TIGERSearch 2.1

User's Manual

Esther König, Wolfgang Lezius, Holger Voormann

IMS, University of Stuttgart

Last modified:2003/9/1

Table of contents

Chapter I - Introduction

1. About this manual

Welcome to the TIGERSearch software suite! We are glad that you have decided to use TIGERSearch for your linguistic research.

What has TIGERSearch been designed for? The answer is simple: The TIGERSearch software let's you *explore* syntactically annotated text corpora. If you are a grammar engineer who is developing a grammar, you might use TIGERSearch to obtain sample sentences for the syntactic phenomena you are interested in. If you are a lexicographer or terminologist, you can employ TIGERSearch to find out lexical properties of a word like the collocations the word is used in. Generally speaking, the TIGERSearch software can be used to *visualize* treebanks and to *extract information* from treebanks.

This manual describes the concept and handling of the TIGERSearch software suite. The manual is delivered in three different versions: The hyperlinked versions (HTML and PDF) can be found in the `doc/html/` and `doc/pdf/` subdirectories of your TIGERSearch distribution. Another version has been integrated as an help system into the TIGERSearch software. You can activate the help system by selecting one of the items in the *Help* menu or by clicking the *Help* button in the main window of both the TIGERSearch or TIGERRegistry tool.

The present version is the hyperlinked PDF version of the manual. It has been developed for browsing through a high-quality print version. To navigate through the manual, we recommend you to start with the table of contents or the index.

What are the central chapters of this manual? Users who just want to use the query functionality of the TIGERSearch software suite should have a look at chapter III and chapter IV which describe the TIGERSearch corpus query language and the TIGERSearch query tool. Corpus administrators should also read chapter V and chapter VI which explain the TIGER-XML corpus import/export format and the TIGERRegistry corpus administration tool, respectively.

Users who are interested in the concept, design, and implementation of the TIGERSearch software should also consult Wolfgang Lezius' Ph.D. thesis [Lezius2002] (in German).

⚠**Please note:** The TIGERSearch manual has been developed using the *JManual* environment. The JManual project is a joint initiative of the projects NITE and TIGER (cf. *http://www.tigersearch.de* for further information).

2. The TIGERSearch software tools

Searching on syntactically annotated corpora is much more time-consuming than searching on corpora which are only annotated on the word level (word, lemma, part-of-speech, morphological description etc.). Thus, an index structure which enables fast access to the results of many partial searches can greatly improve efficiency. For this reason we have decided to use an index-based architecture for the design of the TIGERSearch software suite.

Thus, the TIGERSearch software suite is divided into two tools:

- TIGERSearch

  The *TIGERSearch* tool is the corpus query processor. You can process queries, view the results, and export your favourite matches. The TIGERSearch query language is introduced in chapter III. The TIGERSearch tool is described in chapter IV.

- TIGERRegistry

  The import and preprocessing of corpora is realized in a tool called *TIGERRegistry*. For the import of treebanks, we have developed the TIGER-XML corpus encoding format. New corpora have to be converted into this format first. To support as many existing treebanks as possible, TIGERRegistry also comprises import filters (i.e. converters to TIGER-XML) for many popular treebank formats. The TIGER-XML corpus encoding format and the implemented import filters are described in chapter V and subsection 3.5, chapter VI, respectively. The TIGERRegistry tool is described in chapter VI.

3. New features in the 2.x versions

In TIGERSearch 2.1 the following features have been introduced:

- Query processing efficiency has been hugely improved for queries using regular expressions.
- Additional indexing strategies have been implemented. These strategies increase query processing efficiency up to 50%.
- The statistical viewer (introduced as statistical export in TIGERSearch 2.0) has been improved in many details.
- There have been several minor improvements and bugfixes.

In TIGERSearch 2.0 the following features have been introduced:

- The graphical user interface of the TIGERSearch software suite has been reimplemented. All important features can now be accessed directly from the GUI.
- The following utilities have been added to the TIGERSearch query tool: bookmarks, syntax highlighting of corpus queries, and statistical export.
- Corpus queries can be specified graphically, i.e. queries can be *drawn* in an intuitive way.
- TIGERSearch has been internationalized, i.e. Unicode characters can be used in corpus queries and any Unicode character will also be rendered in corpus visualization.
- Templates, a modularization concept for corpus queries, have been introduced.
- The TIGERSearch manual can now be accessed from the GUI. It is also available as a PDF file for high-quality printing.

- The TIGERSearch implementation is based on Java 1.4. Thus, many platform-dependent dialogs such as printing or file dialogs are much easier to handle.

4. Support

For up-to-date information about the TIGERSearch project, you should consult the TIGERSearch homepage: *http://www.tigersearch.de*

If you have any further question about the TIGERSearch software suite, just send a mail to the following mail address: *tigersearch@ims.uni-stuttgart.de*

Chapter II - Software installation

1. General system requirements

Hardware requirements

- min. 256 MB main memory (512 MB recommended)
- min. 150 MB free hard disk space (includes space required during installation)
- high-resolution monitor (1024x768 or higher, 256 colors)
- mouse or other pointing device

Windows

- Intel Pentium II/233 MHz or higher (or compatible)
- Microsoft Windows 98 (SE), 2000 (SP 2), XP, or NT 4.0 (SP6a)

Linux

- Intel Pentium II/233 MHz or higher (or compatible)
- Red Hat Linux 6.2 or higher; SuSE Linux 6.4 or higher

Solaris

- UltraSPARC II or higher
- Solaris 7 (2.7) or 8 (2.8)

Mac

- Power Mac G3, G4, G4 Cube; iMac; PowerBook G3, G4; iBook; eMac
- Mac OS X, version 10.2.4 or higher

⚠**Please note:** For Microsoft Windows, Linux and Solaris, the required Java Runtime Environment (JRE) is bundled with the respective download version of the TIGERSearch software. For Mac OS X, the JRE is part of the operating system installation.

2. Installation instructions

2.1 Windows

Installation

The installation on the Windows platform is realized by a self-extracting `.exe` file that also comprises the Java Virtual Machine which is necessary for using TIGERSearch. To install the TIGERSearch software suite on the Windows platform the following steps are necessary:

1. Download the installation file (*http://www.tigersearch.de*).

2. Install TIGERSearch on your system:

- Double-click the installation file. The installation of TIGERSearch will be started.
- Choose the destination directory. We recommend you to use the default destination directory `c:\TIGERSearch`.

  ⚠**Please note:** Do not choose a destination path that comprises space characters (e.g. `c:\Program Files\TIGERSearch`). Java-based programs often do not work properly if installed in such a destination directory.
- Choose the shortcut directory. The installation tool will place links to the TIGERSearch and TIGERRegistry program in this directory. Please note that links to the HTML and PDF versions of the TIGERSearch User's Manual are also placed in the shortcut directory. It is recommended to choose the option *Create a new Program Group* for the TIGERSearch tool.
- Now the program files are copied to your system.
- Start the TIGERSearch or TIGERRegistry program from the created Program Group (cf. subsection 1.1, chapter IV and subsection 2.1, chapter VI, respectively).

Uninstall

Enter the Windows System Control. Open the Software Properties. Mark the TIGERSearch entry in the software list and click the *Remove* button to start the uninstall process.

2.2 Unix (Linux/Solaris)

Installation

The installation on the Unix platform (Linux and Solaris) is realized by self-extracting `.bin` files that also comprise the Java Virtual Machines which are necessary for using TIGERSearch on Linux and Solaris. To install the TIGERSearch software suite on the Linux or Solaris platform the following steps are necessary:

1. Download the installation file for your platform (*http://www.tigersearch.de*).

2. Install TIGERSearch on your system:

- Start the binary installation file.

  ⚠**Please note:** The binary file must be executable, i.e. the file permission must be set to `rwx` (`chmod u+rwx filename`).

- Choose the shortcut directory. The installation tool will place links to the TIGERSearch and TIGERRegistry program in this directory. For example, you might choose `/usr/local/bin`, `/usr/bin` or `~/bin`. Please note that HTML and PDF versions of the TIGERSearch User's Manual are placed in the `doc/` subdirectory of the destination directory.
- Now the program files are copied to your system.
- Start the TIGERSearch or TIGERRegistry program by using the shortcuts created by the installation tool. If you did not create shortcuts, `TIGERSearch` and `TIGERRegistry` executables are located in the `bin/` subdirectory of the installation directory. (cf. subsection 1.1, chapter IV and subsection 2.1, chapter VI, respectively).

Uninstall

Enter the `UninstallerData/` subdirectory of the destination directory. Start the uninstall script `Uninstall_TIGERSearch`.

⚠**Please note:** The script must be executable, i.e. the file permission must be set to `rwx` (`chmod u+rwx filename`).

2.3 Mac OS X

Installation

The installation on the Mac OS X platform is realized by a self-extracting `.zip` file. To install the TIGERSearch software suite on the Mac OS X platform the following steps are necessary:

1. Download the installation file:

After downloading, the Mac installer included in the `.zip` file will be automatically recognized and decoded by *Stuffit Expander*. If your system does not handle the file automatically, download and install a current version of the StuffIt Expander software (cf. *http://www.aladdinsys.com/expander/*).

2. Install TIGERSearch on your system:

- After downloading (*http://www.tigersearch.de*), a new icon named `tssetup` is placed on your desktop. Double-click this icon. The installation of the TIGERSearch suite will be started.
- Choose the destination directory (typically the directory where programs are installed

on your system).

- Choose the alias folder. The installation tool will place links to the TIGERSearch and TIGERRegistry program in this directory. We recommend you to choose the option *Place into Home Folder*. Please note that links to the HTML and PDF versions of the TIGERSearch User's Manual are also placed in the alias folder.
- Now the program files are copied to your system.
- Start the TIGERSearch or TIGERRegistry program by clicking the corresponding icon in your dock.

Uninstall

Enter the `UninstallerData/` subdirectory of the destination directory. Double-click the `Uninstall_TIGERSearch` icon.

3. Internationalization

3.1 Unicode fonts

As the TIGERSearch software suite has been entirely implemented in Java, it is able to process corpora using the Unicode encoding. For the import of corpora, an XML-based approach is used to read any Unicode characters (cf. chapter V). The query processor is also able to process Unicode characters (cf. subsection 3.3, chapter IV).

As a platform-independent software, the TIGERSearch software suite is not able to analyze the font configuration of the user's platform in order to automatically detect an appropriate Unicode font. Therefore only the following two popular Unicode fonts are supported by the software:

- Arial Unicode MS (`Arialuni.ttf`, 52,000 characters; 23 MB installed)

  This font package is not freely available. However, it is included in the following commercial software packages: Microsoft Windows XP, Microsoft Office 2000, and Microsoft Publisher. You can easily check in the Windows System Control if the font package has already been installed on your computer. Otherwise, you will find the package on the CD-ROM of your commercial software.

- Cyberbit Bitstream (`Cyberbit.ttf`, 30,000 characters; 12.5 MB installed)

  This font package is freely available on the following web page: *ftp://ftp.netscape.com/pub/communicator/extras/fonts/windows/*. Please pay attention to the license agreement of the font package.

If one of these two font packages has been installed on your system, the TIGERSearch software will automatically detect and use it. Please consult the manual of your operating system how to install font packages on your computer.

⚠**Please note:** If you are working on a system where you do not have the user rights to install a font package, but you have already installed the TIGERSearch system on your computer, there is a workaround to install the font package to be used by the TIGERSearch software **only**: Just copy the font file (suffix `.ttf`) to the following subdirectory of the TIGERSearch installation directory: `jre/lib/fonts/`

3.2 Input methods

If you have installed a Unicode font package to be used by the TIGERSearch software, TIGERSearch will be able to display any Unicode character which is supported by the font package. However, typing in Unicode characters is a different story. So how can you type in a Unicode character in the corpus query editor, e.g. the Greek capital letter Omega?

Input methods of the operating system

On most platforms, specialized tools have been developed for this purpose. These tools are usually called *input methods*. As e.g. Greek characters do not exist on a German keyboard, these charaters are typed in as an abbreviation. For example, the string *Omega* might be used as an abbreviation for the Greek character that will be automatically expanded if the abbreviation has been typed in. Please consult the manual of your operating system to find out which tools are available for your platform.

Input methods of TIGERSearch

For the TIGERSearch software we have developed Java-based input methods for 16 European languages (in alphabetic order): Czech, Danish, Dutch, Finnish, French, German, Greek (classic and modern), Hungarian, Italian, Latin, Norwegian, Portugese, Romanian, Spanish, Swedish, and Turkish.

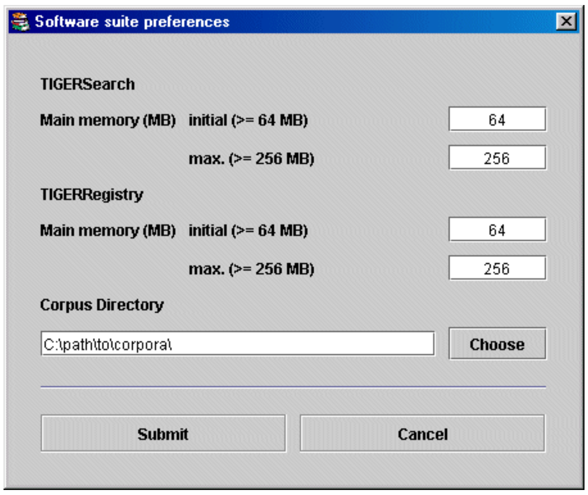
These input methods are automatically plugged into the TIGERSearch software suite during the installation process. In subsection 3.3, chapter IV you will find detailed instructions how to use the TIGERSearch input methods for typing in Unicode characters in your corpus queries. A description of the language mapping tables (file `europe.pdf`) is placed in the `doc/pdf/` subdirectory of your TIGERSearch installation.

Readers who are interested in the implementation of the input methods should consult the TIGERSearch homepage for additional information.

4. Software suite preferences

The main memory configuration of both the TIGERSearch and TIGERRegistry tool and the corpus base directory used by the two tools can be modified in the *Software suite preferences* window. It is opened by selecting the *Software preferences* item in the *Options* menu of the

TIGERRegistry tool:



TIGERSearch

Main memory (MB) initial (>= 64 MB) 64

max. (>= 256 MB) 256

TIGERRegistry

Main memory (MB) initial (>= 64 MB) 64

max. (>= 256 MB) 256

Corpus Directory

C:\path\to\corpora\

Figure: Software suite preferences

The following software suite parameters can be specified:

Memory configuration

Whereas the TIGERRegistry tool will always work fine if you have the required 256 MB main memory on your system, the TIGERSearch tool could lack on main memory if you work with very large corpora. Unfortunately, a typical feature of Java applications is that the initial main memory size (used when starting an application) and the maximum main memory size (that can be used by an application) have to be specified **before** the application is launched. So if you have more than 256 MB main memory installed on your system, this additional memory will not be used automatically.

If you want to increase the maximum main memory used by the TIGERSearch tool, just type in the new memory size in the *max* text field. We recommend you also to increase the initial memory used when an application is started - preferably a quarter of the maximum main memory. This will increase the corpus loading speed.

The TIGERRegistry tool should work without any problems using 256 MB memory. Nevertheless you can also change its memory configuration.

Corpus base directory

The corpus base directory used by the TIGERSearch software suite is created during the software installation as a subdirectory `TIGERCorpora` of the installation directory.

You can move the directory to another location (e.g. to another hard disc drive) as follows:

1. First create the new (empty) corpus directory.
2. Then open the *software preferences* window in the TIGERRegistry tool and specify the new directory in the *Corpus directory* text field by typing in the absolute path or using the file chooser. After submitting, the TIGERRegistry tool will be closed automatically.

3. Move the content of the old corpus directory to the new corpus directory.

Submit modifications

If you modified the memory or corpus directory configuration, press the *Submit* button. The modifications will be checked and the software suite configuration will be changed. As the status of the sofware suite has changed, the TIGERRegistry tool will be closed automatically. To profit from the new software configuration you will have to restart your current TIGERSearch session.

⚠**Please note:** The modification of the software configuration files is realized in a platform-independent way. Thus, the file permissions of the modified configuration files may change, depending on your default settings. So if you have installed the TIGERSearch software on a multi-user system, please check the file permissions of the modified configuration files. These are the two files `TIGERSearch.lax` and `TIGERRegistry.lax` in the `bin/` subdirectory of your TIGERSearch installation directory.

Chapter III - The TIGERSearch query language

1. Introduction

In this chapter, the TIGER language will be introduced, the query language for the TIGERSearch query engine. Actually, the TIGER language is not only a query language, but a general decription language for *syntax graphs*, i.e. restricted directed acyclic graphs. Syntax graphs are close relatives to (syntax) trees, to feature structures and to dependency graphs. Syntax graphs are syntax trees with two additions: Edges may have labels, and crossing edges are permitted. On the other hand, syntax graphs differ from feature structures due to the following two properties: First, the leaf nodes are ordered (like in syntax trees). Second, two edges must not join in a common node. This means that the structure sharing mechanism of feature structures is ruled out. However, structure sharing can be expressed on the additional level of so-called secondary edges.

We call the TIGER language a 'description language' since we want to emphasize that, in principle, the language serves to represent *corpus annotations* as well as *corpus queries*. In the TIGER system, for corpus annotation only a proper sublanguage of the TIGER description language may be used which does not include any possibility for underspecification. Note that, for reasons of technical convenience, the corpus annotation has to be encoded in the XML-counterpart of the corpus annotation sublanguage (TIGER-XML, cf. chapter V). The TIGER language accomodates a wide range of common treebank formats.

Acknowledgements and references

Syntax graphs are a formalization of the so-called Negra data structure, which has been used to encode the first major treebank for German, the Negra corpus (cf. [SkutEtAl1997]).

The design and the formal definition of the TIGER language has been influenced by other work on formal languages:

- unification-based formalisms

  (cf. [DoerreDorna1993], [Doerre1996], [DoerreEtAl1996], [EmeleZajac1990], [Emele1997], [HoehfeldSmolka1988], [Schmid1999])

- tree description languages

  (cf. [BlackburnEtAl1993], [DuchierNiehren1999], [MarcusEtAl1993], [RogersEtAl1992])

- corpus query languages

  (cf. [Christ1994], [ChristEtAl1999])

We want to thank our colleagues in the TIGER and DEREKO projects (cf. *TIGER project homepage* and *DEREKO project homepage*) for their creative input and their patience in discussing the details of earlier versions of this chapter.

2. Language overview

Like other formal languages, the TIGER language has been defined recursively, i.e. by nested layers of formal constructs. The invidual *nodes* in syntax graphs can be described with *feature constraints*, i.e. Boolean expressions over *feature-value pairs*. For the sake of computational simplicity, we do not admit nested feature structures. This means that *feature value descriptions* must denote constants or more precisely, strings. For this reason, we rather talk about *feature records* instead of feature structures. Here are some sample queries for the *TIGER-Sampler corpus* which is part of the TIGERSearch distribution (cf. [Smith2002] for an introduction to the corpus annotation).

```
[word="Abend" & pos="NN"]
[word=/Ma.*/ & pos= ("NN"|"NE")]
```

There are two elementary *node relations*, precedence (`.`) and labelled dominance (`>L`), and a range of derived node relations such as unlabelled direct dominance (`>`) and general dominance (`>*`). Example:

```
[cat="NP"] > [pos="ART"]
```

The *graph descriptions* are made from (restricted) Boolean expressions over node relations. Feature values, feature constraints, and nodes can be refered to by logical *variables* (e.g. `#n`) which are bound existentially at the outmost formula level. Example:

```
(#n:[cat="NP"] > [pos="ART"]) & (#n >* [pos="ADJA"])
```

Queries can include template calls and type names. *Template definitions* help to modularize lengthy queries. Types are a means to structure the universe of feature-value pairs. *Type definitions* include the declaration of features with domain and range types and the definition of type hierarchies.

In the subsequent sections, we introduce the TIGER language in an informal manner, i.e. by the way of examples. All sample queries should work on the *TIGERSampler corpus*, which is distributed with the TIGERSearch software. A formal definition of the TIGER language is given in a separate document (cf. [KoenigLezius2001]).

In section 12, the reader will find a quick reference of all language elements.

3. Feature value descriptions

3.1 String

Strings are to be marked by quotation marks, e.g. `"NN"`. To function as an actual TIGERSearch query, we must use the string as the value in a *feature-value pair*, surrounded by brackets:

```
[pos="NN"]
```

Characters which are TIGER reserved symbols must be preceded by the \-symbol (cf. following query). TIGER reserved symbols are listed in section 12.

```
[word="\."]
```

3.2 Types

Constant-denoting type symbols can serve as descriptions of feature values (cf. section 8). If a feature value symbol comes without quotes, it is interpreted as a type name:

```
[pos=proform]
```

3.3 Boolean expressions

Descriptions of feature values may be complex in the sense that type symbols and strings can be combined as Boolean expressions. The operators are `!` (negation), & (conjunction), and `|` (disjunction). Here are uses of a Boolean expressions as a feature value descriptions:

```
[pos = ("NN" | "NE")]
[pos = ! ("NN" | "NE") ]
```

By the way, the query above can be written as:

```
[pos != ("NN" | "NE") ]
```

⚠**Please note:** Boolean expressions for feature values which involve binary operators (conjunction and disjunction) must always be put into parentheses. See the example above where the outer parentheses cannot be omitted!

The operator precedence is defined as follows: `!`, &, `|`. This definition is illustrated by the following examples:

Example	Interpretation
`! "NN" & "NE"`	`(!"NN") & ("NE")`
`"NN" & "NE" | "PREL"`	`(NN"&"NE") | ("PREL")`

3.4 Variables

A Boolean feature value description can be refered to by a variable (in the example: `#c`):

```
[pos= #c:("NN"|"NE")]
```

A variable name has to start with a #-symbol. See subsection 7.2 for more meaningful applications of variables.

3.5 Regular expressions

One can also use *regular expressions* as feature value descriptions. Regular expressions are marked by enclosing slashes `/`. The syntax of regular expressions in TIGER is compatible with the syntax of regular expressions in Perl 5.003 (cf. [WallEtAl1996]). In our implementation, the following expression types are available:

Single characters

`a`	the character `a`
`.`	any character

Character classes

`[ace]`	any of the characters `a, c, e`
`[a-z]`	any lower-case letter
`[^a-f]`	any character except `a` to `f`

Special characters

`\s`	whitespace (space, tab, return)
`\d`	digit (0-9)

Alternatives

`(abc|de)`	the string `abc`, or `de`

Quantifiers

`(ab)*`	no or any number of `ab` (empty string, `ab`, `abab`, ...)
`(ab)+`	at least one `ab` (`ab`, `abab`, ...)
`(ab)?`	no or exactly one `ab`
`(ab){m,n}`	from `m` to `n` occurences of `ab`

Grouping

`ab+`	`a` followed by at least one `b` (`a`, `ab`, `abb`, ...)
`(ab)+`	at least one `ab` (`ab`, `abab`, ...)

⚠**Please note:** In our notation `/x/` means `/^x$/` in the Perl notation.

The following example means 'find words which start with *spiel*':

```
[word = /spiel.*/]
```

With the following query, one can locate the words *das* and *der*, irrespective of capitalization of the first letter:

```
[lemma = /[dD](as|er)/]
```

The following query finds words which contain at least one uppercase letter or a figure at a non-initial position, i.e. hyphenated compounds, and potential abbreviations and product names:

```
[word = /.+([0-9A-Z])+.*/]
```

⚠**Please note:** There is a difference between `.` and `\.` in the context of regular expressions. The following example denotes all strings starting with the prefix *sagt*, whereas the subsequent query means all strings with the prefix *sagt* followed by an arbitrary, possibly empty number of full stops:

```
[word = /sagt.*/]
[word = /sagt\.*/]
```

⚠**Please note:** The TIGER language compiler performs only a rough check of the syntax of regular expressions. The fine-grained syntax check for regular expressions will be carried out when a query is evaluated. Therefore it may involve more effort for you to discover the syntax errors you have made in a regular expression.

3.6 Boolean expressions vs. types vs. regular expressions

Regular expressions for feature values should be reserved for 'open-ended' feature values such as the *word* values. For features with a restricted range of a values such as syntactic categories, the use of types and Boolean expressions is suggested in order to increase readability and processing efficiency. If possible, types should be used instead of Boolean expressions.

4. Feature constraints

4.1 Feature-value pairs

Simple feature constraints, i.e. feature-value pairs such as `pos="NN"` have been already introduced in section 3.

4.2 Boolean expressions

Complex feature constraints are Boolean expressions over feature-value pairs:

```
[word="das" & pos="ART"]
[word = /sp.*/ | pos = "VVFIN"]
[word="das" & !(pos="ART")]
```

Basically, the last query is equivalent to the following two queries. Note that this kind of equivalence is not generally valid in a typed system (cf. section 8)!

```
[word="das" & pos != "ART"]
[word="das" & pos = (!"ART")]
```

The operator precedence is defined as follows: `!`, &, `|`. This definition is illustrated by the following examples:

Example	Interpretation
`[! pos="NN" & word="der" ]`	`[(!pos="NN") & (word="der")]`
`[pos="NN" & word="Haus" | pos="NE"]`	`[(pos="NN" & word="Haus") | (pos="NE")]`

4.3 Variables

A feature constraint can be prefixed by a variable. For further discussion see subsection 7.2.

```
[#f: (word="das" & pos="ART")]
```

4.4 Unspecific feature constraint

The *unspecific feature constraint* is written as `[]`. It denotes the whole universe of nodes.

5. Node descriptions

A *node description*, or simply a *node*, is a pair of a node identifier and a feature constraint. In queries, node identifiers must be *node variables* (in the example: `#n1`):

```
#n1:[word="das" & pos="ART"]
```

⚠**Please note:** The use of *constant node identifiers* is reserved for corpus definitions only. It may not be used in corpus queries:

```
(*) "id1":[word="das" & pos="ART"]
```

In queries, the node variable can be completely omitted from a node description unless it is needed for coreference with some other node description. This means that a plain feature constraint is interpreted as a node description with an unspecified node identifier.

```
[word="das" & pos="ART"]
```

On the other hand, a node variable by itself, `#n1`, is interpreted as an (unspecified) node description, i.e. as `#n1:[ ]`.

6. Node relations

6.1 Elementary node relations

Since syntax graphs are two-dimensional objects, we need two operators to express the spatial relations. We simply take the nomenclature from linguistics, and call the vertical dimension the *dominance* relation and the horizontal dimension the *precedence* relation.

Labelled direct dominance

The symbol > means direct dominance. It is further specified by an edge label, for example HD. This means that *labelled direct dominance* is expressed by e.g. >HD. The constraint that an edge which is labelled HD leads from node #n1 to node #n3 (or that #n3 is a direct HD-successor of #n1) is written as follows:

```
#n1 >HD #n3
```

As a more comprehensive example, the following labelled dominance constraints encode the vertical dimension of the tree in the graph presented below. Note that labelled dominance is a relation among nodes, not a function like it is the case for feature structures. This means that there may be more than one edge with the same label leading out of one mother node, cf. the NK-edges in the presented graph.

```
#n1 >SB #n2
#n1 >HD #n3
#n2 >NK #n4
#n2 >NK #n5
```

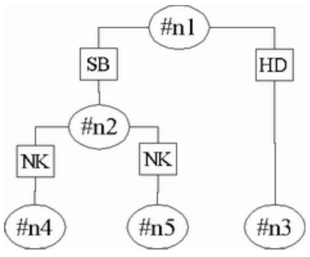


Figure: A simple syntax graph

On the basis of the directed edges, which are defined by the dominance relation, the nodes of a syntax graph in the corpus annotation are classified in the following manner:

- Inner nodes (nonterminal nodes)

  Nodes with outgoing edges, i.e. nodes with children, are called *inner nodes* or *nonterminal nodes*. In the presented figure only the nodes #n1 and #n2 are inner nodes.

- Leaf nodes (terminal nodes)

  Nodes without successors are named *leaf nodes* or *terminal nodes*. For example, the nodes #n4, #n5, and #n3 are the leaves of the tree in the presented figure.

Direct precedence of leaf nodes

A syntax graph is not only defined by constraining the vertical arrangement of its nodes, but also by the horizontal order of its leaf nodes, i.e. by the precedence relation among the leaves. We use the `.` symbol to represent *direct precedence*, since it serves as the concatenation operator in some programming languages. Now, the horizontal dimension of the tree in the presented figure is determined by the two precedence constraints:

```
#n4 . #n5
```

```
#n5 . #n3
```

6.2 Derived node relations

In queries, one wants to refer to indeterminate portions of a graph. Therefore, generalized notions of dominance and precedence ('wildcards') are necessary. Furthermore, certain convenient abbreviations should be introduced like a sibling operator.

Dominance

Dominance in general means that there is a *path* from one node to another one via a connected series of direct dominance relations. The *distance* of a dominance relation is the length of the path between the two nodes, i.e. the number of direct dominance relations to be traversed. The minimum distance is 1, i.e. in TIGERSearch, a node does not dominate itself.

We introduce the following generalizations of the labelled direct dominance relation `>L`:

`>`	unlabelled direct dominance
`>*`	dominance (minimum distance 1)
`>n`	dominance with distance `n` (`n>0`)
`>m,n`	dominance with distance between `m` and `n` (`0<m<n`)
`>@l`	leftmost terminal successor ('left corner')
`>@r`	rightmost terminal successor ('right corner')
`!>L`	negated labelled direct dominance
`!>`	negated unlabelled direct dominance
`!>*`	negated dominance
`!>n`	negated dominance, distance `n`
`!>m,n`	negated dominance, distance `m`...`n` (`0<m<n`)
`!>@l`	negated left corner
`!>@r`	negated right corner

For unlabelled edges, the label is considered to be unspecified, i.e. an unlabelled edge can match any edge.

The *left corner* resp. the *right corner* relation is reflexive, i.e. the leftmost resp. rightmost successor of a leaf node is the leaf node itself.

For the example tree in subsection 6.1, the following relations hold:

```
#n1 >* #n4
#n1 >2 #n4
#n1 >@l #n4
#n4 >@l #n4
```

Concerning the negated node relations, remember that variables are bound existentially at the outermost formula level. For example, the following query has to be read as 'find a syntax graph with an NP node and an NE node which are not connected by an NK-edge.'

```
[cat="NP"] !>NK [pos="NE"]
```

Precedence

So far, we have only talked about the precedence of leaf nodes. Whereas, for syntax trees, the precedence relation among its inner nodes is obvious, the situation is not clear at all for syntax graphs, which admit crossing edges. For example, the following two figures are different visualizations of the same syntax graph wrt. the arrangement of the nodes #n2 and #n3.

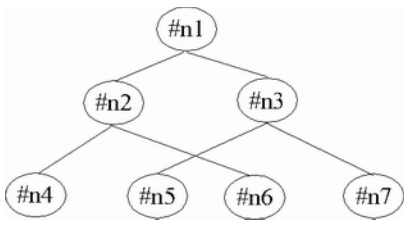


Figure: Precedence of inner nodes, first layout

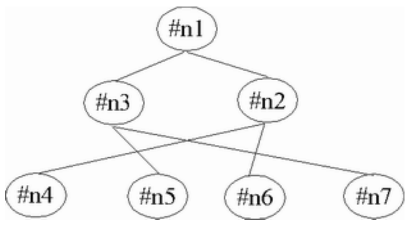


Figure: Precedence of inner nodes, second layout

There are several alternatives to define the precedence of inner nodes. One could adopt the strict precedence definition of proper trees: A node #n1 precedes a node #n2 if all nodes dominated by #n1 precede all nodes dominated by #n2 (cf. [SteinerKallmeyer2002]), i.e. if the right corner of #n1 precedes the left corner of #n2. Based on this strict definition, there would be no precedence relation at all between the nodes #n2 and #n3 in the first syntax graph shown above.

We decided for a somewhat weaker definition, which admits precedence statements even for overlapping graphs: A node #n1 is said to precede another node #n2 if the left corner of #n1 precedes the left corner of #n2 (left corner precedence).

According to the left corner based definition, node #n2 precedes node #n3 in the first syntax graph shown above. The user may define his own version of precedence as a template (cf. section 9), e.g. with the help of the right corner operator >@r.

The *distance* n between two non-identical leaf nodes is the number of leaf nodes between these two nodes increased by 1. For example, the distance between two direct neighbour leaves is 1. The distance between two inner nodes #x and #y is the distance between their respective left corners. Two nodes which share the same left corner, do not precede each other. In particular, a node does not precede itself. For the precedence relation, there are the derived operators listed below:

.*	precedence (minimum distance 1)
.n	precedence with distance n (n>0)
.m,n	precedence with distance between m and n (0>m>n)
!.*	negated precedence
!.n	negated precedence, distance n
!.m,n	negated precedence, distance m...n

Sibling relation

Two nodes #n1 and #n2 are *siblings* if they are directly dominated by the same node #n0, i.e. if they have the same mother node #n0. We use the following writing conventions:

$	siblings
$.*	siblings with precedence
!$	negated siblings

We did not include the negated $.* relation since it seems rather counter-intuitive.

6.3 Secondary edges

Although, in the kernel of the TIGER language, a node must not be dominated by more than one node, we have added so-called secondary edges as an extra layer in order to allow for structure sharing in syntax graphs. A secondary edge defines an additional parent node for a given node.

>~L	labelled secondary edge
>~	secondary edge
!>~L	negated labelled secondary edge
!>~	negated secondary edge

7. Graph descriptions

7.1 Boolean expressions

Graph descriptions or *graph constraints* are (restricted) Boolean expressions over node relations and node descriptions. Currently, conjunction & and disjunction | are available as logical connectives. For example, with the help of the &-operator, the following node relations can be joined into a graph constraint which retrieves the tree shown below.

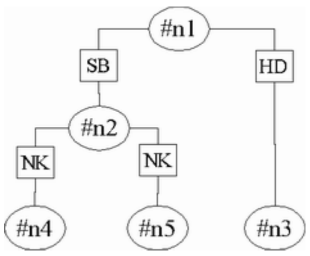


Figure: A simple syntax graph

```
(#n1 >SB #n2) & (#n1 >HD #n3) & (#n2 >NK #n4) & (#n2 >NK #n5)
```

Parentheses can be omitted in the usual fashion:

```
#n1 >SB #n2 & #n1 >HD #n3 & #n2 >NK #n4 & #n2 >NK #n5
```

The operator precedence is defined as follows: `Relation`, &, |. This definition is illustrated by the following examples:

Example	Interpretation
`#v > #w & #x`	`(#v > #w) & #x`
`#v & #w | #x`	`(#v & #w) | #x`

7.2 Use of variables

Variables for feature values

Variables for feature values are typically used to express agreement constraints. The following query looks for two adjacent nodes which are labelled with `NN` or `NE`.

```
[pos = #noun] . [pos = #noun:("NN" | "NE")]
```

Variables for feature constraints

With variables for feature constraints, we can search e.g. for sentences which contain the same preposition (the same word form!), twice:

```
[#f:(pos="APPR")] .* [#f]
```

⚠**Please note:** There is a subtle difference if we used a feature value variable instead. If we only require the identity of the feature value, i.e. of the part-of-speech tag, we get all sentences which contain at least two prepositions (not necessarily the same word form!):

```
[pos = #v:"APPR"] .* [pos=#v]
```

Node variables

Node variables are necessary to express multiple node relations with respect to one node, e.g. to list the children of a node like in the example in subsection 7.1:

```
#np:[cat="NP"] & #np > [pos="ADJA"] & #np > [pos="NN"]
```

Node (in)equality

Two nodes variables `#n1` and `#n2` may match the same node in the corpus. If this causes problems, the inequality of two node variables can be enforced e.g. by adding the following subformula which requires the variables `#n1` and `#n2` to match distinct nodes (due to the irreflexivity of the precedence relation):

```
((#n1 .* #n2) | (#n2 .* #n1))
```

In the case your corpus contains unary transitions (nonterminal nodes with one single nonterminal daughter), you should use a weaker constraint for node inequality:

```
((#n1 .* #n2) | (#n2 .* #n1)) | ((#n1 >* #n2) | (#n2 >* #n1))
```

7.3 Graph predicates

In principle, by now there are all the operators to describe syntax graphs. For reasons of convenience, and to a certain extent for reasons of completeness, we have added so-called graph predicates, e.g. to designate the root of a graph.

Root predicate

The root of a graph (for a whole sentence) can be identified by the predicate `root`.

```
root(#n1)
```

Arity predicates

The following graph description describes all graphs which contain a certain node `#n1` with **at least** two children `#n2` and `#n3`:

```
(#n1 > #n2) & (#n1 > #n3)
```

However, one would like to state that there must be **exactly** two children. For this reason, we introduce a two-place operator `arity` in order to be able to restrict the number of children of a node `#n1`, e.g. to two children:

```
(#n1 > #n2) & (#n1 > #n3) & arity(#n1,2)
```

The `arity` predicate can also come with three arguments in order to indicate an interval of number of children, e.g. from two to four children:

```
(#n1 > #n2) & (#n1 > #n3) & arity(#n1,2,4)
```

Similarly, there is a `tokenarity` operator to constrain the number of leaves which are dominated by this node. For example, the following means that node `#n1` must dominate exactly 5 terminal nodes. And the subsequent example states that node `#n1` must have between 5 and 7 leaves.

```
tokenarity(#n1,5)
tokenarity(#n1,5,7)
```

Continuity predicates

It may be useful to state that the leaves which are dominated by a node must form a continuous string or not. For this purpose, the two unary operators `continuous` and `discontinuous` have been introduced:

```
continuous(#n1)
discontinuous(#n1)
```

8. Type definitions

If the user has to declare the symbols which will be used in a corpus and in queries, inconsistencies in the corpus annotation and in the corpus query can be detected much more easily. TIGER allows for the declaration of type hierarchies (cf. subsection 8.2) and features (cf. subsection 8.4).

Type hierarchies have to be linked to a corpus. The linking of type hierarchies is described in subsection 4.3, chapter VI.

8.1 Built-in types

There are the following built-in types in the TIGER language:

- `String` for feature values not listable such as the values of the features `word` or `lemma`
- `UserDefConstant` for user-defined ('listable') feature values
- `Constant` comprises both `String` and `UserDefConstant`
- `NT` for feature constraints of nonterminal nodes
- `T` for feature constraints of terminal nodes
- `FREC` stands for all feature records (feature constraints), i.e. `NT` and `T`
- `Node` stands for node descriptions
- `Graph` means all graphs
- `Top` means anything (in the world of syntax graphs)

The hierarchy of built-in types is visualized in the following figure. Defining a type hierarchy is introduced in the following subsection.

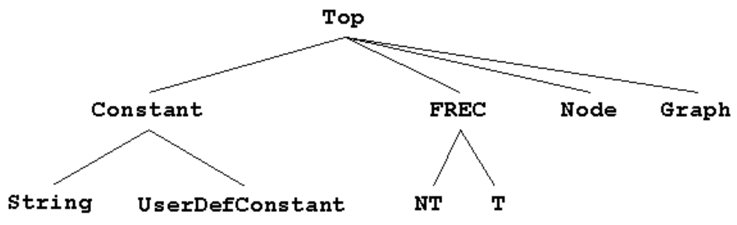


Figure: Built-in types

The built-in types `Top`, `Node` and `Graph` are only required on the conceptual level. In the current implementation, they cannot be referred to in the description language.

