OpenSextant Developers Guide

© The MITRE Corporation 2012

Table of Contents

[1 Overview 2](#_Toc345669384)

[2 Document Processing 3](#_Toc345669385)

[2.1 Document Reset 4](#_Toc345669386)

[2.2 Tokenizer 4](#_Toc345669387)

[2.3 Sentence Splitter 4](#_Toc345669388)

[2.1 Case Detector 4](#_Toc345669389)

[2.2 OpenSextant Vocabulary Tagger 5](#_Toc345669390)

[2.3 Phonetic Tagger 5](#_Toc345669391)

[2.4 Naïve Tagger 6](#_Toc345669392)

[2.5 NaiveTagger PostProcessing rules 6](#_Toc345669393)

[2.6 Geocoord Finder 6](#_Toc345669394)

[2.7 OpenSextant Rules Engine 6](#_Toc345669395)

[2.8 Cantilever 7](#_Toc345669396)

[2.9 Scoring 8](#_Toc345669397)

[3 Output Formats 8](#_Toc345669398)

[3.1 KML 8](#_Toc345669399)

[3.2 ESRI Shapefiles 8](#_Toc345669400)

[3.3 CSV 10](#_Toc345669401)

[3.4 JSON 10](#_Toc345669402)

[3.5 WKT 10](#_Toc345669403)

[4 Build OpenSextant 10](#_Toc345669404)

[4.1 Prerequisites: 10](#_Toc345669405)

[4.2 Building OpenSextant 11](#_Toc345669406)

[4.2.1 Get GATE 11](#_Toc345669407)

[4.2.2 Get OpenSextant 11](#_Toc345669408)

[4.2.3 Configure 12](#_Toc345669409)

[4.2.4 Build 12](#_Toc345669410)

[4.2.5 Rebuild gazetteers (optional) 12](#_Toc345669411)

[4.3 The OpenSextant Projects 13](#_Toc345669412)

[4.4 Dependencies 14](#_Toc345669413)

[4.4.1 Runtime Dependencies for the CREOLE Plugin 14](#_Toc345669414)

[4.4.2 Dependencies Between OpenSextantToolbox and Gazetteer 14](#_Toc345669415)

[4.5 Build Routine Reference 15](#_Toc345669416)

[5 APPENDIX. Gazetteer Reference 15](#_Toc345669417)

# Overview

OpenSextant identifies and disambiguates geospatial information in unstructured text. Geospatial information includes named places as well as explicit spatial coordinates such as latitude-longitude pairs and Military Grid References. Named places can be any geospatial feature, such as countries, cities, rivers, and so on. Disambiguation refers to matching a named place in a document with one or more entries in a gazetteer and providing a relative confidence level for each match.

OpenSextant is implemented in Java and built on top of the General Architecture for Text Engineering (GATE) framework (<http://gate.ac.uk/>). OpenSextant currently handles only English documents, but GATE has been extended to a number of languages and OpenSextant could be as well.

A high level view of OpenSextant document processing is shown in Figure 1. Processing is performed in the following sequential stages:

* Identify candidate named places in the document. These are text mentions that match gazetteer entries; the gazetteer is typically stored in a database.
* Identify geocoordinates (lat/lon, Universal Transverse Mercator (UTM), and Military Grid Reference System (MGRS) explicitly stated in the text, using regex patterns.
* Collect and associate supporting and contradicting evidence for a "this-is-a-place" decision for each candidate. This is accomplished with Java Annotation Pattern Engine (JAPE) rules (<http://gate.ac.uk/sale/tao/splitch8.html#chap:jape>) that utilize vocabularies of specific words that may appear in the text.
* Aggregate evidence into a single "place-or-not-place" confidence score.
* For candidates that match multiple gazetteer entries, rank the set of candidate-gazetteer matches giving each a "this-specific-place" confidence score.

A thresholding of the "place-or-not-place" confidence score gives the final set of candidates which are labeled "place".

