WikiWord: Integrator

Table of Contents

Intro

WikiWord is a system for extracting a thesaurus from Wikipedia. It was originally developed by Daniel Kinzler as part of his diploma thesis at the University of Leipzig in 2007 and 2008. In 2008 and 2009, development was continued for Wikimedia Deutschland and as part of contract work for Knewco, Inc. WikiWord is free software released under the LGPL.

WikiWord's Integrator module is designed top use this data as a glue between different data sets, that is, to map between different vocabularies, standardized or natural.

Process

This section outlines the process of semantic vocabulary integration using WikiWord.

Building the WikiWord Thesaurus

The WikiWord thesaurus is extracted from Wikipedia's XML dumps.

Building the WikiWord thesaurus is a long-running task, best performed centrally on appropriately powerful hardware. It is assumed that this will be performed as a service by Wikimedia.

Thesaurus Extraction

For each Wikipedia dump (I.e. for each Language), a thesaurus is extracted,

Thesaurus Merging

The thesauri are then merged into a multi-lingual thesaurus.

Building Concept Properties

Semantic integration means mapping concepts from one vocabulary to another. In the context of this document, concepts from a foreign authority are mapped to WikiWord concepts.

Concept properties are the basis for mapping concepts. Properties include information supplied from the foreign authority, properties extracted from Wikipedia articles as well as the information in the thesaurus proper, that is, the terms/labels used to refer to a given concept.

Property Extraction

WikiWord allows for additional properties to be extracted from Wikipedia pages to be attached to an existing thesaurus. This way, a generic thesaurus can be used for a variety of domains, while the properties extracted can be tailored to the desired domain. Concept properties are most often extracted from so called "infobox" templates contained in Wikipedia articles, but also from other special purpose templates and categorization tags.

Property Import

Concepts defined by a third party authority are imported into the WikiWord database as sets of properties attached to a concept id. They are represented in the database as generic triples of concept id, property name and property value.

External concepts can be imported directly from CSV/TSV files or from the result of an SQL query.

Mapping Concepts

Mapping concepts is the task of the Integrator module. "Integrator" because it is designed to integrate vocabularies from different source, using the WikiWord thesaurus as a kind of glue between them. All entry points for loading properties, building associations, building mappings and filtering mappings follow the same pattern: a stream of records (depending on task, of `FeatureSet`s, `Association`s, or `MappingCandidates`) is generated from a data source (generally a CSV file or a database query). The stream is represented by an instance of `DataCursor`, which also implements most of the logic of transforming, merging, and filtering records. A processor (`WikiWordProcessor`) steps through each record from the cursor and passes it to storage, implemented by an instance of an interface derived from `WikiWordStoreBuilder`. The store is generally backed by one or more database tables.

The process performed by an Integrator application looks roughly like this:

1. The application is invoked, from the command line for from the BeanShell environment.
2. The application sets up the Store (target) and DataCursor (source)
3. The application then creates a Processor around the Store and calls it on the DataCursor.
4. The Processor fetches on entry after another from the DataCursor and passes it to the StoreBuilder. Note that any logic for filtering, grouping and converting of entries is usually implemented in the DataCursor, not in the Processor.

Building Concept Associations

Concept *associations* represent individual links between foreign authority concepts and WikiWord concepts. There may be several such associations between the same pair of concepts. Associations may be annotated with a variety of information about how the association was derived and how it is weighted.

Building Concept Mappings

Concept *mappings* represent aggregated links between foreign authority concepts and WikiWord concepts. There may be only one mapping between a given pair of concepts. However, the same foreign concept may be mapped to several WikiWord concepts, and a single WikiWord concept may be mapped to several different foreign concepts.

Mappings are generally derived from associations by a kind of "grouping" operation: all associations between a given pair of concepts are grouped into a single mapping entry. Annotation of the mappings are reduced to figures aggregated from the associations that defined the respective mappings.

Filtering Concept Mappings

To get the mappings that are desired for a given purpose, different kind of filters can be applied. One very common filter uses a threshold: all mappings that are below a given value according to some measure (often, the value of a specific "weight" annotation) are ignored.

Often, it is desired to get *unique* mappings, that is, to have a given foreign concept map to only one WikiWord concept. There are two default ways to achieve this: either by using the *best* available mapping for a given foreign concept, according to some measure. Or my simply ignoring all ambiguous mappings; This of course reduces the amount of mappings, but it also improved the level of confidence in the mappings.