8.2 Definition of a type hierarchy

In the current implementation, the user can only add type definitions for feature values, i.e. for the type `UserDefConstant`. Let us start with a sample type hierarchy for the `pos` feature:

```
<typedeclaration base="pos" version="1.0">
...
  <!-- base type: pos -->
  <type name="pos">
     <subtype nameref="openclass"/>
     <subtype nameref="closedclass"/>
     <subtype nameref="punctuation"/>
     <subtype nameref="misc"/>
  </type>
...
</typedeclaration>
```

Type declaration are encoded in an XML-based format: The type declarations for a feature constitute the contents of a `typedeclaration` root element. The `base` attribute defines the base type (or root type) of the type system.

Type definition rules are encoded by `type` elements. The value of the `name` attribute is the type `t` which is being defined. The (direct) subtypes are given by the `nameref` attribute values of the individual `subtype` child elements. An occurrence of a type `t'` in an `subtype` element is called a *use* of `t'`.

In this way, *type hierarchies* can be defined. The 'terminal nodes' of a type hierarchy (of constant denoting types) are constants:

```
  <!-- open word classes-->
  <type name="openclass">
     <subtype nameref="noun"/>
     <subtype nameref="verb"/>
     <subtype nameref="adjective"/>
     <!-- adverb -->
     <constant value="ADV" comment="schon, bald, doch"/>
  </type>

  <!-- noun -->
  <type name="noun">
     <!-- common noun -->
     <constant value="NN" comment="Tisch, Herr, [das] Reisen"/>
     <!-- proper noun -->
     <constant value="NE" comment="Hans, Hamburg, HSV"/>
  </type>
```

On the basis of these type definitions, some disjunctions of feature values can be written in a more concise manner, e.g. the following query can now be replaced by the subsequent query:

```
[pos = ("NE"|"NN")]
```

```
[pos = noun]
```

Restrictions for type definitions

- A type must be defined at most once.
- A type must be used exactly once in a type definition.

The first restriction means that neither recursion nor cross-classification (alternative definitions of the same type symbol) can be expressed. If you think you need cross-classification, template definitions (section 9) might be a way out. The second restriction enforces that every type must be hooked up in the *type hierarchy*. In total this means that type definitions define a tree-shaped type hierarchy. Undefined types may only occur as the leaves of a type hierarchy.

8.3 Type definition example

In this subsection, the part-of-speech type hierarchy used for the TIGER Corpus Sampler (based on a modified version version of the STTS tag set) is listed as an example. The file is also placed in the `doc/examples/` subdirectory of your TIGERSearch installation.

```
<typedeclaration base="pos" version="1.0">

  <!-- base type: pos -->
  <type name="pos">
     <subtype nameref="openclass"/>
     <subtype nameref="closedclass"/>
     <subtype nameref="punctuation"/>
     <subtype nameref="misc"/>
  </type>

  <!-- open word classes-->
  <type name="openclass">
     <subtype nameref="noun"/>
     <subtype nameref="verb"/>
     <subtype nameref="adjective"/>
     <!-- adverb -->
     <constant value="ADV" comment="schon, bald, doch"/>
  </type>

  <!-- closed word classes-->
  <type name="closedclass">
     <!-- definite or indefinite article -->
     <constant value="ART" comment="der, die, das, ein, eine"/>
     <subtype nameref="proform"/>
     <!-- cardinal number -->
     <constant value="CARD" comment="zwei [Männer], [im Jahre] 1994"/>
```

```
      <subtype nameref="conjunction"/>
      <subtype nameref="adposition"/>
      <!-- interjection -->
      <constant value="ITJ" comment="mhm, ach, tja"/>
      <subtype nameref="particle"/>
   </type>

   <!-- noun -->
   <type name="noun">
      <!-- common noun -->
      <constant value="NN" comment="Tisch, Herr, [das] Reisen"/>
      <!-- proper noun -->
      <constant value="NE" comment="Hans, Hamburg, HSV"/>
   </type>

   <!-- verb -->
   <type name="verb">
      <subtype nameref="finite"/>
      <subtype nameref="nonfinite"/>
   </type>

   <!-- finite verbform -->
   <type name="finite">
      <!-- finite full verb -->
      <constant value="VVFIN" comment="du] gehst, [wir] kommen [an]"/>
      <!-- finite auxiliary verb -->
      <constant value="VAFIN" comment="[du] bist, [wir] werden"/>
      <!-- finite modal verb -->
      <constant value="VMFIN" comment="dürfen"/>
   </type>

   <!-- non-finite verbform -->
   <type name="nonfinite">
      <subtype nameref="infinitive"/>
      <subtype nameref="participle"/>
      <subtype nameref="imperative"/>
      <!-- infinitive with zu, full verb -->
      <constant value="VVIZU" comment="anzukommen, loszulassen"/>
   </type>

   <!-- infinitive verbform -->
   <type name="infinitive">
      <!-- inifinitive, full verb -->
      <constant value="VVINF" comment="gehen, ankommen"/>
      <!-- infinitive, auxiliary verb -->
      <constant value="VAINF" comment="werden, sein"/>
      <!-- infinitive, modal verb -->
      <constant value="VMINF" comment="wollen"/>
   </type>

   <!-- past participle -->
   <type name="participle">
      <!-- past participle, full verb -->
      <constant value="VVPP" comment="gegangen, angekommen"/>
      <!-- past participle, auxiliary verb -->
      <constant value="VAPP" comment="gewesen"/>
      <!-- past participle, modal verb -->
      <constant value="VMPP" comment="gekonnt, [er hat gehen] können"/>
   </type>

   <!-- past participle -->
   <type name="participle">
      <!-- past participle, full verb -->
      <constant value="VVPP" comment="gegangen, angekommen"/>
      <!-- past participle, auxiliary verb -->
      <constant value="VAPP" comment="gewesen"/>
      <!-- past participle, modal verb -->
      <constant value="VMPP" comment="gekonnt, [er hat gehen] können"/>
   </type>

   <!-- imperative -->
   <type name="imperative">
      <!-- imperative, full verb -->
      <constant value="VVIMP" comment="komm [!]"/>
      <!-- imperative, auxiliary verb -->
      <constant value="VAIMP" comment="sei [ruhig !]"/>
   </type>

   <!-- adjective -->
   <type name="adjective">
      <!-- attributive adjective -->
      <constant value="ADJA" comment="[das] große [Haus]"/>
      <!-- adverbal or predicative adjective -->
      <constant value="ADJD" comment="[er fährt] schnell, [er ist] schnell"/>
   </type>

   <!-- proform -->
   <type name="proform">
   <subtype nameref="prodemon"/>
   <subtype nameref="proindef"/>
   <!-- irreflexive personal pronoun -->
   <constant value="PPER" comment="ich, er, ihm, mich, dir"/>
   <subtype nameref="propos"/>
   <subtype nameref="prorel"/>
   <!-- reflexive pronoun -->
   <constant value="PRF" comment="sich, einander, dich, mir"/>
   <subtype nameref="prointer"/>
   <!-- pronominal adverb, bug, should be "PAV" -->
   <constant value="PROAV" comment="dafür, dabei, deswegen, trotzdem"/>
   </type>

   <!-- demonstrative pronoun -->
   <type name="prodemon">
      <!-- substitutive demonstrative pronoun -->
      <constant value="PDS" comment="dieser, jener"/>
      <!-- attributive demonstrative pronoun -->
      <constant value="PDAT" comment="jener [Mensch]"/>
   </type>

   <!-- indefinite pronoun -->
   <type name="proindef">
      <!-- substitutive indefinite pronoun -->
      <constant value="PIS" comment="keiner, viele, man, niemand"/>
      <!-- attributive indefinite pronoun -->
      <constant value="PIAT" comment="kein [Mensch], irgendein [Glas]"/>
   </type>

   <!-- posessive pronoun -->
   <type name="propos">
      <!-- substitutive possesive pronoun -->
      <constant value="PPOSS" comment="meins, deiner"/>
      <!-- attributive possessive pronoun -->
      <constant value="PPOSAT" comment="mein [Buch], deine [Mutter]"/>
   </type>

   <!-- relative pronoun -->
   <type name="prorel">
      <!-- substitutive relative pronoun -->
      <constant value="PRELS" comment="[der Hund,] der"/>
      <!-- attributive relative pronoun -->
      <constant value="PRELAT" comment="[der Mann ,] dessen [Hund]"/>
   </type>

   <!-- interrogative pronoun -->
   <type name="prointer">
      <!-- substitutive interrogative pronoun -->
      <constant value="PWS" comment="wer, was"/>
      <!-- attributive interrogative pronoun -->
      <constant value="PWAT" comment="welche [Farbe], wessen [Hut]"/>
      <!-- interrogative adverb or adverbial relative pronoun -->
      <constant value="PWAV" comment="warum, wo, wann, worüber, wobei"/>
   </type>

   <!-- conjunction -->
   <type name="conjunction">
   <subtype nameref="conjsub"/>
   <!-- coordinating conjunction -->
   <constant value="KON" comment="und, oder, aber"/>
   <!-- comparative conjunction -->
   <constant value="KOKOM" comment="als, wie"/>
```

```
  </type>

  <!-- subordinating conjunction -->
  <type name="conjsub">
     <!-- subordinating conjunction with zu-infinitive -->
     <constant value="KOUI" comment="um [zu leben], anstatt [zu fragen]"/>
     <!-- subordinating conjunction with sentence -->
     <constant value="KOUS" comment="weil, daß, damit, wenn, ob"/>
  </type>

  <!-- adposition -->
  <type name="adposition">
     <!-- preposition -->
     <constant value="APPR" comment="in [der Stadt], ohne [mich], von [jetzt an]"/>
     <!-- preposition + article -->
     <constant value="APPRART" comment="im [Haus], zur [Sache]"/>
     <!-- postposition -->
     <constant value="APPO" comment="[ihm] zufolge, [der Sache] wegen"/>
     <!-- circumposition, right part -->
     <constant value="APZR" comment="[von jetzt] an"/>
  </type>

  <!-- particle -->
  <type name="particle">
     <!-- "zu" before infinitive -->
     <constant value="PTKZU" comment="zu [gehen]"/>
     <!-- negating particle -->
     <constant value="PTKNEG" comment="nicht"/>
     <!-- separated verb particle -->
     <constant value="PTKVZ" comment="er kommt] an, [er fährt] rad"/>
     <!-- answer particle -->
     <constant value="PTKANT" comment="ja, nein, danke, bitte"/>
     <!-- particle with adjektive or adverb -->
     <constant value="PTKA" comment="am [schönsten], zu [schnell]"/>
  </type>

  <type name="punctuation">
     <!-- comma -->
     <constant value="$," comment=","/>
     <!-- final punctuation -->
     <constant value="$." comment=". ? ! ; :"/>
     <!-- other punctuation marks -->
     <constant value="$(" comment="- [,]()"/>
  </type>

  <type name="misc">
     <!-- foreign material -->
     <constant value="FM" comment="[Er hat das mit ``] A big fish ['' übersetzt]"/>
     <!-- nonword, with special characters -->
     <constant value="XY" comment="3:7, H2O, D2XW3"/>
     <!-- truncated element -->
     <constant value="TRUNC" comment="An- [und Abreise]"/>
     <!-- untagged -->
     <constant value="--"/>
     <!-- tagging of the token unknown -->
     <constant value="UNKNOWN"/>
  </type>

</typedeclaration>
```

8.4 Feature declarations

Feature declarations are part of the corpus definition (cf. section 11). A feature declaration states the following information:

- It declares the symbol under consideration as a feature name.
- It restricts the *domain* of the feature, i.e. it states for which type the feature may be used.

- It restricts the feature values (i.e. the *range*).

In the current implementation, features can only be declared for the built-in types `NT` and `T`. Furthermore, one cannot use a type to restrict the range of a feature, but the possible values for a feature have to be enumerated. One reason is that we want to keep the number of dependencies between corpus definition and type definitions as small as possible. The other reason is that such simple feature declarations can be also be constructed automatically - for those corpora which do not come with feature declarations.

If the value enumeration is omitted from a feature declaration, the default range of that feature is `String`.

For example, for the type `T` of feature constraints for terminal nodes, the features `word`, `lemma`, and `pos` may be defined as follows.

```
<feature name="word" domain="T"/>

<feature name="lemma" domain="T"/>

<feature name="pos" domain="T">
  ...
  <value name="VAFIN">...comment...</value>
  <value name="VAIMP"/>
  <value name="VAINF"/>
  <value name="VAPP"/>
  <value name="VMFIN"/>
  <value name="VMINF"/>
  ...
</feature>
```

⚠**Please note:** Each feature must be declared exactly once. The exclusion of multiple declarations for the same feature means that polymorphic overloading of a feature symbol is not permitted.

⚠**Please note:** In the TIGER description language being a typed language, the following two queries are **not** equivalent!

```
  [word="das" & !(pos="ART")]
  [word="das" & pos != "ART"]
```

The reason is that `!(pos="ART")` equals `!(T & pos="ART")` due to the corresponding feature declaration. The latter formula again is equivalent to `!(T) | (pos != "ART")`, i.e. either the feature `pos` is not defined on a type or it is defined and its value is not equal to `"ART"`.

9. Template definitions

When working with a syntactic representation formalism such as the TIGER language, certain pieces of code will be used over and over again. Furthermore, it is good programming and grammar writing style to define generalizations. Therefore, there is an urgent need for a means for defining abbreviations, templates (cf. [Shieber1986]), or macros. We chose TIGER templates to be *non-recursive logical relations*. This means that, although template calls may be embedded into template definitions, there must be neither direct nor indirect self-reference in template definitions. Hence, templates realize the 'database programming' part of a logic programming language such as Prolog (cf. [SterlingShapiro1986]).

A simple scheme `PrepPhrase` of a prepositional phrase which consists only of a preposition (`APPR`) and a proper noun (`NE`) can be defined as follows:

```
PrepPhrase(#n0) <-
   #n0:[cat="PP"] > #n1:[pos="APPR"]
   & #n0 > #n2:[pos="NE"]
   & #n1.#n2
   & arity(#n0,2) ;
```

The arrow `<-` marks a *defining clause* of a template definition. The `<-` operator is a two-place operator which takes a *template head* on its lefthand side and a *template body* on its righthand side. The template head consists of the *template name* (e.g. `PrepPhrase`) and a list of variables, the *argument parameters* of the template clause. The list of argument parameters must have at least one element, because otherwise no information flow between the template body and the calling environment would be possible, i.e. the template definition would be useless. An argument parameter `#x` must come with enough information (in the template body or in the query context) so that one can decide which of the built-in types `Constant`, `FRec`, `Node`, or `Graph`, the variable `#x` belongs to. The template body consists of a graph description. The end of a defining clause is marked by the `;` symbol. A template definition can consist of several defining clauses (not yet implemented).

The `PrepPhrase` template can now be *used* or *called* in a query:

```
#n0 & PrepPhrase(#n0) ;
```

This query will return all graphs from the given corpus, which contain a subtree (rooted in `#n0`) with the desired shape.

⚠**Please note:** The '`#n0 &`' part is important. TIGERSearch assumes that variables which occur only in the template call but not elsewhere, do not make sense. If you omit the '`#n0 &`' you will get an error message.

The template `PrepPhrase` can also be used in the body of some other template definition, e.g. in the definition of a pattern for `VerbPhrase` like *'geht nach Stuttgart'*.

```
VerbPhrase(#n0) <-
   #n0:[cat="VP"] > #n1:[pos="VVFIN"]
   & #n0 > #n2
   & #n1.#n2
   & arity(#n0,2)
   & PrepPhrase(#n2) ;
```

⚠**Please note:** The scope of a variable is the defining clause it occurs in. E.g. the variable `#n2` in the definition of `PrepPhrase` is distinct from the variable `#n2` in the definition of `VerbPhrase`. The TIGER language interpreter will *resolve* the call to `PrepPhrase` in the following manner. It will:

1. look up the defining clause of `PrepPhrase`.
2. replace all the variable names by new ones (in order to avoid unintended confusion of

variables from the 'calling' clause and the 'called' clause).

3. identify the node id variable `#n2` in `VerbPhrase` with the argument `#n0` of `PrepPhrase`'s defining clause. Identification of variables means that the constraints for both variables are joined into a single constraint by a logical conjunction.

After the resolution step, the `VerbPhrase`-clause has the following shape:

```
VerbPhrase(#n0) <-
   #n0:[cat="VP"] > #n1:[pos="VVFIN"]
   & #n0 > #n2
   & #n1.#n2
   & #n2:[cat="PP"] > #n21:[pos="APPR"]
   & #n2 > #n22:[pos="NE"]
   & #n21.#n22
   & arity(#n0,2)
   & arity(#n2,2) ;
```

If we want to find out which verbs go with which prepositions, this can be done by adding the appropriate arguments or parameters.

```
PrepPhrase(#n0,#prep) <-
   #n0:[cat="PP"] > #n1:[word=#prep & pos="APPR"]
   & #n0 > #n2:[pos="NE"]
   & #n1.#n2
   & arity(#n0,2) ;

VerbPhrase(#n0,#verblemma,#prep) <-
   #n0:[cat="VP"] > #n1:[pos="VVFIN" & lemma=#verblemma]
   & #n0 > #n2
   & #n1.#n2
   & arity(#n0,2)
   & PrepPhrase(#n2,#prep) ;
```

After a while, we might get interested in a more complex notion of prepositional phrases. For that reason, a template definition may consist of several, alternative defining clauses (not yet implemented):

```
PrepPhrase(#n0,#prep) <-
   #n0:[cat="PP"] > #n1:[word=#prep & pos="APPR"]
   & #n0 > #n2:[pos="NE"]
   & #n1.#n2
   & arity(#n0,2) ;

PrepPhrase(#n0,#prep) <-
   #n0:[cat="PP"] > #n1:[word=#prep & pos="APPR"]
   & #n0 > #n2:[pos="ART"]
   & #n0 > #n3:[pos="NN"]
   & #n1.#n2
   & #n2.#n3
   & arity(#n0,3) ;
```

The above definition is equivalent to a disjunction of the two defining clauses:

```
PrepPhrase(#n0,#prep) <-
   ( #n0:[cat="PP"] > #n1:[word=#prep & pos="APPR"]
     & #n0 > #n2:[pos="NE"]
     & #n1.#n2
     & arity(#n0,2) )
   |
   ( #n0:[cat="PP"] > #n1:[word=#prep & pos="APPR"]
     & #n0 > #n2:[pos="ART"]
     & #n0 > #n3:[pos="NN"]
     & #n1.#n2
     & #n2.#n3
     & arity(#n0,3) );
```

or, in a more packed manner:

```
// packed definition of PrepPhrase
PrepPhrase(#n0,#prep) <-
   #n0:[cat="PP"]
   & #n1:[word=#prep & pos="APPR"]
   & #n1.#n2
   & ( ( #n0 > #n1
         & #n0 > #n2:[pos="NE"]
         & arity(#n0,2) )
       |
       ( #n0 > #n1
         & #n0 > #n2:[pos="ART"]
         & #n0 > #n3:[pos="NN"]
         & #n2.#n3
         & arity(#n0,3) ) ) ;
```

The `//`-symbol marks the remainder of a line as a comment.

Types vs. templates

One might think of abbreviating a disjunction of feature values by a template name:

```
gen-dat(#case) <-
    [ case = #case: ( "gen" | "dat" ) ] ;
```

But we recommend to use a type definition instead, if possible, since this does not introduce an explicit disjunction into the structure.

```
gen-dat := "gen", "dat" ;
```

However, TIGERSearch admits only a single type hierarchy. Therefore, views which are orthogonal to this primary type hierarchy, must be expressed by templates.

10. Possible extensions of the TIGER language

In this section, we discuss some possible extensions of the TIGER language and the reasons why we have not adopted them (yet).

10.1 Variables for edge labels

There are already implicit ('don't care') variables for edge labels, i.e. the unlabelled dominance operator > and the dominance wildcard `>*` etc. If explicit variables for edge labels were available, the user could require co-reference of edge labels. To us it is unclear what kind of computational complexity will be introduced by this additional expressivity. And furthermore, there is a work-around: Since the number of edge labels (grammatical functions) tends to be rather small, it does not seem too inconvenient to define appropriate templates which enumerate pairs of edges with the same labels.

10.2 Negated graph descriptions

Negation on the level of graph descriptions seems somewhat difficult to be grasped conceptually. Since it would have to be pushed down to the level of node relations and feature values anyway, and there is negation available on these lower levels, this might give enough expressivity.

10.3 Universal quantifier

You might be tempted to state the following expression to find trees which are rooted in a `VP`, but which do **not** contain **any** `NP`:

```
  [cat="VP"] >* [cat=(!"NP")]
```

But in the TIGER language, this query has the interpretation: 'Find a (sub-)graph rooted in a `VP` with **at least one** inner node `#n` which is **not** labelled `NP`':

```
exists #child: ([cat="VP"] >* #child:[cat=(!"NP")])
```

The desired constraint of finding a graph which does **not** contain **any** `NP` requires universal quantification and the implication operator (i.e. negation of graph descriptions):

```
exists #node: forall #child: ( (#node:[cat="VP"] >* #child) =>
                               (#child:[cat=(!"NP")])
                             )
```

The use of the universal quantifier causes computational overhead since universal quantification usually means that a possibly large number of copies of logical expressions have to be produced. For the sake of computational simplicity and tractability, the universal quantifier is (currently) not part of the TIGER language.

11. Appendix: Corpus definition

Actually, corpora for TIGERSearch have to be defined in TIGER-XML. However, TIGER-XML is a direct translation of the corpus definition sublanguage of the TIGER description language. The restrictions for corpora are as follows:

Declaration of required features

A corpus definition must include the feature declarations for the type `NT` of nonterminal constraints and the type `T` of terminal constraints.

⚠**Please note:** If no explicit feature declarations are given, in most cases, they can be derived automatically from the corpus.

Single Root Node, Connectedness, No Structure Sharing

Every node in a graph except for one distinguished node (*root node*) has to be directly dominated by exactly one other node.

⚠**Please note:** A multi-rooted graph (unconnected subgraphs) can be turned automatically into a graph with unique root node by adding an 'artificial' root node plus the edges which point to the individual subgraphs.

⚠**Please note:** A structure sharing mechanism (multi-dominance) is provided by the additional layer of 'secondary edges'.

Acyclicity

No node may (indirectly) dominate itself.

Full Disambiguation

- The graph constraints may only include the basic node relations labelled direct dominance (`>L`) and direct precedence (`.`).
- The feature constraint for a node must be a conjunction of feature-value pairs either for all features which have been declared for `NT` or for those which have been defined for `T`.
- For each node, its arity (i.e. the number of its children) has to be fixed.
- The precedence relation on terminal nodes (leaf nodes) must be a total order.
- The only logical connective on all structural levels is the conjunction operator &.
- Neither types nor template calls nor regular expressions are admitted.

12. Appendix: Query Language Quick Reference

Reserved symbols

All the operators and built-in types which are listed below are reserved symbols. In particular, this means that the use of the following characters

```
 ! " # $ & ( ) * + , - . / : ; = > ? @ | { }
```

in edge labels, strings, constants, predicate or type names may cause problems. In these cases, a preceding operator \ will help. Example:

```
[word = "\,"]
```

Built-in types

The sample queries for the built-in types are meant to illustrate the context where a built-in type may occur. However, these queries are not really meaningful, since they are too general.

Symbol	*Meaning*	*Sample query*
`Constant`	constants	`[word = Constant]`
`String`	strings	`[word = String]`
`UserDefConstant`	user defined constants	`[pos=UserDefConstant]`
`FREC`	feature records	`[FREC]`
`NT`	feature records for nonterminals	`[NT]`
`T`	feature records for terminals	`[T]`

Constants

`"`	constant mark	`[word = "Geld"]`

Regular expressions for constants

`/`	regular expression mark	`[word = /Geld/]`
`.`	unspecified character	`[word = /sag./]`
`*`	unrestricted repetition	`[lemma = /spiel.*/]`
`+`	repetition with minimum 1	`[word = /.+[0-9A-Z]+.*/]`
`?`	optionality	`[word = /(Leben)s?/]`
`[ ]`	character set	`[word = /.+[0-9A-Z]+.*/]`
`^`	negated character sets	`[word = /[^0-9A-Z].*/]`

( )	grouping	[word = /([lmnp][aeiou])+/]
|	disjunction	[word = /[dD](as|er)/]
\	escape for reserved characters	[word = /.*\-.*/]

Feature-value pairs

=	feature-value pair	[pos = "NN"]
!=	negated feature-value pair	[pos != "NN"]

Graph predicates

root()	root of a graph	root(#n)

arity(,)
arity(, ,)
	arity of a node	arity(#n,2)
		arity(#n,2,4)

tokenarity(,)
tokenarity(, ,)
	number of dominated leaves	tokenarity(#n,5)
		tokenarity(#n,5,8)

continuous( )	continuous leaves	continuous(#n)
discontinuous( )	discontinuous leaves	discontinuous(#n)

Dominance relation

>L	labelled direct dominance	[cat="NP"] >NK [cat="NP"] [cat="NP"] >OA\-MOD [cat="NP"]
>	direct dominance	[cat="NP"] > [pos="NE"]
>*	dominance	[cat="NP"] >* [pos="NE"]
>$	dominance, distance n	[cat="NP"] >2 [pos="NE"]
>m,n	dominance, distance m..n	[cat="NP"] >2,3 [pos="NE"]
>@l	left corner	[cat="NP"] >@l [word="die"]
>@r	right corner	[cat="NP"] >@r [word="Jahr"]
$	siblings	[word="die"] $ [cat="NP"]
$.*	siblings with precedence	[word="etwas"] $.* [cat="NP"]
!>L	neg. labelled direct dominance	[cat="NP"] !>GR [cat="NP"]
!>	neg. direct dominance	[cat="NP"] !> [pos="NE"]
!>*	neg. dominance	[cat="NP"] !>* [pos="NE"]
!>n	neg. dominance, distance n	[cat="NP"] !>2 [pos="NE"]
!>m,n	neg. dominance, distance m..n	[cat="NP"] !>2,3 [pos="NE"]

`!>@l`	neg. left corner	`[cat="NP"] !>@l [word="etwas"]`
`!>@r`	neg. right corner	`[cat="NP"] !>@r [word="etwas"]`
`!$`	neg. siblings	`[word="etwas"] !$ [cat="NP"]`
`>~L`	labelled secondary edge	`[cat="VP"] >~HD [cat="NP"]`
`>~`	secondary edge	`[cat="VP"] >~ [cat="NP"]`
`!>~L`	neg. labelled secondary edge	`[cat="VP"] !>~HD [cat="NP"]`
`!>~`	neg. secondary edge	`[cat="VP"] !>~ [cat="NP"]`

Precedence relation

`.`	direct precedence	`[word="die"] . [pos=noun]`
`.*`	precedence	`[word="die"] .* [pos="NN"]`
`.n`	precedence, distance n	`[word="die"] .2 [pos="NN"]`
`.m,n`	precedence, distance m..n	`[word="die"] .2,4 [pos="NN"]`
`!.`	neg. direct precedence	`[word="etwas"] !. [pos="NN"]`
`!.*`	neg. precedence	`[word="etwas"] !.* [pos="NN"]`
`!.n`	neg. precedence, distance n	`[word="etwas"] !.2 [pos="NN"]`
`!.m,n`	neg. precedence, distance m..n	`[word="etwas"] !.2,4 [pos="NN"]`

Variables

`[f=#v]`	variable for a *feature value*	`[pos=#x:("NN"|"NE")] . [pos=#x]`
`[#x]`	variable for a *feature description*	`[#x:(pos="APPR")] .* [#x]`
`#n`	variable for a *node identifier*	`#n:[cat="NP"] & #n > [pos="ADJA"]) & #n > [pos="NN"]`

Boolean expressions

`( )`	bracketing	`[pos=(!("NN" | "ART"))]`
`!`	negation (feature values)	`[pos=(!"NN")]`
	negation (feature constraints)	`[!(pos="NN")]`
`&`	conjunction (feature values)	`[pos=(!"NN" & !"NE")]`
	conjunction (feature constraints)	`[pos="NE" & word="Bonn"]`
	conjunction (graph descriptions)	`#n1>#n2 & #n3.#n2`

`|`	disjunction (feature values)	`[pos=("NN" | "NE")]`
	disjunction (feature constraints)	`[pos="NE" | word="es"]`
	disjunction (graph descriptions)	`#n1>#n2 | #n1>#n3`

⚠**Please note:** Negation (`!`) must not have scope over variables.

⚠**Please note:** A Boolean expression for a feature value must always be put into parentheses (e.g. `pos=("NN"|"NE")`).

Comments

`//`	line comment	`[pos="NE"].[pos="NE"] // 2-word proper nouns`

Chapter IV - The TIGERSearch query tool

1. An introduction to TIGERSearch

1.1 Starting the TIGERSearch tool

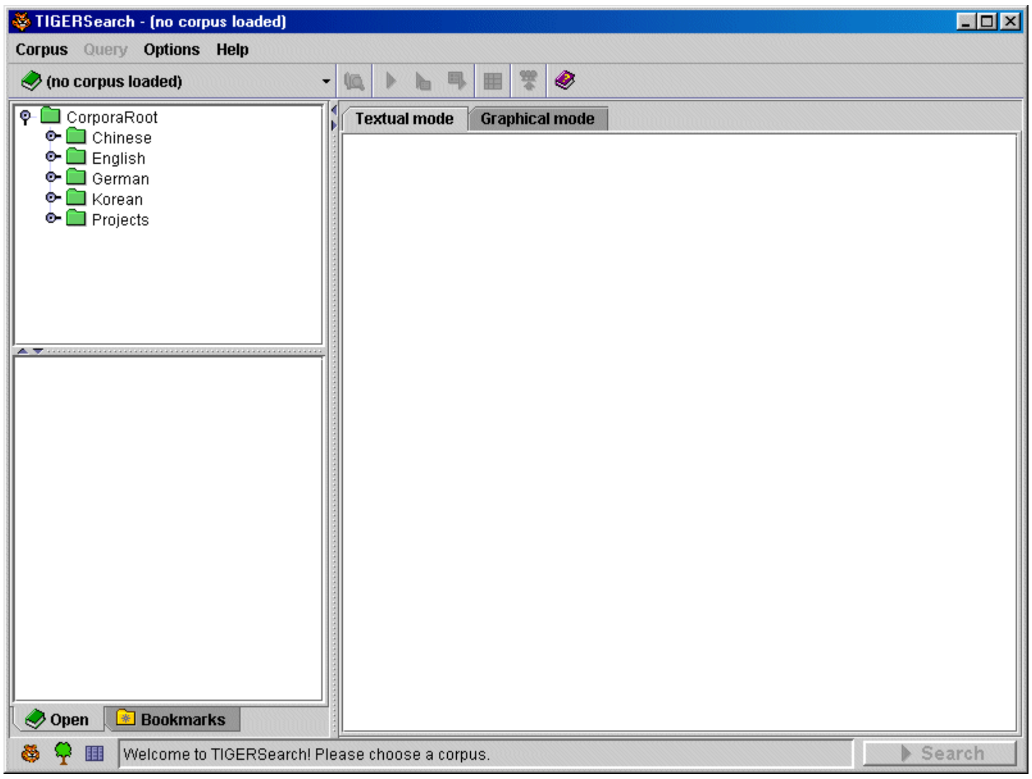
The way you can start the TIGERSearch tool depends on your operating system. On Windows machines, a program group called TIGERSearch has been created during the installation - so you just have to select the TIGERSearch program in the start menu.

On Unix machines, symbolic links have been created. If your general path is set properly, you may just need to type in `TIGERSearch`. However, the TIGERSearch start program can always be found in the TIGERSearch installation path:

```
INSTALLATIONPATH/bin/TIGERSearch
```

Please note: Any relative path specified in a dialog window is evaluated with regard to the so-called working directory. On Unix machines this directory is defined as the TIGERSearch starting directory (i.e. the directory TIGERSearch has been started from). On Mac and Windows machines the working directory is defined as the user's home directory.

When you start the TIGERSearch tool, the TIGERSearch main window pops up (cf. screenshot). Position and size of the window are saved when leaving the tool, so the arrangement of your windows will be restored in the next TIGERSearch session.



Welcome to TIGERSearch! Please choose a corpus.

Figure: TIGERSearch main window

The main window of TIGERSearch ist divided into two parts. On the left you find the so-called information panel. It comprises a list of all corpora (cf. subsection 2.1), some special information about the currently loaded corpus (cf. subsection 2.2, subsection 2.3 and subsection 2.4), and the user's bookmarks (cf. subsection 3.4). On the right there is the query input area which consists of a textual and a graphical query editor (cf. section 3 and section 4, respectively).

1.2 The TIGERSearch help window

The TIGERSearch User's Manual can be accessed directly within the TIGERSearch user interface. The TIGERSearch help window can be activated by pressing the *Help* button in the upper toolbar or selecting one of the items in the *Help* menu.

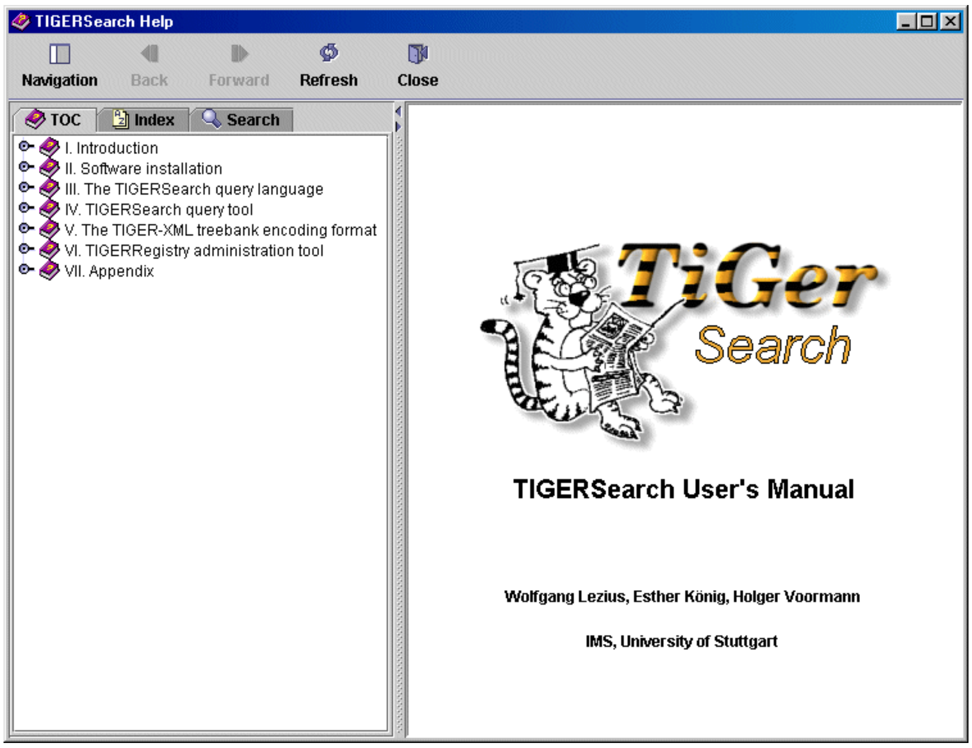
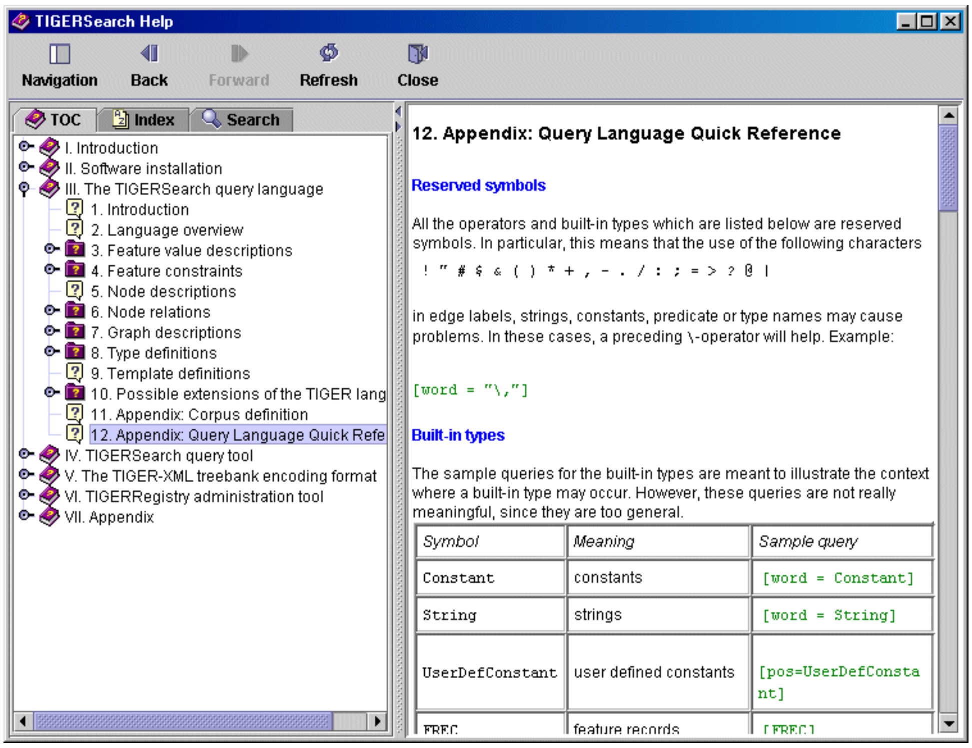

TIGERSearch Help
Navigation
Back
Forward
Refresh
Close
TOC
Index
Search
I. Introduction
II. Software installation
III. The TIGERSearch query language
IV. TIGERSearch query tool
V. The TIGER-XML treebank encoding format
VI. TIGERRegistry administration tool
VII. Appendix
TiGer Search
TIGERSearch User's Manual
Wolfgang Lezius, Esther König, Holger Voormann
IMS, University of Stuttgart

Figure: The TIGERSearch help window

The help window is divided into two parts: On the left there is the manual navigation area. Here you find a table of contents, an index, and a search engine. Just browse through the table of contents or through the index and click on the topic you are interested in:



I. Introduction
II. Software installation
III. The TIGERSearch query language
1. Introduction
2. Language overview
3. Feature value descriptions
4. Feature constraints
5. Node descriptions
6. Node relations
7. Graph descriptions
8. Type definitions
9. Template definitions
10. Possible extensions of the TIGER lang
11. Appendix: Corpus definition
12. Appendix: Query Language Quick Refe
IV. TIGERSearch query tool
V. The TIGER-XML treebank encoding format
VI. TIGERRegistry administration tool
VII. Appendix

12. Appendix: Query Language Quick Reference

Reserved symbols

All the operators and built-in types which are listed below are reserved symbols. In particular, this means that the use of the following characters

```
! " # $ & ( ) * + , - . / : ; = > ? @ |
```

in edge labels, strings, constants, predicate or type names may cause problems. In these cases, a preceding \-operator will help. Example:

```
[word = "\,"]
```

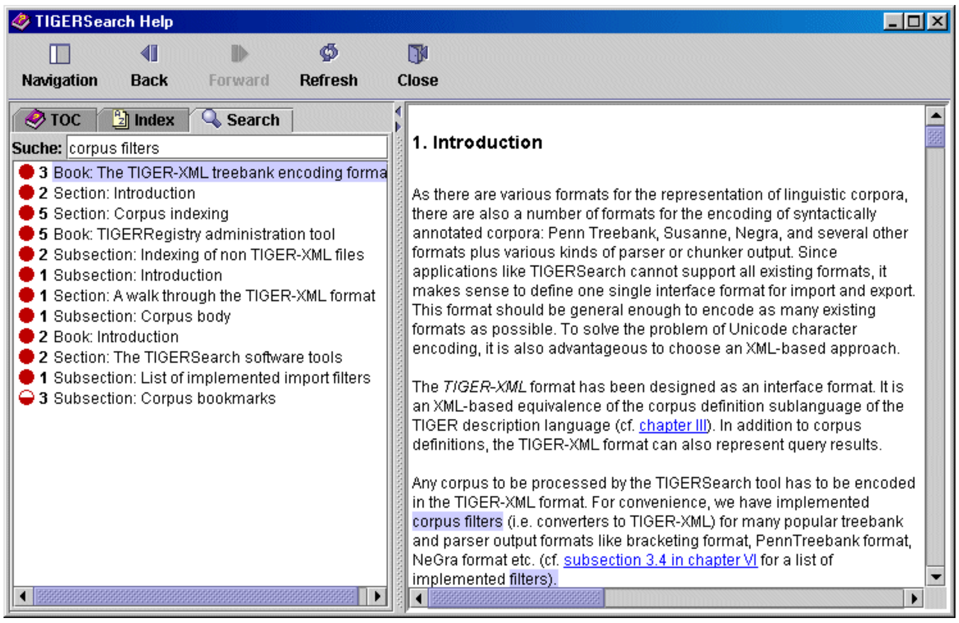
Built-in types

The sample queries for the built-in types are meant to illustrate the context where a built-in type may occur. However, these queries are not really meaningful, since they are too general.

