# Extended Gazetteer in *SProUT*

Jakub Piskorski
German Research Center for Artificial Intelligence
Stuhsatzenhausweg 3, 66123 Saarbrücken, Germany
piskorsk@dfki.de

**VERSION JANUARY 2005**
**(SPROUT VERSION 4.0 AND HIGHER)**

## 1 Introduction

Gazetteer lookup is usually seen as an independent process of linguistic analysis, in which the input stream of characters or tokens is matched against a full name list (e.g., organizations, locations, person names) and keyword list (e.g., company designators). Typically, a gazetteer is used as a subcomponent in the module for named-entity recognition etc. In this paper, we briefly describe the extended gazetteer component, which can be either used in *SProUT* as a linguistic processing resource or independently as a subcomponent in other applications.

Contrary to the first version of the gazetteer module used in *SProUT* that assigns solely a type to each identified occurrence of a gazetteer entry in the analyzed text, the extended gazetteer allows for associating each entry with a list of arbitrary flat attribute-value pairs.[1] Further, ambiguous entries are allowed too. For instance, the entry *albania's* could be associated with the following attribute-value list: <[type: country], [continent: Europe], [case: genitive]>. After identifying an occurrence of the string *alabania's*, the gazetteer would return a feature structure which looks as follows:

$$\begin{bmatrix} start : 20 \\ end : 28 \\ type : country \\ continent : Europe \\ case : genitive \end{bmatrix}$$

[1] In comparison to the simple gazetteer, we optimized the underlying finite-state model with the respect to space complexity and also applied path compression to guarantee further space reduction.

Note, that the start/end fields hold values to the start and end positions of the matched text fragment. The values may either refer to character positions in the text or token IDs (i.e., first and last token of the matched text fragment). The user may switch between these two options. This can be done at a very low-level or by choosing an appropriate option in *SProUT* IDE. A further option allows to switch between case-sensitive and case-insensitive processing. **IMPORTANT: Working in the case-insensitive mode means that the input text is temporarily downcased; on the other side the gazetteer entries are not influenced in any manner.** It is important to notice, that the gazetteer entries are not downcased at any time. Finally, there are two modes in which the gazetteer can be applied to an input text:

MODE A: In this mode the gazetteer does not perform any tokenization and does not rely on any tokenization information for the input text. Instead, the user may enforce that each identified occurrence of a gazetteer entry in the text must be preceded and succeeded by whitespaces (this clearly depends on the application). The whitespaces are directly defined by the user. Obviously, the start/end fields in the output structures contain start/end character positions since no tokenization is performed.

This mode is realized by a call to the method:

```
de.dfki.lt.sprout.runtime.gazetteer.findMatchInText(char[])
```

MODE B: In this mode the gazetteer relies on the tokenization information, computed by *SProUT's* tokenizer. The gazetteer processes in this mode a stream of tokens. An option for defining a symbol which is used to 'separate' the tokens in the input stream is available (**Note** that the tokenizer does not represent whitespaces as tokens!). For instance, Indoeuropean languages should use a simple space, whereas for asian languages an empty string seems to be convenient.

This mode is realized by a call to the method:

```
de.dfki.lt.sprout.runtime.gazetteer.findMatchInText(TokenIterator,char[],char)
```

The *SProUT* system applies the gazetteer in mode B. We will now turn to the matters concerning preparing and compiling gazetteer resources and running the gazetteer.

## 2 Preparing and Compiling Gazetteer Resources

The resources needed for compiling a gazetteer consist of two files, an attribute file and an entry file. The attribute file is just a list of all attributes as shown in the example beneath:

```
type
continent
case
```

Each line of the entry file includes an entry and a list of associated attribute-value pairs, separated by a symbol which does not occur elsewhere in this file. The example below illustrates an entry file.

```
aequatorial guinea | type:country | continent:africa
aequatorial guineas | type:country | continent:africa | case:genitive
afghanistan | type:country | continent:asia
afghanistan's | type:country | continent:asia | case:genitive
afghanistans | type:country | continent:asia | case:genitive
```

Entry files may include comments and empty lines. A comment line is preceded with a # symbol. When entry file and attribute file are provided, we can turn to the compilation process. For running the gazetteer compiler, a configuration file is needed, which defines the following properties:

- a path to the attribute file
- a path to the entry file
- a name of the character set used for decoding input files (e.g., US-ASCII, ISO-8859-1, UTF-8, UTF-16BE). Please refer for further details to `java.nio.charset.Charset` in the JAVA API documentation available at `http://www.java.sun.com`
- a special symbol for separating attribute-value pairs
- a list of whitespaces (unicode encoding), only relevant when applying the gazetteer in mode A
- a path to the output file, which constitutes the only resource needed for applying the gazetteer

The example given below illustrates how a configuration file should look like.

```
InputFeaturesFile=sample.features
InputDataFile= sample.txt
CharacterSet=ISO-8859-1
InputSeparator=|
WhiteSpaces=u0020,u0009,u000a,u000d,u000c
OutputFile=sampleGazetteer.sgz
```

When the definition of the configuration file is completed we can finally compile the resources by running the following script[2]:

[2] Please note that currently compiling gazetteer resources is only possible under Linux or MS Windows since the dynamic library providing the functionality of the regular compiler that is used in the compilation process, is not available for other operating systems.

MS Windows version:

```
java -Xmx640M -cp %1 de.dfki.lt.sprout.runtime.gazetteer.CompilerForExtendedGazetteer %2
```

Linux version:

```
export LD_LIBRARY_PATH=.  // specifies the path where dynamic link libraries are stored
java -Xmx640M -cp $1 de.dfki.lt.sprout.runtime.gazetteer.CompilerForExtendedGazetteer $2
```

where %1 ($1) specifies the path to: (a) the directory containing *SProUT* Java archive (`sprout.jar` or `sprout.zip`) or/and a path to the root directory containing the compiled `CompilerForExtendedGazetteer` class, and (b) the log4j-1.2.8.jar archive[3] (used for logging since version 3.7.1). The second parameter %2 ($2) is a path to the configuration file described earlier. Any messages concerning errors encountered during the compilation are displayed via standard output or according to logging configuration. Since the regular compiler is invoked in the compilation process, make sure that the corresponding dll-library (`RC_JNI.dll` and `libRC_JNI.so` libraries for MS Windows and Linux respectively) is visible by your operating system. In the MS Windows environment it is sufficient to store this library in the working directory or to make the library visible via adding the corresponding absolute path of the library to the environment variable `PATH`. In case of Linux, an additional initial line containing the export directive has to be added to the script which defines the environment variable LD_LIBRARY_PATH ant sets its value to the path where the library is stored. In our script example above, this variable is set to the current directory. Do not forget to modify it if necessary.

Further, the directory in which you compile your resources must be writable, since the compiler produces some intermediate temporary files. It is recommended that users do not use any of the internal *SProUT* directories for compiling their resources.

## 3 Running the Gazetteer

When using the gazetteer within the SProUT grammar development environment, you have to choose the option *Menu -> Tools -> Configure components* which leads you to the configuration window in which you can launch an instance of an extended gazetteer in the similar way to installing other processing resources. Further, you can switch the case-sensitive option on and off. When referring to the matches recognized by the gazetteer in your grammars, please use the type *gazetteer*, as demonstrated in the following rule example:

[3] For internal stuff: The latest versions of all jar archives and external libraries which are used by *SProUT* can be found in DFKIs internal directory /project/cl/sprout/test/resource/lib

```
City:> gazetteer & [surface #N, type city, continent #C] -> [name #N, continent #C]
```

If you wish to see the gazetteer in full effect or test it outside of *SProUT* IDE on any textual data, the following script can be used:

```
java -Xmx640M -cp <LibPath> de.dfki.lt.sprout.runtime.gazetteer.ApplyExtendedGazetteer
     [-s <TokenizerConfig>] -[c][w][t] <GazetterFile> <InputFile> <OutputFile>

        <LibPath> specifies: (a) the path to the directory containing the SProUT Java archive
                  or/and a root directory containing the compiled ApplyExtendedGazetteer
                  class, and (b) the path to the log4j-1.2.8.jar archive

        <GazetteerFile> a path to the file containing the gazetteer

        <InputFile> a path to the file containing the input text

        <OutputFile> a path to the output file

        -s <TokenizerConfig> enforces tokenization, where the argument is a path to the
        tokenizer configuration file (see Tokenization documentation for details)

        -c tells the gazetteer to switch to the case-sensitive mode

        -w tells the gazetteer to enforce preceding and succeeding whitespaces when running
        the gazetteer in mode A

        -t tells the gazetteer to output token positions instead of character positions in the
        output structures
```

