Packages of the **Topic Tracking Framework (ttf)**

- **package ttf.analysis**
  - **class AnalysisController** - controls a **Processor** (`ttf.analysis.processor.Processor`) which will process all the **Articles** (`ttf.model.article.Article`) provided by an **Article Provider** (`ttf.analysis.input.ArticleProvider`)

- **package ttf.analysis.command**
  - **class EntityDetectionCommand** - implementation of **Command** (`org.apache.commons.chain.Command`) thus overriding the **execute(Context)** method. The **Context** (`org.apache.commons.chain.Context`) to be considered is the **ttf** implementation - the **Analysis Context** (`ttf.analysis.context.AnalysisContext`). By executing this **Command** upon the given **Context**, the **Entity Detector** (`ttf.util.alchemyapi.EntityDetector`) of the **Context** is used to extract the **Alchemy Entities** (`ttf.util.alchemyapi.AlchemyEntity`) using the **Article**'s **address** (`ttf.model.article.Article`). The information provided by the resulting **Alchemy Entities** is then added to the **Context** in the form of the **Article's Entity Group**. More explicitly, the text content of each discovered entity is put together with its relevance value, as a key:value pair, in the **Entity Group**. This is the first **Command** to be executed in the **ttf analysis process**.

  - **class ModelPersistenceCommand** - implementation of **Command** for **ttf** (see **class EntityDetectionCommand**). By executing this **Command** upon the given **Context,** the changes that have be brought to the **Context** during the **analysis process** are made persistent. The altered **Topic** and **Article** are made persistent by the **Model Store** provided by the Context. This is most likely to be the last **Command** to be executed in the **ttf analysis process**, the one that makes the changes of analysis permanent.

  - **class TfIdfDetectionCommand** - implementation of **Command** for **ttf** (see **class EntityDetectionCommand**). By executing this **Command** upon the given **Context**, the **Tf-Idf Detector** (`ttf.util.tfidfapi.TfIdfDetector`) of the **Context** is used to extract the **Tf-Idf Entity** (`ttf.util.tfidfapi.TfIdfEntity`) using the **Article**'s **address** (`ttf.model.article.Article`). The information provided by the resulting **Tf-Idf Entities** is then added to the **Context** in the form of the **Article's Term Group**. More explicitly, the text content of each discovered entity is put together with its frequency value, as a key:value pair, in the **Term Group**. This is most likely to be the the second **Command** to be executed in the **ttf analysis process**.

  - **class TopicLoadingCommand** - implementation of **Command** for **ttf** (see **class EntityDetectionCommand**). By executing this **Command** upon the given **Context**, the **Model Store** (`ttf.persistence.ModelStore`) if

solicited for the **Topics** (`ttf.model.topic.Topic`) which are present in the persistence layer of the framework, in order to add the retrieved **Topics** to that **Context**. This is most likely to be the the third **Command** to be executed in the **ttf analysis process**, preceding the **Command** for **Topic Selection**.

- **class TopicSelectionCommand** - implementation of **Command** for **ttf** (see **class EntityDetectionCommand**). By executing this **Command** upon the given **Context**, the **Similarity Computer** (`ttf.analysis.computation.SimilarityComputer`) is used to compute the similarity factors between the processed **Article** (`ttf.model.article.Article`) and all the previously considered **Topics** (`ttf.model.topic.Topic`). All the needed information (**Similarity Computer**, **Article**, **Topic**) is part of the **Context**. After computing all the values, the **Article** will be assigned to the **Topic** to which it is most similar, provided that their similarity factor is greater, or at least equal to the **minimum similarity** threshold. If none of the topics is sufficiently similar to the article, then a new topic will be created bearing the article's title. This is most likely to be the fourth **Command** to be executed in **the ttf analysis process**.

- **class TopicUpdateCommand** - implementation of **Command** for **ttf** (see **class EntityDetectionCommand**). By executing this **Command** upon the given **Context**, the **Topic** (`ttf.model.topic.Topic`) information is updated in order to reflect the addition of an **Article** (`ttf.model.article.Article`). Therefore, the **Term Group** and **Entity Group** of the **Article** are added to the **Topic** ones. This is most likely to be the the fifth **Command** to be executed in the **ttf analysis process**, after the selection of a suitable **Topic**.