Symbol	*Meaning*	*Sample query*
`Constant`	constants	`[word = Constant]`
`String`	strings	`[word = String]`
`UserDefConstant`	user defined constants	`[pos=UserDefConstant]`
`FREC`	feature records	`[FREC]`

Figure: Navigating through the manual

If you are looking for a special topic which you cannot find in the table of contents or in the index, you might use the search engine. Just type in your search item(s) and the search engine finds all topics in which one of the items is used:



Suche: corpus filters

● 3 Book: The TIGER-XML treebank encoding forma
● 2 Section: Introduction
● 5 Section: Corpus indexing
● 5 Book: TIGERRegistry administration tool
● 2 Subsection: Indexing of non TIGER-XML files
● 1 Subsection: Introduction
● 1 Section: A walk through the TIGER-XML format
● 1 Subsection: Corpus body
● 2 Book: Introduction
● 2 Section: The TIGERSearch software tools
● 1 Subsection: List of implemented import filters
● 3 Subsection: Corpus bookmarks

1. Introduction

As there are various formats for the representation of linguistic corpora, there are also a number of formats for the encoding of syntactically annotated corpora: Penn Treebank, Susanne, Negra, and several other formats plus various kinds of parser or chunker output. Since applications like TIGERSearch cannot support all existing formats, it makes sense to define one single interface format for import and export. This format should be general enough to encode as many existing formats as possible. To solve the problem of Unicode character encoding, it is also advantageous to choose an XML-based approach.

The *TIGER-XML* format has been designed as an interface format. It is an XML-based equivalence of the corpus definition sublanguage of the TIGER description language (cf. chapter III). In addition to corpus definitions, the TIGER-XML format can also represent query results.

Any corpus to be processed by the TIGERSearch tool has to be encoded in the TIGER-XML format. For convenience, we have implemented corpus filters (i.e. converters to TIGER-XML) for many popular treebank and parser output formats like bracketing format, PennTreebank format, NeGra format etc. (cf. subsection 3.4 in chapter VI for a list of implemented filters).

Figure: The help window search engine

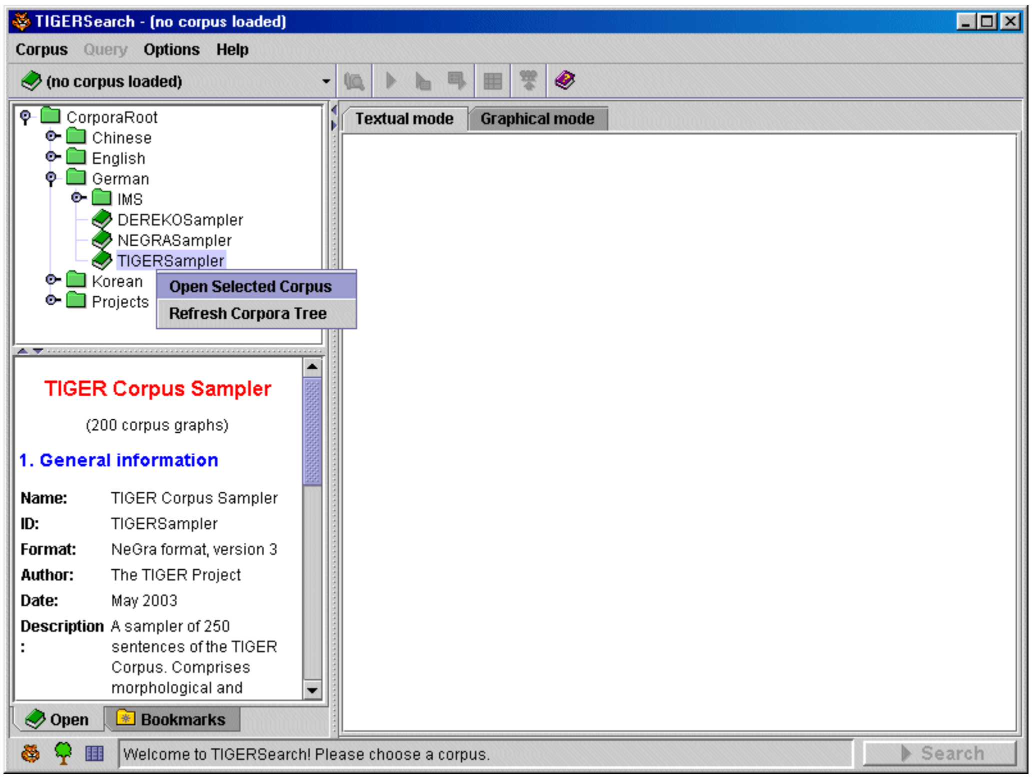
The help window toolbar comprises the following buttons: The *Navigation* button lets you hide/show the navigation bar. The *Back* and *Forward* buttons let you navigate through the topics you have viewed before. The *Refresh* button reloads a topic, and the *Close* button closes the help window.

Corpus queries are displayed as green-colored hyperlinks. If you click on a query hyperlink, the query text is automatically copied into the query text editor of the TIGERSearch main application.

2. Loading a corpus

2.1 Selecting a corpus

TIGERSearch corpora are organized in a hierarchical file system, i.e. related corpora are grouped in folders. To see the corpora available, select the *Open* tab in the information panel. Now you can browse through the corpus tree (upper-hand side of the tab), have a look at the corpus properties (lower-hand side), and finally load a corpus. The corpus properties are displayed if you mark a corpus symbol:



TIGER Corpus Sampler

(200 corpus graphs)

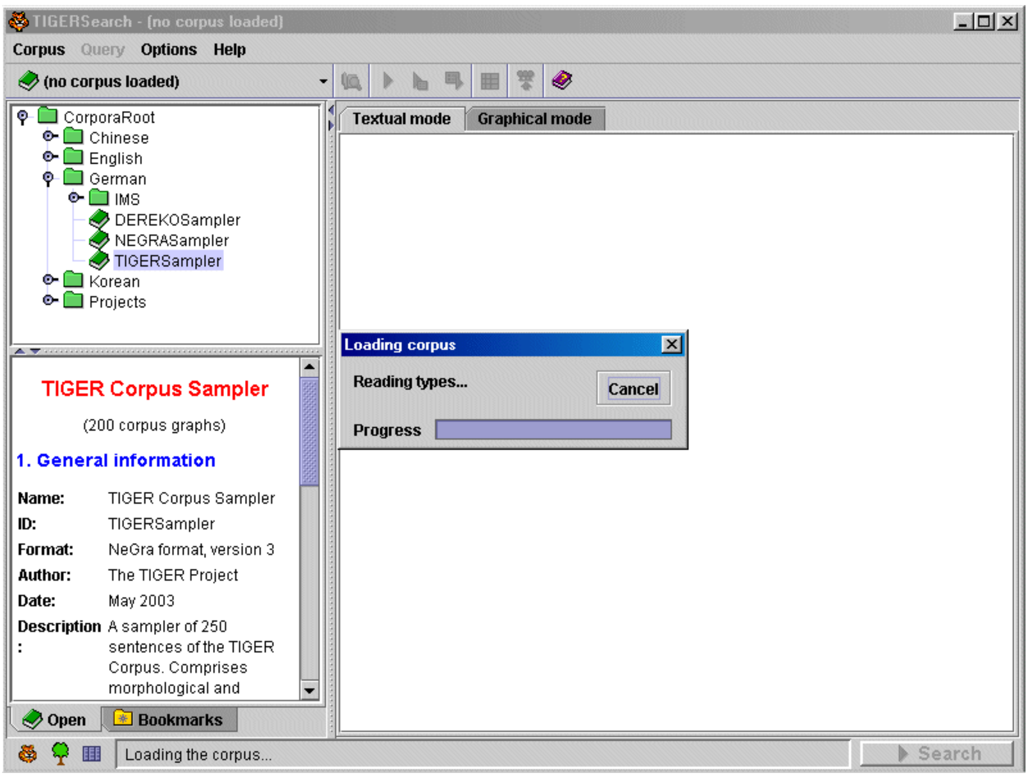
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003
Description:	A sampler of 250 sentences of the TIGER Corpus. Comprises morphological and

Welcome to TIGERSearch! Please choose a corpus.

Figure: The TIGERSearch corpus tree

Corpus loading is activated by double-clicking the respective corpus symbol, or selecting the *Open selected corpus* item in the context menu (activated by right mouse click on the corpus symbol). The corpus loading process can be aborted any time by pressing the *Cancel* button in the corpus loading progress window:



TIGER Corpus Sampler

(200 corpus graphs)

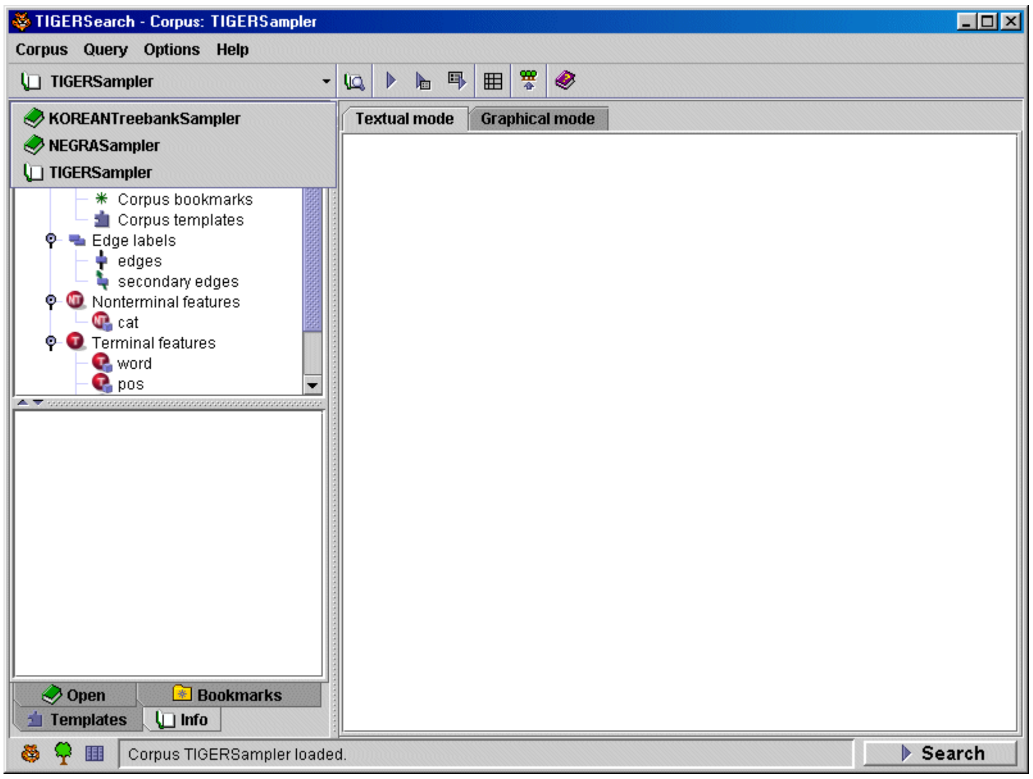
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003
Description:	A sampler of 250 sentences of the TIGER Corpus. Comprises morphological and

Figure: Corpus loading progress window

Corpus hotkey

An interesting alternative to load a corpus is the so-called corpus hotkey. This is a brief list of the corpora last opened by the user (up to 15 corpora). You can view the list by pressing the hotkey button in the upper-left corner of the main window (cf. the following screenshot). If you select a corpus in the shorthand list, the corpus loading process will be started immediately.



Corpus TIGERSampler loaded.

Figure: Corpus hotkey

(Un)Succesful corpus loading

If the corpus has been successfully loaded, there will be up to two additional corpus information tabs in the information panel which are described in the following subsections (*corpus bookmarks* and *corpus templates*). The currently loaded corpus is also indicated by the corpus hotkey and by the title bar of the main window.

If there are any problems during the corpus loading process, all warning messages are stored for inspection. These messages can be displayed by clicking on the warning symbol in the corpus information tab. In the case of corpus loading problems, please check the corpus configuration in the TIGERRegistry tool.

Corpus autoload

An interesting feature is the so-called *corpus autoload*. If this feature has been activated (cf. *Preferences* item in the *Options* menu), the corpus opened when leaving the tool will be automatically loaded when the tool is started for the next time. By default, the autoload feature is activated.

2.2 Corpus documentation

After corpus loading, the corpus information tab is activated. It includes the following pieces of information which are presented as groups within an information tree (cf. screenshots):

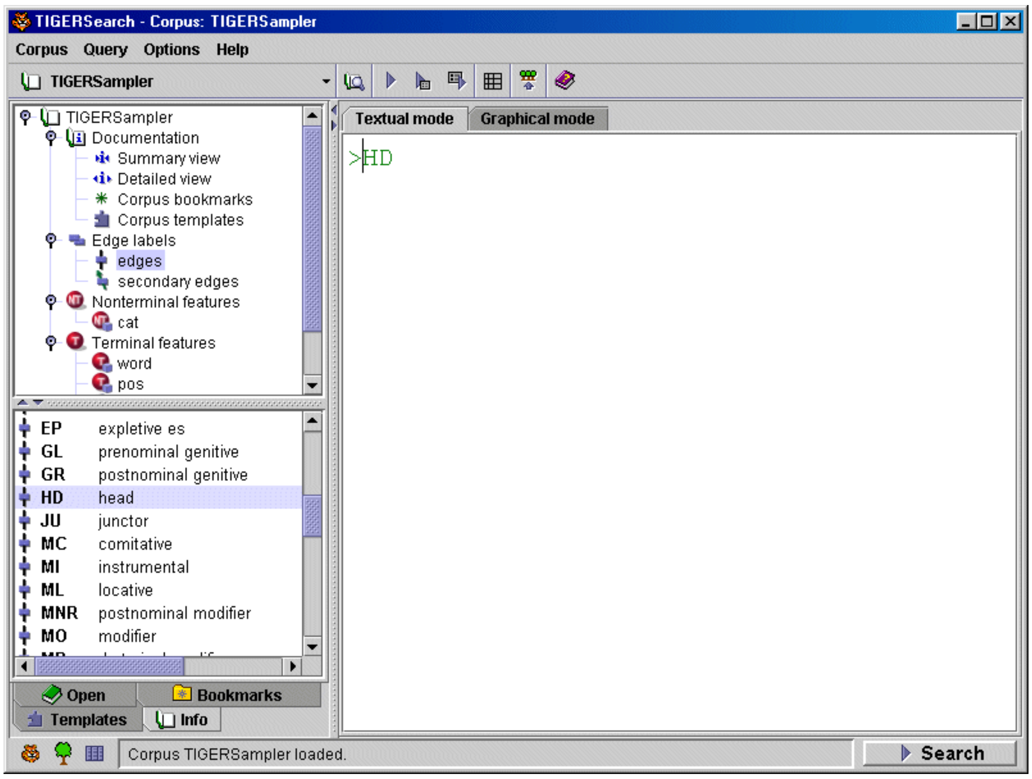
General documentation

The first group comprises general corpus documentation. Pressing one of the group icons displays the corresponding document in the lower part of the tab. The so-called *Summary view* contains some meta information about the corpus which has been specified in the corpus process (cf. subsection 4.2, chapter VI). The *Detailed view* also lists all corpus features and their corresponding feature values used in the corpus. Both information pages can be printed by selecting the *Print Current Documentation* item in the context menu which is activated by a right-button mouse click (in the lower left information panel):

If the loaded corpus comprises corpus bookmarks or corpus templates, corresponding indication icons are also placed in the documentation group (cf. screenshot above). Corpus bookmarks and corpus templates are described in subsection 2.3 and subsection 2.4, respectively.

Edge labels

The second group contains documentation about optional edge labels and secondary edge labels. All (secondary) edge labels and an optional short description are listed. If you type in a character, the cursor jumps to the first item which begins with this character. If you double-click a (secondary) edge label, it is copied into the corpus query editor:



EP	expletive es
GL	prenominal genitive
GR	postnominal genitive
HD	head
JU	junctor
MC	comitative
MI	instrumental
ML	locative
MNR	postnominal modifier
MO	modifier

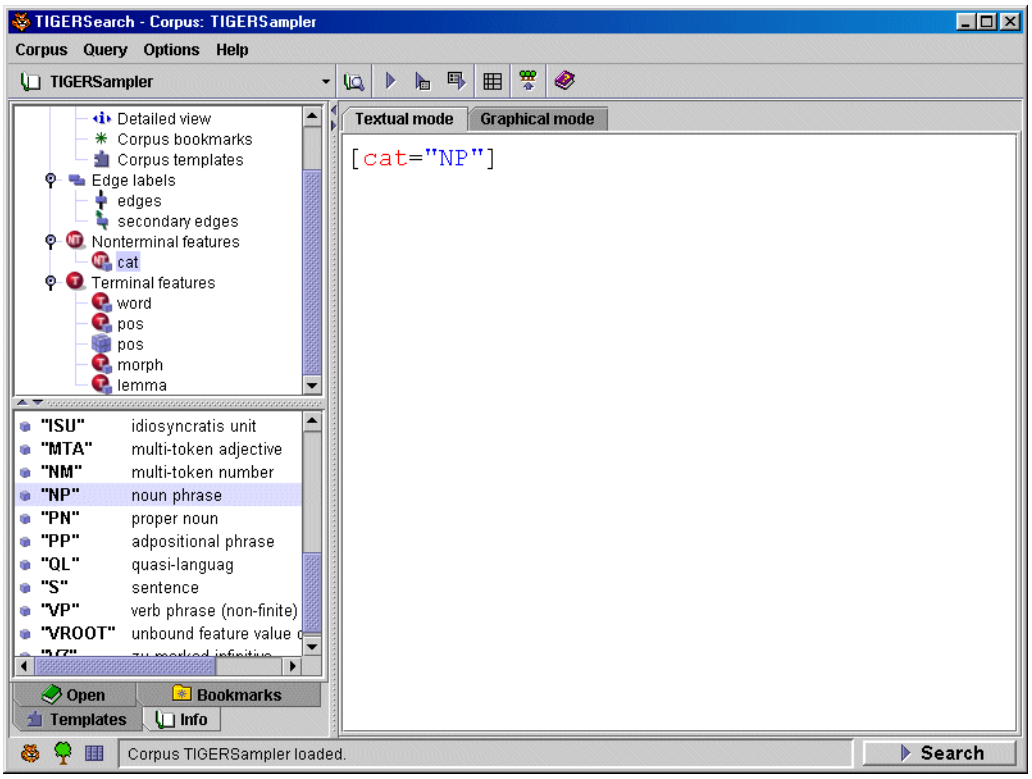
```
>HD
```

Corpus TIGERSampler loaded.

Figure: Edge labels / Secondary edge labels

Features

The third and fourth groups comprise the nonterminal and terminal features of the corpus. All feature values and an optional description are listed. If you type in a character, the cursor jumps to the first item which begins with this character. If you double-click a feature values, the corresponding feature-value pair is copied into the corpus query editor:

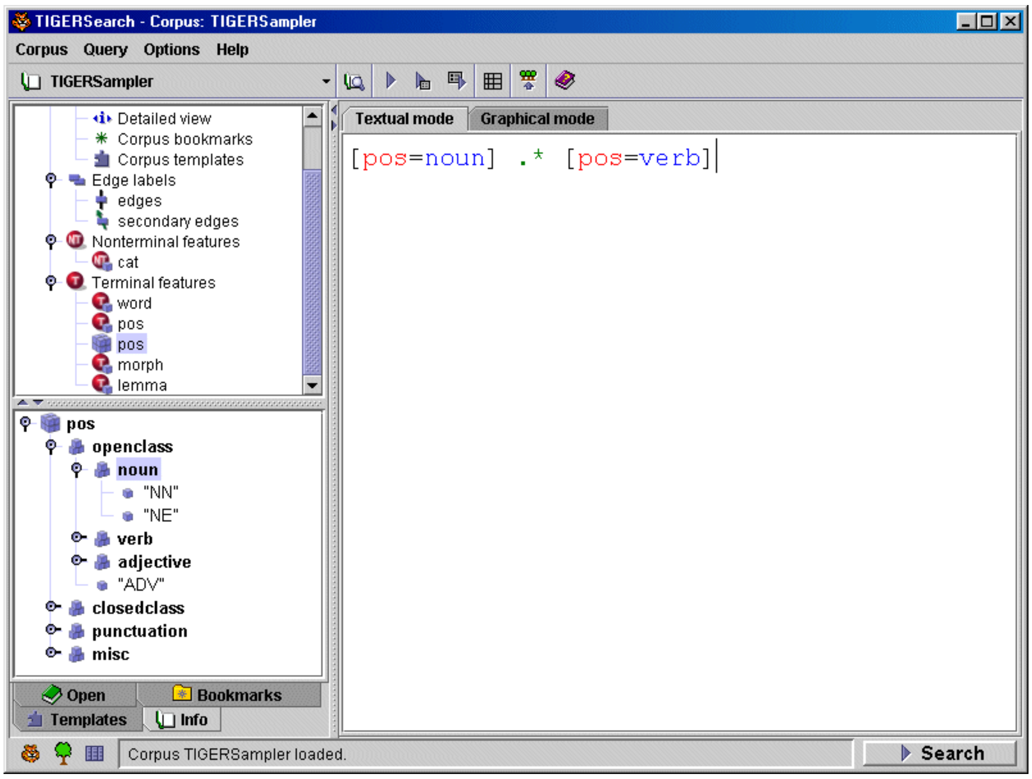


```
[cat="NP"]
```

"ISU"	idiosyncratis unit
"MTA"	multi-token adjective
"NM"	multi-token number
"NP"	noun phrase
"PN"	proper noun
"PP"	adpositional phrase
"QL"	quasi-languag
"S"	sentence
"VP"	verb phrase (non-finite)
"VROOT"	unbound feature value

Figure: Corpus features

If a type system has been defined for a corpus feature (cf. section 8, chapter III), it is also documented. Just click on the type icon which is placed under the corpus feature icon (cf. screenshot below). The type system is presented as a type tree. If you click on a type symbol or on a feature value, the corresponding feature-value / feature-type pair is copied into the corpus query editor:



```
[pos=noun] .* [pos=verb]
```

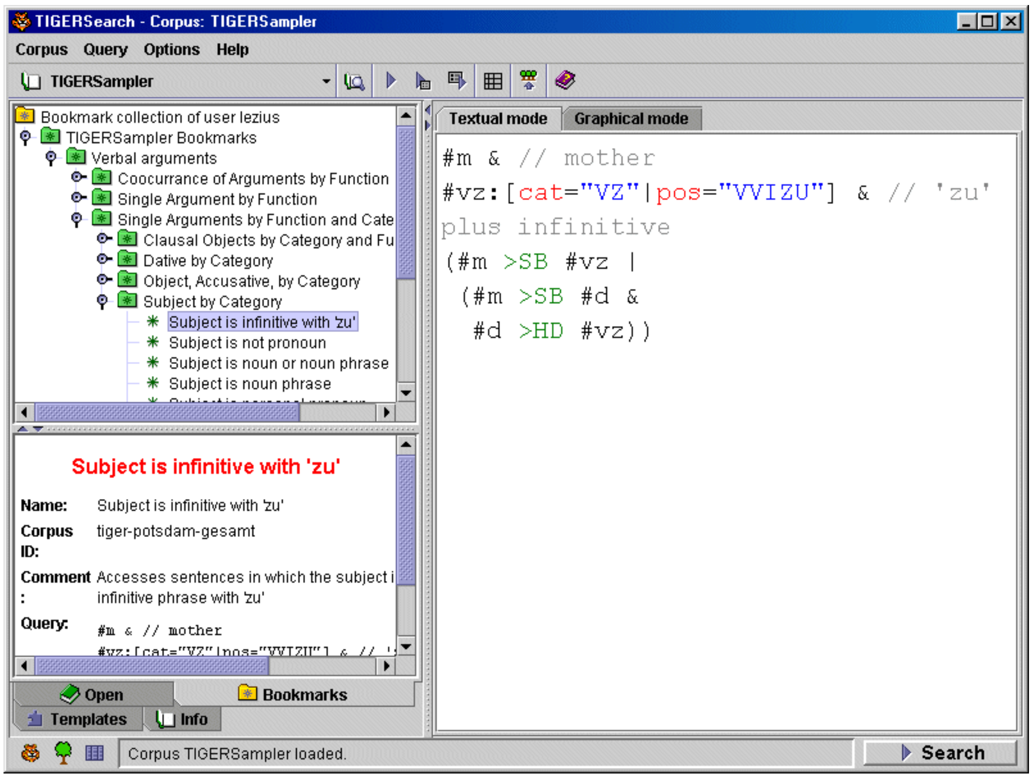
Corpus TIGERSampler loaded.

Figure: Feature types

2.3 Corpus bookmarks

The *Bookmarks* tab comprises the user's favourite queries. Queries can be saved as bookmarks in the query editor (cf. subsection 3.4). In the TIGERRegistry tool, such bookmarks can be linked to a corpus as the so-called *corpus bookmarks* (cf. subsection 4.4, chapter VI). So if the user opens a corpus, the predefined bookmarks will be available in the *Bookmarks* tab (cf. screenshot below).

In order to differentiate between corpus bookmarks and the user's bookmarks, corpus bookmarks are displayed green-colored. To see the bookmark's name, the corpus used by the bookmark query, and the bookmark query itself, press the bookmark icon. To copy a bookmark query into the query editor, just double-click the bookmark icon:



Bookmark collection of user lezius
TIGERSampler Bookmarks
Verbal arguments
Coocurrance of Arguments by Function
Single Argument by Function
Single Arguments by Function and Cate
Clausal Objects by Category and Fu
Dative by Category
Object, Accusative, by Category
Subject by Category
Subject is infinitive with 'zu'
Subject is not pronoun
Subject is noun or noun phrase
Subject is noun phrase

Subject is infinitive with 'zu'

Name: Subject is infinitive with 'zu'
Corpus ID: tiger-potsdam-gesamt
Comment: Accesses sentences in which the subject i infinitive phrase with 'zu'
Query: #m & // mother

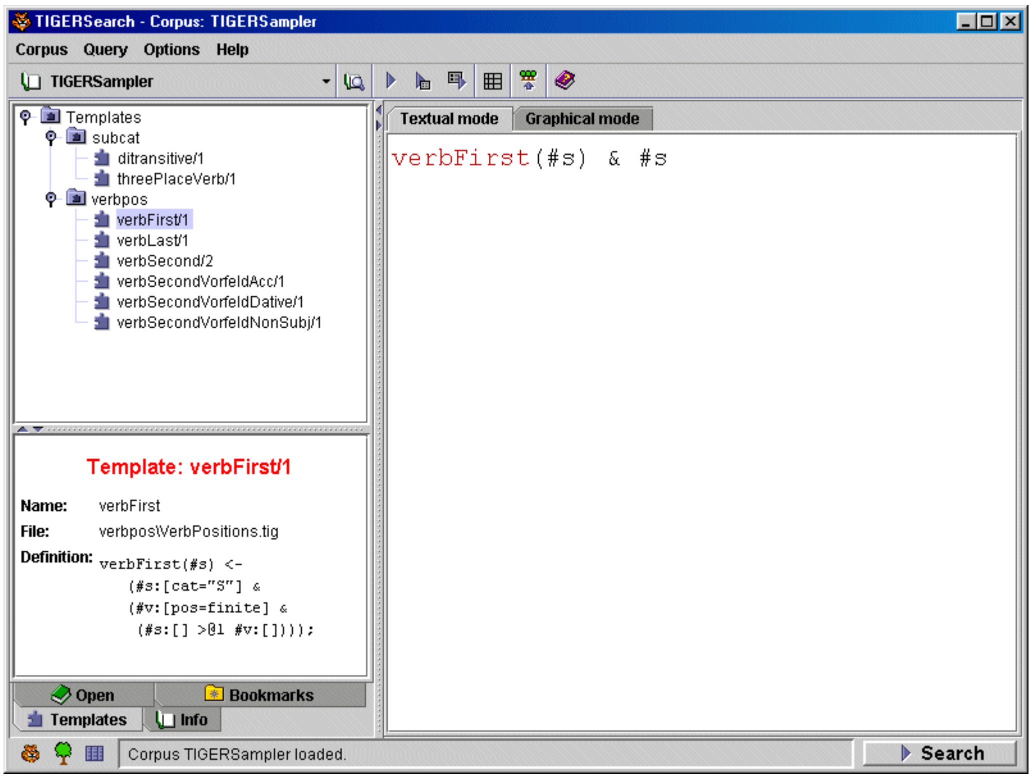
```
#m & // mother
#vz:[cat="VZ"|pos="VVIZU"] & // 'zu'
plus infinitive
(#m >SB #vz |
 (#m >SB #d &
  #d >HD #vz))
```

Corpus TIGERSampler loaded.

Figure: Corpus bookmarks

2.4 Corpus templates

If templates have been declared for the corpus, they are presented in the *Templates* tab. In the upper part of the tab, all templates are presented. If you press a template icon, the corresponding template name, template path, and template definition are displayed. To copy the template call into the query editor, just double-click the template icon:



```
verbFirst(#s) & #s
```

Template: verbFirst/1

Name: verbFirst

File: verbpos\VerbPositions.tig

Definition:

```
verbFirst(#s) <-
    (#s:[cat="S"] &
    (#v:[pos=finite] &
     (#s:[] >@l #v:[])));
```

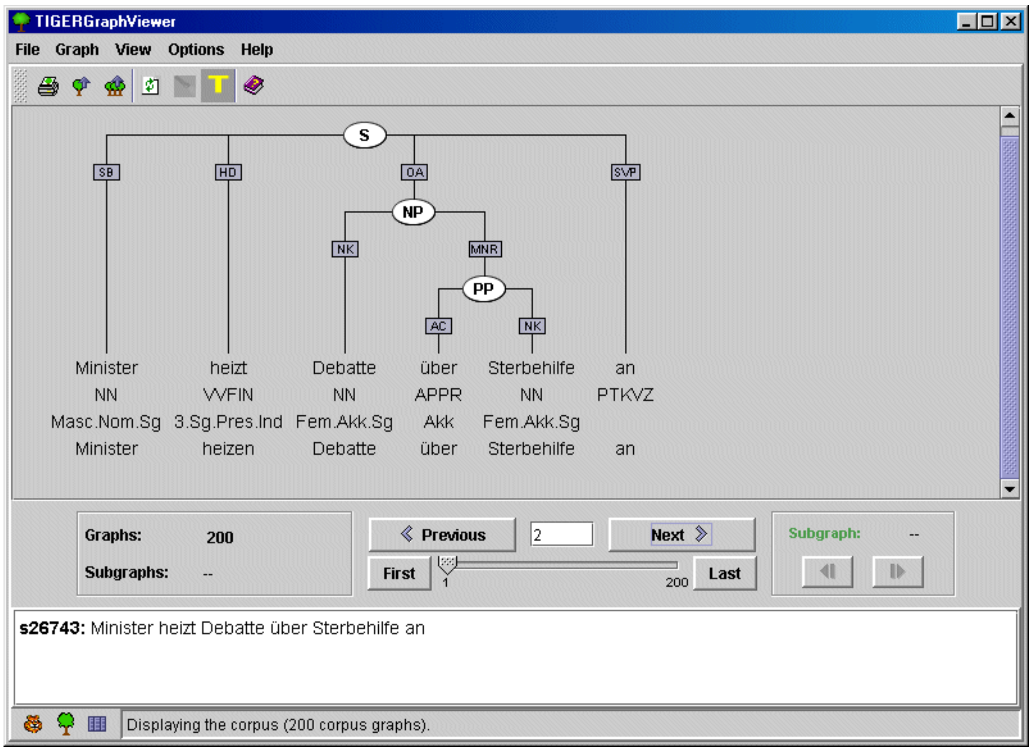
Corpus TIGERSampler loaded.

Figure: Corpus templates

2.5 Exploring the corpus

After corpus loading, the corpus exploration button in the toolbar (leftmost button, right neighbour of the corpus hotkey) is activated. Pressing this button will open the TIGER-GraphViewer which visualizes the corpus graphs (cf. section 7). So you can browse through the corpus without processing corpus queries. This feature is very helpful if you are not familiar with a corpus and its annotations.

⚠**Please note:** You can easily switch between the TIGERSearch main window and the GraphViewer window using the shortcut buttons in the lower left corner of both windows. If you press a shortcut button, the corresponding window will be moved in front of all other windows on your desktop.



s26743: Minister heizt Debatte über Sterbehilfe an

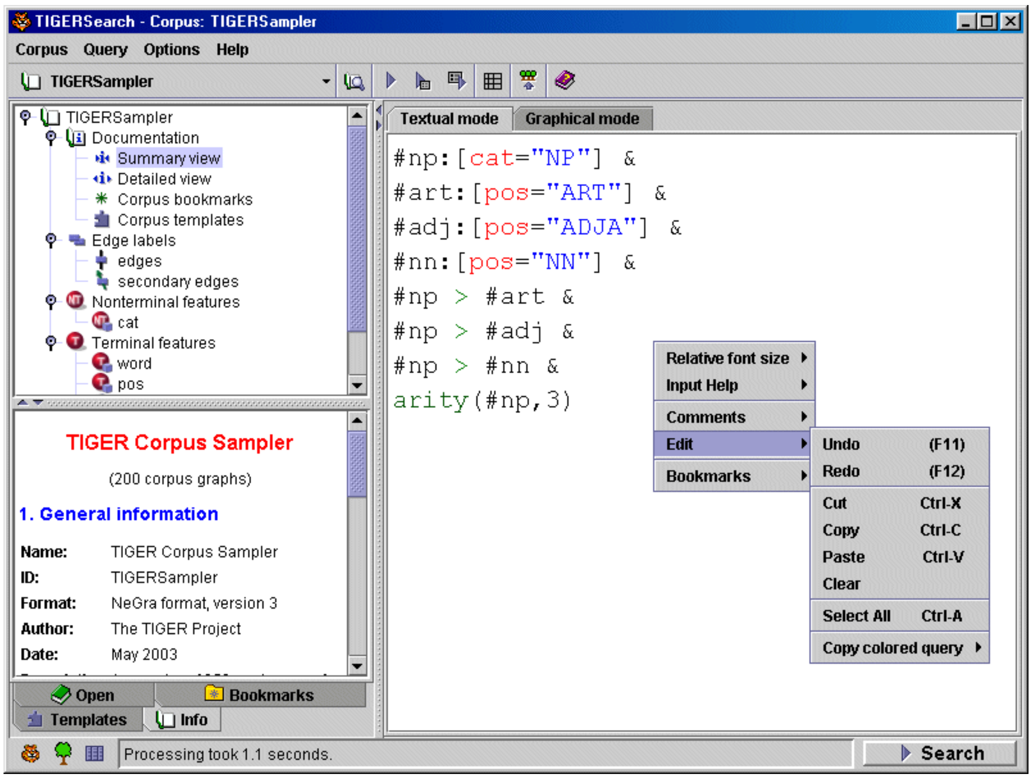
Displaying the corpus (200 corpus graphs).

Figure: Exploration of the corpus

3. Textual query editor

3.1 Query editor basics

The corpus query editor is the central part of the TIGERSearch tool. For a description of the query language please see chapter III. The query text editor is quite sophisticated, i.e. its features include query syntax highlighting and copy and paste functionality to exchange query texts with other applications (cf. *Edit* menu in the context menu). Using the *Undo* and *Redo* items you can restore corpus queries that have already been submitted.



```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
#np > #art &
#np > #adj &
#np > #nn &
arity(#np,3)
```

TIGER Corpus Sampler

(200 corpus graphs)

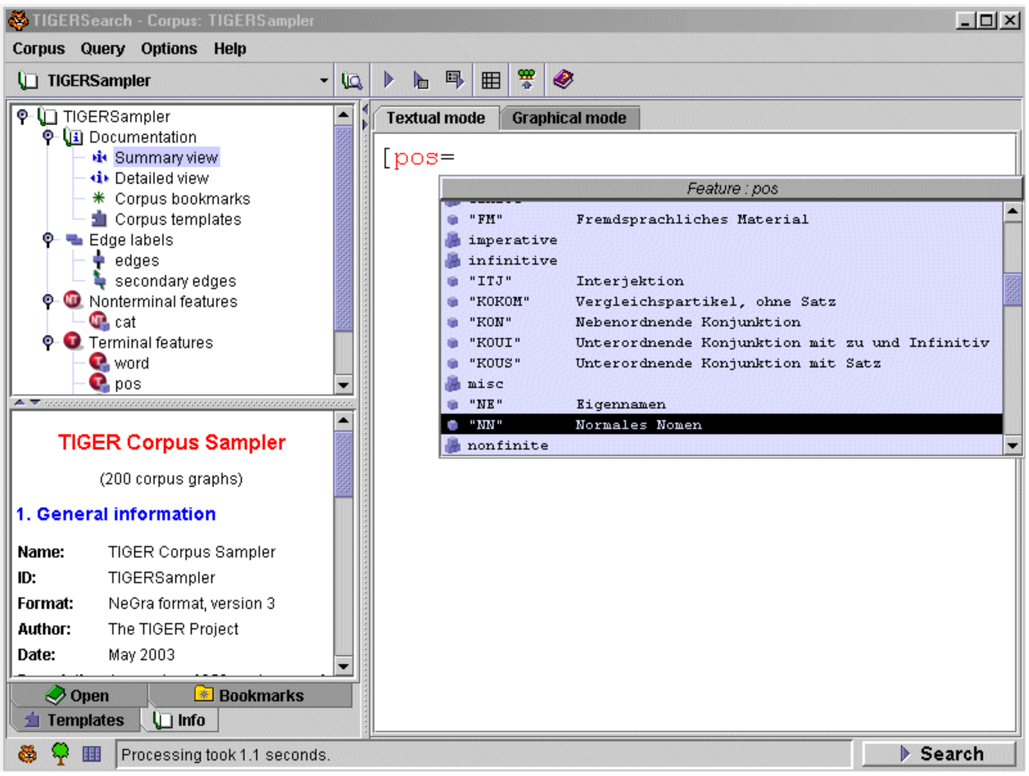
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: The corpus query editor

A helpful input help feature is the so-called feature popup window. If you type in a feature name followed by the equality character (e.g. `pos=`), the feature popup window appears. It comprises all feature values and types declared for this feature. You can browse through this list by using the cursor arrows or the page up and page down keys. If you press a character key, the cursor moves to the first item which begins with the corresponding characters:



```
[pos=
```

"FM"	Fremdsprachliches Material
imperative	
infinitive	
"ITJ"	Interjektion
"KOKOM"	Vergleichspartikel, ohne Satz
"KON"	Nebenordnende Konjunktion
"KOUI"	Unterordnende Konjunktion mit zu und Infinitiv
"KOUS"	Unterordnende Konjunktion mit Satz
misc	
"NE"	Eigennamen
"NN"	Normales Nomen
nonfinite	

TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

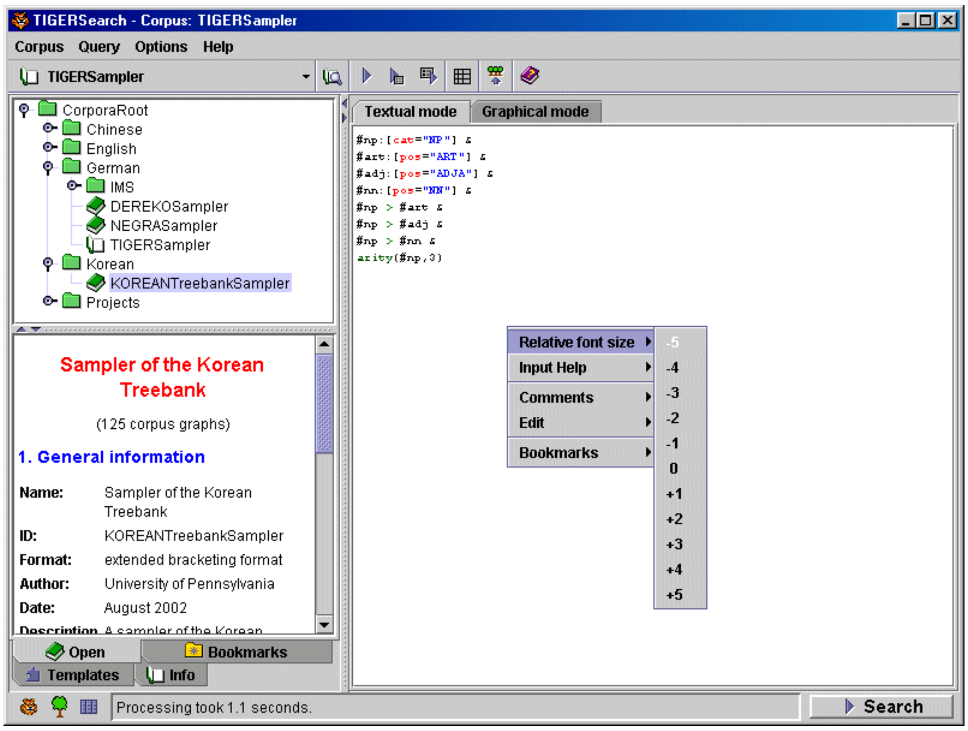
Processing took 1.1 seconds.