Example: In order to apply the gazetteer in the case-sensitive mode to the file **`sample.txt`** in the mode B (with tokenization), whereas the gazetteer is stored in the file **`gazetteer.sgz`**, the tokenizer configuration is saved in the file **`TokConf.txt`**, and the output (with token IDs in the start/end fields) should be written to the file **`result.txt`**, following script must be used:

```
java -Xmx640M -cp sprout.jar;log4j-1.2.8
de.dfki.lt.sprout.runtime.gazetteer.ApplyExtendedGazetteer -s TokConf.txt -t -c gazetteer.sgz
sample.txt result.txt
```

Developers who would like to perform the same action in their own applications would need to generate the following piece of code (in a simplified form):

```
import java.io.*;
import java.util.*;
import de.dfki.lt.sprout.runtime.tokenizer.*;
import de.dfki.lt.sprout.runtime.gazetteer.*;

// Initializing the gazetteer

// Create an instance of type ExtendedGazetteer
ExtendedGazetteer E = new ExtendedGazetteer();

// Set the basic logger
E.setBasicLogger();

// Initialize the gazetteer with the resources from file
E.readFromFile(GazetteerFile);
```

```
// Token positions should be returned in the output structures
E.ReturnsTokenPositions();

// Switch to the case-sensitive mode
E.CaseSensitive

// Prepare the input text

String INPUT = .....

// Run the tokenizer

TokenIterator pTI = null;
pTI = RunTokenizer.RunTok("sample.txt","TokConf.txt");
pTI.reset();

// Apply the gazetteer

// Define the placeholders for the output
ExtendedGazetteerInfo[] Res = null;
ArrayList ResList = null;

// Run the gazetteer
ResList = E.findMatchInText(pTI,INPUT.toCharArray(),' ');

// Write the results to the output file

try { BufferedWriter out = new BufferedWriter
            (new OutputStreamWriter(new FileOutputStream(OutputFile),E.getCharset()));
      if(ResList!=null)
        { for(int i=0;i<ResList.size();i++)
          { Res = (ExtendedGazetteerInfo[])ResList.get(i);
            for(int j=0;j<Res.length;j++)
             { if(E.ReturnsTokenPositions())
                  { int start = pTI.getToken(Res[j].getStartPos()).getStartPos();
                    int end = pTI.getToken(Res[j].getEndPos()).getEndPos();
                    out.write("Text: " + INPUT.substring(start,end+1));
                    out.newLine();
                    Res[j].print(out);
                    out.newLine();
                  }
             }
          }
        }
       out.close();
     }
    catch(IOException e) { System.out.println("IO Exception"); System.exit(0); }
```

For further details, please refer to the java code documentation of the corresponding *SProUT* package.

## APPENDIX A - JAVA DOCUMENTATION

## JAVA DOCUMENTATION FOR THE CLASS ExtendedGazetteer

### de.dfki.lt.sprout.runtime.gazetteer
# Class ExtendedGazetteer

java.lang.Object
└ **de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteer**

public class **ExtendedGazetteer**
extends java.lang.Object
This class implements extended gazetteer which allows for storing a list of character sequences (also multi words) - gazetteer entries, where each such gazetteer entry can be associated with a list of attribute/value pairs. Further, a gazetteer can be applied to a given input text in order to identify all occurenceis of gazetteer entries in this text. Methods for compiling, configuring and applying a gazetteer are provided.

IMPORTANT: There are two modes in which the gazetteer can be applied to an input text

MODE A: In this mode the gazetteer does not perform any tokenization and does not rely on any tokenization information of the input text. Instead the user may enforce that each identified occurence of a gazetteer entry in the text must be preceeded and succeeded by a whitespace. The whitespaces are defined explicitely by the user. Further option allows for switching between case-sensitive and case-insensitive modus (i.e., the input text is downcased, but not the gazetteer entries). For each identified occurence of a gazetteer entry a list of interpretations is returned (note that gazetteer may include more entries associated with the same word (or multi word). An interpretation consists of start/end (character) position in the text, and a list of attribute/value pairs.