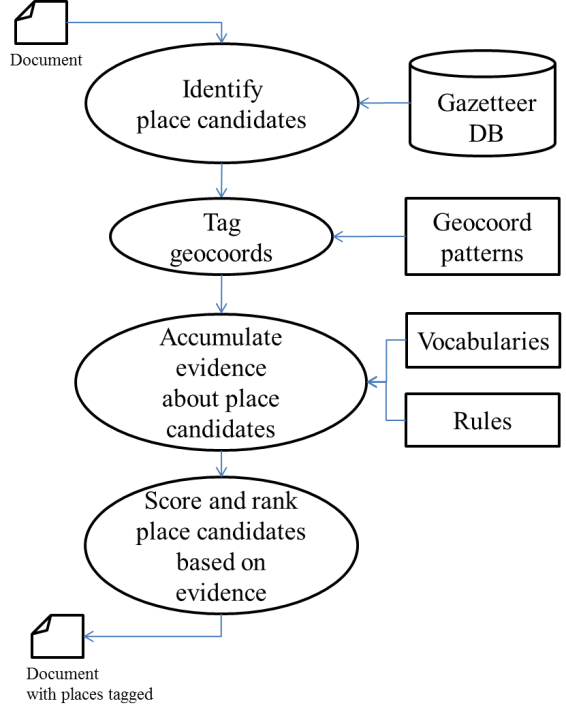


Figure . OpenSextant document processing.

OpenSextant processes any document type recognized by GATE. This includes plain text, Microsoft Word, PDF, and HTML. Multiple documents can be submitted as a zip or tar archive. Geospatial information is written out in commonly used geospatial data formats, such as KML, ESRI Shapefile, CSV, JSON or WKT.

# Document Processing

OpenSextant document processing is executed by a GATE application. This is a serial pipeline of GATE processing resources specified at run time by a GATE application file, referred to as a GAPP file. The document processing pipeline is show schematically in Figure 2.

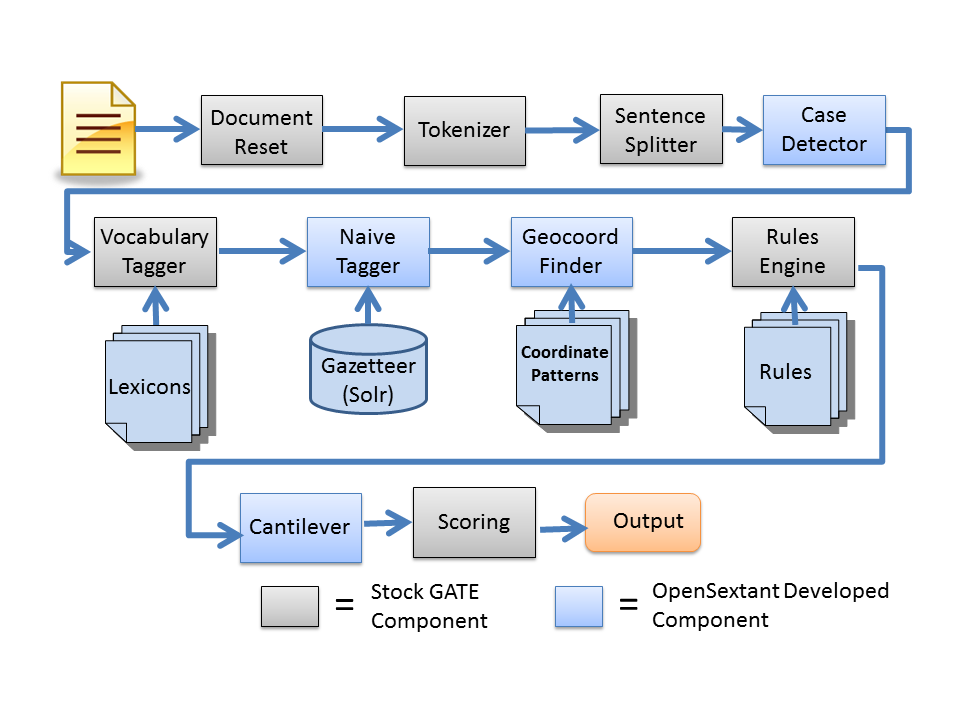


Figure . The OpenSextant document processing pipeline.