Figure: The feature pop-up window

If you select an item (by double-clicking or pressing the return key), it is copied into the query text editor. The feature popup window can be enabled / disabled in the *Input Help* menu of the context menu.

3.2 Advanced query editor features

To change the editor's font size, increase or decrease the relative font size from -5 to +5 in the editor's context menu (right mouse click in the editor, item *relative font size* in the context menu):



```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
#np > #art &
#np > #adj &
#np > #nn &
arity(#np,3)
```

Sampler of the Korean Treebank

(125 corpus graphs)

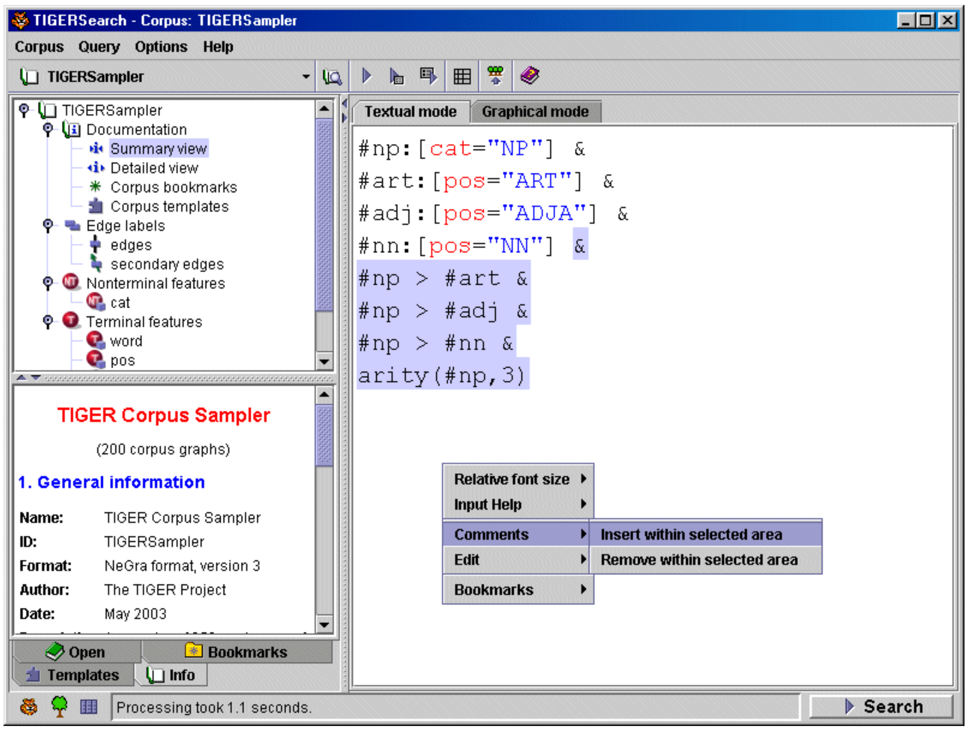
1. General information

Name:	Sampler of the Korean Treebank
ID:	KOREANTreebankSampler
Format:	extended bracketing format
Author:	University of Pennsylvania
Date:	August 2002
Description	A sampler of the Korean

Processing took 1.1 seconds.

Figure: Changing the editor's font size

To comment or uncomment lines in the query text, just mark the corresponding text area. Afterwards select the *Insert within selected area* or *Remove within selected area* option in the *Comments* menu of the context menu, respectively:



```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
#np > #art &
#np > #adj &
#np > #nn &
arity(#np,3)
```

TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: (Un)commenting query text

If you want to include a corpus query into an electronic document, you can use the *Copy* item of the *Edit* menu. However, the syntax highlighting of the query will be ignored. For this purpose we have implemented the *Copy colored query* feature in the *Edit* menu. You can copy the colored query text as an HTML or LaTeX fragment. Afterwards you can paste the clipboard content into your HTML or LaTeX editor.

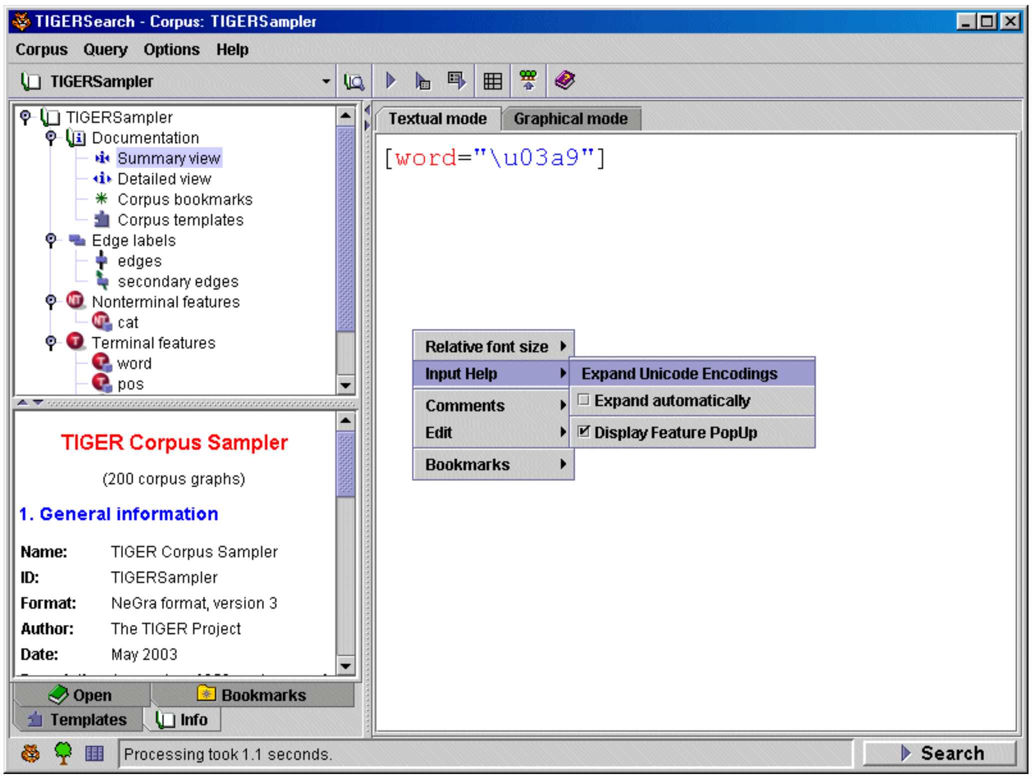
3.3 Internationalization of the query editor

A common problem for applications such as TIGERSearch is the keyboard input of characters which are not included in the ISO-Latin-1 character set. If you are working with a corpus that makes uses of such characters, you should consider the following three alternatives:

Please note: Typing in Unicode characters implies that Unicode charaters can be displayed (rendered) by the software. Thus, one of the Unicode fonts supported by TIGERSearch must have been installed on your system. Please consult section 3, chapter II for instructions.

Unicode encoding

The first alternative to encode a Unicode character is to type in its hexadecimal Unicode encoding. For example, the Greek capital letter Omega is represented by `\u03a9`. If you have typed in the Unicode encoding, just select the *Expand Unicode Encodings* option in the *Input Help* menu of the context menu to expand the character:



```
[word="\u03a9"]
```

TIGER Corpus Sampler

(200 corpus graphs)

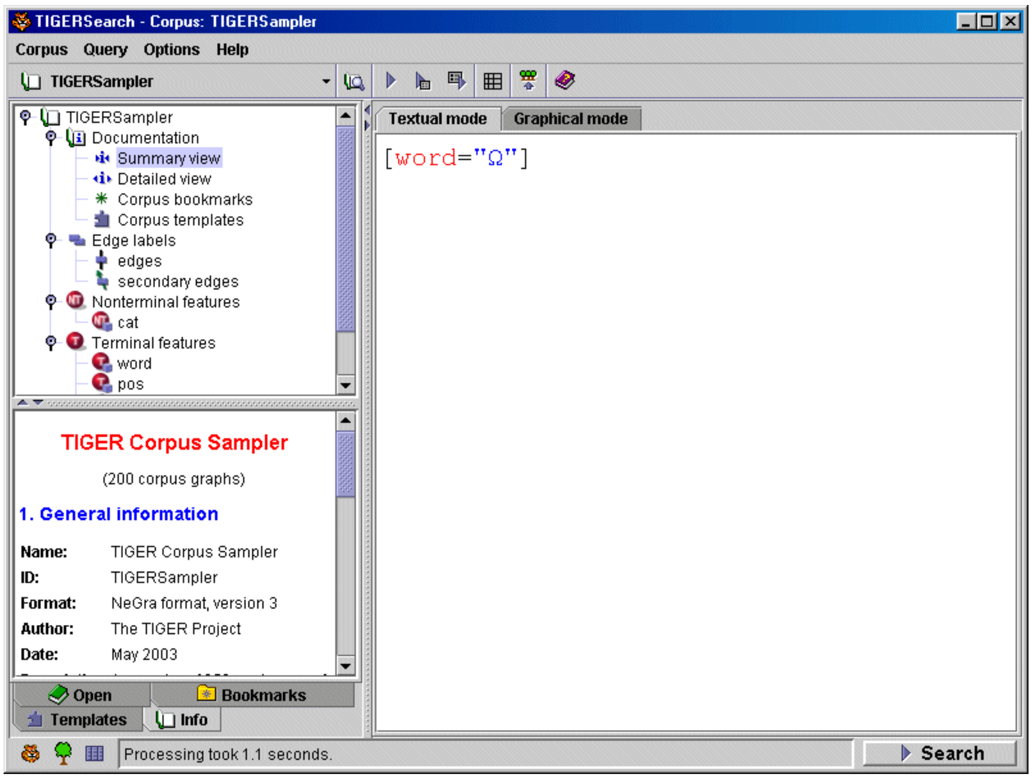
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: Expanding Unicode encodings

The Unicode encoding will be replaced by its corresponding character (cf. screenshot below). Please remind that a Unicode font must be installed to render the character properly.



```
[word="Ω"]
```

TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: Expansion of Unicode encodings

If you are frequently working with corpora using characters outside the ISO-Latin-1 character set, you should activate the *Expand automatically* option in the *Input Help* menu of the context menu.

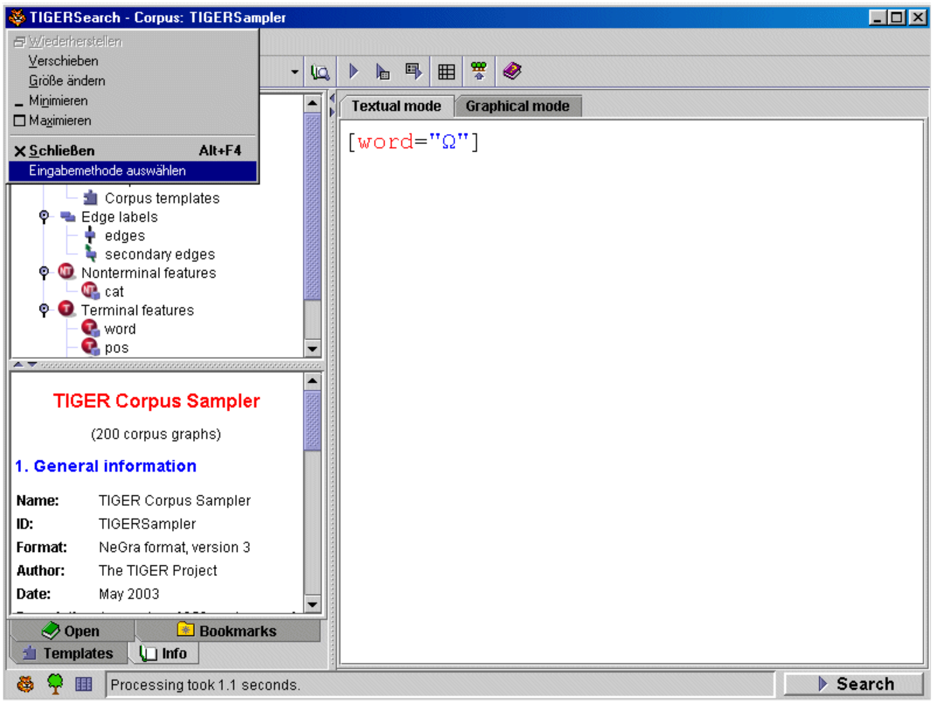
Input help (operating system)

On many platforms, specialized tools have been developed to type in characters outside the ISO-Latin-1 character set. These tools are usually called *input methods*. As e.g. Greek characters do not exist on a German keyboard, these charaters are typed in as an abbreviation. For example, the string *Omega* might be used as an abbreviation for the Greek character that will be automatically expanded if the abbreviation has been typed in. Please consult the manual of your operating system to find out which tools are available for your platform.

Input help (TIGERSearch)

In the TIGERSeach Project we have implemented specialized input methods for 16 European languages which can be used in the TIGERSearch query editor (cf. subsection 3.2, chapter II). To activate the TIGERSearch input methods, press the upper left corner of the TIGERSearch window (Windows: press the tiger icon) and select the last option in the corresponding menu (usually called *Choose input method*).

The following screenshot shows how the input method is activated on a German Windows platform. The display will look similar on different platforms.



```
[word="Ω"]
```

TIGER Corpus Sampler

(200 corpus graphs)

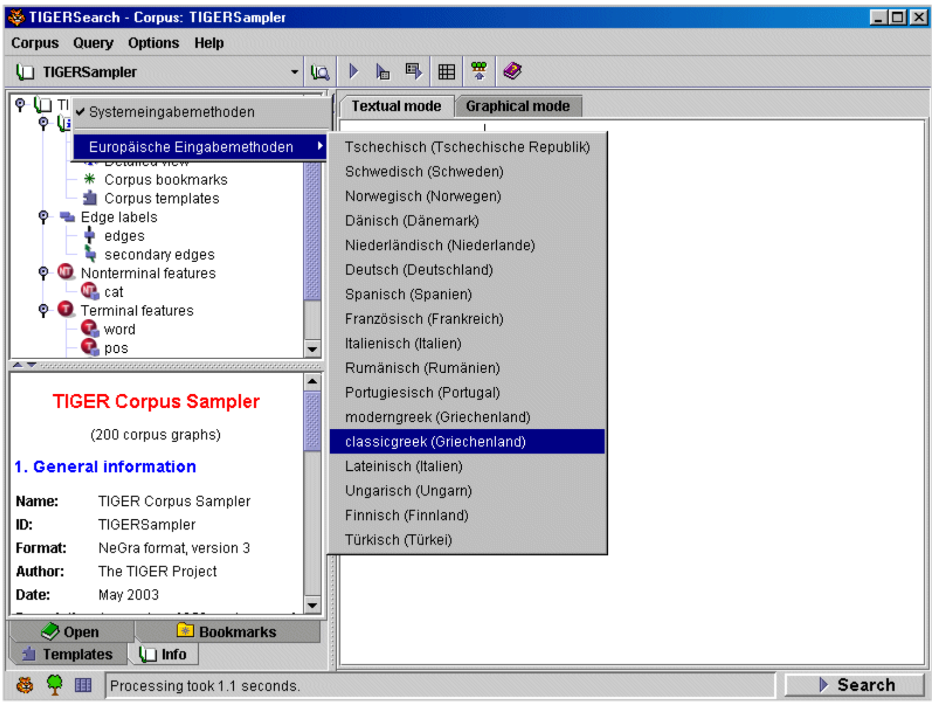
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: Activating the TIGERSearch input methods (1)

Now you are asked to choose one of the supported European languages. In the following screenshot, the Greek language (modern) is chosen:



TIGER Corpus Sampler

(200 corpus graphs)

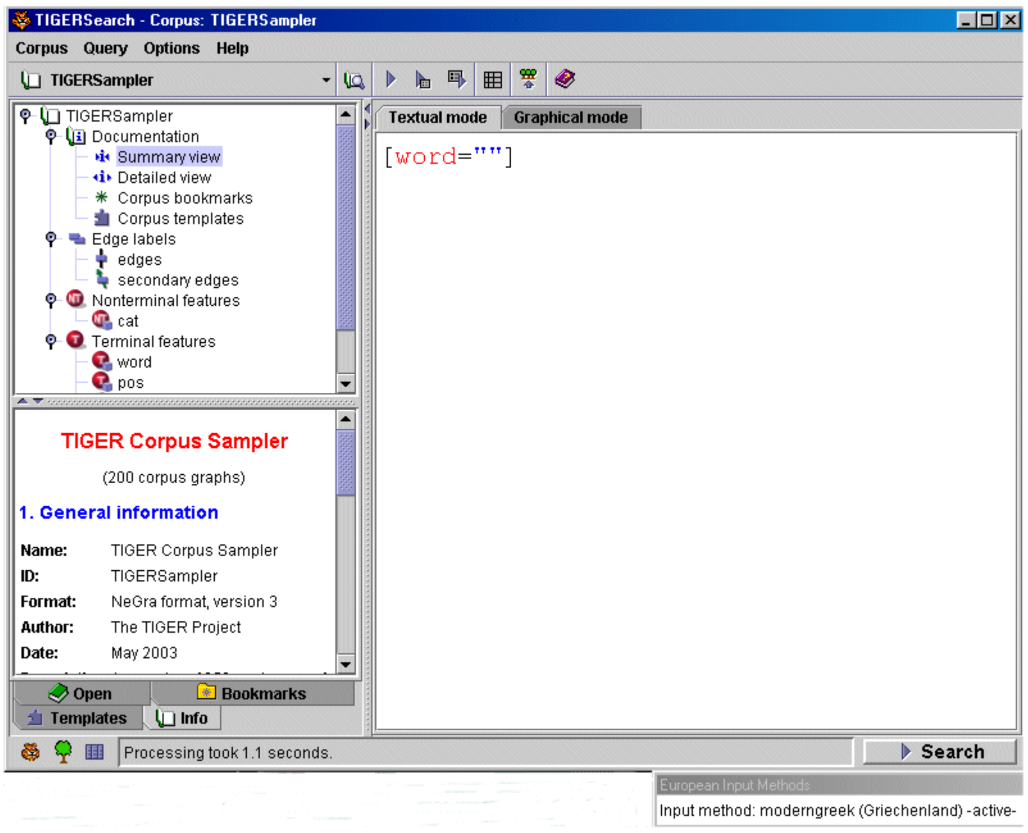
1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: Activating the TIGERSearch input methods (2)

The input method mode has been activated. A small status window is placed in the lower right corner of the screen. This window shows which language has been chosen and whether the input method is activated or deactivated:



```
[word=""]
```

TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Input method: moderngreek (Griechenland) -active-

Figure: Input method status window

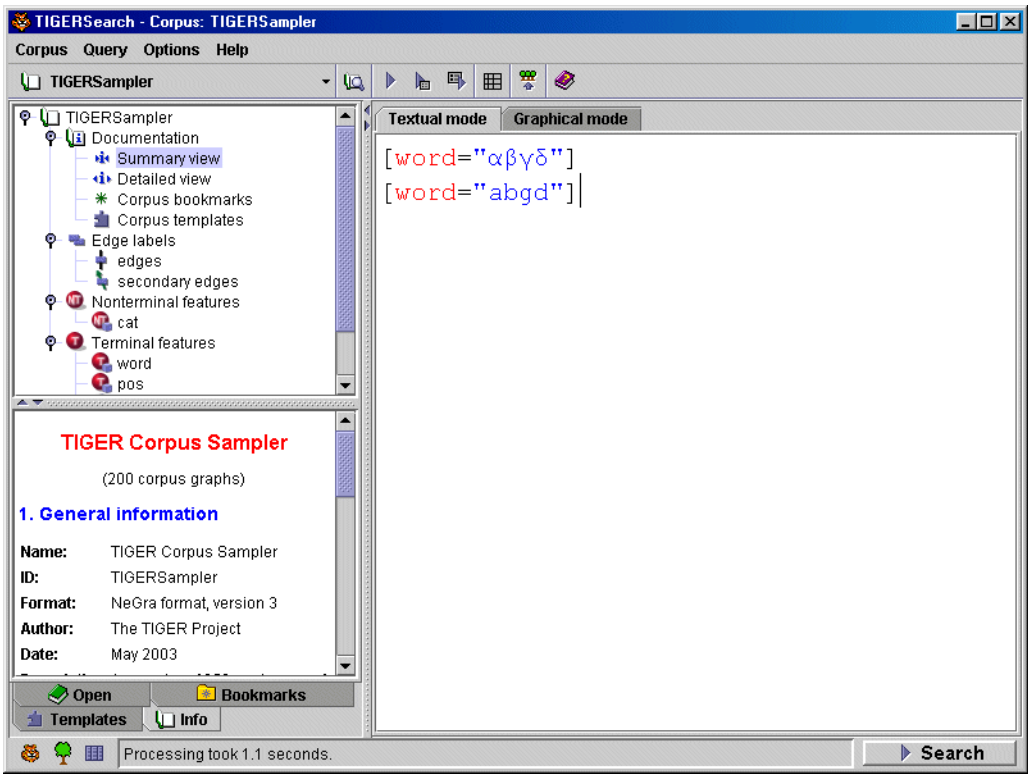
To select a different language, you can either process the input method activation procedure described before or you can switch between the languages using the `F7` key. To activate or deactivate the current input method please use the `F8` key.

⚠**Please note:** To deactive the TIGERSearch input methods (especially to deactivate the input method status window), start the input method selection procedure again, but choose `system input methods` in the input method menu.

How is the input method used in the query editor? All characters that are not included in the ISO-Latin-1 character set are represented by special abbreviations. To allow the input of the Latin characters as well as the special characters side by side in one mode, we have chosen encodings conventions used in the LaTeX system. For example, the German character `ä` is represented as `\"a` which is its LaTeX encoding. So if you have chosen the German keyboard mapper and you type in the character sequence `\"a`, it will be automatically expanded to `ä` by the TIGERSearch input help system.

⚠**Please note:** Of course, all German characters are included in the ISO-Latin-1 character set. However, German special characters (ä,ö,ü,ß) can only be typed in on keyboards manufactured for the German market. Otherwise, an input method for the German language is necessary in order to work with German treebanks such as the TIGER treebank.

For languages such as Greek which comprises many special characters, a side by side usage of Latin and Greek characters is not possible. In this case, most Greek characters are represented by Latin characters. For example, the capital letter Omega is represented by the Latin character V. So if you type in V in the query editor, this input string is automatically expanded as the capital letter Omega. The following screenshot illustrates how Greek characters are typed in:



```
[word="αβγδ"]
[word="abgd"]
```

TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003

Processing took 1.1 seconds.

Figure: Typing in Greek characters

The mapping tables for the 16 supported European languages can be found in the file `europe.pdf` which is placed in the `doc/pdf/` subdirectory of your TIGERSearch installation. It can also be downloaded from the TIGERSearch homepage (cf. *http://www.tigersearch.de*).

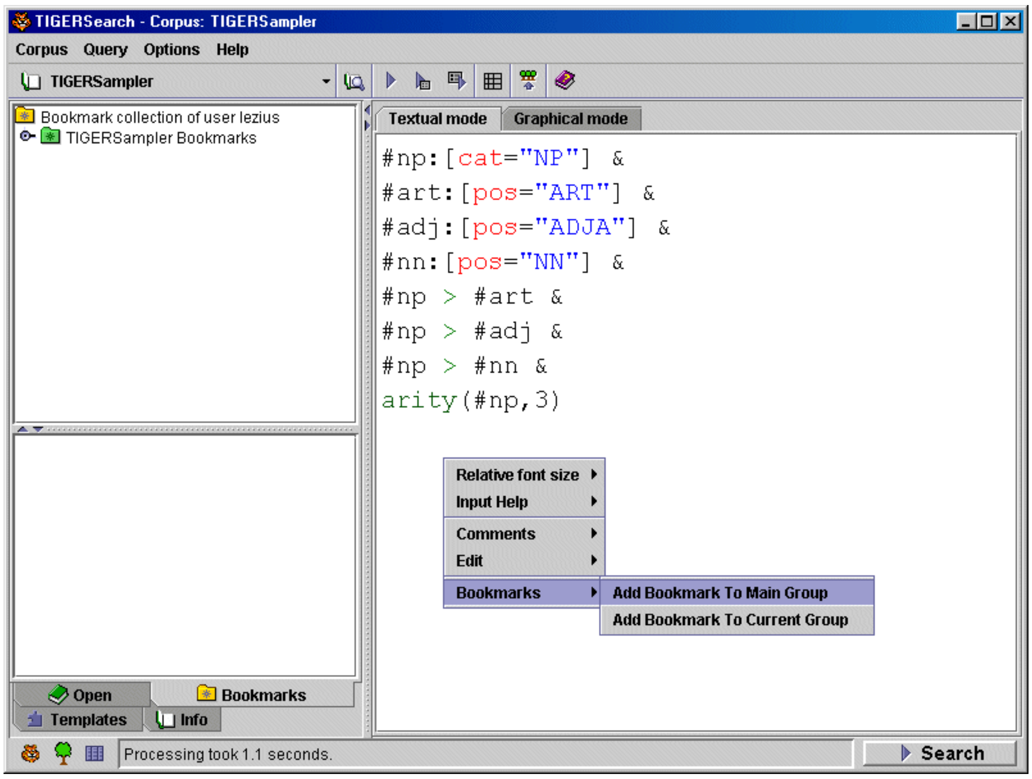
3.4 Bookmarks

TIGERSearch provides a bookmark concept to store your favourite corpus queries. As an option, you can also store the results of your queries. This makes sense especially for queries which took a long time to evaluate. The present subsection describes how to add and open bookmarks and the concept of bookmark maintenance.

Adding a bookmark

If you would like to file a bookmark, first mark the preferred bookmark parent folder in the

Bookmarks tab (cf. screenshot). Afterwards, select the *Add Bookmark to Main Group* item in the *Bookmarks* menu in the context menu of the query editor. If you do not want to specify the parent folder, just select the *Add Bookmark to Main Group* option.

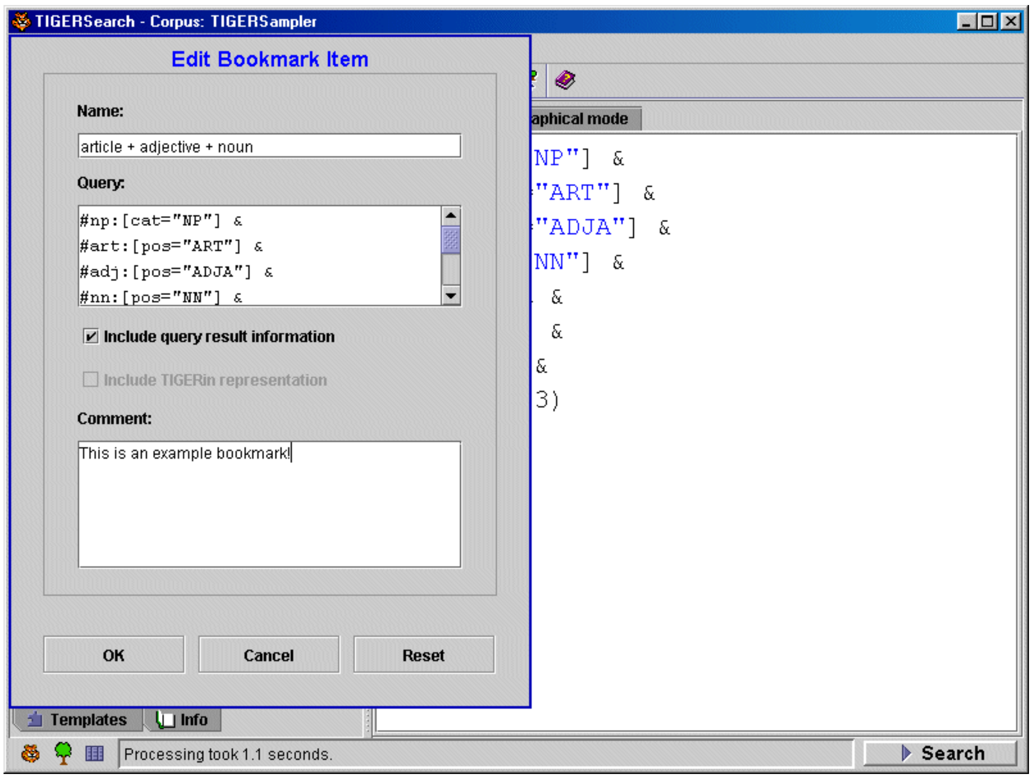


```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
#np > #art &
#np > #adj &
#np > #nn &
arity(#np,3)
```

Processing took 1.1 seconds.

Figure: Adding a bookmark

Next, the bookmark properties window is presented. Here you can specify the name of the bookmark and a comment that describes the bookmark. If the query has already been processed, the query results can also be stored. To confirm the bookmark properties and insert the bookmark in the specified parent folder, press the *OK* button:



Name:
article + adjective + noun

Query:
```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
```

[✓] Include query result information

[ ] Include TIGERin representation

Comment:
This is an example bookmark!

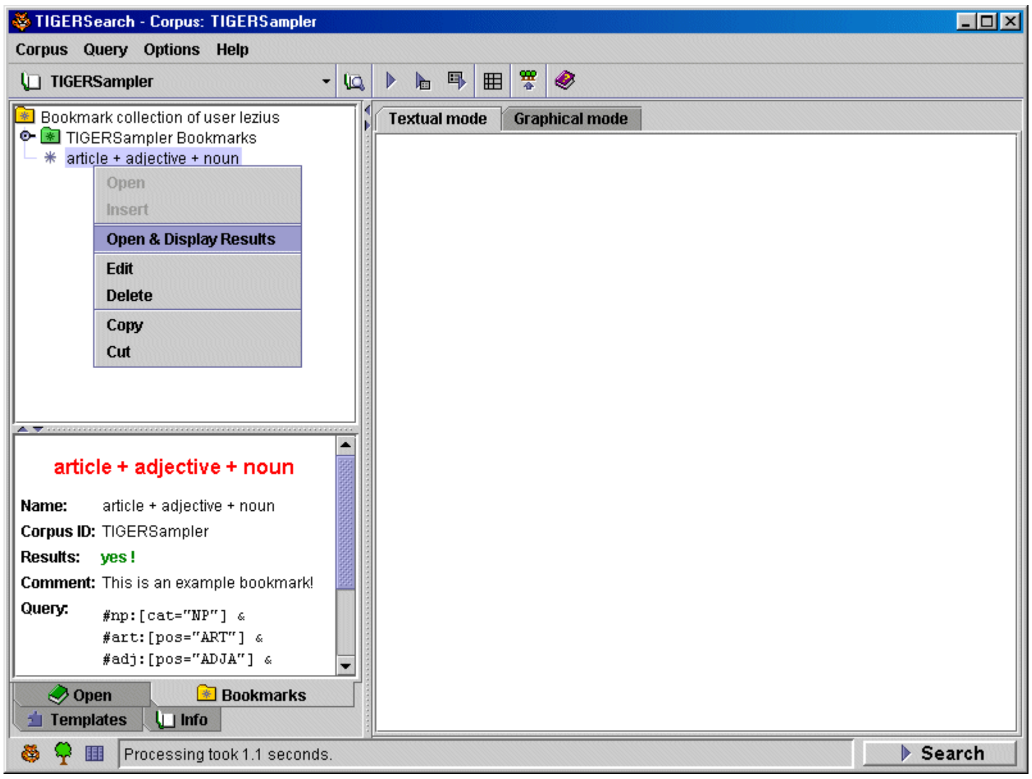
Processing took 1.1 seconds.

Figure: Bookmark properties

Please note: Storing query results is a helpful feature, but it consumes hard disc memory in the user's home account. If you do not plan to reuse the results, you should better do without it.

Opening a bookmark

The user's bookmarks are presented in the *Bookmarks* tab in the corpus information panel. If you press a bookmark icon, the bookmark name and definition are shown in the lower part of the tab. If you double-click on a bookmark or select the *Open* item in the bookmark's context menu, the query text of the bookmark is copied into the query editor. If query results are also stored, they are restored and can be inspected in the GraphViewer.



article + adjective + noun

Name: article + adjective + noun
Corpus ID: TIGERSampler
Results: yes !
Comment: This is an example bookmark!
Query:

```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
```

Processing took 1.1 seconds.

Figure: Opening a bookmark

Bookmark maintenance

The bookmark maintenance is realized by context menus for bookmarks and bookmark folders (right mouse click on a bookmark folder or item, respectively). You can edit bookmark properties, delete a bookmark, cut a bookmark, and paste a bookmark which has been copied or cut before. You can also edit the properties of bookmark folders, cut/copy and paste a folder, and add a new subfolder or bookmark. In order to deallocate hard disc memory used by query results, users can mark a bookmark folder and select the *Delete Results Information* item in the context menu to delete the query results of all bookmark queries which are placed under this folder or any of its subfolders.

⚠**Please note:** Corpus bookmarks cannot be deleted. If you like to use corpus bookmarks as a basis for defining derived queries, you might duplicate them by using the copy and paste feature.

In order to exchange bookmarks with other users, you can import and export bookmarks. Just mark the parent folder of the preferred bookmarks (e.g. the root folder to export all bookmarks) and select the *Export as Bookmark File* item in the context menu.

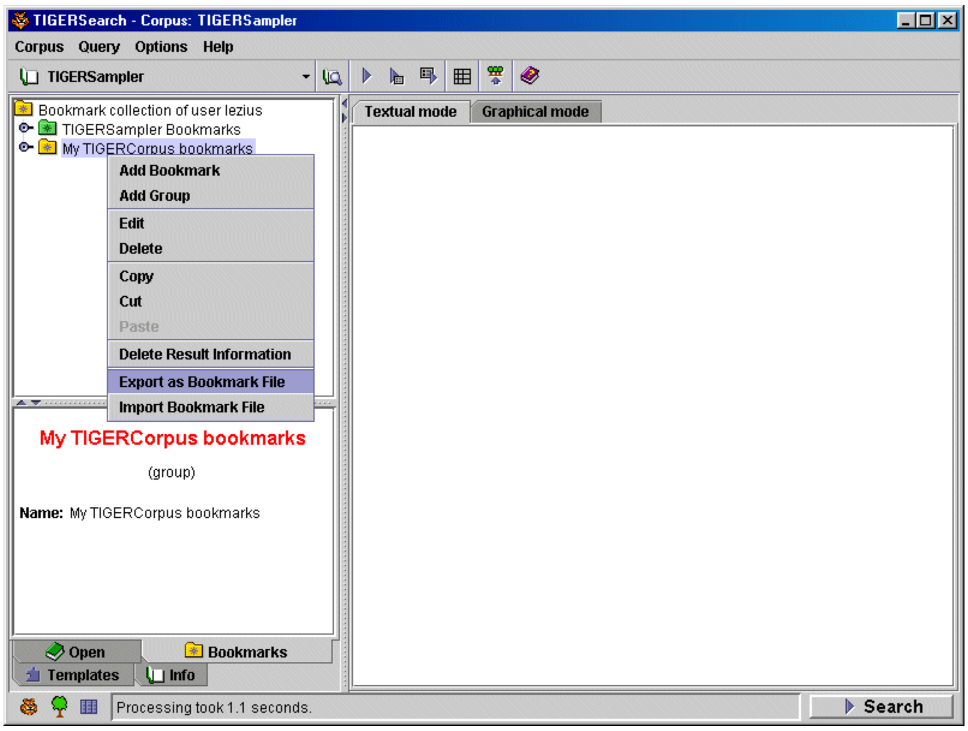


Figure: Exporting bookmarks

Now the marked bookmark folder is saved. This file can be imported by other users using the *Import bookmarks file* item in the context menu of the preferred parent folder. A bookmark file can also be used as a corpus bookmarks file (cf. subsection 4.4, chapter VI).

4. Graphical query editor

4.1 Introduction

The graphical query editor

The graphical query editor can be used to create queries without knowledge of the TIGERSearch query language. For example, you can create nodes with a few mouse clicks, and change values by selecting an item from the list of a pull down menu. You can also create edges between nodes, which represent dominance and precedence relations.

The graphical query editor is the result of Holger Voormann's diploma thesis ([Voormann2002]; in German).

An alternative to the textual mode

The graphical editor has been designed for beginners as well as for casual users. Power users will likely create queries faster textually than graphically. It is also possible to start constructing a query graphically, automatically generate the textual equivalent, and make some further textual edits.

Graphical vs. textual queries

Because the graphical mode has also been designed for people who want to learn the TIGERSearch query language (cf. chapter III), the graphical construction of a query follows the concept of the textual query language. For example, nodes are specified with an arbitrary number of feature-value pairs.

The graphical query editor as a preprocessor

To process a graphical query, it is necessary to convert it to a textual representation. You can see the textual mode as a plugin, which is converting your graphical query to the TIGERSearch query language (see figure below).