MODE B: In this mode the gazetteer relies on the tokenization information, which is represented as tokenizer Iterator returned by SProUTs tokenizer (see tokenizers documentation). Gazetteer processes in this mode a stream of tokens. An option for defining a symbol which is used to 'separate' the tokens is available (e.g., for indoeuropean languages a simple space should be used, whereas for asian languages an empty string should be used). Analogously to mode A, an option that allows for switching between case-sensitive and case-insensitive modus is provided (i.e., the inut text is downcased, but not the gazetteer entries). For each identified occurence of a gazetteer entry a list of interpretations is returned (note that gazetteer may include more entries associated with the same word (or multi word). An interpretation consists of start/end position in the text, and a list of

attribute/value pairs. There is an option to allow for returning token start/end positions instead of expilict character positions.

## Constructor Summary

| | |
|---|---|
| **ExtendedGazetteer**() | Simple constructor |

## Method Summary

| | |
|---|---|
| void | **CaseSensitive**() Switches to the case-sensitive mode. |
| java.util.ArrayList | **CheckConsistency**(ShUG grammar) This function performs a consistency check in order to test if all feature structure types and attributes used by the Extended Gazetteer are present in the given type hierarchy. |
| boolean | **configureLogger**(java.lang.String LoggerPropertyFile) Configure the logger according to the Logger property file |
| void | **DoNotuseTrimmingWhileCompiling**() Deactivates the 'trimming' mode during compilation process on, i.e., abcdef \| TYPE:X ----> 'abcdef' is an entry (no trailing space!!!) |
| void | **enforceWhitespaces**() Enforces the Gazetteer to check if the candidate text fragment is preceeded and succeeded by a whitespace in order to qualify it as a match (default). |
| ExtendedGazetteerInfo[] | **findMatch**(char[] S, int startAt) Finds for a given string and given starting index, a longest match, and returns an array of objects of type ExtendedGazetteerInfo. |
| ExtendedGazetteerInfo[] | **findMatch**(char[] S, int startAt, int Len) Finds for a given string, given starting index, and a given length an exact match (if such exists), and returns a list of objects of type ExtendedGazetteerInfo. |
| java.util.Vector | **findMatchInText**(char[] S) Applies the gazetteer to the whole input text, and returns a vector of arrays, each holding objects of type ExtendedGazetteerInfo, where each such array represents a match in the input text (@see de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteerInfo). |
| java.util.Vector | **findMatchInText**(TokenIterator TI, char[] S, char SpecialSeparator) Applies the gazetteer in MODA B to the whole input text, and returns a vector of arrays, each of type ExtendedGazetteerInfo, where each such array represents a match in the input text (@see de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteerInfo). |

| | |
|---|---|
| java.lang.String | **getAttributeName**(int AttrIndex)<br>Returns the name of the attribute with a given ID. |
| java.lang.String | **getCharset**()<br>Returns the Character set used by the gazetteer. |
| int | **getNumAttributes**()<br>Returns the number of attributes in the current gazetteer (number of possible indices). |
| int | **getNumValues**(int AttrIndex)<br>Returns the number of possible values for a given attribute (number of possible inidices). |
| java.lang.String | **getValueName**(int AttrIndex, int ValIndex)<br>Returns the value for a given attribute ID and value ID. |
| void | **ignoreWhitespaces**()<br>Executing this method causes that the preceeding and succeeding whitespaces are not required in order to qualify a text fragment as a match (use this option for Chinese and Japanese gazetteeers). |
| boolean | **initializeFromData**(java.lang.String InputFeaturesFile, java.lang.String InputDataFile, java.lang.String InputCharSet, char InputSeparator, java.lang.String WhiteSpaces)<br>This method creates a new Gazetteer from input data. |
| boolean | **initializeFromDataAndSave**(java.lang.String InputFeaturesFile, java.lang.String InputDataFile, java.lang.String InputCharSet, char InputSeparator, java.lang.String WhiteSpaces, java.lang.String GazFile)<br>This method creates a new Gazetteer from input data and saves it to a given file. |
| boolean | **isActiveTrimmingWhileCompiling**()<br>Checks if the 'trimming' mode for compilation is active |
| boolean | **IsCaseSensitive**()<br>For checking if the case-sensitive mode is on |
| void | **NotCaseSensitive**()<br>Switches to the case-insensitive mode. |
| boolean | **readFromFile**(java.lang.String filename)<br>Reads an ExtendedGazetteer object from a binary file. |
| void | **readFromStream**(java.nio.ByteBuffer b)<br>Reads an ExtendedGazetteer object from a Bytebuffer. |
| void | **ReturnCharacterPositions**()<br>Switches to the mode in which character positions are returned in the output structures. |
| boolean | **ReturnsTokenPositions**()<br>For checking the type of returned start/end position information. |
| void | **ReturnTokenPositions**()<br>Switches to the mode in which token positions are returned in the output structures. |