The following GATE components comprise the OpenSextant pipeline:

## Document Reset

This is a processing resource from the set of standard GATE resources referred to as ANNIE (<http://gate.ac.uk/sale/tao/splitch6.html#chap:annie>) that deletes any tags originally present in the document.

## Tokenizer

This is another stock ANNIE resource. It identifies the tokens in the document. Tokens are atomic units of text. They can be words, symbols, or whitespace characters.

## Sentence Splitter

Also a stock ANNIE resource, the Sentence Splitter does exactly what its name implies: identify sequences of tokens that constitute sentences.

## Case Detector

The Case Detector attempts to determine if a document is written in all/mostly uppercase, all lowercase or “proper” mixed case (i.e. first word in sentence and proper nouns have an initial uppercase ..). The purpose of this information is to allow rules which make assumption about case to only be applied when their case assumptions actually apply. This combination of case detector and case form specific rules improves accuracy when handling certain types of document form and contents such as informal documents (emails, blogs tweets), audio transcripts and certain government documents which are not in “proper” case.

## OpenSextant Vocabulary Tagger

This resource consists of the stock ANNIE Gazetteer and 51 lexicons, or lists of words, organized by category. In this context, a gazetteer is a software module that finds and tags occurrences of words in a document; the term is also used to refer to the set of words to be found. The ANNIE Gazetteer tags each word it finds with the category it belongs to. A word can consist of more than one token.

The 51 lexicons comprise the OpenSextant vocabulary. Some of the categories are:

* Cardinal directions
* Units of distance
* Nationalities
* Organizations
* Prepositions that relate to places (e.g., outside of, throughout, towards)
* Verbs that relate to movement between places (e.g., arrive, depart, drive)
* Common names in various languages

The words tagged by the gazetteer are used in subsequent processing to determine whether a place name in the text actually refers to a place or not.

## Phonetic Tagger

Phonetic algorithms are used to match different words with similar pronunciation. The Phonetic Tagger tags each token with a string representing its pronunciation. Encoders for well-known phonetic algorithms are available from the Apache Software Foundation (see <http://commons.apache.org/codec/api-release/org/apache/commons/codec/language/package-summary.html>). Phonetic matching has the potential to improve geotagging performance for texts containing transliterations of foreign place names.

The current Phonetic Tagger currently supports 15 non-exact/phonetic algorithms which range from very simple (remove case and diacritics) to more sophisticated language based ones ( Double Metaphone and Beider-Morse algorithms). The default algorithm removes case information, reduces diacritics to their base character, and removes certain "noisy" punctuation. If the Phonetic Tagger is included in the processing pipeline, then the Naïve Tagger will use the transformed strings in place of the original strings when matching tokens to place names.

(Note that this component is not used with the Solr based version of the NaiveTagger (see below) since that component does its own phonetic reduction.)

## Naïve Tagger

The Naïve Tagger identifies the names of geographic locations in the text and creates a place candidate object for each name it finds. Place candidates store information from the document relevant to determining whether the candidate is a place or not, along with the possible geographic locations it could represent.

There are currently two Naïve Tagger implementations available:

* Micro – Uses the stock ANNIE gazetteer.
* MySQL – Uses a MySQL database.
* Solr - Uses a Solr instance

The Micro Tagger works well for relatively small gazetteers that contain no more than a million or so places. The MySQL Tagger can handle much larger gazetteers. The Solr based tagger also scales to very large data sets and is substantially faster than the MySQL version.

## NaiveTagger PostProcessing rules

This stage resolves any potential overlapping or nested candidates created by the NaiveTagger. It essentially chooses the longest and rightmost candidate from any set of overlapping/intersecting candidates.

This stage also generates some convenience annotations (Country, Admin1, and NationalCapital) from the NaiveTagger generated candidates. These annotations are used in follow-on rule stages.

## Geocoord Finder

The Geocoord Finder uses regular expressions to identify lat/lon, UTM, and MGRS coordinates explicitly stated in the text. The regexes are intended to include all the various formats commonly used to represent coordinates, such as decimal degrees, degrees-minutes-seconds, or hemispheres before or after the angular measures.