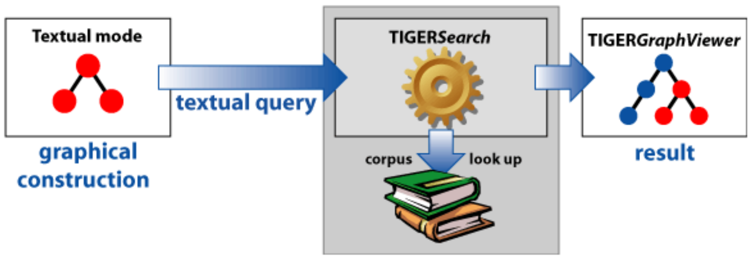


Figure: Before query processing, the graphical construction is converted to a textual query.

Limitations

Because of the architecture (see paragraph above), the graphical mode may, in principle, have the power of the TIGERSearch query language. However, in the current version there are some limitations:

- The graphical editor supports disjunctions between feature-value pairs and feature values, but not between graph relations or any grouped parts of a query (cf. section 7, chapter III).
- It is not possible to create an unspecified node (cf. subsection 4.3).
- Variables, used for equality of node specifications, are not yet supported (cf. subsection 7.2, chapter III).
- Templates are not yet supported (cf. section 9, chapter III).

4.2 Starting

Switch to *Graphical Mode*

To start creating a new graphical query you first have to select the *Graphical mode* tab instead of the *Textual mode* at the top of the query input area (see screenshot):

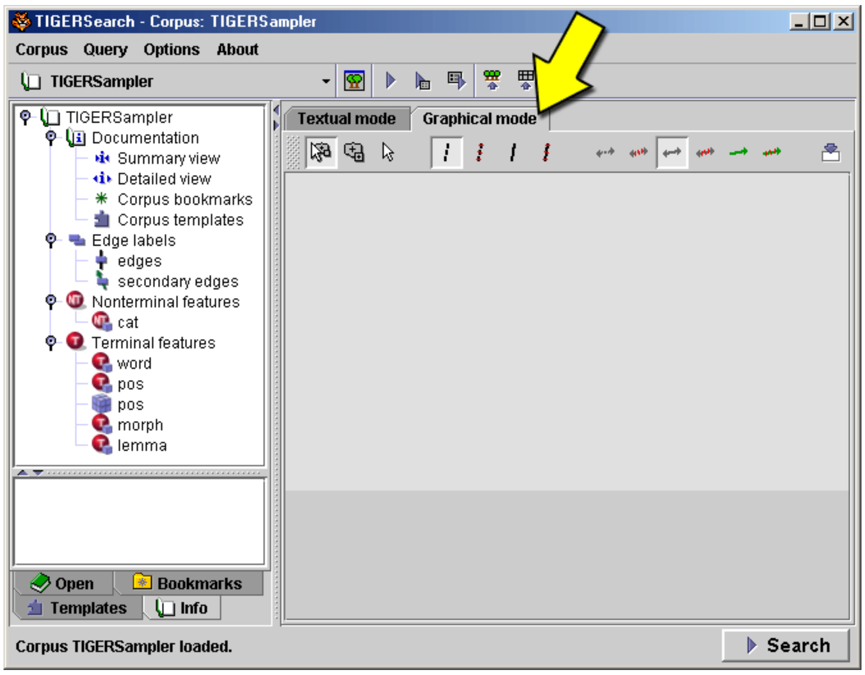


Figure: Switch to the Graphical Mode by clicking on the tab.

Before a graphical query can be created, a corpus must be loaded (cf. section 2).

⚠**Please note:** If the corpus is closed or reloaded or another corpus is opened, the current graphical query will be lost.

Creating a graphical query

To create a graphical query, you first have to create nodes (cf. subsection 4.3). In the second step you might specify these created nodes (cf. subsection 4.4) or create edges between existing nodes (cf. subsection 4.5).

4.3 Nodes

Input field

After switching to the *Graphical mode* you can see the horizontally divided input field. Be sure that either the *Move/Create* tool or the *Create* tool is activated (cf. subsection 4.6). By clicking into the upper part, *phrase nodes* (so-called *inner nodes* or *nonterminal nodes*) will be created. A left button mouse click into the bottom area creates a *token* (so-called *leaf nodes* or *terminal node*).

⚠**Please note:** Because of the divided input field either phrase nodes or token nodes can be created. An unspecified node cannot be created.

The different parts of a node

The different parts of a node are separated by small lines. You can disable these lines (so-called plug borders) via the context menu (cf. subsection 4.7).

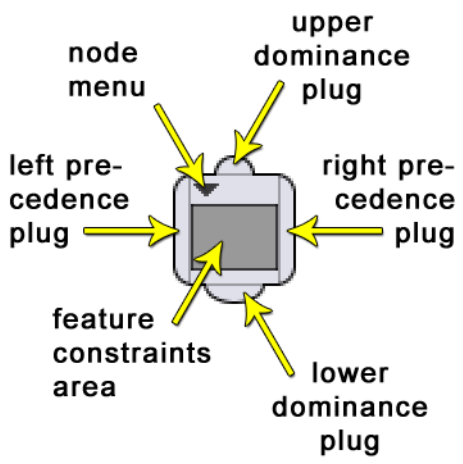


Figure: A phrase node with its four plugs, the node menu, and the feature constraints area.

The four (token nodes: three) plugs are used to create edges (cf. subsection 4.5). The darker inner area contains the node specification (cf. subsection 4.4).

Node menu

Every node has a pull down menu in its upper left corner. Use this menu to delete the node and to enable or disable graph predicates (cf. subsection 7.3, chapter III).

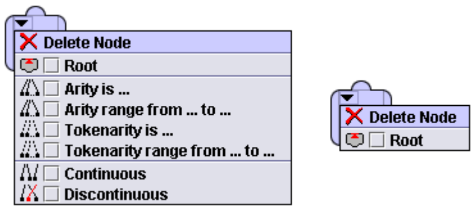


Figure: The menu of a phrase node (left) and of a token node (right).

The (token)arity and (dis)continuous predicates are only useful for phrase nodes.

- **× Delete Node**
  Deletes the node, including its feature constraints and all edges starting or ending at this node.
- **Root**
  Enables/disables the root predicate. Only one node might use the root predicate. If you specify this predicate on one node, the root predicate will be disabled on all other nodes. The root predicate is visualized by disabling the dominance top plug (all edges ending at this node will be deleted) and displaying a red triangle instead.
- **Arity is ...**
  Shows/hides an input field for the arity value below the inner node specification area.
- **Arity range from ... to ...**

Shows/hides two input fields for the start and end arity values below the inner node specification area.

- **Tokenarity is ...**
  Shows/hides an input field for the tokenarity value below the inner node specification area.
- **Tokenarity range from ... to ...**
  Shows/hides two input fields for the start and end tokenarity values below the inner node specification area.
- **Continuous**
  Enables/disables the node predicate *continuous*. By enabling this predicate the predicate *discontinuous* will be disabled, because both predicates cannot be enabled at the same time.
- **Discontinuous**
  Enables/disables the node predicate *discontinuous*. By enabling this predicate the predicate *continuous* will be disabled, because both predicates cannot be enabled at the same time.

Arity and tokenarity

After enabling an arity or tokenarity predicate via the node menu, input fields are shown below the inner node specification area at the bottom of the node. One input field for entering the exact (token)arity value is shown if the predicate *(token)arity is* ... is selected. If *(token)arity range from* ... *to* ... is selected, two input fields are shown: the left for the start value and the right for the end value of the range.

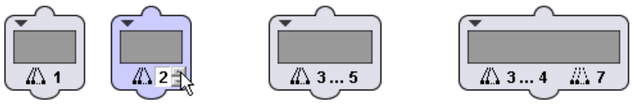


Figure: The arity and tokenarity input fields are only shown if the predicate is enabled.

The values can be changed by either typing in a new value or by pressing the increase or decrease button, which are shown at the right of the input field when the mouse cursor is over the input field or the input field is activated (see screenshot above).

Continuous and discontinuous

Only one of the *continuous* or the *discontinuous* node predicates can be set. These predicates are visualized by the non-crossing edges and the crossing edges symbol at the bottom of the node.

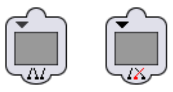


Figure: The (dis)continuous predicate visualized by the (non-)crossing edges symbol.

4.4 Feature Constraints

Creating a feature-value pair

A node can be specified by an arbitrary number of feature-value pairs (cf. section 3, chapter III). To create a feature-value pair click into the inner node specification area. A box comprising three menus appears (see image below). If the current corpus supports more than one phrase or token feature, the upper menu (the feature selection menu) opens automatically.

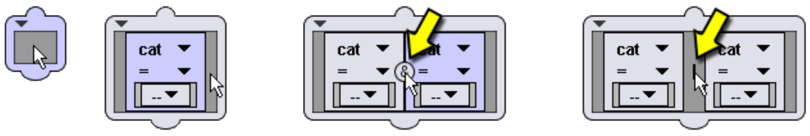


Figure: Creating feature-value pairs and toggling between conjunction and disjunction.

To create another feature-value pair you have to click into the inner node specification area, left or right of the existing feature-value pair. By default, these feature-value pairs are connected by conjunction. To toggle between conjunction and disjunction, click on the corresponding symbol.

1. Feature selection menu

The feature selection menu is primarily used to choose the feature, but can also be used to delete or negate the feature-value pair (visualized by a red line from lower left to upper right).

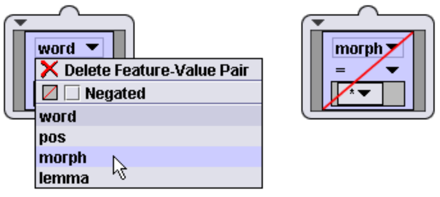


Figure: Feature selection menu (left) and negated feature-value pair (right).

2. Operator selection menu

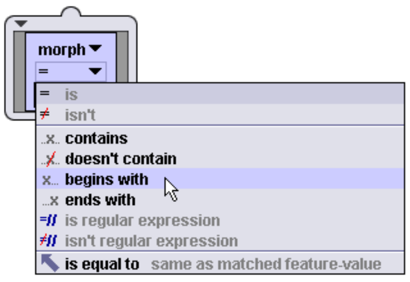

morph
=
= is
≠ isn't
..x.. contains
..x.. doesn't contain
x... begins with
...x ends with
=// is regular expression
≠// isn't regular expression
is equal to same as matched feature-value

Figure: The operator selection menu.

The *is* operator is selected by default. The following operators are available:

- **is**
  The feature must agree with the specified value.
- **isn't**
  The feature must not agree with the specified value.
- **contains**
  The feature value must contain the given string.
- **doesn't contain**
  The feature value must not contain the given string.
- **begins with**
  The feature value must begin with the given string.
- **ends with**
  The feature value must end with the given string.
- **is regular expression**
  The given pattern must match the feature value.
- **isn't regular expression**
  The given pattern must not match the feature value.
- **is equal to**
  The feature value is equal to the feature value of another node (cf. paragraph below).

The operators *contains*, *doesn't contain*, *begins with*, and *ends with* do not have a textual equivalent. They are converted to regular expressions (cf. subsection 3.5, chapter III). For example *begins with 'abc'* will be converted to `/abc.*/`.

3. Value specification

By selecting the operator *is* or *isn't*, the feature value can be specified by one or more values or types. In this case the third item of the feature-value pair is a field, such as the inner node specification area. By clicking at the left or right of the existing primary value/type, further values/types can be created. You can switch between disjunction and conjunction by a mouse click on the connector.

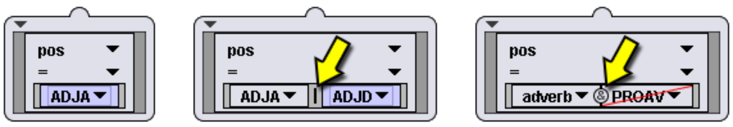


Figure: One or more values/types can be used to specify a feature value.

The first item of a value/type menu is used for deleting.

⚠**Please note:** If there is only one value/type, the delete function does not work.

The second menu item is for negation, also visualized by a red line from lower left to upper right.

Types are marked by the symbol and (by convention) type names are written in small letters. All possible feature values are listed, marked by the symbol . If there is no feature value list available e.g. for the word feature), there is an input field you can type in a value (see figure below).

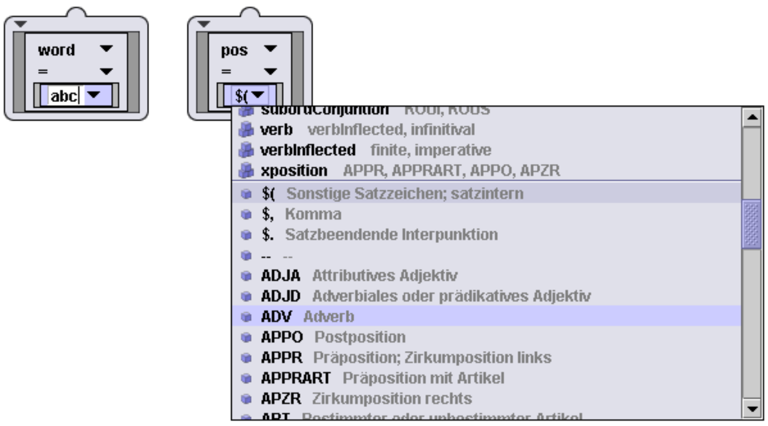

word
=
abc
pos
=
$(
subordConjunction KOUI, KOUS
verb verbInflected, infinitival
verbInflected finite, imperative
xposition APPR, APPRART, APPO, APZR
$(Sonstige Satzzeichen; satzintern
$, Komma
$. Satzbeendende Interpunktion
-- --
ADJA Attributives Adjektiv
ADJD Adverbiales oder prädikatives Adjektiv
ADV Adverb
APPO Postposition
APPR Präposition; Zirkumposition links
APPRART Präposition mit Artikel
APZR Zirkumposition rechts
ART Bestimmter oder unbestimmter Artikel

Figure: Input field (left) and menu for selecting a value or type (right).

Equality

To specify a feature value to be equal to a feature value of another node, variables are used (cf. subsection 7.2, chapter III). To create such a variable you have to select the operator *is equal to*. The mouse cursor will change into an arrow . Now click either into a free space of an inner node specification area of another node or on the same feature-value pair of another node (see figure below).

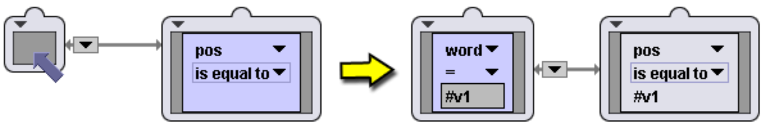


Figure: A click into an inner node specification area will create a new feature-value pair.

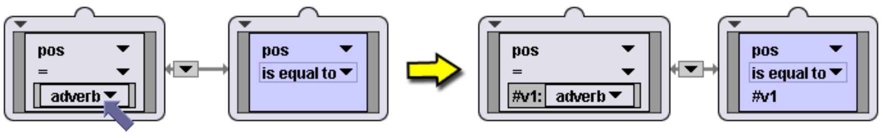
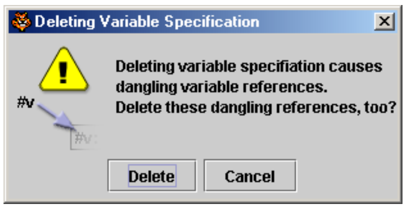


Figure: A click on an existing feature-value pair will set its value equal to the source feature value.

If you want to delete a feature-value pair which is used as a reference for an *is equal to* feature value, the following dialog will appear:


Deleting Variable Specification

Deleting variable specifiation causes dangling variable references. Delete these dangling references, too?

Figure: Deleting a variable specification, which is referenced by another feature-value pair, is not allowed.

4.5 Edges

Creating

Edges represent graph relations: *dominance* and *precedence* (cf. section 6, chapter III). An edge can be created between two existing nodes. To create a *dominance edge* you have to select a *lower dominance plug* of a node (cf. subsection 4.3) and a second partner node. It is also possible to do this in inverse order by first selecting an *upper dominance plug* and a second node as the start node of the relation.

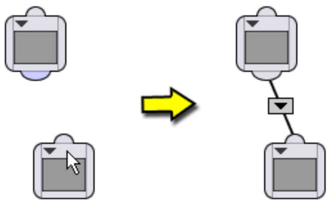


Figure: Dominance edges are created by clicking on dominance plugs.

To create a *precedence edge* you have to first select a *right precedence plug* or, to do it in inverse order, a *left precedence plug* and afterwards another node.

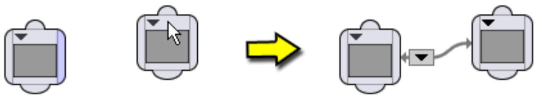


Figure: Precedence edges are created by clicking on the precedence plugs.

The type of the created edge (i.e. direct or non-direct, negated or not negated) depends on the preselection, which can be set in the toolbar (cf. subsection 4.6). After creating an edge, the type can be changed in the edge menu.

Multiple selection and creation

Two or more nodes or plugs can be selected by holding down the SHIFT or CTRL key while clicking. This can be used to create more than one edge simultaneously (see figure below).

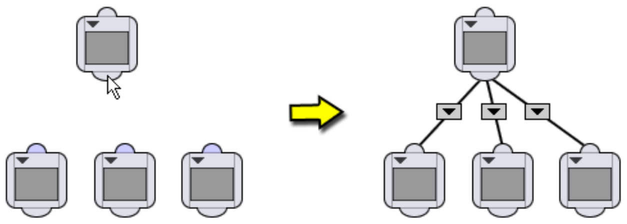


Figure: By pressing the SHIFT or CTRL key two or more nodes or plugs can be selected.

Dominance

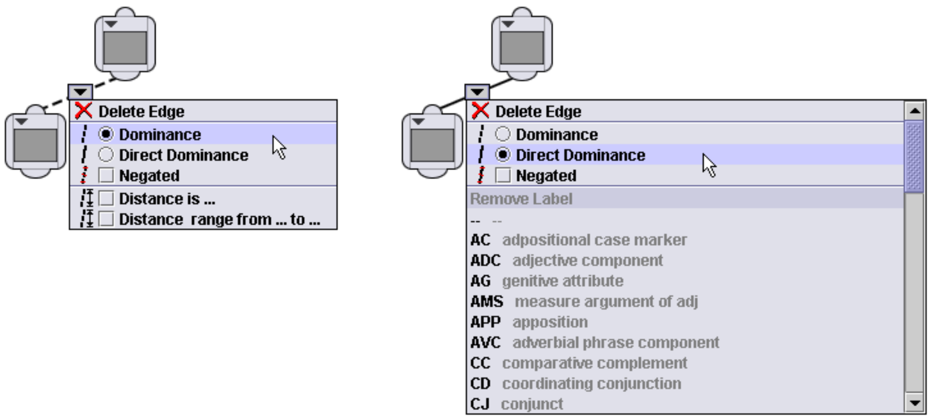


Figure: The menu of a dominance (left) and direct dominance (right) edge.

The first item of the edge menu is for deleting. The following three menu items provide functions to change the type of an edge after creating it.
If it is a non-direct dominance edge, the distance can be specified, either by the exact value or by a range. If the *Distance is ...* or *Distance range from ... to ...* item is enabled in the edge menu, one resp. two input fields are shown left of the edge menu. These values can be changed by either typing in a new value or by pressing the increase or decrease button, which are shown at the right of the input field when the mouse cursor is over the input field or the input field is activated. If the edge type is *direct dominance*, an optional edge label can be chosen from a list.

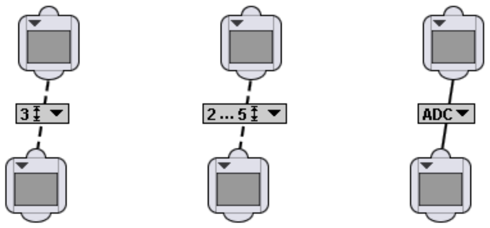


Figure: Distance specification (left) and edge label (right).

Left/right corner

If the end node of the (direct) dominance relation is a token, three additional edge menu items are available for the optional corner specification (cf. subsection 6.2, chapter III). Left or right corners are visualized with black filled triangles (see figure below).

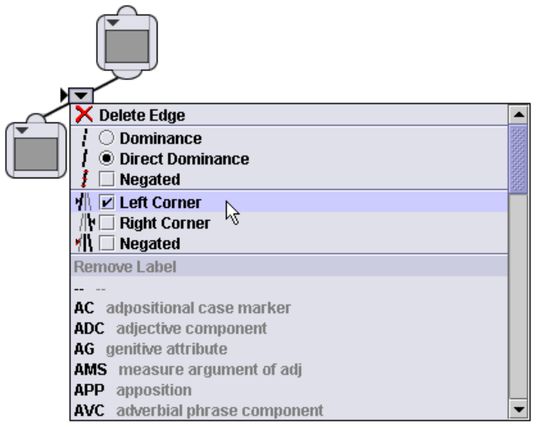


Figure: The menu items for corner specification.

Horizontal edges

Beside the dominance relation there are two more relations. With the *precedence* relation you can specify the horizontal order and distance between nodes. *Secondary edges* are often used as a kind of additional dominance relation. Secondary edges are not supported by all corpora.

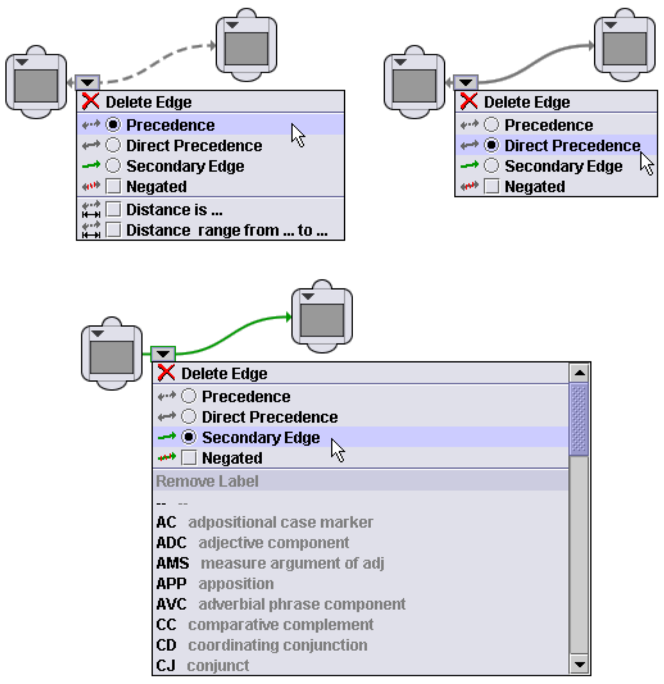


Figure: The menu of a precedence, direct precedence, and secondary edge.

The first menu item can be used to delete the edge. Below there are four (three if the current corpus does not support secondary edges) items to choose the type of the edge and to negate it (see figure above). If the edge type is *precedence*, input fields for either the exact distance value or a distance range can be enabled by choosing either the *Distance is ...* or *Distance range from ... to ...* menu item. The values can be changed by either typing in a new value or by pressing the increase or decrease buttons, which are shown at the right of the input field when the mouse cursor is over the input field or the input field is activated.

⚠**Please note:** Not every corpus supports secondary edges and secondary edge labels.

4.6 Toolbar

The buttons of the toolbar are divided into four parts: *tool selection*, *dominance edge type preselection*, *horizontal edge type preselection*, and *switch to textual mode*.

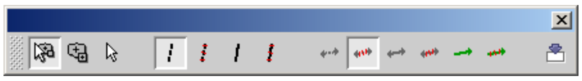


Figure: The toolbar.

Tool selection: *Moving/Creating*, *Creating*, *Moving*

With the leftmost three buttons you can specify how to create objects and how to move nodes:

- **Move/Create**
  Objects like nodes, feature-value pairs, and feature values will only be created by a mouse click if the selection is empty. If the selection is not empty, a mouse click in a free area deselects every object.
- **Create**
  Regardless whether the selection is empty or not, nodes, feature-value pairs, and feature values can be created by a mouse click at the appropriate location. For deselection click on a selected object or use the context menu (cf. subsection 4.7).
- **Move**
  Nodes, feature-value pairs, feature values, and edges cannot be created if this tool is activated. In this mode you can only move nodes.

Dominance edge type preselection

Before creating a dominance edge the type of the new edge can be preselected (cf. also subsection 4.5). Of course the type of an edge can be changed after creating, but preselection is very useful for creating two or more edges simultaneously. There are four different dominance edge types:

- **Dominance**
- **Negated Dominance**
- **Direct Dominance**

- **Negated Direct Dominance**

Horizontal edge type preselection

The type of a new horizontal edge can also be preselected. (cf. subsection 4.5).

- **Precedence**
- **Negated Precedence**
- **Direct Precedence**
- **Negated Direct Precedence**
- **Secondary Edge**
- **Negated Secondary Edge**

⚠**Please note:** *Secondary edge* and *negated secondary edge* are only shown if the current corpus supports secondary edges.

Switch to Textual Mode

By clicking the rightmost button marked by the symbol the textual representation of the current graphical query will be copied into the text query editor of TIGERSearch (cf. section 3). The query text can now be edited and submitted.

⚠**Please note:** In the current version of TIGERSearch a textual query cannot be converted to a graphical query (cf. subsection 4.1).

4.7 Context Menu

Popping up the context menu depends on the platform. Either the second or third mouse button has to be pressed or released. The context menu and its submenus are shown below:

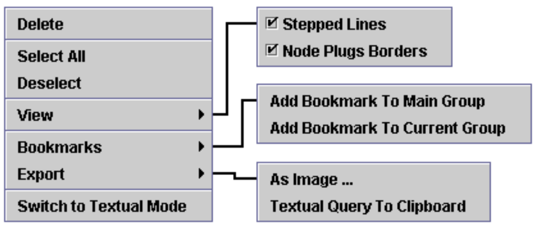


Figure: The context menu and its submenus.

Delete

Deletes all selected objects. Edges cannot be selected but will be deleted if the start or end node is deleted. Nodes are also deleted if only a node plug is selected.

Select All

If the context menu is called from the inner node specification area, all feature constraints of the current node are selected.

Deselect

Removes the current selection.

View - Stepped Lines

The dominance relation can be visualized either by stepped lines (cf. TIGERGraphViewer) or by a straight line edge:

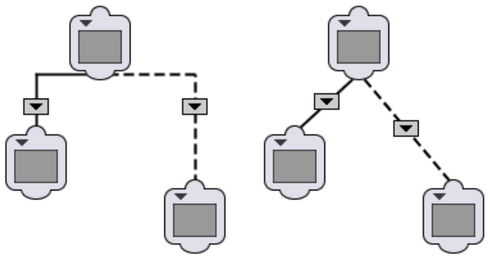


Figure: Stepped lines (left), straight lines (right).

View - Node Plug Borders

To point out the different areas of a node (cf. subsection 4.3) small lines show the borderlines. These lines can be disabled.

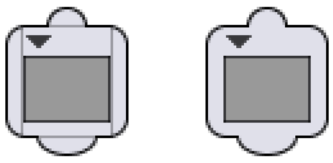


Figure: Nodes with (left) or without (right) node plug borders.

Bookmarks

The **textual** representation of the graphical query can be saved for later use as a bookmark (cf. subsection 3.4). Selecting *Add Bookmark To Main Group* adds the textual representation of the current graphical query into the main group of the bookmark folder, *Add Bookmark To Current Group* will add a bookmark into the currently selected bookmark folder.

⚠**Please note:** In the current version, the graphical query representation cannot be saved for later usage.

Export - As Image

The current graphical query can be exported as an image. There are five export formats available:

- **SVG (Scalable Vector Graphic)**
  An XML-based vector graphics format. This means that images can be scaled without loss of quality.
- **TIF (Tag Image File Format)**
- **PNG (Portable Network Graphics)**
- **JPG (JPEG File Interchange Format)**
- **PDF (Portable Document Format)**

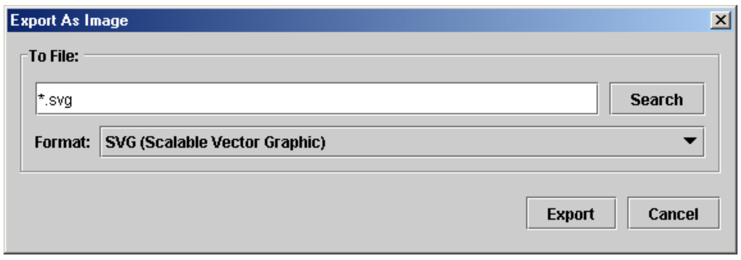


Figure: Export As Image dialog.

Export - Textual Query to Clipboard

The textual representation of the current graphical query is copied to the clipboard to be used in other applications.

Switch to Textual Mode

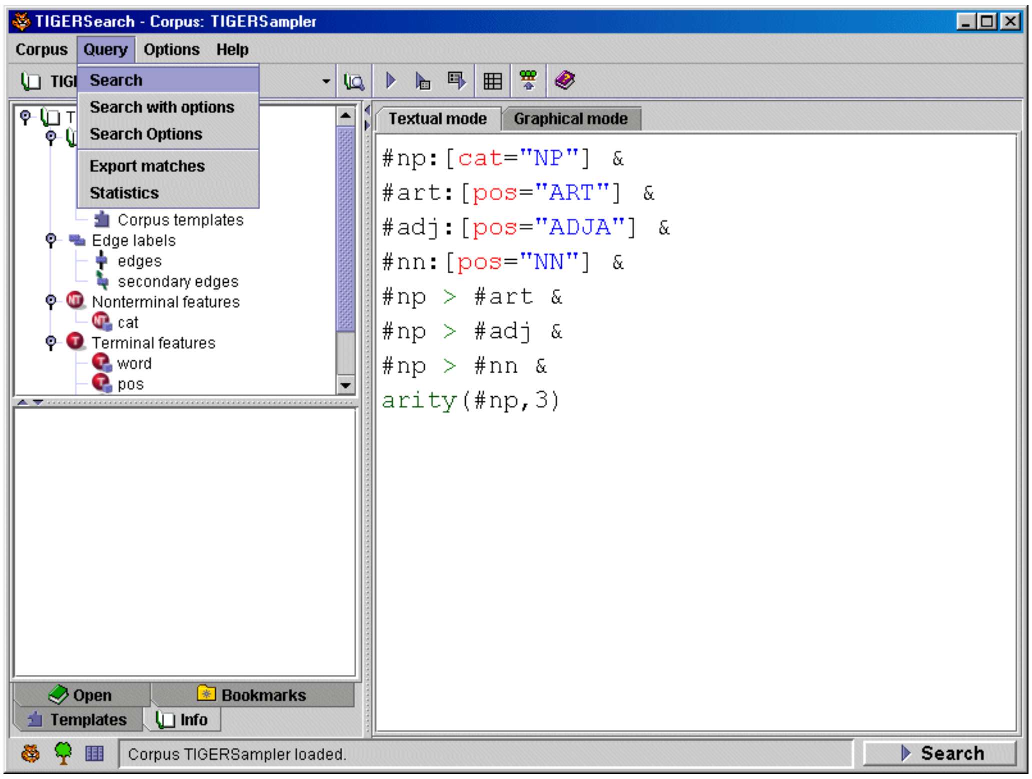
The textual representation of the current graphical query will be copied to the text query editor of TIGERSearch (cf. section 3). The query text can now be modified and submitted.

⚠**Please note:** In the current version of TIGERSearch a textual query cannot be converted to a graphical query (cf. subsection 4.1).

5. Processing a query

5.1 Basic query processing

Query processing is started by pressing the *Search* button in the lower right corner of the TIGERSearch window, or pressing the *Search* button in the button toolbar, or selecting the *Search* item in the *Query* menu of the TIGERSearch window.

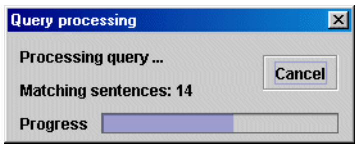


```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
#nn:[pos="NN"] &
#np > #art &
#np > #adj &
#np > #nn &
arity(#np,3)
```

Corpus TIGERSampler loaded.

Figure: Starting query processing

The query processing progress window will keep you informed about the current status of query processing (cf. screenshot). Pressing the *Cancel* button will stop query processing. The matching corpus graphs found up to this point will be displayed by the GraphViewer (cf. section 7).

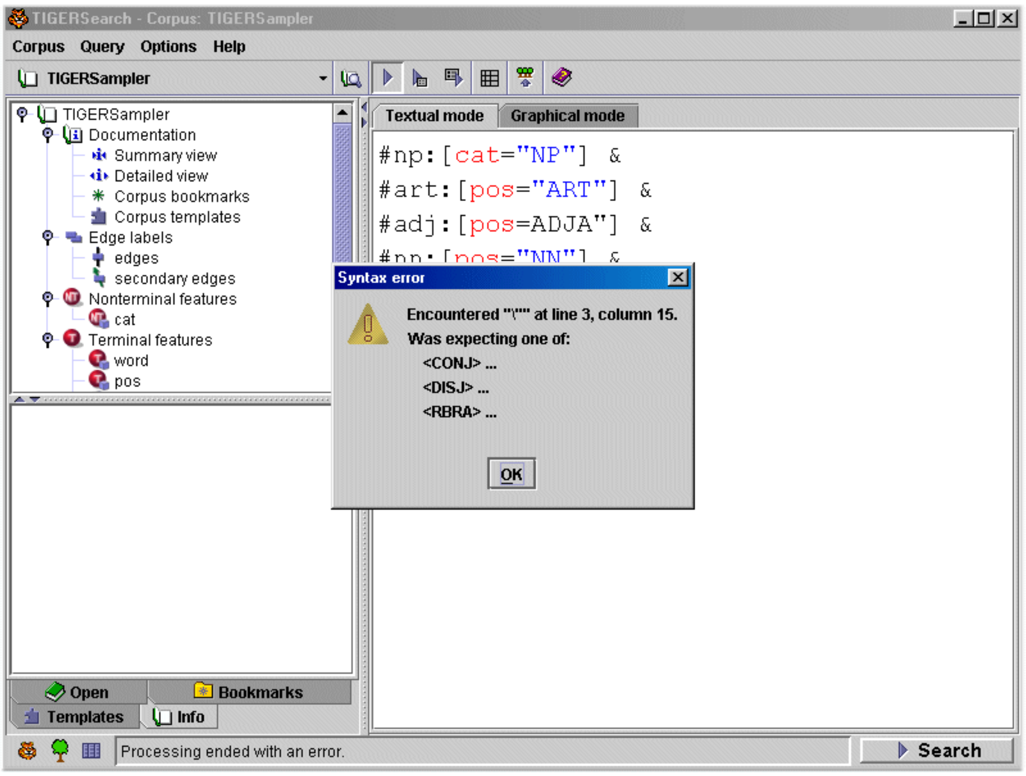


Processing query ...

Matching sentences: 14

Figure: The query processing progress window

A common error that might occur during query processing is a syntax error detected by the query parser. In this case, a message window will be displayed that informs you about the error type (cf. screenshot). After pressing the *OK* button, the cursor will automatically jump to the error position.



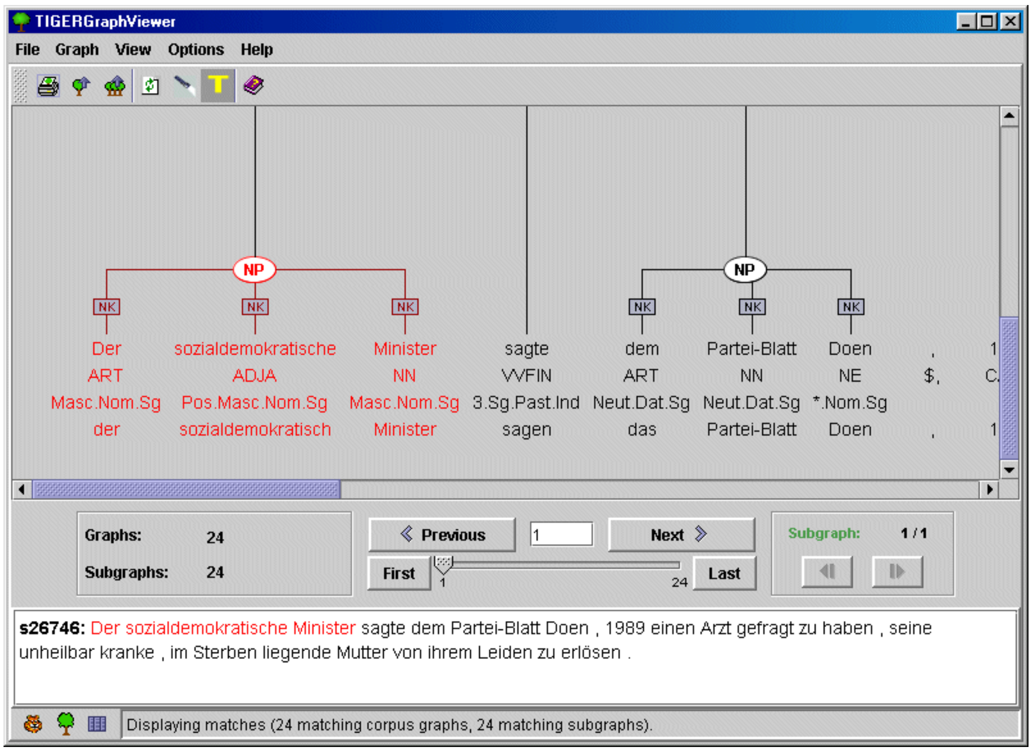
```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos=ADJA"] &
#nn:[pos="NN"] &
```

Encountered "\"" at line 3, column 15.
Was expecting one of:
<CONJ> ...
<DISJ> ...
<RBRA> ...

Processing ended with an error.

Figure: Handling of syntax errors

When query processing has finished, the matching corpus graphs will be displayed by the GraphViewer (cf. screenshot and section 7 for a detailed description). You can also view the matches in the statistics mode (cf. section 8) or export your favourite matches using the export module (cf. section 9).



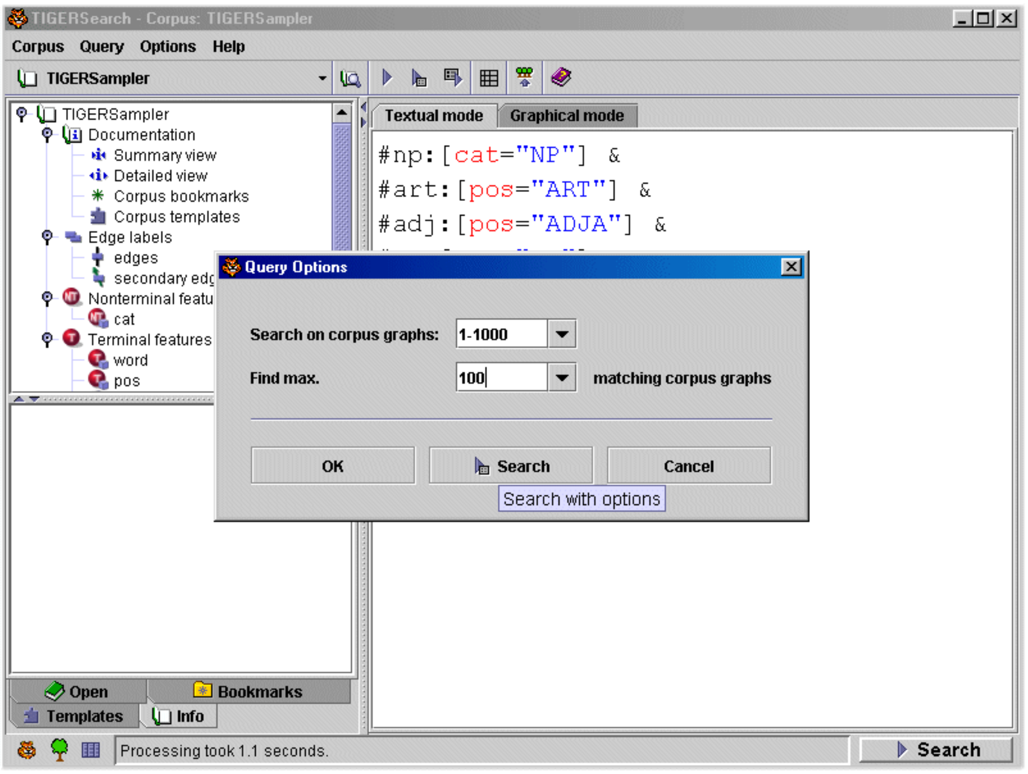
s26746: Der sozialdemokratische Minister sagte dem Partei-Blatt Doen , 1989 einen Arzt gefragt zu haben , seine unheilbar kranke , im Sterben liegende Mutter von ihrem Leiden zu erlösen .