| | |
|---|---|
| boolean | **saveToFile**(java.lang.String filename)<br>Saves an ExtendedGazetteer object in binary file. |
| void | **setBasicLogger**()<br>Activates the default basic logger (standard output) |
| void | **testPrint**() |
| void | **useTrimmingWhileCompiling**()<br>Activates the 'trimming' mode during compilation process on, i.e., abcdef \| TYPE:X ----> 'abcdef ' is an entry |
| void | **writeToStream**(java.io.DataOutputStream s)<br>Writes an ExtendedGazetteer object in to a DataOutputStream |

| Methods inherited from class java.lang.Object |
|---|
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

## ExtendedGazetteer

public **ExtendedGazetteer**()
Simple constructor

# Method Detail

## initializeFromDataAndSave

public boolean **initializeFromDataAndSave**(java.lang.String InputFeaturesFile,
java.lang.String InputDataFile,
java.lang.String InputCharSet,
char InputSeparator,
java.lang.String WhiteSpaces,
java.lang.String GazFile)

This method creates a new Gazetteer from input data and saves it to a given file. This involves several compilation steps. Note, that several temporary files are created and deleted during the compilation process.

**Parameters:**

`InputFeaturesFile` - path to the input file containing a list of attribute names, where name is stored in a separate line

`InputDataFile` - path to the input file containing in each line an entry together with the associated attribute/value pairs in the following format: entry separator attribute-1:value-1 separator attribute-2:value-2 .... separatpor attribute-k:value-k

`InputCharSet` - string representation of the Character Set that will be used

`InputSeparator` - separator used in the input files and automaton

`WhiteSpaces` - list of whitespaces (only relevant for [MODE A])

`GazFile` - path to a file where the created gazetteer will be written to

**Returns:**

`true` if the operation was successful, or `false` otherwise.

---

## useTrimmingWhileCompiling

public void **useTrimmingWhileCompiling**()

Activates the 'trimming' mode during compilation process on, i.e., abcdef | TYPE:X ---> 'abcdef ' is an entry

---

## configureLogger

public boolean **configureLogger**(java.lang.String LoggerPropertyFile)

Configure the logger according to the Logger property file

**Parameters:**

`LoggerPropertyFile` - the property file used for configuring the logger

**Returns:**

`true` if the configuration was successful, or `false` otherwise.

---

## setBasicLogger

public void **setBasicLogger**()

Activates the default basic logger (standard output)

---

## DoNotuseTrimmingWhileCompiling

public void **DoNotuseTrimmingWhileCompiling**()

Deactivates the 'trimming' mode during compilation process on, i.e., abcdef | TYPE:X ----> 'abcdef' is an entry (no trailing space!!!)

---

## isActiveTrimmingWhileCompiling

public boolean **isActiveTrimmingWhileCompiling**()

Checks if the 'trimming' mode for compilation is active

---

## getCharset

public java.lang.String **getCharset**()

Returns the Character set used by the gazetteer.

**Returns:**

returns the Character set used by the gazetteer.

## ReturnTokenPositions

public void **ReturnTokenPositions**()

Switches to the mode in which token positions are returned in the output structures. Note that using this method makes only sense in the MODE B !!!

## ReturnCharacterPositions

public void **ReturnCharacterPositions**()

Switches to the mode in which character positions are returned in the output structures. Note that using this method makes only sense in the MODE B !!!

## ReturnsTokenPositions

public boolean **ReturnsTokenPositions**()

For checking the type of returned start/end position information. Note that using this method makes only sense in the MODE B !!!

**Returns:**

`true` if the gazetteer returns in the current mode token positions, or `false` otherwise.

## NotCaseSensitive

public void **NotCaseSensitive**()

Switches to the case-insensitive mode. Note that this method is applicable either in MODE A or MODE B.

## CaseSensitive

public void **CaseSensitive**()

Switches to the case-sensitive mode. Note that this method is applicable either in MODE A or MODE B.

## IsCaseSensitive

public boolean **IsCaseSensitive**()

For checking if the case-sensitive mode is on

**Returns:**

`true` if the gazetteer is case-sensitive or `false` otherwise.