- **package ttf.analysis.computation**
  - **class SimilarityComputer** - responsible for computing the similarity factor between an **Article** (`ttf.model.article.Article`) and a **Topic** (`ttf.model.topic.Topic`). More accurately, a relation can is established between a given **Article** and a given **Topic** based on the cosine similarity of their **Entitiy Groups**. (http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html#Cosim)

- **package ttf.analysis.context**
  - **class AnalysisContext** - implementation of the **Context Base** interface (`org.apache.commons.chain.impl.ContextBase`) which detains and permits access to all the data members that make a consistent article processing context:
    - **Article Factory** (`ttf.model.article.ArticleFactory`)
    - **Topic Factory** (`ttf.model.topic.TopicFactory`)
    - **Model Store** (`ttf.persistence.ModelStore`)
    - **Alchemy API** (`com.orchestr8.api.AlchemyAPI`)
    - **Entity Detector** (`ttf.util.alchemyapi.EntityDetector`)
    - **TfIdf Detector** (`ttf.util.tfidfapi.TfIdfDetector`)

        - **Similarity Computer** (`ttf.analysis.computation.SimilarityComputer`)
        - **Processed Article** (`ttf.model.article.Article`)
        - **Loaded topics** (`Set<ttf.model.topic.Topic>`)
        - **Selected Topic** (`ttf.model.topic.Topic`)

    - **class ContextFactory** - factory class resposible for building **Analysis Contexts** which considers fixed instances for some of the necessary data members

- **package ttf.analysis.input**
    - **interface ArticleProvider** - describes an **Article Provider** as being an **Article** (`ttf.model.article.Article`) source, offering one such **Article** by means of the method **poll()**

    - **class InternalProvider** - implementation of the **ArticleProvider** interface as a **LinkedList**

- **package ttf.analysis.processor**
    - **interface Processor** - describes a Processor as being able to process an article (`ttf.model.article.Article`)

    - **class ChainProcessor** - implementation of the **Processor** interface, in which processing a given article implies executing a chain of **Commands** (`org.apache.commons.chain.Command`) on a fixed **Context** (`org.apache.commons.chain.impl.Context`) obtained by means of a **Context Factory** (`ttf.analysis.context.ContextFactory`) to which the **Article** is added (`ttf.analysis.context.AnalysisContext`)

- **package ttf.analysis.tfidf**
    - **class TfIdf** - responsible for obtaining **Tokens** from a Web page specified by its URL. The extraction of visual textual content of a web page (what a browser would display) is done by employing a **String Extractor** which is parte of the `org.htmlparser.parserapplications.StringExtractor` package (http://htmlparser.sourceforge.net/). A Sentence Tokenizer and afterwards a Word Tokenizer are used for separating the text into words. Also a **Stop Filter** (`org.apache.lucene.analysis.StopFilter`) which is set to eliminate the Strings that are mention by a given **stop set of Strings**. Once a word is accepted, it is considered to be the appearance of a **Token** , thus either a new such **Token** is created, or an allready existing **Token** contracts one more appearence.

- **package ttf.analysis.tfidf.tokenizer**
    - **class ParagraphTokenizer** - uses the **Rule Base Break Iterator** (see **WordTokenizer**) to break a text into paragraphs. The **nextParagraph()** will provide the String considered to be the next paragraph in the text, if any.

- **class RegexBasedWordBreakIterator** - extends the **Break Iterator** by modifying a **Break Interator** for word boundaries analyzed by the default locale's language rules. Changes in behaviour consist of deciding the boundaries for XML input (by checking the "<" and ">" marks) and for composite words (by checking the hyphenation).

- **class SentenceTokenizer** - uses the **Rule Base Break Iterator** (see **WordTokenizer**) to break a text into sentences. The **nextSentence()** will provide the String considered to be the next sentence in the text, if any.