Sometimes, it is desired to only get *exact, exclusive* matches — that is, not only to exclude any foreign concept for which there exists more than one mapping to WikiWord, but to also to exclude all WikiWord concepts mapped to more than one foreign concept. This yields a strict 1:1 relationship and avoids any mismatches in scope or granularity. This is particularly useful when transferring definitions from one authority to another.

Architecture

Modules

BrightByteUtil

This library contains basic utilities.

de.brightbyte.abstraction	Abstract access to properties of objects, be it bean properties, map entries, or something else.
de.brightbyte.application	Utilities for top level applications, such as command line parameter handling.
de.brightbyte.audit	Debugging and introspection utilities.
de.brightbyte.data	Data structures and processing. Matrices, sparse vectors, multi-maps, etc.
de.brightbyte.data.cursor	Data cursor implementations
de.brightbyte.data.filter	Generic filtering framework
de.brightbyte.data.measure	Generic measuring framework, especially for similarity resp. distance.
de.brightbyte.io	I/O utilities
de.brightbyte.job	Concurrent job handling utilities.
de.brightbyte.text	Text handling and parsing utilities.
de.brightbyte.util	General system level utilities.
de.brightbyte.xml	XML utilities and convenience functions.

BrightByteDB

This library contains classes for database access.

de.brightbyte.db	Database utilities and abstraction.
de.brightbyte.db.testing	Database test case utilities

WikiWord

Base module for WikiWord.

de.brightbyte.wikiword	Base classes for application level entry points. Some entity classes for commons entities like Dataset or Namespace.
de.brightbyte.wikiword.disambig	Automated disambiguation
de.brightbyte.wikiword.model	Data beans for WikiWord concepts
de.brightbyte.wikiword.rdf	RDF export and bindings
de.brightbyte.wikiword.schema	Database schema definitions
de.brightbyte.wikiword.store	Read-only DAO interfaces and implementations

WikiWordBuilder

Builder module for WikiWord, used to generate thesauri from Wikipedia dumps.

de.brightbyte.wikiword.analyzer	WikiText analysis code
de.brightbyte.wikiword.analyzer.extractor	Extractor framework, for determining high level properties of a wiki article.
de.brightbyte.wikiword.analyzer.mangler	Code for processing and converting wiki text
de.brightbyte.wikiword.analyzer.matcher	Code for finding and extracting features of wiki text.
de.brightbyte.wikiword.analyzer.sensor	Code for detecting features of wiki text.
de.brightbyte.wikiword.analyzer.template	Template parameter parsing framework
de.brightbyte.wikiword.builder	Application level entry points for building thesauri in the database.
de.brightbyte.wikiword.extract	Application level entry points for extracting data from dumps into files.
de.brightbyte.wikiword.output	Serial output abstraction, for use by applications in de.brightbyte.wikiword.extract
de.brightbyte.wikiword.processor	Processors for stepping through the pages of a dump and processing their contents.
de.brightbyte.wikiword.store.builder	Thesaurus stores (DAO layer)
de.brightbyte.wikiword.wikis	Wiki-specific knowledge, encoded in subclasses of `WikiConfiguration`.

WikiWordIntegrator

The Integrator module uses the WikiWord data to extend and match vocabularies for external sources. Integrator applications are based on a data stream of records, read from a data source, which are processed and then written to a store.

de.brightbyte.wikiword.integrator	Application level entry points to the Integrator module.

de.brightbyte.wikiword.integrator.data	Entity classes representing the various types of data used in the data streams.
de.brightbyte.wikiword.integrator.data.filter	Filter for filtering data streams
de.brightbyte.wikiword.integrator.processor	Code for processing data streams and handing records to the appropriate store
de.brightbyte.wikiword.integrator.store	DAO layer for storing properties, associations and mappings.