## OpenSextant Rules Engine

This stage generates "this-is-a-place" evidence by identifying patterns of words from the OpenSextant vocabulary. The patterns are identified using finite state transducers implemented in GATE’s Java Annotation Patterns Engine (JAPE) language; see <http://gate.ac.uk/sale/tao/splitch8.html#chap:jape> for details. The patterns that are identified include:

* Overlapping place candidates
* Combinations of vocabulary and proper nouns that are more likely to be the names of organizations rather than places
* Associations between mentions of geographical feature types (e.g. village, river, bridge ) and place candidates
* Nested place expressions (e.g. "Wazi village, Watusi District, Iraq")
* Topological relationships between places mentioned (e.g. "this is inside that")
* Basic Encyclopedia (BE) numbers

Evidence is accumulated using these patterns, and patterns of these patterns, with 10 phases of transduction in all. This stage also includes the case-form-specific rules i.e. rules which are only applied to documents which are written in all upper case, all lower case or “proper” mixed case, as determined by the above Case Detector stage.

## Cantilever

This module is intended to deal with non-local (i.e. document level) evidence to support place/no-place and which-place decisions. It deals with two situations:

* It is very common for one mention (usually the first) of a place name (or person/organization..) in a document to have more contextual evidence than other later mentions, e.g. the first mention might be "the village of Wazi, Afghanistan" and later mentions might be merely "Wazi".
* It is also common for a place name to be mentioned with no local evidence, assuming the human reader can extrapolate from some larger context, e.g. if the document or document section is about Afghanistan, place names likely would not repeat Afghanistan for even the first mention.

To handle both of the above phenomena, the cantilever module propagates the local contextual evidence for each mention to all mentions in the document with the same base name. In addition to the propagated evidence the Cantilever module also collects "unconnected" evidence i.e. evidence that is not associated with any candidate. This evidence is used to handle the case of no local and no propagated evidence for a mention.

After the evidence has been collected and propagated , an internal Scorer module does the "which-place" scoring. The Scorer attempts to quantify (using a range of 0.0 - 1.0) the similarity between a place candidate which has been found in the document with potential matches from the gazetteer. The score consists of a weighted average of four one dimensional similarity measurements:

1. The similarity between the name in the document and the name in the gazetteer
2. The geometric distance between a geocoord associated with the candidate in the document and the geocoordinate of the entry in the gazetteer
3. The similarity/compatibility of a feature type associated with the candidate in the document and the feature type of the entry in the gazetteer
4. The similarity/compatibility between the admin structure associated with the candidate in the document and the structure of the entry in the gazetteer. Currently, this only considers country.

Note that all but the first characteristic (name) may be absent from a candidate. Each of four sub-scorers use both local information (primarily provided by the rules in the OpenSextant Rules phase) and (appropriately weighted) document-level information (primarily provided by the Cantilever module). This is to handle the common case, where for instance the document mentions a number of places and a one or more countries but doesn’t contain a phrase which contains both a place candidate and a country. The set of all countries mentioned in the document would be attributed as non-local evidence to the candidate. The score returned in this case would be reduced to indicate lower confidence.

## Scoring

This final stage applies a threshold to the "place-not-place" score, with candidates with a score above the designated threshold (currently 0.0) being labeled as places.

The two primary outputs from the pipeline are:

* Ranked lists (by confidence) of places associated with gazetteer entries
* Lat/lon, UMT and MGRS coordinates extracted from the text

These are the results that appear in the output. Only the highest ranked place for each gazetteer entry appears in the output, although it is possible to show the entire list for each entry for testing and debugging purposes.

# Output Formats

OpenSextant uses MITRE’s GISCore software to export results in common geospatial data formats. For each geolocation, the output contains the maximum amount of information the format supports. This can include the following data:

* The sentence or line of text in which the place name appears
* Federal Information Processing Standards (FIPS) country code
* GEOnet Name Server (GNS) feature class
* GNS feature code
* Confidence score

The supported data formats are described below.