Displaying matches (24 matching corpus graphs, 24 matching subgraphs).

Figure: Query results

5.2 Advanced query processing

If you do not want to search the whole corpus, but just on a part of it, you can specify a corpus area. This feature is very useful for experimenting with queries. To specify a corpus area, open the *Query Options* window by clicking the corresponding toolbar icon or selecting the *Search Options* item in the *Query* menu of the TIGERSearch tool:



```
#np:[cat="NP"] &
#art:[pos="ART"] &
#adj:[pos="ADJA"] &
```

Search on corpus graphs: 1-1000

Find max. 100 matching corpus graphs

Processing took 1.1 seconds.

Figure: Query processing options

Now select one of the preselected search spaces or type in the search space using the syntax *start-end*. If you press the *OK* button, the defined search space will be saved for later processing. If you like to save the search space and immediately start query processing using the defined option, just click the *Search* button in the *Query Options* window.

⚠**Please note:** After leaving the *Query Options* window, query processing using the search area option is only activated when clicking the *Search with options* toolbar icon or selecting the *Search with options* item in the *Query* menu. If you start query processing using the *Search* button, the defined search option will **not** be used.

If you do not want to view all matching corpus graphs, you can set an upper bound for the number of matching corpus graphs determined by the query processor. If you specify to search for n matching corpus graphs, query processing will be stopped automatically when n matching corpus graphs have been found. This option is also specified in the *Query Options* window.

6. Use of templates

The syntax of template definitions and of queries which involve template calls is described in section 9, chapter III. In the current section, you learn how templates are loaded into the TIGERSearch tool, and how to handle error messages concerning templates.

6.1 Template path and template files

Currently, template definitions have to be 'hard-wired' individually for each corpus. This means that only 'power users' can change the template definitions for a corpus. Or, to put it the other way round: If you want to develop your own template definitions, you need your personal copy of a corpus. Please refer to subsection 4.5, chapter VI to see how a corpus template path, i.e. the directory which holds the template definitions, is associated with a corpus.

Template definitions have to be stored in *template files*, i.e. files which have the extension `.tig` (*tiger template*). The corresponding, automatically created binary code will be stored in files with extension `.tgc` (*tiger template, compiled*).

⚠**Please note:** File access for templates must be set in the following manner:

- Users of a corpus which comes with template definitions, need read access not only to the corpus itself but also to the whole directory hierarchy below the template path.
- Template developers need read and write access to the whole template directory hierarchy.

⚠**Please note:** In certain situations,corpus users may get write access error messages wrt. the template directories. In this case, a person with write access to the template directory hierarchy should reload the corpus which produced the error messages in order to have the binary template files created newly.

6.2 Template loading

Template loading is carried out automatically when a corpus is (re-)loaded (cf. subsection 2.1). For template developers, this means that you have to reload the corpus in order to make changes of the template definitions known to the TIGERSearch tool.

The template definitions which are associated with a corpus are loaded from the given template path in the following order:

- depth-first

  Files from subdirectories will be loaded before the `.tig` files in the given directory.
- alphabetical order

  Subdirectories resp. files are loaded in alphabetical order.

⚠**Please note:** Based on the above order, only the first template definition for a template head (template name/arity) will be loaded. All other definitions for the same template head will be ignored.

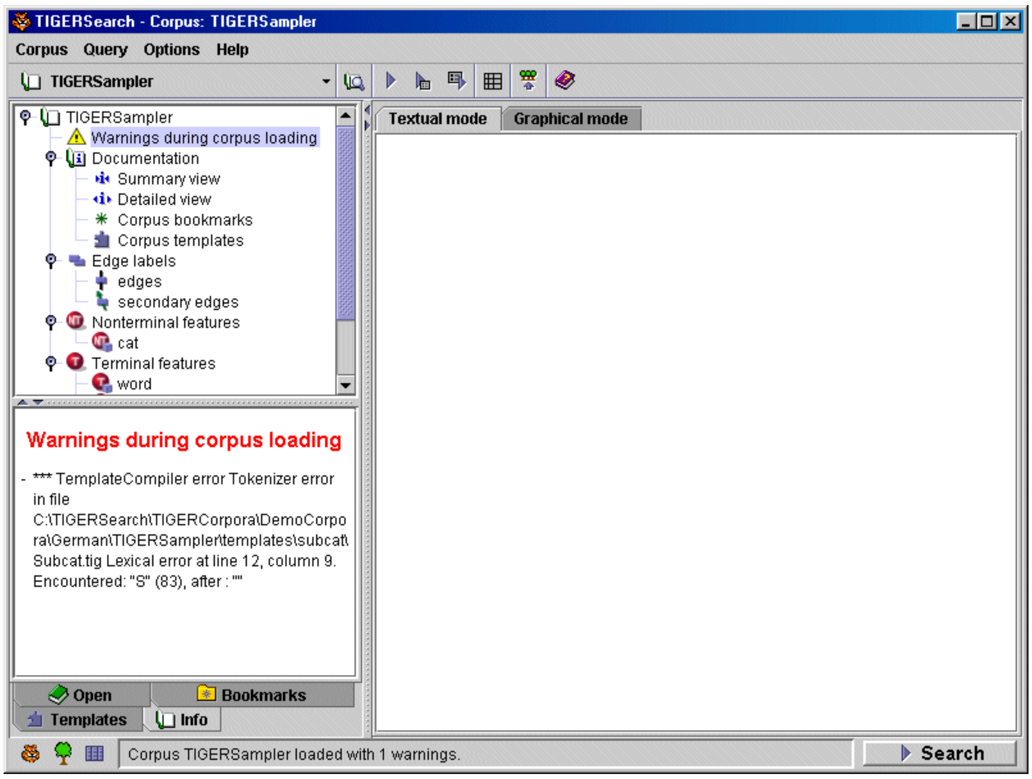
For faster processing, templates are translated ('compiled') into Java objects. In order to avoid unnecessary recompilation steps, compiled template code is stored on the hard disk in binary form (files with extension `.tgc`). The compilation of a template source file `filename.tig` is triggered automatically during the template loading process, in the following situations:

- The compiled file `filename.tgc` does not exist yet.
- The compiled file `filename.tgc` is older than source file `filename.tig`.
- The TIGERSearch system has been updated in the meantime, i.e. the internal data structures of the compiled files are outdated.

The fact that compiled code is written on hard-disk means that file access properties have to be set in an appropriate manner (cf. subsection 6.1).

6.3 Error messages during template loading

If there are any template loading problems during the corpus loading process, all error messages are stored for inspection. These messages can be inspected by clicking on the warning symbol in the corpus information tab (cf. also subsection 2.2):



Warnings during corpus loading

- *** TemplateCompiler error Tokenizer error in file C:\TIGERSearch\TIGERCorpora\DemoCorpora\German\TIGERSampler\templates\subcat\Subcat.tig Lexical error at line 12, column 9. Encountered: "S" (83), after : ""

Corpus TIGERSampler loaded with 1 warnings.

Figure: Inspection of template loading errors

The handling of file access errors has already been explained in subsection 6.1.

If a template call does not match the head of any template definition with respect to template name and number of parameters, an *undefined template error message* is caused if the template is called. Therefore, you should

- verify the template name and the number of argument parameters, both in the template call and in the corresponding template definition. Correct the template call according to your template definitions, or vice versa.

- check whether the corresponding template definition has been compiled properly. If the compilation process has failed for a template definition, e.g. due to syntax errors or read/write access errors, obviously, the definition is not available in the internal template store.

Variable-related error messages, e.g. 'undefined type of variable' or 'variable type clash' will be produced in the following situations:

- The type of an argument parameter in a template call cannot be determined from the context of the template call.
- The type of an argument parameter of a template head cannot be determined from the template body.
- An argument parameter of a template call and the corresponding parameter in a template head have disjoint types.

For example, the type of variable `#n5` cannot be derived from the other constraints in the body of the following template definition:

```
VerbPhrase(#n0) <-
   #n0:[cat="VP"] > #n1:[pos="VVFIN"]
   & #n0 > #n2
   & #n1.#n2
   & arity(#n0,2)
   & PrepPhrase(#n5) ;
```

This problem is solved by either correcting the name of the variable, or by inserting the same variable name somewhere else in the graph description, so that the type of the variable becomes clear (node variable vs. feature constraint variable vs. feature value variable).

A *cyclic template definition error* will be issued if a template definition embeds a call to itself, either directly or indirectly. The error message lists the 'ancestors' of the cyclic call in order to give you a hint how the cycle could have been created. Currently, the ancestors are listed in arbitrary order - which may be somewhat confusing.

6.4 Restrictions of the current implementation

- Although a file may contain several defining clauses for the same or for different templates, only the first clause for each template will be retained.

  Workaround: Use the disjunction operator `|` in order to pack alternative template bodies into a single defining clause.

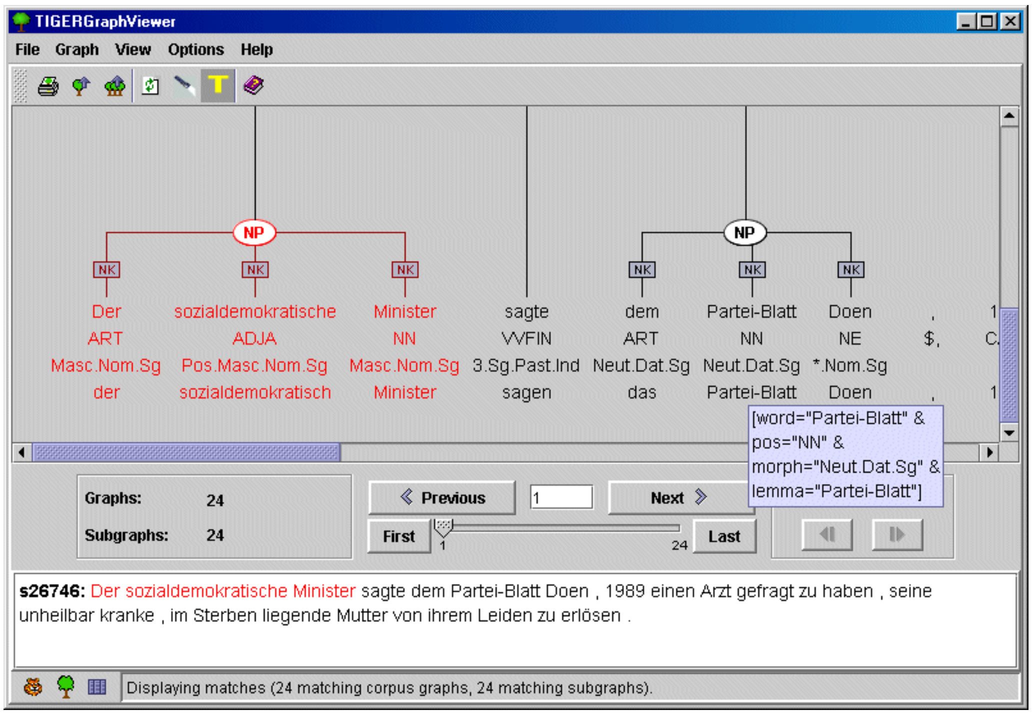
7. Viewing the matches - The GraphViewer

7.1 Navigation through the corpus graphs

The GraphViewer is used to browse through the whole corpus (corpus exploration) or to browse through the matching corpus graphs (match visualization). The corpus exploration mode is activated by pressing the *Explore corpus* button or selecting the *Explore* item in the *Corpus* menu. The match visualization is automatically activated after query processing. When opened, the GraphViewer will display the first corpus graph or the first matching corpus graph, respectively.

⚠**Please note:** You can easily switch between the TIGERSearch main window and the GraphViewer window using the shortcut buttons in the lower left corner of both windows. If you press a shortcut button, the corresponding window will be moved in front of all other windows on your desktop.

The tokens of the currently displayed corpus graph is presented at the bottom of the window. You can navigate through the matching corpus graphs by using the navigation panel at the bottom of the GraphViewer. You can either view them one by one in the order they have been found using the *Previous* and *Next* buttons, or browse through the corpus graphs using the slider above these buttons.


TIGERGraphViewer
File Graph View Options Help
NP
NK
NK
NK
NP
NK
NK
NK
Der ART Masc.Nom.Sg der
sozialdemokratische ADJA Pos.Masc.Nom.Sg sozialdemokratisch
Minister NN Masc.Nom.Sg Minister
sagte VVFIN 3.Sg.Past.Ind sagen
dem ART Neut.Dat.Sg das
Partei-Blatt NN Neut.Dat.Sg Partei-Blatt
Doen NE *.Nom.Sg Doen
, $, ,
Previous
1
Next
First
Last
1
24

[word="Partei-Blatt" & pos="NN" & morph="Neut.Dat.Sg" & lemma="Partei-Blatt"]

Graphs: 24
Subgraphs: 24

s26746: Der sozialdemokratische Minister sagte dem Partei-Blatt Doen , 1989 einen Arzt gefragt zu haben , seine unheilbar kranke , im Sterben liegende Mutter von ihrem Leiden zu erlösen .

Displaying matches (24 matching corpus graphs, 24 matching subgraphs).

Figure: Browsing through the (matching) corpus graphs

You can move right to the last graph by pressing the *Last* button and back to the first one pressing the *First* button. You may also move to a graph with a certain position within the results. Just type in the position number in the input field between the *First* and *Last* button and press *Return*.

If there are two or more matching subgraphs in the same corpus graph, the green navigation arrows in the *Subgraph* box on the right hand side of the navigation bar will be activated. Now you can navigate from one matching subgraph to the next using the green buttons. In the example screenshot, there are four NP subgraphs within the current corpus graph matching the query `[cat="NP"]`.

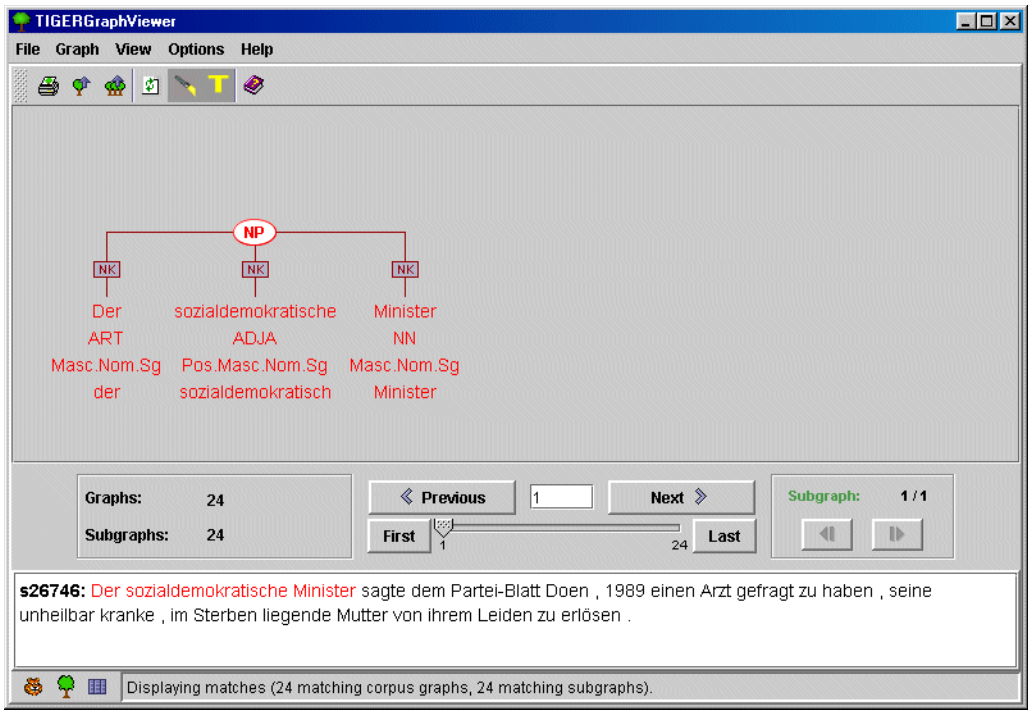
The information box on the left shows the total number of matching corpus graphs and the total number of matching subgraphs within all corpus graphs. Thus, we have 176 corpus graphs which comprise at least one NP and 448 NPs in the whole corpus.

7.2 Advanced navigation

Focus on match

If you do not want to see the whole graph, but only the matching subgraph, you can turn on the *Focus on match* button in the toolbar (represented as a flash light turned on/off). The GraphViewer will display just the matching subgraph. Turning off the *Focus on match* button will display the initial view of the graph.

⚠**Please note:** The *Focus on match* feature is not accessible in the corpus exploration mode.



Graphs: 24
Subgraphs: 24

s26746: Der sozialdemokratische Minister sagte dem Partei-Blatt Doen , 1989 einen Arzt gefragt zu haben , seine unheilbar kranke , im Sterben liegende Mutter von ihrem Leiden zu erlösen .

Displaying matches (24 matching corpus graphs, 24 matching subgraphs).

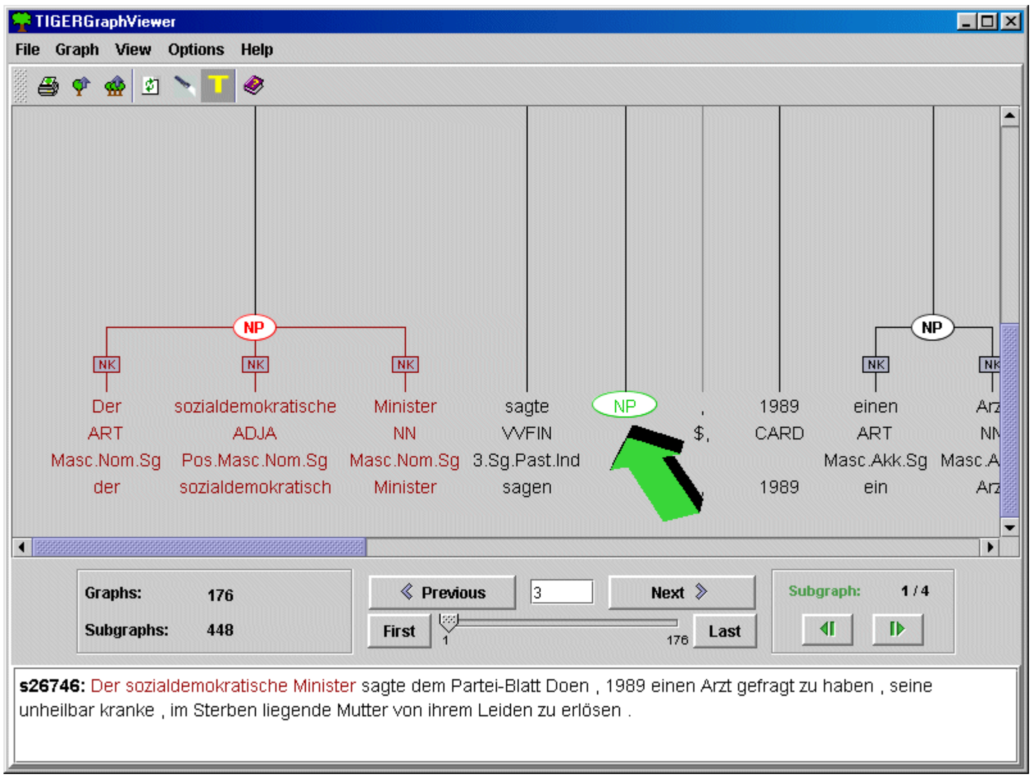
Figure: Focussing the matching structure

Sentence text field

If you do not want to see the tokens of the graph you can turn off the *Token* button in the toolbar (represented by a T character turned on/off). Turning on the *Token* button will display the tokens again. To change the context size of the sentence text field (current sentence +- 0/1/2 context sentence(s)) you can use the display options menu.

Imploding subgraphs

If you want to hide parts of the graph you can implode subgraphs pressing the middle mouse button on the node which is the root of the subgraph you want to hide (cf. screenshot). If you are using a two-button mouse, clicking the left and right button simultaneously should emulate the middle button click. Clicking the imploded node again will expand it to original size. Of course, node imploding does not make sense for the root node and terminal nodes.

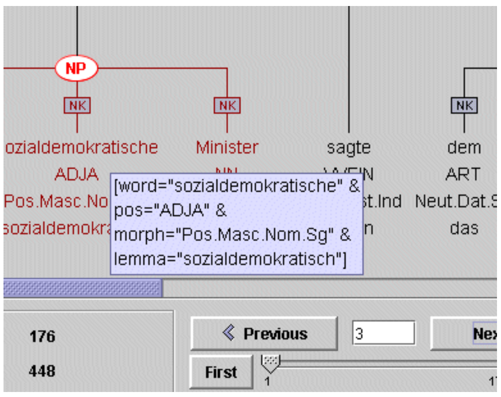

TIGERGraphViewer
File Graph View Options Help
NP
NK
Der ART Masc.Nom.Sg der
sozialdemokratische ADJA Pos.Masc.Nom.Sg sozialdemokratisch
Minister NN Masc.Nom.Sg Minister
sagte VVFIN 3.Sg.Past.Ind sagen
NP
, $,
1989 CARD 1989
einen ART Masc.Akk.Sg ein
Graphs: 176
Subgraphs: 448
Previous
Next
First
Last
Subgraph: 1 / 4

s26746: Der sozialdemokratische Minister sagte dem Partei-Blatt Doen , 1989 einen Arzt gefragt zu haben , seine unheilbar kranke , im Sterben liegende Mutter von ihrem Leiden zu erlösen .

Figure: Imploding a subgraph

Node tooltip

If the mouse pointer is placed over a node (token or inner node), a node tooltip windows pops up. This windows comprises the annotation of all features:



[word="sozialdemokratische" &
pos="ADJA" &
morph="Pos.Masc.Nom.Sg" &
lemma="sozialdemokratisch"]

Figure: Node tooltip

Refresh

If the graph is not displayed properly, use the *Refresh* button in the toolbar to repaint the graph.

7.3 Communicating with the Statistical Viewer

If you are simultaneously working with the GraphViewer and the Statistical Viewer, you can easily switch between these two views using the shortcut icons in the lower left corner of both windows.

However, the two windows have also been designed to communicate with each other. The corpus graph currently displayed in the GraphViewer will be displayed and marked in the Statistical Viewer if you select the *Display in Statistics* item of the GraphViewer's context menu (activated by pressing the right mouse button in the GraphViewer panel).

7.4 Image export

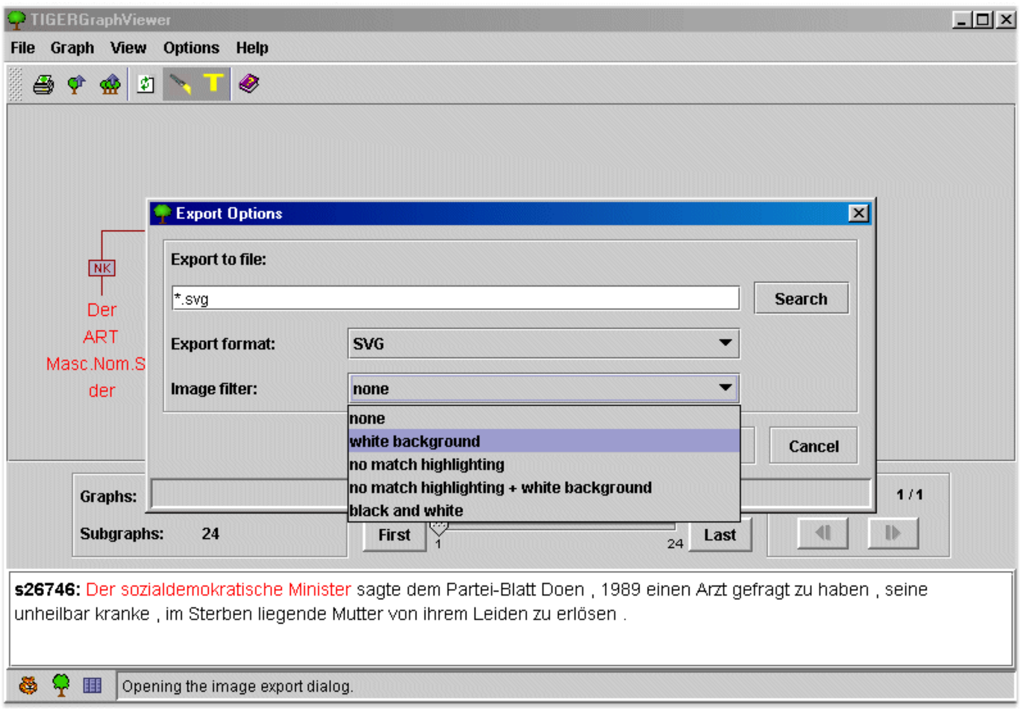
Printing the current graph / Postscript export

You can print the current graph pressing the *Print* button. The print options can be selected in a platform-dependent print dialog. Some general options such as paper format can be specified in the page setup window which is activated by selecting the *Pape Setup* item in the *Graph* menu of the GraphViewer.

⚠**Please note:** If you choose a Postscript printer in the print dialog (e.g. Apple Laserwriter), you can use the *Print into file* option to create a Postscript version of the displayed corpus graph. The created Postscript file may be used by applications such as LaTeX. Please note that the quality of the Postscript output depends on the selected printer and its parameters.

Exporting the current graph (image export)

You can save the current graph to several image formats (SVG, JPG, PNG, TIF, and PDF) using either the *Export as image* item of the context menu (activated by pressing the right mouse button), or the *Export image* icon (cf. picture icon) of the toolbar, or the *Export image* item in the *Graph* menu. A dialog window pops up, and you can select the options you prefer for saving the current graph (cf. screenshot).



Export to file: *.svg

Export format: SVG

Image filter: none

Graphs:

Subgraphs: 24

s26746: Der sozialdemokratische Minister sagte dem Partei-Blatt Doen , 1989 einen Arzt gefragt zu haben , seine unheilbar kranke , im Sterben liegende Mutter von ihrem Leiden zu erlösen .

Opening the image export dialog.

Figure: Exporting the current corpus graph as an image

The export is based on the SVG format which is an XML-based vector graphics format. In contrast to binary formats such as JPG format, images encoded using SVG can be manipulated (e.g. scaled) without loss of quality. Thus, you can change a token or a syntactic category within the exported SVG file. If you selected an non-SVG image, first an SVG representation is generated and afterwards converted into the preferred format.

The SVG output has been designed in such a way that the different node types (inner/outer node, matching node, imploded node etc.) can be identified by additional XML attributes. Thus, this information can be used to modify the generated SVG image, e.g. by Cascading Stylesheets (CSS).

The image filters you can choose (cf. screenshot above) are realized by inserting predefined CSS stylesheets into the SVG document. In subsection 10.1 we explain how to add your own image filter to the SVG export.

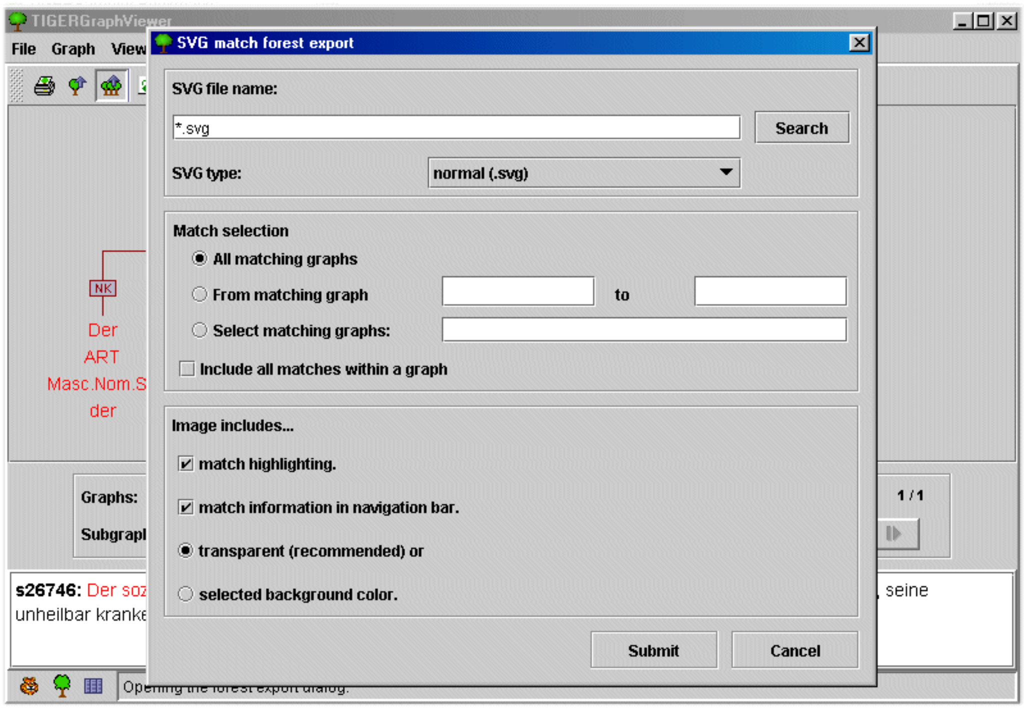
⚠**Please note:** Relative paths used to specify the export file are evaluated with regard to the working directory.

Exporting the match forest (animated SVG image)

If you like other users to have a look at your favourite matches, you may want to export the matches in a format that does not depend on the TIGERSearch software suite. Of course, you might export all the matching corpus graphs as single images, but this solution would not be practical.

As an alternative, you can use the *Export match forest* option to export all matching corpus graphs to one single SVG file. The individual graphs are combined by SVG animations. Thus, you can use any SVG viewer to navigate through the match forest. Please note that the SVG viewer has to support SVG animations.

You can open the *Match forest export* window by pressing the *Export match forest* button (cf. forest icon) in the button toolbar or selecting the *Export forest as SVG* item in the *Graph* menu. Now you can specify the following parameters:



SVG match forest export

SVG file name: *.svg [Search]

SVG type: normal (.svg)

Match selection

- [● All matching graphs]
- ○ From matching graph ______ to ______
- ○ Select matching graphs: ______
- ☐ Include all matches within a graph

Image includes...

- [✓ match highlighting.]
- [✓ match information in navigation bar.]
- [● transparent (recommended) or]
- ○ selected background color.

[Submit] [Cancel]

Figure: Exporting a match forest (animated SVG image)

- Parameter: SVG file name

  You can either generate an uncompressed SVG file (*.svg) or a compressed SVG file (*.svgz). Relative paths are evaluated with regard to the working directory. Compression reduces the size of the file to about 10% of its original size.

- Parameter: match selection

  You can restrict the export of all matching corpus graphs to a range of matching graphs, or specify them in a text field (e.g. `1-3;6-7`). Note that you have to specify the number (or position) within the match forest, not the corpus graph ID.

  Since a corpus graph can match a query more than once, you might prefer to export a graph as often as it matches the query. In this case please check the box *Include all matches within a corpus graph*.

- Parameter: image includes

  All corpus graphs are exported in a canonical form, i.e. as fully expanded graphs including match highlighting and match focussing. You can turn on/off the match highlighting, include/exclude the exact description of the match in the SVG navigation bar, and specify the background color. We recommend you to use *transparent*; in this case the SVG viewer is responsible for the selection of the background color.

To start the export process just press the *Submit* button. Please note that the export process **cannot** be stopped.

A popular SVG viewer is the SVG browser plugin which has been developed by Adobe (cf. *http://www.adobe.com/svg/viewer/install/*). This plugin is currently available for the Microsoft Windows and Apple Macintosh platforms. Using this plugin, you can view an SVG image in the Netscape and Internet Explorer browsers.

If you are interested in integrating SVG images in Microsoft PowerPoint presentations, you should visit the following web page: *http://www.indezine.com/products/powerpoint/*. This page comprises a detailed description of the integration process. It also contains interesting general hints to improve your PowerPoint presentations.

7.5 Viewer options

The default configuration of the TIGERGraphViewer has been defined in such a way that the visualization should work fine on all supported platforms. If you like to adapt the configuration according to your individual preferences, you can modify the display colors and the most important display parameters. Your modifications are saved when leaving the tool and restored at the beginning of your next TIGERSearch session.

Color options

To change the color settings of the GraphViewer, please select the *Color options* item in the *Options* menu. The display color windows appers:

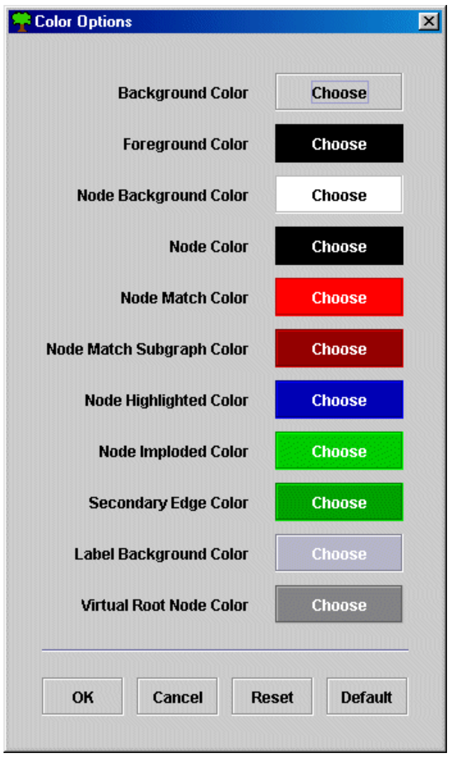


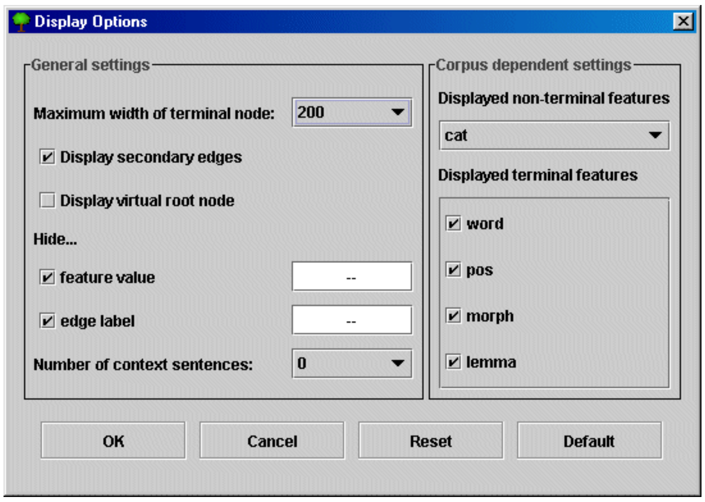
Figure: Changing the color settings of the GraphViewer

In this window all colors used by the TIGERGraphViewer are listed and illustrated by the color of the respective choose button. To change a color, press the corresponding *Choose* button. Another window pops up that lets you select the new color. Select the preferred color and leave this window by pressing the *OK* button.

If you would like to restore your configuration (that has been valid before opening the color options window), just click the *Reset* button. To restore the default values that have been preinstalled for using the TIGERGraphViewer activate the *Default* button. To leave the window without changing the configuration press the *Cancel* button. To submit your changes press the *OK* button.

Display options

Some important display parameters used by the TIGERGraphViewer can also be changed. To open the display options window, please select the *Display options* item in the *Options* menu. The display options window appers.


Display Options

General settings

Maximum width of terminal node: 200

[✓] Display secondary edges

[ ] Display virtual root node

Hide...

[✓] feature value --

[✓] edge label --

Number of context sentences: 0

Corpus dependent settings

Displayed non-terminal features

cat

Displayed terminal features

[✓] word

[✓] pos

[✓] morph

[✓] lemma

Figure: Changing display parameters of the GraphViewer

The following general parameters can be modified:

- Maximum width of terminal nodes

  The maximum width of the terminal node visualization is limited. Thus, some feature values (especially word forms) may be truncated. If this occurs too often, just specify a higher width.

- Display secondary edges

  In order to turn on/off the display of secondary edges check/uncheck this box.

- Display virtual root node

  Some corpus formats such as the Negra format do not integrate special types of tokens (e.g. punctuations marks) into the graph annotation. Unfortunately, the TIGERSearch query processing algorithm only works on connected graphs. Hence, during the indexing process a virtual root node is inserted in any graph that comprises terminal nodes (e.g. punctuations marks) which are not integrated into the corresponding syntax graph. In order to turn on/off displaying of virtual root nodes check/uncheck this box.

- Hide...

  Sometimes annotating a token feature does not make sense for features such as morphological feature or case. In this case, a qualified symbol such as `--` is used instead. We recommend you to use `--` which is also used in our implemented import filters.

  Since these symbols do not represent annotation information, it makes sense to hide them in the visualization. The two boxes must be checked if you want to hide a special feature value or edge label. Please specify the symbol which should be suppressed in the respective text fields.

The following corpus-depending parameters can be modified:

- Displayed non-terminal feature

For non-terminal nodes, only one feature can be displayed in the GraphViewer. If there are two or more features defined for non-terminal nodes, the displayed feature can be changed in this dialog. By default, the non-terminal feature defined first in the TIGER-XML encoding is selected.

- Displayed terminal features

  By default, all features of the terminal nodes are displayed. If you like to reduce the number of displayed features, just deselect the features you are not ineterested in.

⚠**Please note:** As these parameters are corpus-depending, they will be restored when the corpus is used again (even within the next TIGERSearch session).

If you like to restore your configuration (that has been valid before opening the display options window), just click the *Reset* button. To restore the original system defaults that have been pre-installed for using the TIGERGraphViewer activate the *Default* button. To leave the window without changing the configuration press the *Cancel* button. To submit your changes press the *OK* button.

8. Viewing the matches - Statistics

8.1 Introduction

The statistical viewer has been developed as a specialized view on the match results. Users can export their favourite matches as tables. We support a text-based output format, an proprietary XML-based format, and the Excel format.