Classes

Applications	Top level entry points, may be called directly or from BeanShell.
`DatabaseInfo`, `TweakSet`, `FeatureSetSourceDescriptor`	Configuration records, see the section on configuration.
`WikiWordStore`, `WikiWordStoreBuilder`	DAO interfaces, generally wrapping a database
`FeatureSet`	Represents the features (properties) of an arbitrary entity, as a multi-map. This is the basic data unit of all processes in the Integrator module, it is used to represent concepts, foreign entities, attributes of mappings, etc.
`DataCursor`	A cursor represents a record stream. It is similar to a ResultSet or an Iterator. It is used to step through the input data on record at a time. Most of the logic for manipulating, converting, merging and filtering records is implemented in the various cursor classes.
`WikiWordProcessor`	Processors are the “pumps” that read records from a cursor and pass the data to a store. Since most of the logic is implemented in the cursors, usually trivial pass-through implementations are used for processors.
`Association`	An association object represents a link between a foreign entity and a WikiWord concept. It is made up of three FeatureSet objects: one representing the foreign entity (the subject), one representing the concept (object), and one holding the properties of the association itself (fir example, a weight).
`MappingCandidates`	Mapping candidates are all the WikiWord concepts that have been associated to a given subject entity. A MappingCandidates object contains one FeatureSet representing the subject, and a list of FeatureSets representing the different objects of the mapping. Disambiguation is the process of picking one of the object as the correct mapping.
`MappingCandidateFilter`, `MappingCandidateSelector`	A MappingCandidateFilter reduces the set of candidates in a MappingCandidates object according to some rule. A special kind of filter is based on a selector, which picks a single candidate as the "correct" mapping. This is also called disambiguation, because it removes ambiguity from the mapping.
`MappingCnadidateScorer`	A scorer assigns a score value to a given mapping candidate, in the context of the mapping's subject. That is, it measures how good the mapping is. Scores are typically used for filtering,

	either with a threshold or by simply selecting the candidate with the highest score.
`PropertyAccessor`, Aggregator	A property accessor retrieves some value from an object in some way. In the most common case, it simply takes the value of a specific property/feature from a FeatureSet. An Aggregator function is used to combine multiple values of a property into one. Typical aggregators are *sum*, *max*, *concat* and *first*.

Database

Access to the database is handled by DAO objects descendant from `WikiWordStore` or, for write access, `WikiWordStoreBuilder`. When addressing tables directly through the store object, the logical table name must be used. The logical name is then converted to the physical name by prepending the table prefix associated with the current data set.

Data Sets

A data set (internally represented by a `Dataset` object) represents one thesaurus (monolingual or multilingual), with all the associated data. Any data generated by the Integrator module will also become part of the data set is was derived from. The data set identifier is made up of two parts, separated by a colon: the first part is the *`collection`*, the second part is the *`language code`* (or *thesaurus* for a multilingual thesaurus). The collection indicates the scope of the thesaurus: *full* is generally used to indicate a collection of thesauri covering the entire Wikipedia. *health* might be a collection or thesauri covering health topics, etc.

The data set is specified as the first parameter of any WikiWord entry point, in the form *`collection`*:*`language`*. It is used to define the database table prefix used to generate table names for the data set, using the pattern *`collection_language_`*. This way, several data sets can be stored in the same database.

Tables

The database tables most relevant to WikiWord's integrator module (using logical table names):

concept	Contains the individual concepts represented by the thesaurus.
meaning	Associates terms with concepts, that is, phrases with their meanings. Weighting information is also contained in this table.
property	Associates values of arbitrary properties to concepts. For mapping, properties that contain some sort of authoritative external ID are useful.
definition	Associates definitions with concepts.

For the purposes of the Integrator module, it is assumed that these tables already exist. They are generated from Wikipedia dumps by the Build module.

The tables generated by the Integrator module are usually named after the following convention:

`authority`	The table named after an foreign authority contains the properties imported from that authority.
*`authority`*_assoc	Contains associations between foreign entities and WikiWord concepts. Any pair may have multiple associations resulting from different kinds of mappings.
*`authority`*_mapped	Contains mappings condensed from the association table. Each pair of foreign entity and WikiWord concept may occur only once, though a

	foreign entity may be associated with several WikiWord concepts, and vice versa.
*authority*_good	Contains a filtered version of the original mapping table, usually by some threshold.
*authority*_best	Contains a disambiguated version of the original mapping table, with only the "best" mapping remaining for each external entity.
*authority*_unique	Contains a reduced version of the original mapping table, with only the unambiguous mappings from *authority*_good remaining.
*authority*_exclusive	Contains a reduced version of the unique mapping table, with only those mappings from *authority*_unique that are unambiguous both ways. That is, any foreign entity and WikiWord concept may only occur once in this table.

The actual, physical table names will be derived from the logical names given above by prepending the data set's table prefix, *collection_language_*, as described in the previous sections.

Environment

WikiWord's Integrator module installs into a self-contained directory. All JAR files needed are located in the `lib` directory.

