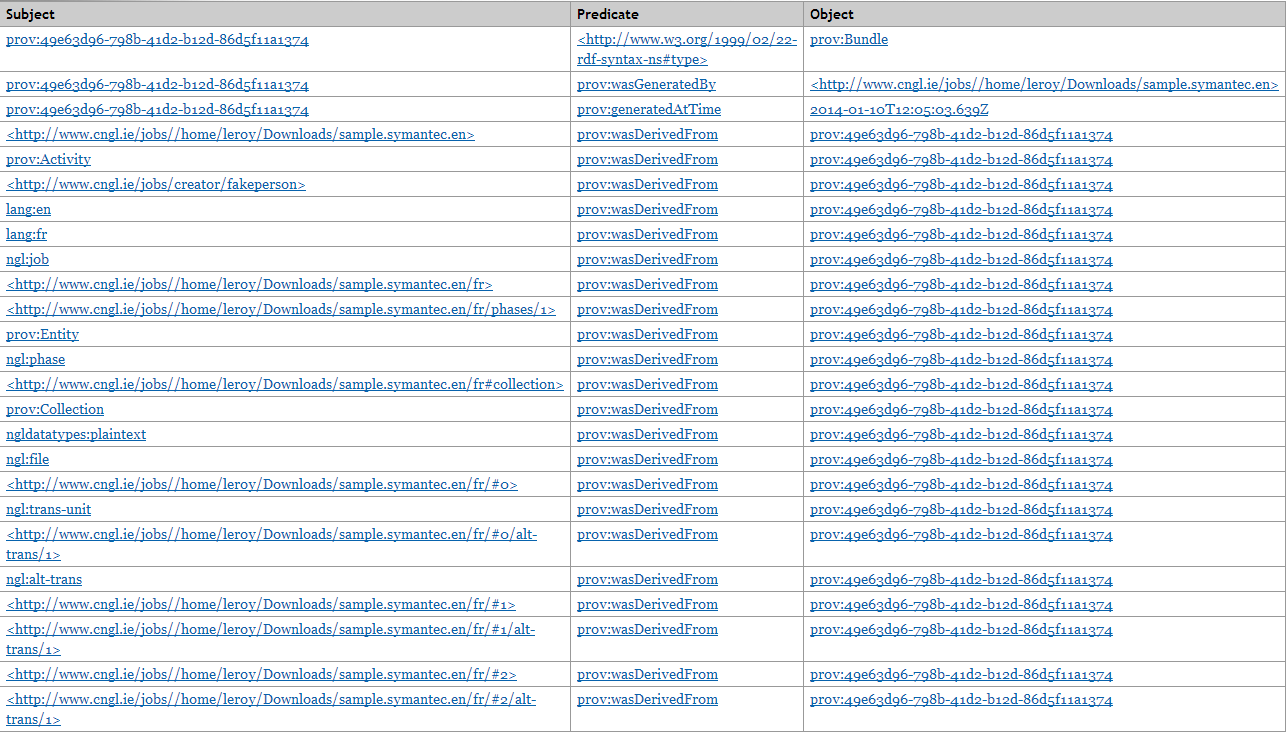
1. Upload a XLIFF document.
2. Parse the XLIFF document using XSLT into N3 statements.
3. Then check the generated N3 statements URIs against the URIs in the triple store. This check is done at the merge stage where new URIs and data are added to the graph and old URIs are being merged with the graph along with old/new data
4. A prov id is generated by a UUID generator which is available **java.util.UUID** package. A UUID is required as the provenance URI has to be unique. Then do the following dependent on the results of step 3:

*if(there are new URIs) then*

*create a new prov record which indicates what file has generated this new prov record, the time the prov record is generated at and the new URIs that were added into the triple store*

*else then*

*just make a prov record which record which indicates what file has generated this prov record and the time the prov record is generated at*



**Figure 2: Sample provenance log**

# 3.2 Updating the XLIFF to RDF transformation

The transformation of XLIFF files to RDF is done through the XSLT document in Logger/src/xliff2n3/xliff2n3.xsl. All statements other than the provenance statements (which are added programmatically in Logger/src/xliff2n3/Graph.java using URIs defined in Logger/src/xliff2n3/Constants.java) originate from this XSLT document, which can be tweaked to replace the grammar and the URIs used. Namespaces are included in this XSLT document too, and will be automatically added to the triple store server namespace list. After making any changes to the XSLT, you will probably want to clear the triple store and re-process the sample files to see the effect of your changes.

# 3.3 Deployment

# 3.3.1 Configuring for triple store server

Create a repository/dataset named for example CNGL. The type (DB or file-based) is up to you.

# 3.3.2 Configuring the Logger for various triple stores

* For Logger with Sesame, edit entries DFLT SESAME URL and DFLT REP ID in config.properties to match the URL of your Sesame Server and the name you have chosen in 3.3.1.
* For Logger with Jena, edit entries DFLT JENA URL SPARQL ENDPOINT and DFLT JENA URL UPDATE ENDPOINT in config.properties to match the URL of your Fueski Server and the name you have chosen in 3.3.1.

# 3.4 Using Java class

Check out Javadoc for more details of the classes and functions in the Logger Java project: <https://github.com/CNGL-repo/IA/tree/master/Demo%20Scenario/logger/javadoc>