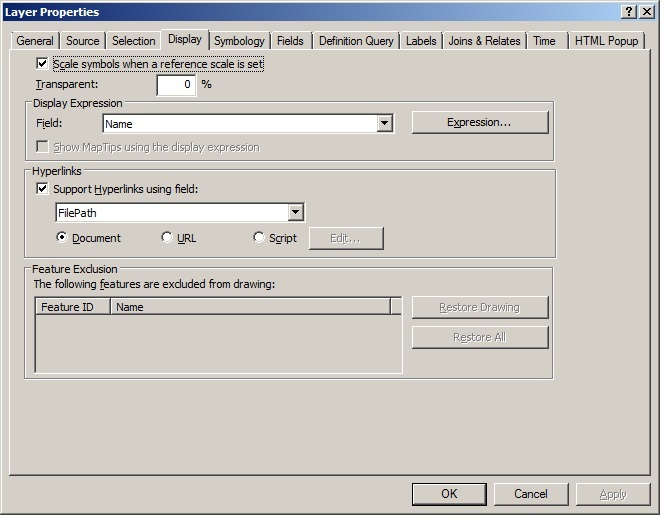
## KML

Within an OpenSextant KML file, placemarks from a document are contained within a folder whose name matches the name of the document. If the input file is a zip or tar archive containing multiple documents, the KML file will have a folder structure that mimics the directory structure of the archive.

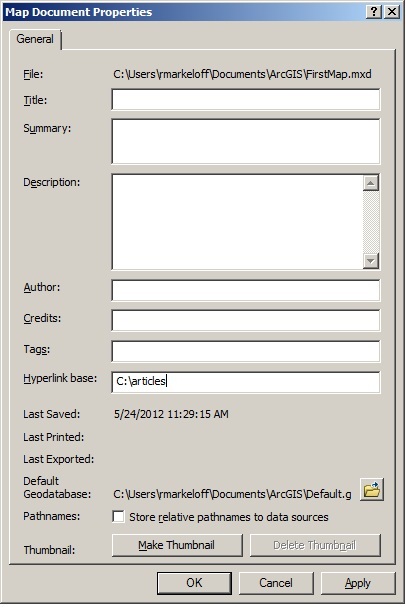
## ESRI Shapefiles

Shapefile output from OpenSextant consists of a zip archive containing .shp, .shx. .dbf, and .prj files. The shapefile also supports the hyperlink feature within ESRI AcGIS. If one zips up a directory structure containing documents for geocoding, processes the zip file through OpenSextant, and opens the shapefile with ArcGIS, it is possible to click on a map symbol and open the document in which that geolocation was found. The following steps are required to enable this feature:

1. In the Layer Properties menu, go to the Display tab and check the “Support Hyperlinks using field” box. Select FilePath for the field, as shown below.



1. In the File → Map Document Properties… menu, set the base path to the directory containing the documents, as shown below.



1. Clicking on a map symbol with the Hyperlink (lightning bolt) tool from the toolbar will now bring up the document in which the geolocation was found.

## CSV

The CSV output fields are described in Table 1.

|  |  |
| --- | --- |
| **Column Header** | **Description** |
| matchtext | The text mention that matches a gazetteer entry |
| start | The start offset for the text mention. Offsets point to positions within a Java String representation of the document. |
| end | The end offset |
| name | The place name from the gazetteer |
| lat | Latitude |
| lon | Longitude |
| cc2 | FIPS country code |
| feat\_class | GNS feature class |
| feat\_type | GNS feature type |
| confidence | Confidence score |
| filename | The name of the document file |
| filepath | The path to the file within an archive directory structure |
| method | For testing purposes only |

Table . CSV output fields.

## JSON

JavaScript Object Notation (JSON) output consists of serialized GeoBean and Geocoord objects. GeoBean objects encapsulate gazetteer records and Geocoord objects describe points on the globe. Each place annotation generated by OpenSextant contains a list of GeoBeans ordered by decreasing confidence score. Only the highest scoring GeoBeans in the lists are serialized. GeoBeans encapsulate their lat-lon coordinates in Geocoord objects. Geocoordinates that explicitly appear in the text are also represented by Geocoords, and these are serialized as well.