Prerequisites

In order to run, WikiWord's Integrator module requires the following environment:

Java 6	Sun's JRE version 6 must be installed. For memory intensive operations, it may be required to use the 64 bit version. This is however generally only the case for the Builder module, the Integrator does not normally use much heap space.
Linux	While WikiWord will run on any platform supporting Java, the launcher scripts are written for the bash shell and use some other utilities from the Un*x world.
MySQL 5	WikiWord needs a MySQL server to connect to.
A WikiWord thesaurus	It is assumed that a WikiWord thesaurus is already present in the database. This would be generated by WikiWord's Builder module. While the Integrator module contains all the classes needed to build a thesaurus, doing so is not subject of this manual.

Command Line

For invoking the different entry points of WikiWord's Integrator module, there are launcher scripts provided in the installation's root directory:

`integrator-launch.sh` This script lets you launch any of the Integrator module's entry points conveniently.

`integrator-launch.sh` {*classname*} {*collection*:*language*} [*args...*]

The *classname* can be provided without the package name `de.brightbyte.wikiword.integrator.`

collection:*language* specify the thesaurus to operate on (in effect, they defined the database table prefix to be used).

`integrator-shell.sh` This script lets you launch the Integrator module's scripting shell.

`integrator-shell.sh` {*collection*:*language*} [*script-file*] [*args...*]

collection:*language* specify the thesaurus to operate on (in effect, they defined the database table prefix to be used). The *script-file* is the BeanShell script to run. If omitted, the shell starts in interactive mode. Any further arguments may be handled by the script.

Configuration files

Several configuration files will be read from the config directory. The config directory is the the installation root, if you use the launcher scripts. Otherwise, it has to be set using the parameter `--config-dir`. Important configuration files are:

db.properties

This file contains the information used by WikiWord to connect to your database. Without this file, WikiWord will not function. `db.properties` contains the following information:

```
url=jdbc:mysql://the-server/the-database?characterEncoding=utf8
driver=com.mysql.jdbc.Driver
user=your-log-in
password=your-password
```

vm.options

Options for the Java virtual machine, used by the launcher scripts. This would typically contain something like *-Xms64m* to specify the RAM available to the JVM.

tweaks.properties

The tweaks file contains a variety of system specific adjustments for WikiWord's operation. It is however mostly relevant to WikiWord's Builder module, the Integrator module does not make use of most of the information.

See `tweaks.properties.sample` for an example and some information.

BeanShell Commands

Built-in BeanShell commands are defined by *.bsh* files in the package `de.brightbyte.wikiword.integrator`.

`runSql(script, source-table, target-file)` Runs the given SQL script. If the script is a plain name with not file extension, it is interpreted as the name of a built in script from the package `de.brightbyte.wikiwird.integrator`.. Otherwise, it is interpreted as the path or URL to an SQL file.

`scriptURL(name)` Utility function to generate a URL relative to the BeanShell script's location. Useful for referencing files that are stored along with the script.

`loadForeignProperties(target-table, source-descriptor)`	Calls the `LoadForeignProperties` application. The data specified by the *`source-descriptor`* will be loaded into the *`target-table`*. *`source-descriptor`* may be a blank BeanShell object, an instance of `FeatureSetSourceDEescriptor`, or the path of a property file.
`buildConceptAssociations(target-table, source-descriptor)`	Calls the `BuildConceptAssociations` application. The associations specified by the *`source-descriptor`* will be written into the *`target-table`*. *`source-descriptor`* may be a blank BeanShell object, an instance of `FeatureSetSourceDEescriptor`, or the path of a property file.
`buildConceptMappings(target-table, source-descriptor)`	Calls the `BuildConceptMappings` application. The mappings specified by the *`source-descriptor`* will be written into the *`target-table`*. *`source-descriptor`* may be a blank BeanShell object, an instance of `FeatureSetSourceDEescriptor`, or the path of a property file.
`filterConceptMappings(target-table, source-descriptor)`	Calls the `FilterConceptMappings` application. The mappings specified by the *`source-descriptor`* will be written into the *`target-table`*. *`source-descriptor`* may be a blank BeanShell object, an instance of `FeatureSetSourceDEescriptor`, or the path of a property file.

Parameters

Source Descriptors

A source descriptor describes where the input data should be obtained from, what columns exist, what special meaning they have and how they are going to be processed. Internally, a source descriptor is represented as an instance of `FeatureSetSourceDescriptor`. Externally, a source descriptor is usually defined by a java property file, in which the values follow a special syntax, similar to the JASON notation. Most importantly:

- There is no semicolon at the end of the line.
- String values *must* be enclosed in quotes.
- Literals for `true`, `false` and `null` are accepted.
- Numeric values are accepted.
- Lists can be defined in square brackets, maps in curly braces, with ":" for separating key and value.
- the keyword *new* followed by a fully qualified path name, optionally followed by a parameter list, instantiates a Java object.