## initializeFromData

public boolean **initializeFromData**(java.lang.String InputFeaturesFile,
java.lang.String InputDataFile,
java.lang.String InputCharSet,
char InputSeparator,
java.lang.String WhiteSpaces)

This method creates a new Gazetteer from input data. This involves several compilation steps. Note, that several temporary files are created and deleted during compilation.

**Parameters:**

`InputFeaturesFile` - path to the input file containing a list of attribute names, where name is stored in a separate line

`InputDataFile` - path to the input file containing in each line an entry together with the associated attribute/value pairs in the following format: entry separator attribute-1:value-1 separator attribute-2:value-2 .... separatpor attribute-k:value-k

`InputCharSet` - string representation of the Character Set that will be used

`InputSeparator` - separator used in the input files and automaton

`WhiteSpaces` - list of whitespaces (only relevant for [MODE A])

**Returns:**

`true` if the operation was successful. or `false` otherwise.

---

## writeToStream

public void **writeToStream**(java.io.DataOutputStream s)
throws java.io.IOException

Writes an ExtendedGazetteer object in to a DataOutputStream

**Parameters:**

`s` - denotes the DataOutputStream to be used

**Throws:**

`java.io.IOException`

---

## saveToFile

public boolean **saveToFile**(java.lang.String filename)

Saves an ExtendedGazetteer object in binary file. This method is not based on standard JAVA serialization, but uses specific file format.

**Parameters:**

`filename` - denotes the file to be used

**Returns:**

`true` if this operation was succesful, or `false` otherwise

---

## readFromStream

public void **readFromStream**(java.nio.ByteBuffer b)

Reads an ExtendedGazetteer object from a Bytebuffer.

**Parameters:**

`b` - denotes the ByteBuffer to read from

---

## readFromFile

public boolean **readFromFile**(java.lang.String filename)

Reads an ExtendedGazetteer object from a binary file. This method is not based on standard JAVA serialization, but uses specific file format.

**Parameters:**

`filename` - denotes the file to be used

**Returns:**

true if this operation was succesful, or `null` otherwise

---

## enforceWhitespaces

public void **enforceWhitespaces**()

Enforces the Gazetteer to check if the candidate text fragment is preceeded and succeeded by a whitespace in order to qualify it as a match (default). Note, that using this method makes only sense in MODE A.

---

## ignoreWhitespaces

public void **ignoreWhitespaces**()

Executing this method causes that the preceeding and succeeding whitespaces are not required in order to qualify a text fragment as a match (use this option for Chinese and Japanese gazetteeers). Note, that using this method makes only sense in MODE A.

---

## findMatchInText

public java.util.Vector **findMatchInText**(TokenIterator TI,
char[] S,
char SpecialSeparator)

Applies the gazetteer in MODA B to the whole input text, and returns a vector of arrays, each of type ExtendedGazetteerInfo, where each such array represents a match in the input text (@see de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteerInfo). By default start/end token positions are returned, and the gazetteer works in the case-sensitive modus. In order to return start/.end character positions and to work in the case-sensitive modus a call to this method must be preceeded by a call to the method

'ExtendedGazetteer.CaseSensitive()' and 'ExtendedGazetteer.ReturnsCharacterPositions()' respectively.

**Parameters:**

`TI` - Token iterator returned by SProUTs tokenizer

`S` - input text

`SpecialSeparator` - symbol used for 'separating' tokens (e.g., for indoeuropean languages a simple space should be used, whereas for asian languages an empty string should be used)

**Returns:**

a vector of arrays of objects of type ExtendedGazetteerInfo, or `null` if no matches were found.

---

## findMatchInText

public java.util.Vector **findMatchInText**(char[] S)

Applies the gazetteer to the whole input text, and returns a vector of arrays, each holding objects of type ExtendedGazetteerInfo, where each such array represents a match in the input text (@see de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteerInfo). Note, that in order to accept an occurence of a certain entry from the gazetteer in the input text, it must be preceeded and succeeded by a whitespace character (except the first\last match which does not require preceeding\succeding whitespace). The last condition may be switched on\of by a preceeding call to the method 'enforceWhitespaces' and 'ignoreWhitespaces()' respectively. Further in order to switch to the case-sensitive modus (case-insensitive is default setting) a call to this method must be preceeded by a call to the method 'ExtendedGazetteer.CaseSensitive()'.

**Parameters:**

`S` - input text

**Returns:**

a vector of arrays of objects of type ExtendedGazetteerInfo, or `null` if no matches were found.