## WKT

Well-known text (WKT) output consists of a list of Point objects. WKT is only capable of describing geometry, and no other information is included.

# Building OpenSextant

This section is for those who wish to build OpenSextant from source code. If you want to use OpenSextant without modification see the *OpenSextant Users Guide* for instructions on installing and using the pre-built OpenSextant.

## Prerequisites:

In order to build OpenSextant from source code, you will need the following development tools:

1. Java - JDK 6.0 or later
2. Ant - version 1.8.1 or later
3. Ivy - version 2.3 or later
4. Maven (only needed if rebuilding the SolrTextTagger or GISCORE subcomponents)
5. Kettle (only needed if reprocessing source gazetteer data)
6. GATE (version 7.x or later)

## Overview

To build OpenSextant:

1. Get and install a distribution of the GATE framework
2. Get the OpenSextant distribution
3. Configure the build process
4. Do the Build

These steps are detailed below.

### Get GATE

A GATE distribution can be downloaded from its SourceForge page (<https://sourceforge.net/projects/gate/>)

Detailed instructions for how to install GATE can be found at <http://gate.ac.uk/sale/tao/splitch2.html#chap:gettingstarted>

Install GATE to somewhere: <OS\_HOME>/GATE

### Get OpenSextant

An OpenSextant distribution can be downloaded from SourceForge at <https://sourceforge.net/projects/opensextant/>

TBD: how will OpenSextant be packaged on SourceForge. Source only, source and prebuilt? Gaz data included? Prebuilt gaz solr\_home?

Unpack the OpenSextant distribution into <OS\_HOME>. You should end up with a directory structure that looks like:

* <OS\_HOME>
  + GATE
  + Gazetteer
  + giscore
  + LanguageResources
  + OpenSextantClient
  + OpenSextantToolbox
  + OpenSextantWebservice
  + SolrTextTagger
  + Xponents

### Configure

What actually needs to be set to do a basic build?

### Build

To build the OpenSextant Demo application :  
 cd <OS\_HOME>/OpenSextantClient   
ant

To build the OpenSextant WebService :  
 cd <OS\_HOME>/OpenSextantWebService  
ant

There are two final products of the build process:

* OpenSextantRunner.zip - A zip archive containing Windows and Unix shell scripts for running OpenSextant as a standalone application, either from a GUI or the command line.
* OpenSextant.war- A WAR for deploying OpenSextant as a RESTful Web service. WARs can be generated for both Tomcat and JBoss servers, with the default being Tomcat. The JBoss WAR excludes some JARs that are shipped with JBoss. See the note *Deploying OpenSextant on JBoss* for details.

The core products incorporate the same input/output functionality. They only differ in their interfaces in how they are invoked.

For on how to install/deploy, configure and run the OpenSextant client and web service, see the *OpenSextant Users Guide* .

## The OpenSextant Projects

OpenSextant consists of 5 separate projects, as shown in Figure 1. Building is accomplished with Apache Ant with help from Ivy and Maven . Invoking the Ant script default targets in the OpenSextantRunner and OpenSextantWebService projects produces the products described above. These targets trigger a cascade of builds across the dependent projects shown in Figure 1.

The remainder of this document describes the individual OpenSextant projects and the dependencies between them.



Figure . The OpenSextant projects and their dependencies. The final products are built from the OpenSextantRunner and OpenSextantWebService projects.

The 5 major projects within OpenSextant are:

* **OpenSextantToolbox**. The GATE-based toolbox plugin. It consists of components for building a GATE pipeline. This project also contains implementations of the algorithms and processes used by the GATE components.
* **OpenSextantClient**. Contains code for GUI and command line wrappers around the OpenSextant core. As well, the full release packaging is managed here.
* **OpenSextantWebService**. Contains code and other resources necessary for deploying OpenSextant as a Web service.
* **LanguageResources**. Contains various files required by OpenSextant at run time. It does not have an Ant script and does not need to be built.
* **Gazetteer**. Primarily contains software and data for constructing the OpenSextant gazetteers. At build time, files to support a text-based gazetteer option are copied from this project and included in the final products. Also copied from the Gazetteer project is a JAR used by both OpenSextant and the gazetteer construction process. The Gazetteer project does not have an Ant script and does not need to be built.