In the BeanShell environment, a source descriptor may be defined as a blank object created by calling `object()`. The individual parameters can then be defined by setting members on that object (i.e. defining variables in the object's scope). Lists may be given as arrays or instances of `List`.

Note

In parameter names, "-" and "_" are interchangeable. By convention, "-" is used ion property files, but "_" must be used in the BeanShell environment.

Source Descriptor Parameters

association-annotation-field	The field/column that contains the annotation string. The annotation could be any additional info attached to a mapping. Used with `BuildConceptMappings`.
association-value-field	The field/column that contains the association value. That is the value that was used to derive the association. Used with `BuildConceptAssociations`.
association-weight-field	The field/column that contains the association weight. The weight may be used for filtering. Used with `BuildConceptAssociations` as well as `BuildConceptMappings` and `FilterConceptMappings`.
authority	The name of an external authority; the authority name serves as the namespace for foreign property names and foreign entity IDs. Instead of setting *authority* to a fixed value, it can also be taken from a data field/column specified by *foreign-authority-field*.
concept-fields	The list of fields names that will be taken to belong to the concept (mapping target resp. object) when building an `Association` from a single `FeatureSet` instance. Used with `BuildConceptAssociations`.
concept-id-field	Field that contains the WikiWord concept's ID. Usually *concept*.
concept-name-field	Field that contains the WikiWord concept's name. Usually *concept_name*.
concept-property-field	Field that contains the WikiWord property. Usually *property* or, in the case of terms, *term_text*.
concept-property-freq-field	Field that contains a frequency value of a WikiWord property. When matching terms from the `meaning` table, this is usually *freq*, otherwise it remains unused.
concept-property-source-field	Field that contains a source ID of a WikiWord property. When matching terms from the `meaning` table, this is usually *rule*, otherwise it remains unused.
csv-backslash-escape	Boolean indicating if backslash escapes are allowed when parsing CSV input. Ignored if *csv-chunker* is set.
csv-chunker	Instance of `Chunker` used to split the lines if the input file. This overrides *csv-separator*, *csv-backslash-escape*, etc.
csv-separator	Field separation character to used when parsing CSV input. Ignored if *csv-chunker* is set.
csv-skip-bad-rows	Boolean indicating if bad rows in the CSV input file should be skipped. If false, bad lines will cause the import to be aborted.
csv-skip-header	Boolean indicating if the first row of the CSV file is to be skipped. This defaults to `true` if *fields* is set, to `false` otherwise. If *fields* is set but the file contains a header row, *csv-skip-header* must be set to true explicitly.

defaults
: Default source descriptor to load. The name refers to one of the descriptors files located in the package `de.brightbyte.wikiword.integrator` with the file extension *.properties*, namely:

Default Source Descriptors

best-mappings
: Settings for filtering mappings to find the best candidate for each foreign entity.

build-mappings
: Settings for building mappings from associations.

list-mappings
: Settings for listing mappings.

match-properties
: Settings for matching concepts using properties.

match-terms
: Settings for matching concepts using terms.

unique-mappings
: Settings for selecting unique mappings.

encoding
: Encoding of the input file. Defaults to *UTF-8*.

field-chunkers
: A `Map` assigning instances of `Chunker` to individual data field, in order to split values. This is useful if lists of multiple values are encoded into individual field values.

fields
: The list of fields (columns) in the data source. If not given, it will be determined automatically. In case of CSV files, field names are taken from the first line in the file.

file
: The name of the input file. May be a full URL. If the file name ends in *.gz* or *.bz2*, the file is automatically decompressed on the fly. If the file extension (ignoring any *.gz* or *.bz2*) is *.sql*, the file's content is executed as an SQL script (the file format detection can be overridden using *file-format*). Otherwise, it is assumed to contain CSV data. The dialect (CSV or TSV) is also determined from the file extension. Alternatively to specifying *file*, you may give the input data source via *query*.

file-format
: Override the file format detection for *file*. If the format is *sql*, the input file will be executed as an SQL script. If it is *csv* resp. *tsv*, it is read as a CSV file with the appropriate separators (comma for csv and tab for tsv).