---

## findMatch

public ExtendedGazetteerInfo[] **findMatch**(char[] S,
int startAt)

Finds for a given string and given starting index, a longest match, and returns an array of objects of type ExtendedGazetteerInfo. This method is used as a subroutine in the 'findMatchInText' method. Note, that it is not necessary like in the 'findMatchInText' method that the S[startAt] must be preceeded with a whitespace character ! However by default a succeeding whitespace is required. In order to enforce\ignore succeeding whitespaces use the methods 'enforceWhitespaces' and 'ignoreWhitespaces()' respectively.

**Parameters:**

`S` - input string

`startAt` - initial position in the input string

**Returns:**

a list of objects of type ExtendedGazetteerInfo for the longest match found, or `null` if no matches were found.

**See Also:**

`ExtendedGazetteerInfo`

---

## findMatch

public ExtendedGazetteerInfo[] **findMatch**(char[] S,
int startAt,
int Len)

Finds for a given string, given starting index, and a given length an exact match (if such exists), and returns a list of objects of type ExtendedGazetteerInfo. Note, that it is not necessary like in the 'findMatchInText' method that the S[startAt] must be preceeded with a whitespace character ! However by default a succeeding whitespace is required. In order to enforce\ignore succeeding whitespaces use the methods 'enforceWhitespaces' and 'ignoreWhitespaces()' respectively.

**Parameters:**

`S` - input string

`startAt` - initial position in the input string

`Len` - length of the string to be matched

**Returns:**

a list of objects of type ExtendedGazetteerInfo for the longest match found, or `null` if no matches were found.

**See Also:**

`ExtendedGazetteerInfo`

---

## getAttributeName

public java.lang.String **getAttributeName**(int AttrIndex)

Returns the name of the attribute with a given ID.

**Parameters:**

`AttrIndex` - the ID of the attribute

**Returns:**

the name of the attribute, or `null` if the value of the ID is invalid

---

## getValueName

public java.lang.String **getValueName**(int AttrIndex,
int ValIndex)

Returns the value for a given attribute ID and value ID.

**Parameters:**

`AttrIndex` - the ID of the attribute

`ValIndex` - the ID of the value

**Returns:**
the value for the given attribute/value IDs, or `null` if any of the indices is invalid

---

## getNumAttributes

public int **getNumAttributes**()

Returns the number of attributes in the current gazetteer (number of possible indices).

**Returns:**
the number of attributes in the gazetteer

---

## getNumValues

public int **getNumValues**(int AttrIndex)

Returns the number of possible values for a given attribute (number of possible inidices).

**Parameters:**
`AttrIndex` - the ID of the attribute

**Returns:**
the number of different values for the given attribute or `null` if the attribute index is not valid

---

## testPrint

public void **testPrint**()

---

## CheckConsistency

public java.util.ArrayList **CheckConsistency**(ShUG grammar)

This function performs a consistency check in order to test if all feature structure types and attributes used by the Extended Gazetteer are present in the given type hierarchy.

**Parameters:**
`grammar` - object containing a type hierarchy for testing the consistency

**Returns:**
an array list of String containing all inconsistencies. If no incosistencies were found, the array list will be empty.

---

# JAVA DOCUMENTATION FOR THE CLASS ExtendedGazetteerInfo

---

## de.dfki.lt.sprout.runtime.gazetteer

# Class ExtendedGazetteerInfo

java.lang.Object
└ **de.dfki.lt.sprout.runtime.gazetteer.ExtendedGazetteerInfo**

public class **ExtendedGazetteerInfo**
extends java.lang.Object

## Constructor Summary

| |
|---|
| **ExtendedGazetteerInfo**()<br>Simple constructor for the class ExtendedGazetteerInfo. |
| **ExtendedGazetteerInfo**(int size, int[] AttrInd, int[] ValInd)<br>Constructor which initializes a new object of the class ExtendedGazetteerInfo with given values, except the values for start and end positions. |
| **ExtendedGazetteerInfo**(int startC, int endC, int size, int[] AttrInd, int[] ValInd)<br>Constructor which initializes a new object of the class ExtendedGazetteerInfo with given values |