In the present section, the description of the statistical viewer is illustrated by the results of the following TIGERSearch query:

```
  #np:[cat="NP"] & #art:[pos="ART"] & #adj:[pos="ADJA"] &
#nn:[pos="NN"] & #np > #art & #np > #adj & #np > #nn &
arity(#np,3)
```

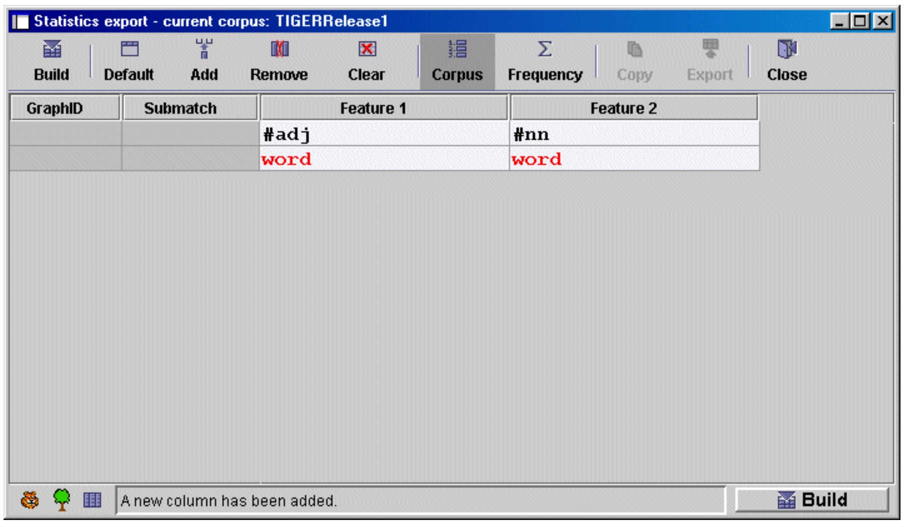
⚠**Please note:** To identify nodes within the statistical viewer in a unique way, every node must be labelled, i.e. it must be identified by a node variable (e.g. `#art`).

To show the statistical viewer, press the *Statistics* button in the button toolbar of the TIGERSearch main window, or select the *Statistics* item in the *Query* menu.

⚠**Please note:** You can easily switch between the TIGERSearch main window, the GraphViewer window, and the statistical viewer using the shortcut buttons in the lower left corner of the three windows. If you press a shortcut button, the corresponding window will be moved in front of all other windows on your desktop.

8.2 Specifying nodes and features

First of all, you have to specify the nodes and node features you are interested in. These are selected in the first two rows of the statistics table. Select the node in the first row and the node feature in the second row (cf. screenshot). If you like to display more than just one node feature, you can add more node feature columns by clicking the *Add* button or selecting the *Add column* item in the context menu of the feature columns. The *Remove* and *Clear* buttons (and its corresponding items in the context menus) can be used to delete a single column or to delete all currently used columns, respectively.



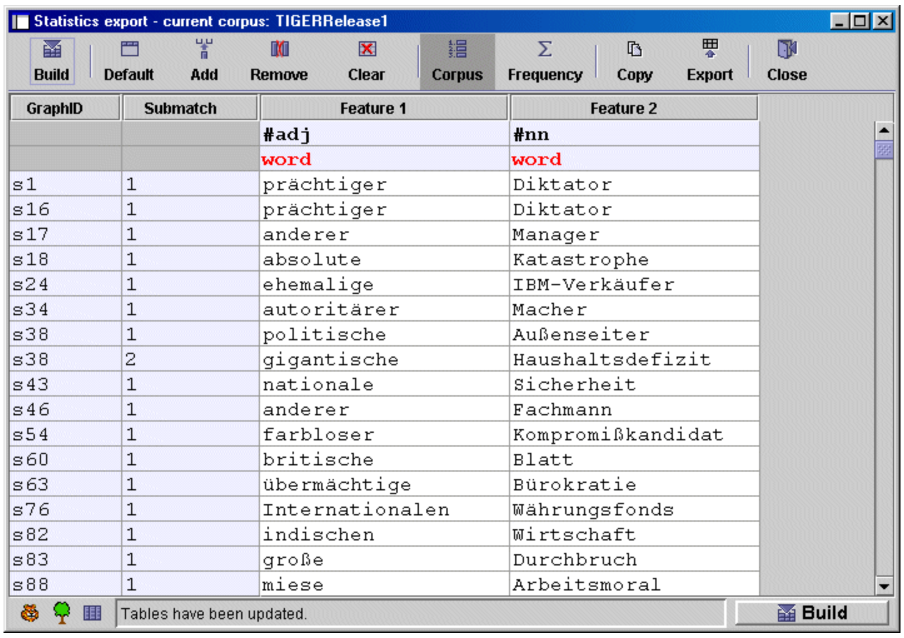
GraphID	Submatch	Feature 1	Feature 2
		#adj	#nn
		word	word

A new column has been added.

Figure: Specifying nodes and features

You might also choose the default arrangement of the nodes and features by pressing the *Default* button in the button toolbar. All terminal nodes specified in the query will be presented as columns and the default feature (usually the *word* feature) will be used.

Next, you have to build the table. Press one of the two *Build* buttons in the upper left or lower right corner, respectively. The statistics table will be filled with feature value information.



GraphID	Submatch	Feature 1	Feature 2
		#adj	#nn
		word	word
s1	1	prächtiger	Diktator
s16	1	prächtiger	Diktator
s17	1	anderer	Manager
s18	1	absolute	Katastrophe
s24	1	ehemalige	IBM-Verkäufer
s34	1	autoritärer	Macher
s38	1	politische	Außenseiter
s38	2	gigantische	Haushaltsdefizit
s43	1	nationale	Sicherheit
s46	1	anderer	Fachmann
s54	1	farbloser	Kompromißkandidat
s60	1	britische	Blatt
s63	1	übermächtige	Bürokratie
s76	1	Internationalen	Währungsfonds
s82	1	indischen	Wirtschaft
s83	1	große	Durchbruch
s88	1	miese	Arbeitsmoral

Tables have been updated.

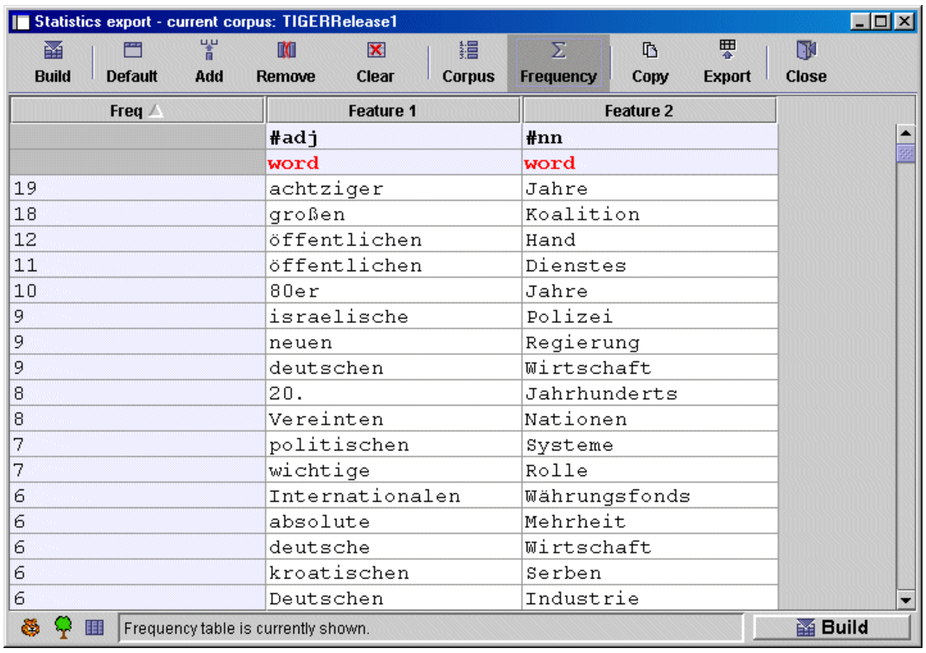
Figure: Building the table

The left two columns of the table show the corpus graph ID and the number of the current submatch (Remember that a query can be matched by a corpus graph more than once.). By default, the rows are ordered corresponding to the ordering of the corpus graphs. If you like to change the sort sequence of the rows, consult subsection 8.4.

To adapt the table layout, you can also change the width of a column or change the column ordering with the help of your mouse device.

8.3 Corpus view and Frequency view

The default view of the statistical viewer is the *Corpus view*, i.e. the rows are ordered with respect to the corpus graph ordering. However, to analyze the data it is sometimes helpful to group indentical rows and display their frequency. To switch to the *Frequency view* click on the *Frequency* button.


Statistics export - current corpus: TIGERRelease1
Build
Default
Add
Remove
Clear
Corpus
Frequency
Copy
Export
Close
Frequency table is currently shown.
Build

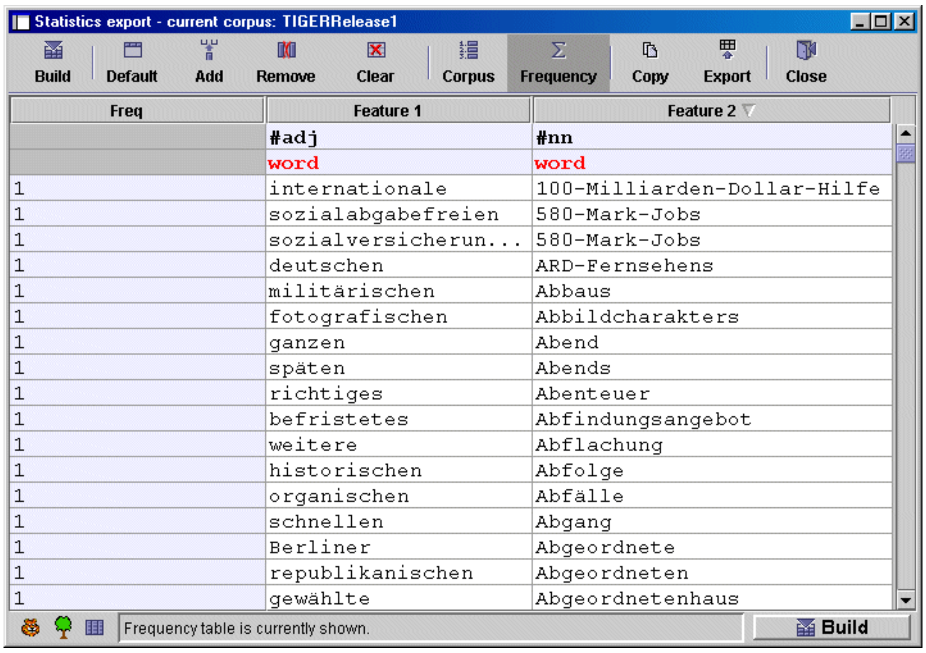
Freq	Feature 1	Feature 2
	#adj	#nn
	word	word
19	achtziger	Jahre
18	großen	Koalition
12	öffentlichen	Hand
11	öffentlichen	Dienstes
10	80er	Jahre
9	israelische	Polizei
9	neuen	Regierung
9	deutschen	Wirtschaft
8	20.	Jahrhunderts
8	Vereinten	Nationen
7	politischen	Systeme
7	wichtige	Rolle
6	Internationalen	Währungsfonds
6	absolute	Mehrheit
6	deutsche	Wirtschaft
6	kroatischen	Serben
6	Deutschen	Industrie

Figure: The frequency view

Now the left column shows the number of occurences of the rows displayed in the table. The rows are ordered by frequency.

8.4 Changing the sort sequence

The ordering of the table rows in both Corpus View and Frequency View can be changed easily. Just double-click on the upper border of the column you like to select as the sort sequence column. Or select the *Sort by column* item in the context menu of the column headline (activated by a right button mouse click on the column headline). In the following example, the second column has been selected. If you double-click the column again, the rows are sorted in reverse order.



Freq	Feature 1	Feature 2
	#adj	#nn
	word	word
1	internationale	100-Milliarden-Dollar-Hilfe
1	sozialabgabefreien	580-Mark-Jobs
1	sozialversicherun...	580-Mark-Jobs
1	deutschen	ARD-Fernsehens
1	militärischen	Abbaus
1	fotografischen	Abbildcharakters
1	ganzen	Abend
1	späten	Abends
1	richtiges	Abenteuer
1	befristetes	Abfindungsangebot
1	weitere	Abflachung
1	historischen	Abfolge
1	organischen	Abfälle
1	schnellen	Abgang
1	Berliner	Abgeordnete
1	republikanischen	Abgeordneten
1	gewählte	Abgeordnetenhaus

Frequency table is currently shown.

Figure: Changing the sort sequence

If you select the *Sort by column* item in the context menu of the GraphID or Submatch column within the Frequency View, the default ordering of the table will be restored.

8.5 Communicating with the GraphViewer

If you are simultaneously working with the GraphViewer and the Statistical Viewer, you can easily switch between these two views using the shortcut icons in the lower left corner of both windows.

However, the two windows have also been designed to communicate with each other. The match currently displayed in the stattistical viewer will be displayed and highlighted in the GraphViewer by double-clicking the corresponding row (mouse device must be on the graph ID or submatch column) or by selecting the *Open match in GraphViewer* item in the context menu of the row (activated by a right mouse button click on the graph ID or submatch column).

8.6 Exporting the results

To export the statistics, first mark the rows you want to export. Use the mouse to mark the rows or select the *Mark all* item in the context menu. To unselect the marked rows, just select the *Clear selection* item in the context menu. Click the export button to display the export dialogue window.

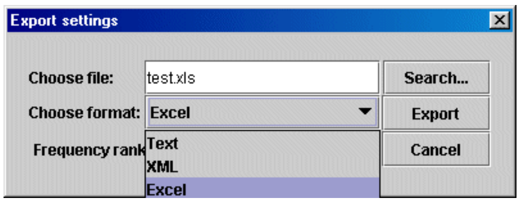


Figure: Exporting the results

We have implemented three export formats:

- Text format

  Columns are separated by tabs, rows are separated by carriage return.

- XML format

  The data is exported in an XML-based format that can be used for further processing.

- Excel format

  The data is exported as an Microsoft Excel table.

As an alternative, you can also copy the text format output into the clipboard. Just select the *Copy* button in the button toolbar.

9. Exporting the matches

9.1 Introduction

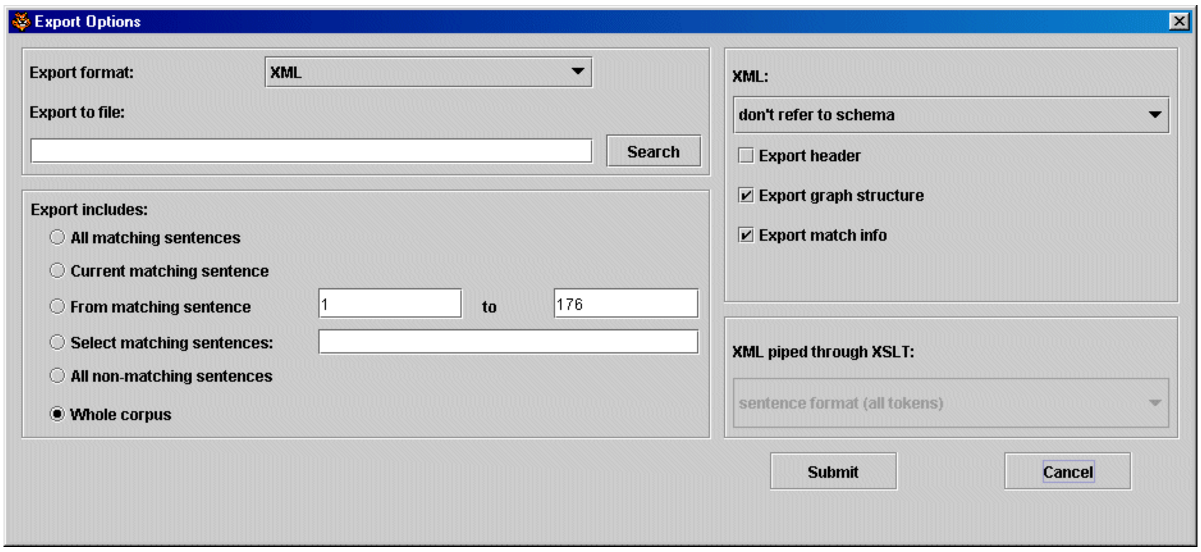
For the export of your favourite matches we have implemented two different approaches. First, you can export single graphs as image data or a set of matching graphs as an animated SVG image (cf. subsection 7.4). Second, TIGERSearch enables you to export matches to XML using its own *TIGER-XML* encoding format (cf. chapter V), or to pipe the TIGER-XML output through an XSLT stylesheet. This section describes how to export a TIGER-XML file (cf. subsection 9.3) and how to pipe an XSLT stylesheet through TIGER-XML output (cf. subsection 9.4).

Please keep in mind that an exported TIGER-XML corpus can be indexed by the TIGERRegistry tool, i.e. exported matches can be reused as a new TIGERSearch corpus!

9.2 Setting up the export mode

After processing a query and viewing its results with the GraphViewer you may want to save your favourite matches for later review or processing. Just choose the *Export Matches* icon in the TIGERSearch main window toolbar or the *Export matches* item in the *Query* menu of the main window. The export feature can also be used if you did not yet submit a query. In this case, you can export the whole corpus.

Next, you have to select the output format: *TIGER-XML format* (cf. subsection 9.3) or *XML* *piped through an XSLT stylesheet* (cf. subsection 9.4).



Export Options

Export format: XML

Export to file:

Search

Export includes:
- ○ All matching sentences
- ○ Current matching sentence
- ○ From matching sentence 1 to 176
- ○ Select matching sentences:
- ○ All non-matching sentences
- [✓ Whole corpus]

XML: don't refer to schema
- ☐ Export header
- ☑ Export graph structure
- ☑ Export match info

XML piped through XSLT: sentence format (all tokens)

Submit Cancel

Figure: Setting up the export mode

Next, you have to specify an output file name. You can either type it in by hand (relative paths are evaluated with regard to the working directory) or use a file dialog by clicking on the *Search* button.

To restrict the export there are several options (cf. screenshot above):

- **All matching corpus graphs**

  no restriction, export all matching corpus graphs

- **Current matching corpus graph**

  export the corpus graph currently displayed in the GraphViewer

- **From matching corpus graph**

  restrict export to a range of corpus graphs

- **Select matching corpus graphs**

  restrict export to a list of matching corpus graphs separated by comma or colon (e.g. `1;3` or `1-2;3-7,19`)

- **All non-matching corpus graphs**

  export all corpus graphs which do **not** match the corpus query

- **Whole corpus**

  export the whole corpus

⚠**Please note:** *Matching corpus graph* in this context means the number (or position) of the graph in the forest of matching corpus graphs. Example: `1;5-9` will export the 1st, 5th, 6th, 7th, 8th, and 9th matching corpus graph, but **not** the corpus graphs with the IDs 1, 5 etc.

Pressing the *Submit* button will start the export process. It can be stopped at any time.

9.3 Exporting to TIGER-XML

If you choose TIGER-XML as the export format, the following options can be specified in the export window.

Schema reference

The structure of a TIGER-XML export file follows the TIGER-XML schema declaration (cf. section 4, chapter V). In the export file you can refer to the schema file:

- on your local computer or network created during the TIGERSearch installation (**refer to local schema**),
- on the TIGERSearch web site (**refer to WWW schema**),
- or make no reference (**don't refer to schema**).

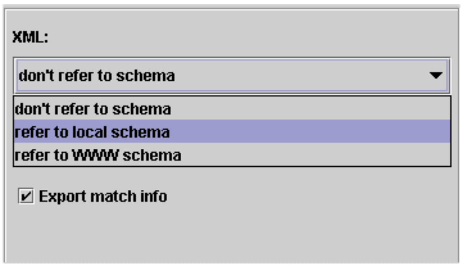


Figure: Schema reference options

Include/exclude options

You can also exclude certain parts of the export file by unchecking some of the boxes:

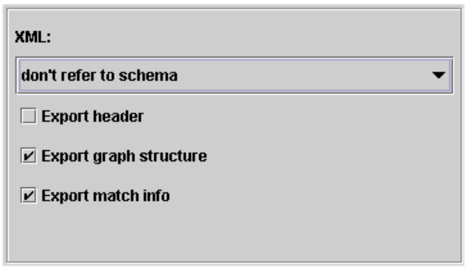


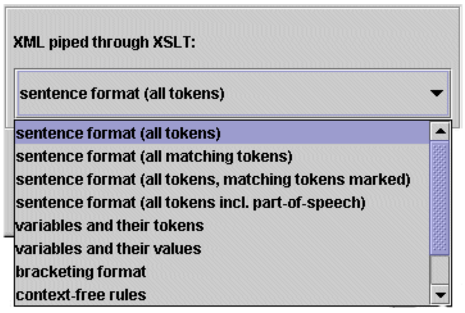
Figure: Include/exclude options

- **Export header:** contains meta information and feature declaration; essential if the exported file should represent a new TIGERSearch corpus; unchecked by default
- **Export graph structure:** tokens, inner nodes, edges etc.
- **Export match info:** indicates which part of a corpus graph actually matches the query

⚠**Please note:** In subsection 2.5, chapter V we describe the encoding of corpus query matches in the TIGER-XML format.

9.4 Exporting with XSLT

Of course, users can first export a TIGER-XML file and afterwards process the XML file with an external stylesheet. However, TIGERSearch offers the feature to do it all in one step. Just choose *XML piped through XSLT* as your export format and choose one of the predefined stylesheets:


XML piped through XSLT:

sentence format (all tokens)

- sentence format (all tokens)
- sentence format (all matching tokens)
- sentence format (all tokens, matching tokens marked)
- sentence format (all tokens incl. part-of-speech)
- variables and their tokens
- variables and their values
- bracketing format
- context-free rules

Figure: Stylesheet selection

TIGERSearch is delivered with several predefined stylesheets. If you have created additional stylesheets you can link your stylesheets into TIGERSearch. The linking mechanism is explained in subsection 10.2.

Some interesting predefined stylesheets are now illustrated by a match of the corpus query `[cat="NP"]`:

- sentence format (all tokens): tokens separated by blanks, sentences separated by line breaks

  ```
  Minister heizt Debatte über Sterbehilfe an
  ```

- sentence format (tokens+pos): same as above, but each token is annotated with a part-of-speech tag in the token/pos format

  ```
  Minister/NN heizt/VVFIN Debatte/NN über/APPR Sterbehilfe/NN an/
  PTKVZ
  ```

- bracketing format: UPenn-style bracketing format

  ```
  ( (S (NN-SB Minister)
       (VVFIN-HD heizt)
       (NP-OA (NN-NK Debatte)
              (PP-MNR (APPR-AC über)
                      (NN-NK Sterbehilfe)))
       (PTKVZ-SVP an)) )
  ```

  ⚠**Please note:** If a corpus graph comprises crossing edges, the tokens of the graph may get disordered. An adequate linguistic representation using traces has not been implemented in this stylesheet.

- context-free rules: lists all rules used in the corpus annotation (dublicates are not removed)

  ```
  PP -> APPR NN
  NP -> NN PP
  S -> NN VVFIN NP PTKVZ
  ```

- corpus graph numbers: list of corpus graph numbers separated by blanks which can be imported by the *Annotate* tool (cf. *http://www.coli.uni-sb.de/sfb378/negra-corpus/annotate.html*)

  ```
  26743 26745 26746 26747 26748 26750 ...
  ```

10. Adapting the TIGERSearch tool

10.1 Image filters

The *Scalable Vector Graphics* format SVG is an XML-based format. Thus, Cascading Stylesheets (CSS) or Extensible Stylesheets (XSLT) can be used to modify an image. The embedding of stylesheets in SVG documents is explained in the SVG specifications located at the W3C web site (*http://www.w3c.org/Graphics/SVG*). The following example comprises an SVG file exported by the TIGERGraphViewer. It illustrates how an Cascading Stylesheet is used to set the background color of an image to white:

```
<svg width="662" height="333">

  <style type="text/css">
    g[type="bgcolor"] > rect { fill:white }
  </style>

  <g type="sentence" id="s10">

    <!-- create background color -->
    <g type="bgcolor">
      <rect x="0" y="0" width="662" height="333" fill="grey"/>
    </g>

    <!-- terminal node "s10_1":[word="There" & pos="EX"] -->
    <g type="t" id="s10_1" match="subgraph" font-family="dialog"
font-style="normal" font-weight="normal" font-size="14"
fill="rgb(150,0,0)">
      <text x="55" y="276" text-anchor="middle">There</text>
      <text x="55" y="297" text-anchor="middle">EX</text>
    </g>
    ...

  </g>

</svg>
```

To develop your own Cascading Stylesheets, we recommend you to export an SVG example corpus graph which makes use of an image filter. The generated SVG file can be used to experiment with your stylesheet. The SVG specification supports Cascading Stylesheets Level 2 which are described in a W3C specification (cf. *http://www.w3c.org/Style/CSS*).

The integration of your own CSS stylesheets is quite simple. All stylesheets that are represented by image filters in the graphical user interface are listed in a file named `tigersearch_svg.xml` which is located in the `config/` subdirectory of your TIGERSearch installation directory. The file looks like the following:

```
<svgfilter version="1.0">

  <filter name="black/white">
     text { fill:black }
     line,rect,ellipse { stroke:black }
     rect { fill:white }
     path { stroke:black }
  </filter>

  <filter name="white background">
    g[type="bgcolor"] > rect { fill:white }
  </filter>

</svgfilter>
```

To add your stylesheets, just insert a new `<filter>` element. Specify the stylesheet name by setting the value of the attribute *name*. Insert the stylesheet text as the element content of the new `<filter>` element.

Be aware of the XML element content conventions, e.g. use `&lt;` instead of < in the content of a `<filter>` element. Please also note that TIGERSearch has to be restarted in order to activate the new stylesheet configuration.

10.2 XSLT stylesheets

All XSLT stylesheets used to transform the export of query matches are placed in the `export/` subdirectory of your TIGERSearch installation directory. If you want to develop a new stylesheet, we recommend you to use the file `testsentence.xml` for testing your stylesheet. As the matching corpus graphs are processed on your stylesheet separately, this test file illustrates the XML representation of one single graph. It is located in the `export/` directory, too.

The XSLT stylesheets offered by the user interface are registered in a file named `TIGERExportRegistry.xml` which is also located in the `export/` subdirectory. This file looks like the following:

```
<registry version="1.0">

  <stylesheet name="bracketing format" sentence_xslt="bracket.xsl"/>

  <stylesheet name="sentence format (tokens)"
```

```
sentence_xslt="sentence.xsl"/>

  <stylesheet name="sentence format (tokens + pos)"
sentence_xslt="pos.xsl"/>

</registry>
```

To add your stylesheet, just insert a new `<stylesheet>` element. Specify the stylesheet name by setting the value of the attribute *name*. Specify the filename by setting the value of the `sentence_xslt` attribute.

⚠**Please note:** TIGERSearch has to be restarted in order to activate the new stylesheet configuration.

Chapter V - The TIGER-XML treebank encoding format

1. Introduction

As there are various formats for the representation of linguistic corpora, there are also a number of formats for the encoding of syntactically annotated corpora: Penn Treebank, Susanne, Negra, and several other formats plus various kinds of parser or chunker output. Since applications like TIGERSearch cannot support all existing formats, it makes sense to define one single interface format for import and export. This format should be general enough to encode as many existing formats as possible. To solve the problem of Unicode character encoding, it is also advantageous to choose an XML-based approach.

The *TIGER-XML* format has been designed as an interface format. It is an XML-based equivalence of the corpus definition sublanguage of the TIGER description language (cf. chapter III). In addition to corpus definitions, the TIGER-XML format can also represent query results.

Any corpus to be processed by the TIGERSearch tool has to be encoded in the TIGER-XML format. For convenience, we have implemented corpus filters (i.e. converters to TIGER-XML) for many popular treebank and parser output formats like bracketing format, PennTreebank format, NeGra format etc. (cf. subsection 3.5, chapter VI for a list of implemented filters).

Why should you read this chapter? First of all, you might have corpora encoded in formats not supported as a corpus filter. So knowledge about TIGER-XML is essential for their conversion. In addition, TIGERSearch supports exporting the matches of a query in the TIGER-XML format. If you like to transform the XML output, e.g. by XSLT stylesheets, you will have to know how the XML document has been designed.

The first section walks through an example of the TIGER-XML format (cf. section 2). In the second section a real-life example is presented (cf. section 3). Finally, you can find a description of the XML schema used to validate TIGER-XML documents (cf. section 4).

If you are interested in the motivations that have influenced the development of the TIGER-XML format, you should have a look at Wolfgang Lezius' Ph.D. thesis [Lezius2002] (in German).

2. A walk through the TIGER-XML format

2.1 TIGER-XML corpora

The following subsections are a kind of guided tour. They walk through the encoding format and explain its properties.

A TIGER-XML document consists of two parts. In the header you can find the corpus declaration and some meta information (cf. subsection 2.2). The body comprises the definition of the corpus graphs and their annotation (cf. subsection 2.3). The corpus body can also be divided into so-called subcorpora (cf. subsection 2.4). Corpus query matches can also be represented within the TIGER-XML format (cf. subsection 2.5).

2.2 Corpus header

The corpus header consists of two parts. General meta information about the corpus is encoded in the first part of the header: name of the corpus, author, date, short description, format and history. This corpus meta information is displayed by the TIGERSearch tool when presenting the corpora available. The ID of the corpus (cf. root element `<corpus>`) should be unique with regard to all indexed corpora.

```
<corpus id="TESTCORPUS">

<head>

  <meta>
    <name>Test corpus</name>
    <author>Wolfgang Lezius</author>
    <date>April 2003</date>
    <description>illustrates the TIGER-XML format</description>
    <format>NeGra format, version 3</format>
    <history>first version</history>
  </meta>
  ...
</head>
...
</corpus>
```

The second part of the corpus header provides information about the features used in the corpus. This feature declaration is obligatory for corpora to be indexed by the TIGERRegistry tool. Feature values and short explanations of the tags might be listed - this kind of meta information will be used by the TIGERSearch GUI as corpus documentation. If it does not make sense to list all the values of a feature in a corpus (e.g. for a feature *word*), the content of the corresponding `feature` element is empty.

In the following example, the feature *word* is declared as a feature of terminal nodes (`T`) and the feature *cat* as a feature of nonterminal nodes (`NT`). If a feature is used in both terminal and nonterminal nodes (e.g. *case*), its domain is called `FREC` (cf. description of the query language; section 8, chapter III). Element content of a feature value declaration is interpreted as an explanation of the feature value. Potential edge labels are declared in an `<edgelabel>` element, secondary edges in an `<secedgelabel>` element.

```
<head>
  ...
  <annotation>

    <feature name="word" domain="T"/>

    <feature name="pos" domain="T">
      <value name="ART">determiner</value>
      <value name="ADV">adverb</value>
      <value name="KOKOM">conjunction</value>
      <value name="NN">noun</value>
      <value name="PIAT">indefinite attributive pronoun</value>
      <value name="VVFIN">finite verb</value>
    </feature>

    <feature name="morph" domain="T">
      <value name="Def.Fem.Nom.Sg"/>
      <value name="Fem.Nom.Sg.*"/>
      <value name="Masc.Akk.Pl.*"/>
      <value name="3.Sg.Pres.Ind"/>
      <value name="--">not bound</value>
    </feature>

    <feature name="cat" domain="NT">
      <value name="AP">adjektive phrase</value>
      <value name="AVP">adverbial phrase</value>
      <value name="NP">noun phrase</value>
      <value name="S">sentence</value>
    </feature>

    <edgelabel>
      <value name="CC">comparative complement</value>
      <value name="CM">comparative concjunction</value>
      <value name="HD">head</value>
      <value name="MO">modifier</value>
      <value name="NK">noun kernel modifier</value>
      <value name="OA">accusative object</value>
      <value name="SB">subject</value>
    </edgelabel>

  </annotation>

</head>
```

2.3 Corpus body

The supported data model is based on so-called *syntax graphs*, i.e. directed acyclic graphs with a single root node. Thus, corpus graphs cannot be encoded by embedding XML elements. As a solution, all terminal and nonterminal nodes are listed and edges are explicitly encoded as elements. The following example illustrates the corpus graph encoding.

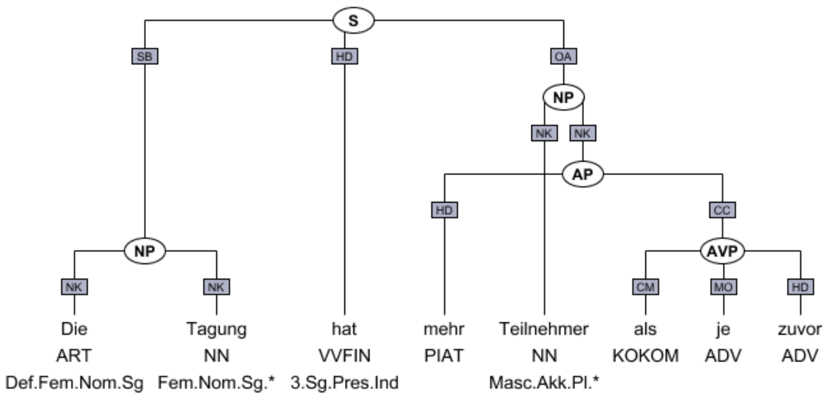


Figure: Example sentence and its annotation

```
<body>

<s id="s5">
  <graph root="s5_504">
    <terminals>
      <t id="s5_1" word="Die" pos="ART" morph="Def.Fem.Nom.Sg"/>
      <t id="s5_2" word="Tagung" pos="NN" morph="Fem.Nom.Sg.*"/>
      <t id="s5_3" word="hat" pos="VVFIN" morph="3.Sg.Pres.Ind"/>
      <t id="s5_4" word="mehr" pos="PIAT" morph="--"/>
      <t id="s5_5" word="Teilnehmer" pos="NN"
morph="Masc.Akk.Pl.*"/>
      <t id="s5_6" word="als" pos="KOKOM" morph="--"/>
      <t id="s5_7" word="je" pos="ADV" morph="--"/>
      <t id="s5_8" word="zuvor" pos="ADV" morph="--"/>
    </terminals>
    <nonterminals>
      <nt id="s5_500" cat="NP">
        <edge label="NK" idref="s5_1"/>
        <edge label="NK" idref="s5_2"/>
      </nt>
      <nt id="s5_501" cat="AVP">
        <edge label="CM" idref="s5_6"/>
        <edge label="MO" idref="s5_7"/>
        <edge label="HD" idref="s5_8"/>
      </nt>
      <nt id="s5_502" cat="AP">
        <edge label="HD" idref="s5_4"/>
        <edge label="CC" idref="s5_501"/>
      </nt>
      <nt id="s5_503" cat="NP">
        <edge label="NK" idref="s5_502"/>
        <edge label="NK" idref="s5_5"/>
      </nt>
      <nt id="s5_504" cat="S">
        <edge label="SB" idref="s5_500"/>
        <edge label="HD" idref="s5_3"/>
        <edge label="OA" idref="s5_503"/>
      </nt>
    </nonterminals>
  </graph>
</s>

</body>
```

⚠**Please note:** Feature values, represented as attribute-value pairs, cannot be omitted. If a feature value or edge label does not make sense for a token or inner node (e.g. in the example sentence the feature *morph* is sometimes unspecified), please use a meaningful symbol instead. We recommend you to use the symbol `--` which is also used in our implemented import filters. When viewing a matching corpus graph using the TIGERGraphViewer, the display of a feature value or edge label such as `--` can be suppressed (cf. subsection 7.5, chapter IV).

2.4 Subcorpora

As a corpus grows, it sometimes needs to be divided into several files. Therefore the concept of subcorpora has been introduced in the TIGER-XML format. In the main corpus a link is placed to a subcorpus. The subcorpus consists of corpus graphs or other embedded subcorpora. It can be validated using the subcorpus subschema of the TIGER-XML format (cf. section 4).

The embedding syntax is the following: Within the corpus body, an element `<subcorpus>` is placed. Its attributes *name* and *external* specify the name of the subcorpus and its URL, respectively.

⚠**Please note:** As the link is represented as an URL, a protocol has to be specified. If the subcorpus is placed within the local file system, use the `file:` protocol. A relative path will be evaluated with regard to the path of the embedding XML file.

The following example illustrates the embedding:

Main corpus (main.xml)

```
<corpus>

  <head>
    ...
  </head>

  <body>
    <subcorpus name="embedded corpus"
external="file:subcorpus.xml"/>
  </body>

</corpus>
```

Subcorpus (subcorpus.xml)

```
<subcorpus name="embedded corpus">

  <s id="s1">
  ...
  </s>

  ...

</subcorpus>
```

2.5 Corpus query matches

To be as flexible as possible, the TIGER-XML format has also been designed to represent corpus query matches. The following example illustrates the encoding of the match information for the query `#v:[cat="NP"] > #w:[pos="NN"]` and the matching corpus graph.

```
<matches>
    <match subgraph="s5_500">
      <variable name="#v" idref="s5_500"/>
      <variable name="#w" idref="s5_2"/>
    </match>
</matches>
```

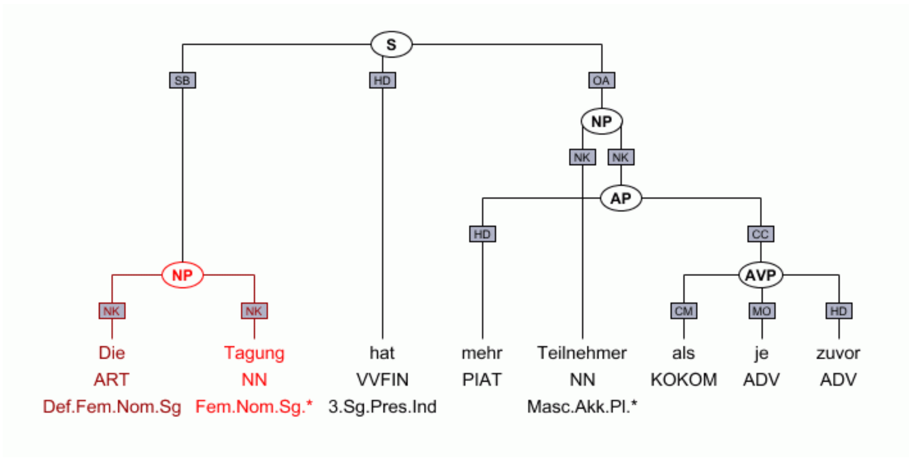


Figure: Example sentence and its match visualization (red-colored)

Matches are represented by `<match>` elements. The `<variable>` elements refer to the corresponding graph nodes matching the variables `#v` and `#w`. Hence the IDs of the `<t>` and `<nt>` elements are essential for both the edge linking and match reference mechanism. The *subgraph* attribute of a `<match>` element refers to the root node of the matching subgraph.

In total, we get the following encoding of the corpus graph and query result:

```
<s id="s5">
  <graph root="s5_504">
    <terminals>
      <t id="s5_1" word="Die" pos="ART" morph="Def.Fem.Nom.Sg"/>
      <t id="s5_2" word="Tagung" pos="NN" morph="Fem.Nom.Sg.*"/>
      <t id="s5_3" word="hat" pos="VVFIN" morph="3.Sg.Pres.Ind"/>
      <t id="s5_4" word="mehr" pos="PIAT" morph="--"/>
      <t id="s5_5" word="Teilnehmer" pos="NN"
morph="Masc.Akk.Pl.*"/>
      <t id="s5_6" word="als" pos="KOKOM" morph="--"/>
      <t id="s5_7" word="je" pos="ADV" morph="--"/>
      <t id="s5_8" word="zuvor" pos="ADV" morph="--"/>
    </terminals>
    <nonterminals>
      <nt id="s5_500" cat="NP">
        <edge label="NK" idref="s5_1"/>
        <edge label="NK" idref="s5_2"/>
      </nt>
      <nt id="s5_501" cat="AVP">
        <edge label="CM" idref="s5_6"/>
        <edge label="MO" idref="s5_7"/>
        <edge label="HD" idref="s5_8"/>
      </nt>
      <nt id="s5_502" cat="AP">
        <edge label="HD" idref="s5_4"/>
        <edge label="CC" idref="s5_501"/>
      </nt>
      <nt id="s5_503" cat="NP">
        <edge label="NK" idref="s5_502"/>
        <edge label="NK" idref="s5_5"/>
      </nt>
      <nt id="s5_504" cat="S">
```

```
        <edge label="SB" idref="s5_500"/>
        <edge label="HD" idref="s5_3"/>
        <edge label="OA" idref="s5_503"/>
      </nt>
    </nonterminals>
  </graph>
  <matches>
    <match subgraph="s5_500">
      <variable name="#w" idref="s5_2"/>
      <variable name="#v" idref="s5_500"/>
    </match>
    <match subgraph="s5_503">
      <variable name="#w" idref="s5_5"/>
      <variable name="#v" idref="s5_503"/>
    </match>
  </matches>
</s>
```

3. Corpus example

The following listing shows the TIGER-XML representation of a small demo corpus. This corpus comprises two corpus graphs of the *Wall Street Journal* corpus of the PennTreebank. It makes use of edge labels and of secondary edges to represent coreference annotation. The TIGER-XML example file `corpus.xml` is also placed in the `doc/examples/` subdirectory of your TIGERSearch installation.