foreign-authority-field
: The field containing the name of the foreign authority which acts as a namespace for the foreign entity's ID and property names. Alternatively, specify *authority*.

foreign-fields
: The list of fields names that will be taken to belong to the foreign entity (mapping source resp. subject) when building an

	`Association` from a single `FeatureSet` instance. Used with `BuildConceptAssociations`.
`foreign-id-field`	Field containing an ID unique in the context of the foreign authority, used to identify foreign entities.
`foreign-name-field`	Field containing a display name for foreign entities. Defaults to the value of *`foreign-id-field`*.
`foreign-property-field`	Field containing the name of the property of a foreign entity. Used with `BuildConceptAssociations`.
`mapping-filter`	Instance of `MappingCandidateFilter` to use for filtering. Used with `FilterConceptMappings`. Overrides mapping-filter-aggregator, mapping-filter-field, mapping-filter-scorer, mapping-filter-selector, etc.
`mapping-filter-aggregator`	Instance of `Functor2` to use as an aggregator to aggregate multiple score values into one. Overridden by mapping-filter, mapping-selector, *`mapping-filter-scorer`* and *`mapping-filter-accessor`*.
`mapping-filter-aggregator-function`	Name of the aggregator function used to merge multiple score values into one; Must be either "max" or "sum", default is "sum". Used with `FilterConceptMappings`. Overridden by mapping-filter, mapping-selector, *`mapping-filter-scorer`*, *`mapping-filter-accessor`* and *`mapping-filter-aggregator`*.
`mapping-filter-field`	The field to take the score for a candidate `FeatureSet` from. Used with `FilterConceptMappings`. Used with `FilterConceptMappings`. Overridden by mapping-filter, mapping-selector, *`mapping-filter-scorer`* and *`mapping-filter-accessor`*.
`mapping-filter-field-accessor`	Instance of `PropertyAccessor` to extract the score from a candidate `FeatureSet`. Used with `FilterConceptMappings`. Overridden by mapping-filter, mapping-selector and *`mapping-filter-scorer`*.
`mapping-filter-scorer`	Instance of `MappingCandidateScorer` to use when filtering (for best candidate, or by threshold). Used with `FilterConceptMappings`. Overridden by mapping-filter and mapping-selector, overrides mapping-filter-aggregator, mapping-filter-field etc.
`mapping-filter-threshold`	Threshold value to use when filtering mapping candidates. If set, all candidates with a score equal to or better than the threshold will pass. Otherwise, the one candidate with the best score will pass. Used with `FilterConceptMappings`. Overridden by mapping-filter and mapping-selector.
`mapping-selector`	Instance of `MappingCandidateSelector` to use for filtering. Used with `FilterConceptMappings`. Overridden by mapping-filter, overrides mapping-filter-aggregator, mapping-filter-field, mapping-filter-scorer etc.
`property-fields`	The list of fields names that will be taken to belong to the association itself (mapping properties) when building an `Association` from a single `FeatureSet` instance. Used with `BuildConceptAssociations`.

property-subject-name-field	Field/column containing the name of the property subject, i.e. of the foreign entity. Used with `LoadForeignProperties`.
query	SQL query to generate the input data. If the query is a single identifier with no white-space, it is interpreted as the name of a built-in script. Built-in scripts are located in the package `de.brightbyte.wikiword.integrator` and have the file extension .sql. Alternatively, use *`file`* or *query-generator*.

Build-in data generator scripts

build-mappings	Builds mappings by grouping entries in an association table by foreign ID and concept ID.
list	Lists the contents of a table.
match-single-property	Match concepts by a single property.
match-terms	Match concepts using terms from WikiWord's meaning table.

query-generator	Instance of `SqlQueryGenerator` that will generate the SQL query to generate the input data. Alternative to *`query`*.
source-table	name of the source table. Used to build a generic query listing all data in the table if no *`query`* or *`file`* is given
sql-comment-subst	A `Map` of substitutions to perform on the SQL script before it is executed. The keys are the names of placeholders. Placeholders in the SQL code are formatted as comments, like */* foo */* and may specify defaults, like */* foo | bar */*. Overridden by *sql-manglers*.
sql-manglers	A list of `Mangler`s to apply to the SQL script before it is executed. Overrides *sql-comment-subst*.

Appendix

Links

WikiWord project page	http://brightbyte.de/page/WikiWord
WikiWord download page	http://brightbyte.de/download/WikiWord
WikiWord SVN repository	http://svn.wikimedia.org/svnroot/mediawiki/trunk/WikiWord/