Related independent sub-projects are

* **Xponents**. Xponents is a family of information extractors for use in text processing pipelines. XCoord and XTemporal extract coordinate and date/time patterns respectively. FlexPat is a flexible pattern definition grammar API that lightly wraps around Java Regular Expressions. All three components have basic unit test facilities (which should be migrated towards JUnit) for self-evaluation.
* **SolrTextTagger** – a generic text tagging capability that uses a reference term list to tag a text input. It is implemented in Solr, as the typical use case is to use term lists that are both highly structured with metadata and have the potential to scale to thousands or millions of terms.
* **GISCore** – a family of I/O routines for manipulating and brokering GIS data formats in Java
* **GATE** - The GATE project is downloaded from the GATE Sourceforge site (<http://sourceforge.net/projects/gate/>). It is used as-is and is never rebuilt. OpenSextant currently uses GATE version 7.0. Upgrading to a newer version of GATE requires updating a list of GATE JAR dependencies (gateDependencies.xml) within the OpenSextantToolbox project. Although it is possible that switching to a newer version of GATE would also require modifications to the OpenSextant code, the GATE interfaces used by OpenSextant are relatively stable and not likely to change from one version to the next.

The sub-projects (giscore, SolrTextTagger, FlexPat, XCoord, and XTemporal) provide Java software used by OpenSextant. Each of these projects has an Ant build script and is built during the OpenSextantToolbox build. The OpenSextantToolbox build copies the JARs produced by these builds to the OpenSextantToolbox project.

## Dependencies

### Runtime Dependencies for the CREOLE Plugin

The OpenSextant GATE CREOLE plugin requires that a set of JARs required at runtime be placed in the OpenSextant directory under the GATE plugin directory. The list of JARs is contained in the file runtimeDependencies.xml in the OpenSextantToolbox project. This list, along with the GATE dependencies list previously mentioned, is accessed by the OpenSextantRunner and OpenSextantWebService builds.

# APPENDIX. Rebuilding the Gazetteer

OpenSextant comes with a pre-built copy (TBD: are we sure we are including a pre-built gazetteer) of gazetteer information which has been created from gazetteer information published by USGS and NGA. Although this pre-built gazetteer should suffice for many purposes, you may wish to update your gazetteer, add your own gazetteer data or otherwise modify the gazetteer contents. In order to do so, you must rebuild the gazetteer using the instructions in this appendix.

## Overview

The gazetteer build process involves three major steps

1. Get the source gazetteer data from the distribution websites
2. Clean, transform and augment that data into a single consistent form
3. Load the cleaned data into Solr

These steps are detailed below.

## Prerequisites

OpenSextant uses Pentaho Data Integration (aka Kettle) open source ETL (extract-transform-load) framework. This package is a general purpose ETL (extract-transform-load) framework, which OpenSextant uses to clean and transform source gazetteer data into a single consistent form.

Download and install a copy of Kettle package from <http://kettle.pentaho.com>. OpenSextant has been tested with Pentaho Data Integration (Community Edition) version 4.2.

## Configuring and Building the Gazetteer

edit build.xml in <OS\_HOME>/Gazetteer and change the following variable to the appropriate values for your installation:

solr\_home – the location where the Solr based gazetteer will be built   
kettle\_home – the location where you installed Kettle  
*NGA\_date – the date of the version of NGA data to use (see build.xml for details)*  
*USGS\_date – the date of the version of USGS data to use (see build.xml for details)*

Once the above values are set, run the default build target in ant:

ant

This will fetch the data from the NGA and USGS websites, run the multiple Kettle processes to clean and transform that data, and create and populate a Solr instance with the data in a form ready to be used by OpenSextant by setting the value of solr\_home in the OpenSextant configuration.

Please note that this entire process can take 1-2 hours depending on network and machine characteristics.

Also note that the above covers OpenSextant’s Solr based tagger. A MySQL and file based versions of this component are also included in the OpenSextant package but will likely be removed in the future.