- **class WordTokenizer** - uses the `com.ibm.icu.text.RuleBasedBreakIterator` (http://icu-project.org/apiref/icu4j/) to break up a piece of text according to a set of rules, thus creating **Tokens** (`ttf.model.token.Token`). In detail, once its data members **text** (`String`) and **break iterator** (`RuleBasedBreakIterator`) have been assigned, the **next()** method produces the next **Token**, if any, taken from the text. The content (value) of the **Token** in filled in by the String produced by means of the **break iterator**, while the its **type** is set accordingly to the rule that decided the token limit (the break rule).

- **package ttf.model**
  - **abstract class Model** - base class meant to be inherited, which establishes the fact that a **Model** is identified by an **id** String. More precisely, two **Models** are considered equal if they share the same id. Because **Model** overrides the **equals()** method of Java Object, it also overrided the **hashCode()**, providing a new int value.

- **package ttf.model.article**
  - **class Article** - extends the **Model** class and presents an **Article** as a composition of the following data members to which it facilitates access by getter and setter methods
    - **address** (`String`)
    - **title** (`String`)
    - **author** (`String`)
    - **publishedAt** (`Date`)
    - **discoveredAt** (`Date`)
    - **content** (`String`)
    - **tags** (`Set<String>`)
    - **score** (`ttf.model.property.NumericalValue`)
    - **topic** (`ttf.model.topic.Topic`)
    - **term group** (`ttf.model.property.PropertyGroup <String, ttf.model.property.NumericalValue>`)
    - **entity group** (`ttf.model.property.PropertyGroup <String, ttf.model.property.NumericalValue>`)

  - **interface ArticleFactory** - states that factory classes for **Article** must provide two factory methods for obtaining an **Article**: **build()** and **build(String)** - with or without the ID parameter

  - **class BasicArticleFactory** - very simple implementation of a factory class for **ArticleFactory**

- **package ttf.model.property**
  - **class NumericalValue** - retains and provides access to a primitive **double** value
  - **interface PropertyGroup<K, V> -** describes a **Property Group** as a **Map** (`java.util.Map`) of **Keys** and **Values**
  - **class HashMapPropertyGroup<K, V>** - simple implementation of a **PropertyGroup** as a **HashMap**

- **package ttf.model.token**
  - **class Token** - the **Token** entity aggregates the followind data members, to which it provides access by getter and setter methods:
    - **value** (`String`)
    - **type** (`ttf.model.token.TokenType`)
    - **count** (`int`)

  - **enum TokenType** - list of all the recognized **Token Types**. These are: **ABBREVIATION, COMBINED, PHRASE, EMOTICON, INTERNET, WORD, STOP_WORD, CONTENT_WORD, NUMBER, WHITESPACE, PUNCTUATION, PLACE, ORGANIZATION, MARKUP, UNKNOWN**. The **UNKNOWN** value is the default one.

  - **interface TokenFactory** - states that factory classes for **Token** must provide two factory methods for obtaining a **Token**: **build()** and **build(String, TokenType)**

  - **class BasicTokenFactory** - very simple implementation of a factory class for **TokenFactory**

- **package ttf.model.topic**
  - **class Topic** - extends the **Model** class and presents the **Topic** entity as a composition of the following data members to which it facilitates access by getter and setter methods
    - **title** (`String`)
    - **articleGroup** (`ttf.model.property.PropertyGroup <String, ttf.model.property.NumericalValue>`)
    - **termGroup** (`ttf.model.property.PropertyGroup <String, ttf.model.property.NumericalValue>`)
    - **entityGroup** (`ttf.model.property.PropertyGroup <String, ttf.model.property.NumericalValue>`)

  - **interface TopicFactory** - states that factory classes for **Topic** must provide two factory methods for obtaining a **Topic**: **build()** and **build(String)**

  - **class BasicTopicFactory** - very simple implementation of a factory class for **Topic**

- **package ttf.persistence**
  - **interface ModelStore** - interface for the persistence layer of the **ttf**. The **Model Store** of the **ttf** must be able to fulfill the functions for **loading** the **Articles** (`ttf.model.article.Article`) and the **Topics** (`ttf.model.topic.Topic`) that are depicted by a given **Query** (`ttf.persistence.query.Query`). Also it must provide a way of making a given **Topic** or **Article** persistent.