## Method Summary

| | |
|---|---|
| java.lang.Object | **clone**()<br>Performs a deep copy of a ExtendedGazetteerInfo object. |
| int | **getAttributeID**(int index)<br>Gets the ID of the given attribute in the A/V list. |
| int | **getEndPos**()<br>Gets the value of the end position. |
| int | **getSize**()<br>Gets the size of the A/V List. |
| int | **getStartPos**()<br>Gets the value of the start position. |
| int | **getValueID**(int index)<br>Gets the ID of the given value in the A/V list. |
| void | **print**()<br>Prints the content of the current ExtendGazetteerInfo object to the standard output. |
| void | **print**(java.io.BufferedWriter out)<br>Prints the content of the current ExtendGazetteerInfo object to an output stream. |
| void | **set**(int size, int[] AttrInd, int[] ValInd)<br>Redefines the A/V List. |
| void | **set**(int startC, int endC, int size, int[] AttrInd, int[] ValInd)<br>Sets new values for all fields. |

| | |
|---|---|
| void | **setEndPos**(int index)<br>Sets a new value for the end field. |
| void | **setStartEndPos**(int start, int end)<br>Sets new values for the start\end fields. |
| void | **setStartPos**(int index)<br>Sets a new value for the start field. |

| **Methods inherited from class java.lang.Object** |
|---|
| equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

# Constructor Detail

## ExtendedGazetteerInfo

public **ExtendedGazetteerInfo**()

Simple constructor for the class ExtendedGazetteerInfo.

---

## ExtendedGazetteerInfo

public **ExtendedGazetteerInfo**(int startC,
int endC,
int size,
int[] AttrInd,
int[] ValInd)

Constructor which initializes a new object of the class ExtendedGazetteerInfo with given values

**Parameters:**

`startC` - start position in text
`endC` - end position in text
`size` - size of the attribut/value list
`AttrInd` - an array containing the IDs of the subsequent attributes in the list
`ValInd` - an array containing the IDs of the subsequent values in the list

---

## ExtendedGazetteerInfo

public **ExtendedGazetteerInfo**(int size,
int[] AttrInd,
int[] ValInd)

Constructor which initializes a new object of the class ExtendedGazetteerInfo with given values, except the values for start and end positions.

**Parameters:**

`size` - size of the attribut/value list
`AttrInd` - an array containing the IDs of the subsequent attributes in the list

`ValInd` - an array containing the IDs of the subsequent values in the list

# Method Detail

## print

public void **print**()

Prints the content of the current ExtendGazetteerInfo object to the standard output.

---

## print

public void **print**(java.io.BufferedWriter out)
throws java.io.IOException

Prints the content of the current ExtendGazetteerInfo object to an output stream.

**Throws:**

`java.io.IOException`

---

## setStartEndPos

public void **setStartEndPos**(int start,
int end)

Sets new values for the start\end fields.

**Parameters:**

`start` - start position

`end` - end position

---

## setStartPos

public void **setStartPos**(int index)

Sets a new value for the start field.

**Parameters:**

`index` - start position

---

## setEndPos

public void **setEndPos**(int index)

Sets a new value for the end field.

**Parameters:**

`index` - start position

---

## set

public void **set**(int startC,
int endC,
int size,
int[] AttrInd,
int[] ValInd)
Sets new values for all fields.
**Parameters:**
`startC` - start position
`endC` - end position
`size` - size of the attribut/value list
`AttrInd` - an array containing the IDs of the subsequent attributes in the list
`ValInd` - an array containing the IDs of the subsequent values in the list

---

## set

public void **set**(int size,
int[] AttrInd,
int[] ValInd)
Redefines the A/V List.
**Parameters:**
`size` - size of the attribut/value list
`AttrInd` - an array containing the IDs of the subsequent attributes in the list
`ValInd` - an array containing the IDs of the subsequent values in the list

---

## getSize

public int **getSize**()
Gets the size of the A/V List.

---

## getStartPos

public int **getStartPos**()
Gets the value of the start position.

---

## getEndPos

public int **getEndPos**()
Gets the value of the end position.

---

## getAttributeID

public int **getAttributeID**(int index)

Gets the ID of the given attribute in the A/V list.

**Parameters:**

`index` - index of the attribute

**Returns:**

ID of the given attribute, or `-1` if the attribute index is out of bound

---

## getValueID

public int **getValueID**(int index)

Gets the ID of the given value in the A/V list.

**Parameters:**

`index` - index of the value

**Returns:**

ID of the given value, or `-1` if the value index is out of bound

---

## clone

public java.lang.Object **clone**()

Performs a deep copy of a ExtendedGazetteerInfo object.

**Returns:**

a deep copy of the current object