  - **class PersistenceException** - **Exception** that is attributed to the persistence layer inside the framework

- **package ttf.persistence.query**
  - **interface Query** - empty interface

- **package ttf.persistence.sql**
  - **class ArticleSaver** - is an implementation of **Model Saver** (`ttf.persistence.sql.ModelSaver`) dedicated to making **Articles** (`ttf.model.article.Article`) persistent. It uses a `org.apache.commons.dbutils.GenKeyQueryRunner` for passing the results to a newly created **Id Result Handler** (`ttf.persistence.sql.IdResultHandler`). Once saved, the **Article's** identifier is set to the key generated when running the query.

  - **class FeatureSaver** - provides methods for composing and runnig Querries in order to update the persistant information regarding **Articles** (`ttf.model.article.Article`) and **Topics** (`ttf.model.topic.Topic`) detained by the framework. A **Querry Runner** (`org.apache.commons.dbutils.QueryRunner`) is used for running **batch queries**.

  - **IdResultHandler** - simple implementation of **Result Set Handler** (`org.apache.commons.dbutils.ResultSetHandler`) as required when working with `org.apache.commons.dbutils.QueryRunner`.

  - **abstract class ModelSaver<M extends Model>** - describes the persistency issue of saving a model. The classes extending **Model Saver** must provide an implementation to override the **save(M)** method. Also, these classes may access the **Data Source** (`javax.sql.DataSource`) and the **Feature Saver** (`ttf.persistence.sql.FeatureSaver`) in order to manage the persistent data.

  - **class SQLStore** - implementation of **Model Store** which deals with data persistence in the framework, saving and loading **Articles** and **Topics**, by employing its data members
    - **Data Source** (`javax.sql.DataSource`)
    - **Topic Factory** (`ttf.model.topic.TopicFactory`)
    - **Article Saver** (`ttf.persistence.sql.ArticleSaver`)
    - **Topic Saver** (`ttf.persistence.sql.TopicSaver`)

- **class TopicListRSH** - implementation of the **Result Set Handle** (`org.apache.commons.dbutils.ResultSetHandler`) dedicated to loading **Topic** (`ttf.model.topic.Topic`) information which it returns as a **collection of Topics.** This implementation is used by the **SQL Store** when retrieving **Topic** data.

- **class TopicSaver** - is an implementation of **Model Saver** (`ttf.persistence.sql.ModelSaver`) dedicated to making **Topics** (`ttf.model.topic.Topic`) persistent. (see **class ArticleSaver** which is very similar).

- **package ttf.util**
  - **class AppContext** - responsible for creating the application Context and therefore meant to be used in the initializing stage. The set up includes creating most of the instances with key roles inside the framework:
    - **Data Source** (`javax.sql.DataSource`)
    - **Article Factory** (`ttf.model.article.ArticleFactory`)
    - **Topic Factory** (ttf.model.topic.TopicFactory)
    - **Model Store** (`ttf.persistence.ModelStore`)
    - **Context Factory** (`ttf.analysis.context.ContextFactory`)
    - **Alchemy API** (`com.orchestr8.api.AlchemyAPI`)
    - **Entity Detector** (`ttf.util.alchemyapi.EntityDetector`)
    - **Tf-Idf-Detector** (`ttf.util.tfidfapi.TfIdfDetector`)
    - **Similarity Computer** (`ttf.analysis.computation.SimilarityComputer`)

  - **class XPathUtil** - contains static methods for evaluating **Objects** using **XPathExpressions** so that a String, int or double value may be easily obtained when parsing **DOM Documents**. The XPathUtil methods are used by the **Entity Detector** (`ttf.util.alchemyapi.EntityDetector`)

- **package ttf.util.alchemyapi**
  - **class EntityDetector** - responsible for obtaining **Alchemy Entities** out of unstructured or HTML texts and URLs. It uses the **Alchemy API** (`com.orchestr8.api.AlchemyAPI`) to fetch a **DOM Document** (`org.w3c.dom.Document`) that contains the Alchemy results. Subsequently, it parses the **Document** in order to obtain **Collections** of **Alchemy Entities** (`Collection<ttf.util.alchemyapi.AlchemyEntity>`).

  - **class AlchemyEntity** - composed with regard to the service provided by **Alchemy API** and the **Entity Detector** which handles this API. It aggregates and provides access to the following data members:
    - **text** (`String`)
    - **type** (`String`)
    - **relevance** (`double`)
    - **count** (`int`)

- **package ttf.util.tfidfapi**

- **class TfIdfEntity** - entity used for managing **Tokens** and their **Term Frequency (tf)** and **Inverse Document Frequency (idf)** (values used in the analysis process). Therefore, the **Tf-Idf-Entity** is composed and provides access to the following data members:
  - **token** (`ttf.model.token.Token`)
  - **tf** (`double`)
  - **idf** (`double`)

- **class TfIdfDetector** - responsible for computing a collection of **Tf-Idf-Entities** by using the **Tf-Idf** (`ttf.analysis.tfidf.TfIdf`) in order to obtain a collection of **Tokens** (`ttf.model.token.Token`). As the **Tf-Idf-Detector** is only used to process one article at a time, the **Term frequency (Tf)** is computed with regard to the nuber of **Tokens** found in the given **Article**, while the **Inverse Document Frequency (Idf)** is set to 1.

- **package ttf.incoming**
  - **class BasicTransformer** - simple implementation of the **Transformer** interface (`ttf.incoming.Transformer`) where all the **Incoming Article**'s fields are one by one read and passed to a new **Article** (`ttf.model.article.Article`) created by means of an **ArticleFactory** (`ttf.model.article.ArticleFactory`)

  - **class FeedEntryParser** - transforms **Synd Entry** (`com.sun.syndication.feed.synd.SyndEntry`) input into **Incoming Article** (`ttf.incoming.IncomingEntry`) output.

  - **class FeedInfo** - responsible for retaining and managing the **Feed Info**. The following data members describe the Feed entities:
    - **address** (`String`) of the Feed
    - **interval** (`int`) for checking the Feed; this value is modified by the **update(SyndFeed)** which takes into consideration the average interval of the past Feed entries.
    - **lastCheck** (`java.sql.Timestamp`)

    Also the **Feed Info** objects are **comparable**, the order established between them is the one in which the Feeds must be checked (the first Feed to be checked is the lesser one).

  - **class FeedReader** - is doing the Feed Reading**.** Information about the **Feed sources** is retrieved from the persistence layer, transformed into **Feed Info** (`ttf.incoming.FeedInfo`) and deposited in a Priority Queue. Using this Queue, the Feeds are one by one checked. The Feed Entries obtained are parsed into **Incoming Articles** (`ttf.incoming.IncomingArticle`) by means of a **Feed Entry Parser** (`ttf.incoming.FeedEntryParser`). These data is added to the **Data Source** by using a **Querry Runner** (`org.apache.commons.dbutils.QueryRunner`).

  - **class IncomingArticle** - describes the **Incoming Article** as a composition of the following data members to which it provides access through getter and setter methods (see **class Article** for correspondence):
    - **address** (`String`)
    - **title** (`String`)

        - **author** (`String`)
        - **publishedAt** (`Date`)
        - **discoveredAt** (`Date`)
        - **content** (`String`)
        - **tags** (`Set<String>`)
        - **score** (`ttf.model.property.NumericalValue`)
        - **topic** (`ttf.model.topic.Topic`)

    - **class IncomingArticleListRSH** - implements the **Result Set Handler** interface (`org.apache.commons.dbutils.ResultSetHandler`) by translating a **Result Set** (`java.sql.ResultSet`) such as obtained by querying a DB**,** into a list of **Incoming Articles** (`ttf.incoming.IncomingArticle`).

    - **interface Transformer** - introduces the way of producing **Articles** (`ttf.model.article.Article`) such as the ones to be processed by the **ttf**, out of **Incoming Articles** (`ttf.incoming.IncomingArticle`) which are the result of some **Feed** application that can provide input for the **ttf**. All the classes that respect the **Transformers** inteface must implement the **Article transform(IncomingArticle)** method.