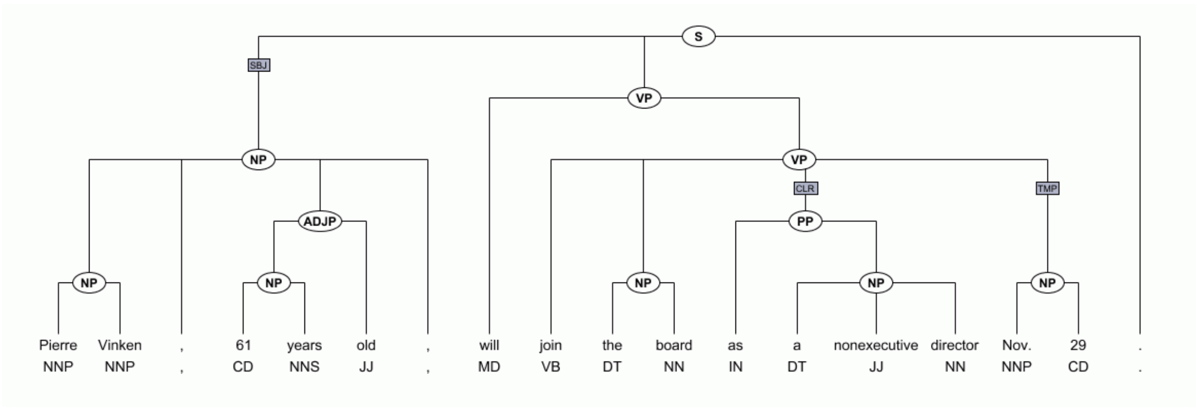


Figure: Sentence 1: Pierre Vinken, 61 years old, will join the board...

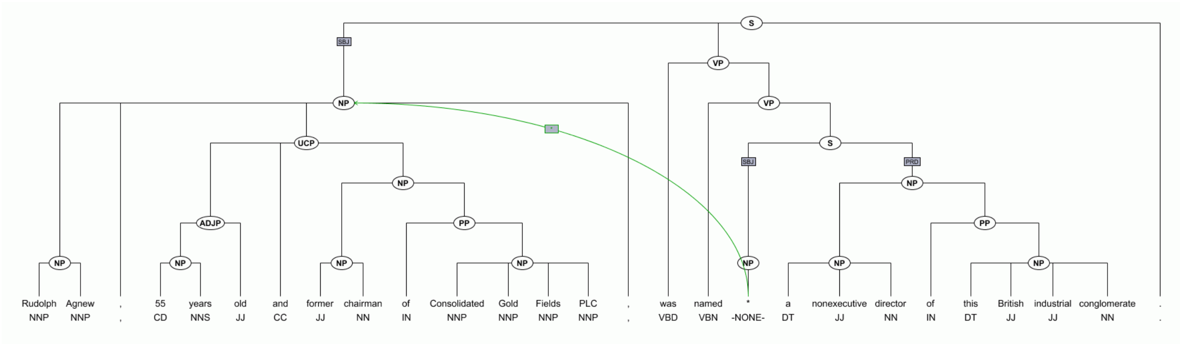


Figure: Sentence 2: Rudolph Agnew, 55 years old and former chairman...

```
<corpus id="DEMO">

<head>
  <meta>
    <name>two sentences of Wall Street Journal corpus</name>
    <format>bracketing format</format>
  </meta>
  <annotation>
    <feature name="word" domain="T"/>
    <feature name="pos" domain="T">
       <value name=","/>
       <value name="-NONE-"/>
       <value name="."/>
       <value name="CC"/>
       <value name="CD"/>
       <value name="DT"/>
       <value name="IN"/>
       <value name="JJ"/>
       <value name="MD"/>
       <value name="NN"/>
       <value name="NNP"/>
       <value name="NNS"/>
       <value name="VB"/>
       <value name="VBD"/>
       <value name="VBN"/>
    </feature>
    <feature name="cat" domain="NT">
       <value name="ADJP"/>
       <value name="NP"/>
       <value name="PP"/>
       <value name="S"/>
       <value name="UCP"/>
       <value name="VP"/>
    </feature>
    <edgelabel>
       <value name="--">not bound</value>
       <value name="CLR"/>
       <value name="PRD"/>
       <value name="SBJ"/>
       <value name="TMP"/>
    </edgelabel>
    <secedgelabel>
       <value name="*"/>
    </secedgelabel>
  </annotation>
</head>

<body>

<s id="s1">
  <graph root="s1_500">
    <terminals>
      <t id="s1_1" word="Pierre" pos="NNP"/>
      <t id="s1_2" word="Vinken" pos="NNP"/>
      <t id="s1_3" word="," pos=","/>
      <t id="s1_4" word="61" pos="CD"/>
      <t id="s1_5" word="years" pos="NNS"/>
      <t id="s1_6" word="old" pos="JJ"/>
      <t id="s1_7" word="," pos=","/>
      <t id="s1_8" word="will" pos="MD"/>
      <t id="s1_9" word="join" pos="VB"/>
      <t id="s1_10" word="the" pos="DT"/>
      <t id="s1_11" word="board" pos="NN"/>
      <t id="s1_12" word="as" pos="IN"/>
      <t id="s1_13" word="a" pos="DT"/>
      <t id="s1_14" word="nonexecutive" pos="JJ"/>
      <t id="s1_15" word="director" pos="NN"/>
      <t id="s1_16" word="Nov." pos="NNP"/>
      <t id="s1_17" word="29" pos="CD"/>
      <t id="s1_18" word="." pos="."/>
    </terminals>
    <nonterminals>
      <nt id="s1_502" cat="NP">
        <edge label="--" idref="s1_1"/>
        <edge label="--" idref="s1_2"/>
      </nt>
      <nt id="s1_504" cat="NP">
        <edge label="--" idref="s1_4"/>
        <edge label="--" idref="s1_5"/>
      </nt>
      <nt id="s1_503" cat="ADJP">
        <edge label="--" idref="s1_504"/>
        <edge label="--" idref="s1_6"/>
      </nt>
      <nt id="s1_501" cat="NP">
        <edge label="--" idref="s1_502"/>
        <edge label="--" idref="s1_3"/>
        <edge label="--" idref="s1_503"/>
        <edge label="--" idref="s1_7"/>
      </nt>
      <nt id="s1_507" cat="NP">
        <edge label="--" idref="s1_10"/>
        <edge label="--" idref="s1_11"/>
      </nt>
      <nt id="s1_509" cat="NP">
        <edge label="--" idref="s1_13"/>
        <edge label="--" idref="s1_14"/>
        <edge label="--" idref="s1_15"/>
      </nt>
      <nt id="s1_508" cat="PP">
        <edge label="--" idref="s1_12"/>
        <edge label="--" idref="s1_509"/>
      </nt>
      <nt id="s1_510" cat="NP">
        <edge label="--" idref="s1_16"/>
        <edge label="--" idref="s1_17"/>
      </nt>
      <nt id="s1_506" cat="VP">
        <edge label="--" idref="s1_9"/>
        <edge label="--" idref="s1_507"/>
        <edge label="CLR" idref="s1_508"/>
        <edge label="TMP" idref="s1_510"/>
      </nt>
      <nt id="s1_505" cat="VP">
        <edge label="--" idref="s1_8"/>
        <edge label="--" idref="s1_506"/>
      </nt>
      <nt id="s1_500" cat="S">
        <edge label="SBJ" idref="s1_501"/>
        <edge label="--" idref="s1_505"/>
        <edge label="--" idref="s1_18"/>
      </nt>
    </nonterminals>
  </graph>
</s>

<s id="s3">
  <graph root="s3_500">
    <terminals>
      <t id="s3_1" word="Rudolph" pos="NNP"/>
      <t id="s3_2" word="Agnew" pos="NNP"/>
      <t id="s3_3" word="," pos=","/>
      <t id="s3_4" word="55" pos="CD"/>
      <t id="s3_5" word="years" pos="NNS"/>
      <t id="s3_6" word="old" pos="JJ"/>
      <t id="s3_7" word="and" pos="CC"/>
      <t id="s3_8" word="former" pos="JJ"/>
      <t id="s3_9" word="chairman" pos="NN"/>
      <t id="s3_10" word="of" pos="IN"/>
      <t id="s3_11" word="Consolidated" pos="NNP"/>
      <t id="s3_12" word="Gold" pos="NNP"/>
      <t id="s3_13" word="Fields" pos="NNP"/>
      <t id="s3_14" word="PLC" pos="NNP"/>
      <t id="s3_15" word="," pos=","/>
      <t id="s3_16" word="was" pos="VBD"/>
      <t id="s3_17" word="named" pos="VBN"/>
      <t id="s3_18" word="*" pos="-NONE-"/>
      <t id="s3_19" word="a" pos="DT"/>
      <t id="s3_20" word="nonexecutive" pos="JJ"/>
      <t id="s3_21" word="director" pos="NN"/>
      <t id="s3_22" word="of" pos="IN"/>
      <t id="s3_23" word="this" pos="DT"/>
      <t id="s3_24" word="British" pos="JJ"/>
      <t id="s3_25" word="industrial" pos="JJ"/>
      <t id="s3_26" word="conglomerate" pos="NN"/>
      <t id="s3_27" word="." pos="."/>
    </terminals>
    <nonterminals>
      <nt id="s3_502" cat="NP">
        <edge label="--" idref="s3_1"/>
        <edge label="--" idref="s3_2"/>
      </nt>
      <nt id="s3_505" cat="NP">
        <edge label="--" idref="s3_4"/>
        <edge label="--" idref="s3_5"/>
      </nt>
      <nt id="s3_504" cat="ADJP">
        <edge label="--" idref="s3_505"/>
        <edge label="--" idref="s3_6"/>
      </nt>
      <nt id="s3_507" cat="NP">
        <edge label="--" idref="s3_8"/>
        <edge label="--" idref="s3_9"/>
      </nt>
      <nt id="s3_509" cat="NP">
        <edge label="--" idref="s3_11"/>
        <edge label="--" idref="s3_12"/>
        <edge label="--" idref="s3_13"/>
        <edge label="--" idref="s3_14"/>
      </nt>
      <nt id="s3_508" cat="PP">
        <edge label="--" idref="s3_10"/>
        <edge label="--" idref="s3_509"/>
      </nt>
      <nt id="s3_506" cat="NP">
        <edge label="--" idref="s3_507"/>
        <edge label="--" idref="s3_508"/>
      </nt>
      <nt id="s3_503" cat="UCP">
        <edge label="--" idref="s3_504"/>
        <edge label="--" idref="s3_7"/>
        <edge label="--" idref="s3_506"/>
      </nt>
      <nt id="s3_501" cat="NP">
        <edge label="--" idref="s3_502"/>
        <edge label="--" idref="s3_3"/>
        <edge label="--" idref="s3_503"/>
        <edge label="--" idref="s3_15"/>
        <secedge label="*" idref="s3_18"/>
      </nt>
      <nt id="s3_513" cat="NP">
        <edge label="--" idref="s3_18"/>
      </nt>
      <nt id="s3_515" cat="NP">
        <edge label="--" idref="s3_19"/>
        <edge label="--" idref="s3_20"/>
        <edge label="--" idref="s3_21"/>
      </nt>
      <nt id="s3_517" cat="NP">
        <edge label="--" idref="s3_23"/>
        <edge label="--" idref="s3_24"/>
        <edge label="--" idref="s3_25"/>
        <edge label="--" idref="s3_26"/>
      </nt>
      <nt id="s3_516" cat="PP">
        <edge label="--" idref="s3_22"/>
        <edge label="--" idref="s3_517"/>
      </nt>
      <nt id="s3_514" cat="NP">
        <edge label="--" idref="s3_515"/>
        <edge label="--" idref="s3_516"/>
      </nt>
      <nt id="s3_512" cat="S">
        <edge label="SBJ" idref="s3_513"/>
        <edge label="PRD" idref="s3_514"/>
      </nt>
      <nt id="s3_511" cat="VP">
        <edge label="--" idref="s3_17"/>
        <edge label="--" idref="s3_512"/>
      </nt>
      <nt id="s3_510" cat="VP">
        <edge label="--" idref="s3_16"/>
        <edge label="--" idref="s3_511"/>
      </nt>
      <nt id="s3_500" cat="S">
        <edge label="SBJ" idref="s3_501"/>
        <edge label="--" idref="s3_510"/>
        <edge label="--" idref="s3_27"/>
      </nt>
    </nonterminals>
  </graph>
</s>

</body>

</corpus>
```

4. The TIGER-XML schema

The TIGER-XML format is validated against an XML schema. XML schema validation is supported by all major XML parsers. The schema is divided into three parts: the main schema, the subschema for the corpus header, and the subschema for subcorpora. The TIGER-XML schema and its two subschemas are placed in the `schema/` subdirectory of your TIGERSearch installation.

Part 1: Main schema - TigerXML.xsd

```
<schema>

 <!-- ================================================================
      XML Schema for the TIGER-XML format
      http://www.ims.uni-stuttgart.de/projekte/TIGER/public/TigerXML.xsd
      ================================================================
      TIGER Project, Wolfgang Lezius
      IMS, University of Stuttgart, 04/01/2003
      ================================================================ -->


  <!-- ======================================================
       INCLUDES DECLARATION OF THE HEADER
       ====================================================== -->
  <include schemaLocation="TigerXMLHeader.xsd"/>


  <!-- ======================================================
       INCLUDES DECLARATION OF SUBCORPORA AND SENTENCES
       ====================================================== -->
  <include schemaLocation="TigerXMLSubcorpus.xsd"/>


  <!-- ======================================================
       DECLARATION OF THE CORPUS DOCUMENT
       ====================================================== -->

  <!-- declaration of the root element: corpus -->

  <element name="corpus">

    <complexType>

      <sequence>

        <choice>
```

```
          <!-- header of the document is optional -->
          <element name="head" type="headType" minOccurs="0" maxOccurs="1"/>
        </choice>

        <element name="body" type="bodyType" minOccurs="1" maxOccurs="1"/>

      </sequence>

      <!-- corpus ID -->
      <attribute name="id" type="idType" use="required"/>

      <!-- optional attribute: TigerXML version; used by TIGERSearch only -->
      <attribute name="version" type="xsd:string" use="optional"/>

    </complexType>

  </element>


  <!-- declaration of the body type -->

  <complexType name="bodyType">

    <choice minOccurs="1" maxOccurs="unbounded">
      <element name="subcorpus" type="subcorpusType" minOccurs="1" maxOccurs="1"/>
      <element name="s" type="sentenceType" minOccurs="1" maxOccurs="1"/>
    </choice>

  </complexType>


</schema>
```

Part 2: Subschema for the corpus header - TigerXMLHeader.xsd

```
<schema>

 <!-- ======================================================================
      XML SubSchema for the header part of the TIGER-XML format
      http://www.ims.uni-stuttgart.de/projekte/TIGER/publicTigerXMLHeader.xsd
      ======================================================================
      TIGER Project, Wolfgang Lezius
      IMS, University of Stuttgart, 04/01/2003
      ====================================================================== -->


  <!-- =====================================================
       DECLARATION OF THE HEADER
       ===================================================== -->


  <!-- declaration of the head element -->

  <element name="head" type="headType"/>


  <!-- declaration of the header type -->

  <complexType name="headType">

     <sequence>
        <element name="meta" type="metaType" minOccurs="0" maxOccurs="1"/>
        <element name="annotation" type="annotationType" minOccurs="0"
maxOccurs="1"/>
     </sequence>

     <!-- optional: reference to external header file

          The header of a TigerXML corpus can also be stored in separate file.
          This attribute points to the external header file. The pointer is
          an URI. Examples: file:relative.xml or file:/path/to/absolute.xml

          Note: If there is a pointer to an external file, the head
                element must be empty. -->
     <attribute name="external" type="xsd:anyURI"/>

  </complexType>


  <!-- declaration of the meta information type -->

  <complexType name="metaType">

    <sequence>
      <element name="name" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="author" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="date" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="description" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="format" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="history" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    </sequence>

  </complexType>


  <!-- declaration of the annotation type -->

  <complexType name="annotationType">

    <sequence>
      <element name="feature" type="featureType" minOccurs="1"
maxOccurs="unbounded"/>
      <element name="edgelabel" type="edgelabelType" minOccurs="0" maxOccurs="1"/>
      <element name="secedgelabel" type="edgelabelType" minOccurs="0"
maxOccurs="1"/>
    </sequence>

  </complexType>


  <!-- declaration of the feature type -->

  <complexType name="featureType">

    <sequence>
       <element name="value" type="featurevalueType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>

    <attribute name="name" type="featurenameType" use="required"/>

    <attribute name="domain" use="required">
       <simpleType>
         <restriction base="xsd:string">
           <enumeration value="T"/>     <!-- feature for terminal nodes -->
           <enumeration value="NT"/>    <!-- feature for nonterminal nodes -->
           <enumeration value="FREC"/>  <!-- feature for both -->
         </restriction>
       </simpleType>
    </attribute>

  </complexType>


  <!-- declaration of the (secondary) edge label type -->

  <complexType name="edgelabelType">

    <sequence>
       <element name="value" type="featurevalueType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>

  </complexType>


  <!-- declaration of the feature value type -->
  <complexType name="featurevalueType">

    <simpleContent>   <!-- element content: documentation of the feature value -->
      <extension base="xsd:string">
        <attribute name="name" type="xsd:string"/>
      </extension>
    </simpleContent>


  </complexType>


  <!-- ========================================================
       HEADER DECLARATIONS THAT SHOULD BE REFINED
       ======================================================== -->

  <!-- declaration of the FEATURE NAMES used in the corpus header;
       this type is unrestricted, but should be refined by a
       specialised, corpus-dependent schema -->

  <simpleType name="featurenameType">

    <restriction base="xsd:string">
      <minLength value="1"/>
      <maxLength value="20"/>
      <whiteSpace value="preserve"/>
    </restriction>

  </simpleType>


</schema>
```

Part 3: Subschema for subcorpora - TigerXMLSubcorpus.xsd

```
<schema>

 <!-- ===========================================================================
      XML Schema for the subcorpus part of the TIGER-XML format
      http://www.ims.uni-stuttgart.de/projekte/TIGER/public/TigerXMLSubcorpus.xsd
      ===========================================================================
      TIGER Project, Wolfgang Lezius
      IMS, University of Stuttgart, 04/01/2003
      ===========================================================================
-->

  <!-- ========================================================
       DECLARATION OF SUBCORPORA AND SENTENCES
       ======================================================== -->


  <!-- declaration of the subcorpus element -->

  <element name="subcorpus" type="subcorpusType"/>


  <!-- declaration of the subcorpus type -->

  <complexType name="subcorpusType">

    <!-- A subcorpus may comprise another subcorpora or sentences -->

    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="subcorpus" type="subcorpusType" minOccurs="1" maxOccurs="1"/>
      <element name="s" type="sentenceType" minOccurs="1" maxOccurs="1"/>
    </choice>

    <!-- required: subcorpus name -->

    <attribute name="name" type="xsd:string" use="required"/>

    <!-- optional: reference to external subcorpus file

         A subcorpus of a TigerXML corpus can also be stored in separate file.
         This attribute points to the external subcorpus file. The pointer is
         an URI. Examples: file:relative.xml or file:/path/to/absolute.xml

         Note: If there is a pointer to an external file, the subcorpus
               element must be empty. -->

    <attribute name="external" type="xsd:anyURI"/>

  </complexType>


  <!-- declaration of the sentence type -->

  <complexType name="sentenceType">

    <sequence>
      <element name="graph" type="graphType" minOccurs="0" maxOccurs="1"/>
      <element name="matches" type="matchesType" minOccurs="0" maxOccurs="1"/>
    </sequence>

    <attribute name="id" type="idType" use="required"/>

  </complexType>


  <!-- declaration of the graph type -->

  <complexType name="graphType">

    <sequence>
      <element name="terminals" type="terminalsType" minOccurs="1" maxOccurs="1"/>
      <element name="nonterminals" type="nonterminalsType" minOccurs="1"
maxOccurs="1"/>
    </sequence>

    <attribute name="root" type="idrefType" use="required"/>

    <!-- indicated that the exported sentence is discontinuous -->
    <attribute name="discontinuous" type="xsd:boolean" default="false"
use="optional"/>

  </complexType>


  <!-- declaration of the terminals type -->

  <complexType name="terminalsType">

    <sequence>
      <element name="t" type="tType" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>

  </complexType>


  <!-- declaration of the t element -->

  <complexType name="tType">

    <!-- secondary edges possible -->
    <sequence>
      <element name="secedge" type="secedgeType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>

    <attribute name="id" type="idType" use="required"/>
    <attributeGroup ref="tfeatureAttributes"/>

  </complexType>


  <!-- declaration of the nonterminals type -->

  <complexType name="nonterminalsType">
```

```
    <sequence>
      <element name="nt" type="ntType" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>

  </complexType>


  <!-- declaration of the nt element -->

  <complexType name="ntType">

    <!-- edge and secondary edges possible -->
    <sequence>
      <element name="edge" type="edgeType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="secedge" type="secedgeType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>

    <attribute name="id" type="idType" use="required"/>
    <attributeGroup ref="ntfeatureAttributes"/>

  </complexType>


  <!-- declaration of the edge type -->

  <complexType name="edgeType">

    <attribute name="idref" type="idrefType" use="required"/>

    <attributeGroup ref="edgelabelAttribute"/>

  </complexType>


  <!-- declaration of the secondary edge type -->

  <complexType name="secedgeType">

    <attribute name="idref" type="idrefType" use="required"/>

    <attributeGroup ref="secedgelabelAttribute"/>

  </complexType>


  <!-- declaration of the matches type -->

  <complexType name="matchesType">

    <sequence>
      <element name="match" type="matchType" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>

  </complexType>


  <!-- declaration of the match type -->

  <complexType name="matchType">

    <sequence>
      <element name="variable" type="varType" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>

    <attribute name="subgraph" type="idrefType" use="required"/>

  </complexType>


  <!-- declaration of the variable type -->

  <complexType name="varType">

    <attribute name="name" type="xsd:string" use="required"/>
    <attribute name="idref" type="idrefType" use="required"/>

  </complexType>


  <!-- =====================================================
       SENTENCE DECLARATIONS THAT SHOULD BE REFINED
       ===================================================== -->

  <!-- declaration of the TERMINAL FEATURE ATTRIBUTES;
       this group is unrestricted, but should be refined by a
       specialised, corpus-dependent schema -->

  <attributeGroup name="tfeatureAttributes">

    <anyAttribute processContents="skip"/>

  </attributeGroup>


  <!-- declaration of the NONTERMINAL FEATURE ATTRIBUTES;
       this group is unrestricted, but should be refined by a
       specialised, corpus-dependent schema -->

  <attributeGroup name="ntfeatureAttributes">

    <anyAttribute processContents="skip"/>

  </attributeGroup>


  <!-- declaration of the EDGE-LABEL ATTRIBUTE;
       the label attribute is optional which should be refined by a
       specialised, corpus-dependent schema -->

  <attributeGroup name="edgelabelAttribute">

    <attribute name="label" type="xsd:string" use="optional"/>

  </attributeGroup>


  <!-- declaration of the SECONDARY-EDGE-LABEL ATTRIBUTE;
       the label attribute is optional which should be refined by a
       specialised, corpus-dependent schema -->

  <attributeGroup name="secedgelabelAttribute">

    <attribute name="label" type="xsd:string" use="optional"/>

  </attributeGroup>


  <!-- =====================================================
       ID and IDREF TYPE DECLARATIONS
       ===================================================== -->

  <!-- Even though XML Schema are a W3C Recommendation, schema
       support of XML parsers is still restricted. Using some
       parsers you might have problems with the ID and IDREF
       attributes in combination with an "anyAttribute"
       declaration. In this case, just modify the base type
       of the following two declarations to "xsd:string".  -->


  <!-- declaration of idType -->

  <simpleType name="idType">

    <restriction base="xsd:ID"/>

  </simpleType>
```

```
  <!-- declaration of idrefType -->
  <simpleType name="idrefType">
    <restriction base="xsd:IDREF"/>
  </simpleType>

</schema>
```

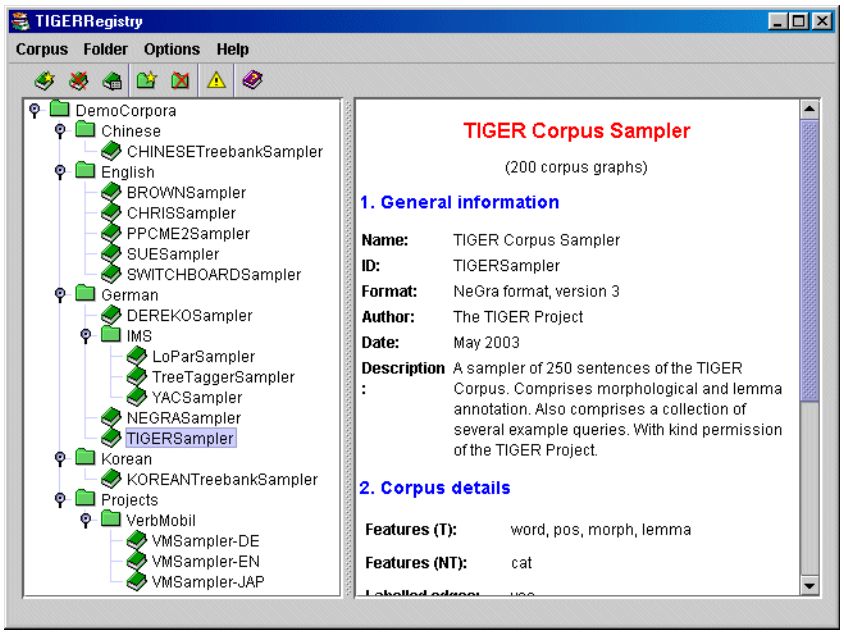
Chapter VI - The TIGERRegistry administration tool

1. An introduction to TIGERRegistry

1.1 Corpus administration

Treebanks to be processed by the TIGERSearch search engine have to be converted into a binary representation first - the so-called index. This index-based approach splits working with the TIGERSearch software suite into two parts: The first part deals with corpus indexing (TIGERRegistry, described in the present chapter), the second part with corpus query processing (TIGERSearch, described in chapter IV). The TIGERSearch corpus query processor can only process corpora that have already been indexed.

The indexed corpora are organized in a hierarchical file system: Each corpus is stored in a folder (i.e. in a directory of your local file system), and corpus folders can be grouped in a common folder as well. The following example illustrates the physical content of a corpus directory and its graphical tree representation. Corpus folders are represented as folder icons and corpora are represented as book icons.



TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name: TIGER Corpus Sampler
ID: TIGERSampler
Format: NeGra format, version 3
Author: The TIGER Project
Date: May 2003
Description: A sampler of 250 sentences of the TIGER Corpus. Comprises morphological and lemma annotation. Also comprises a collection of several example queries. With kind permission of the TIGER Project.

2. Corpus details

Features (T): word, pos, morph, lemma
Features (NT): cat

Figure: Graphical tree representation of the corpus directory

```
DemoCorpora/
DemoCorpora/Chinese/
DemoCorpora/Chinese/CHINESETreebankSampler
DemoCorpora/English/
DemoCorpora/English/BROWNSampler
DemoCorpora/English/CHRISSampler
DemoCorpora/English/PPCME2Sampler
DemoCorpora/English/SUESampler
DemoCorpora/English/SWITCHBOARDSampler
```

```
DemoCorpora/German/
DemoCorpora/German/DEREKOSampler
DemoCorpora/German/IMS/
DemoCorpora/German/IMS/LoParSampler
DemoCorpora/German/IMS/TreeTaggerSampler
DemoCorpora/German/IMS/YACSampler
DemoCorpora/German/NEGRASampler
DemoCorpora/German/TIGERSampler
DemoCorpora/Korean/
DemoCorpora/Korean/KOREANTreebankSampler
DemoCorpora/Projects/
DemoCorpora/Projects/VerbMobil/
DemoCorpora/Projects/VerbMobil/VMSampler-DE
DemoCorpora/Projects/VerbMobil/VMSampler-EN
DemoCorpora/Projects/VerbMobil/VMSampler-JAP
```

1.2 Corpus formats

TIGERSearch supports many existing corpus encoding formats, based on a two-step approach. Since the data model supported by TIGERSearch is more general and expandable than most data models of formats available (PennTreebank format, Negra format, etc.), we have developed the TIGER-XML format. This format maps the supported data model to XML. The TIGER-XML format is described in chapter V. Corpora to be indexed with the TIGERRegistry tool must be encoded in the TIGER-XML format.

To support as many treebank formats as possible, we have also implemented import filters (i.e. converters to TIGER-XML) for many popular formats such as Penn Treebank format or Negra format, and some parser output formats. Indexing of TIGER-XML corpora and indexing of corpora with an import filter are described in subsection 3.2 and subsection 3.3, respectively. A list of implemented import filters can be found in subsection 3.5.

2. Starting TIGERRegistry

2.1 Starting the TIGERRegistry tool

The way you can start the TIGERRegistry tool depends on your operating system. On Windows machines, a program group called TIGERSearch has been created during the installation - so you just have to select the TIGERRegistry program in the start menu.

On Unix machines, symbolic links have been created. If your general path is set properly, you may just need to type in `TIGERRegistry`. However, the TIGERRegistry start program can always be found in the TIGERSearch installation path:

```
INSTALLATIONPATH/bin/TIGERRegistry
```

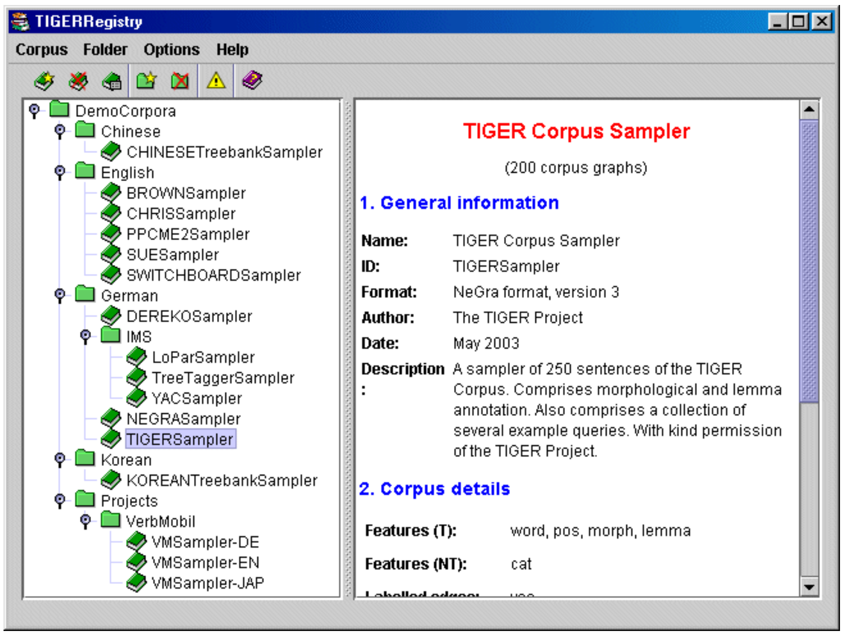
⚠**Please note:** Relative paths specified by the user are evaluated with regard to the working directory. On Unix machines this directory is defined as the TIGERSearch starting directory (i.e. the directory TIGERSearch has been started from). On Mac and Windows machines the working directory is defined as the user's home directory.

2.2 Corpus administration window

When you start the TIGERRegistry tool, it first checks whether you have the permissions to read, write, and create files in the corpus directory. If you do not have the required permissions, an information window pops up. The TIGERRegistry tool will not be started.

If the permission check has been successful, the TIGERRegistry main window appears (cf. screenshot). Position and size of the window are saved when leaving the tool. So the arrangement of your windows will be restored in the next TIGERRegistry session.

The TIGERSearch User's Manual can be accessed directly within the TIGERRegistry user interface. The TIGERRegistry help window can be activated by pressing the *Help* button in the upper toolbar or selecting one of the items in the *Help* menu.



TIGER Corpus Sampler

(200 corpus graphs)

1. General information

Name:	TIGER Corpus Sampler
ID:	TIGERSampler
Format:	NeGra format, version 3
Author:	The TIGER Project
Date:	May 2003
Description:	A sampler of 250 sentences of the TIGER Corpus. Comprises morphological and lemma annotation. Also comprises a collection of several example queries. With kind permission of the TIGER Project.

2. Corpus details

Features (T):	word, pos, morph, lemma
Features (NT):	cat

Figure: TIGERRegistry main window

Now you can browse through the corpus tree (left hand side of the window) and have a look at the corpus properties (right hand side). Just click on a corpus symbol to see the corresponding corpus properties. All operations on this corpus tree (insert a new corpus, delete a corpus etc.) can be activated by pressing the appropriate button in the toolbar, or selecting the appropriate item in the popup-menu (activated by clicking the right mouse button on the corpus symbol), or selecting the corresponding item in the menu bar.

⚠**Please note:** In contrast to file management tools there is no *Undo* function available in the TIGERRegistry tool!

2.3 Folders

To **insert** a new folder (which can contain corpora and other folders) first mark the parent folder of the new folder (in the following figure the folder *Projects* has been marked). Now click the *Insert folder* button in the button toolbar. A new window pops up. Please type in the name of your folder (here: *MyFolder*) and press the *Save* button.

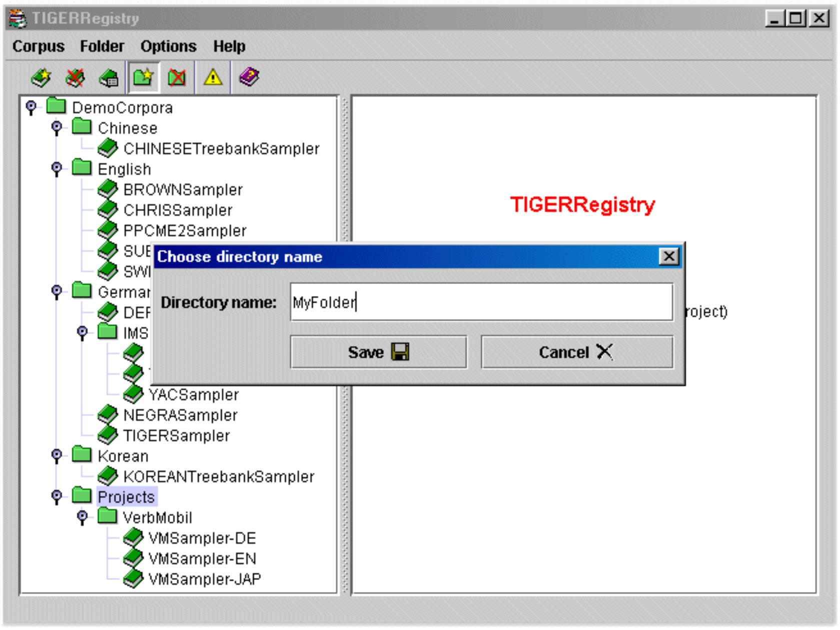


Figure: Inserting a new folder

To **delete** a folder mark it and press the *Delete folder* button in the button toolbar. The folder and its subfolders will be deleted.

To **move** a folder into a new parent folder, use the drag and drop feature of the TIGERRegistry tool: Click the left mouse button on the folder, keep the mouse button pressed (*drag*), move the folder to the new parent folder, and release the mouse button (*drop*).

2.4 Corpora

Please read section 3, section 4, and subsection 3.4 for details about inserting a corpus, changing corpus properties, and viewing corpus log files.

To **delete** a corpus, first mark the corresponding corpus symbol in the corpus administration tree (cf. corpus *TESTCORPUS* in the screenshot). Now click the *Delete corpus* symbol in the button toolbar. A corpus deletion always has to be confirmed:

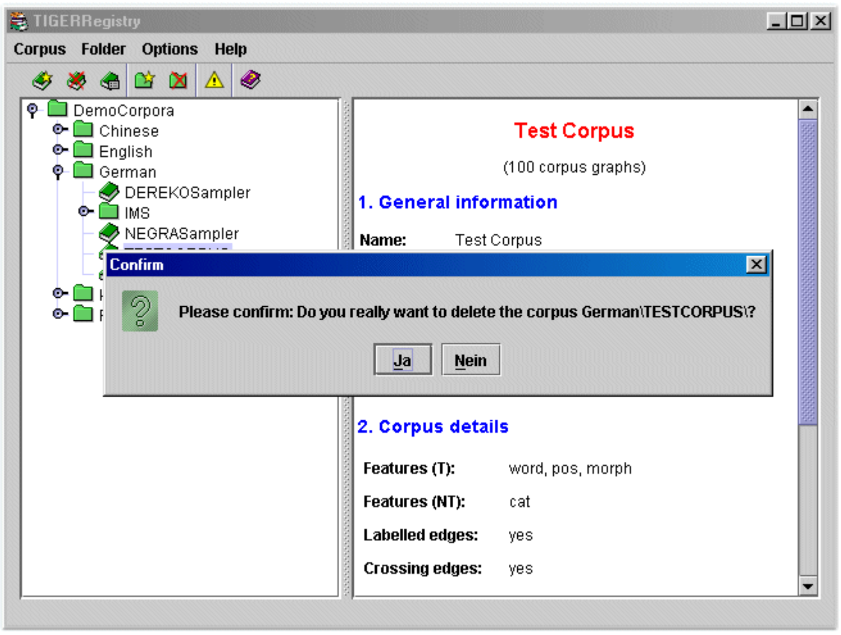


Figure: Deletion of a corpus

Please note: Corpus deletion will delete all data of the selected corpus on the hard disc.

To **move** a corpus into a new parent folder, use the drag and drop feature of the TIGERRegistry tool: Click the left mouse button on the corpus, keep the mouse button pressed (*drag*), move the corpus to the new parent folder, and release the mouse button (*drop*).

2.5 Consistency check

The consistency check runs through the corpus administration tree and checks whether all corpus IDs are distinct. An inconsistency may be caused by corpus administrators with different user rights. To start the consistency check click the corresponding button in the button tool bar, which is presented as a yellow warning symbol.

3. Corpus indexing

3.1 Introduction

The indexing of corpora is based on the TIGER-XML format. Corpora encoded in other formats have to be converted to TIGER-XML first. So if your corpus source file is not encoded in TIGER-XML format, you will have to use one of the existing corpus format filters (i.e. converters to TIGER-XML, cf. subsection 3.5) or convert your corpus to TIGER-XML on your own.

To index a corpus, first mark the parent folder of the new corpus (in the example: *German*). Now click the *Insert Corpus* button in the button toolbar or choose the *Insert Corpus* item in the popup menu (right mouse click):

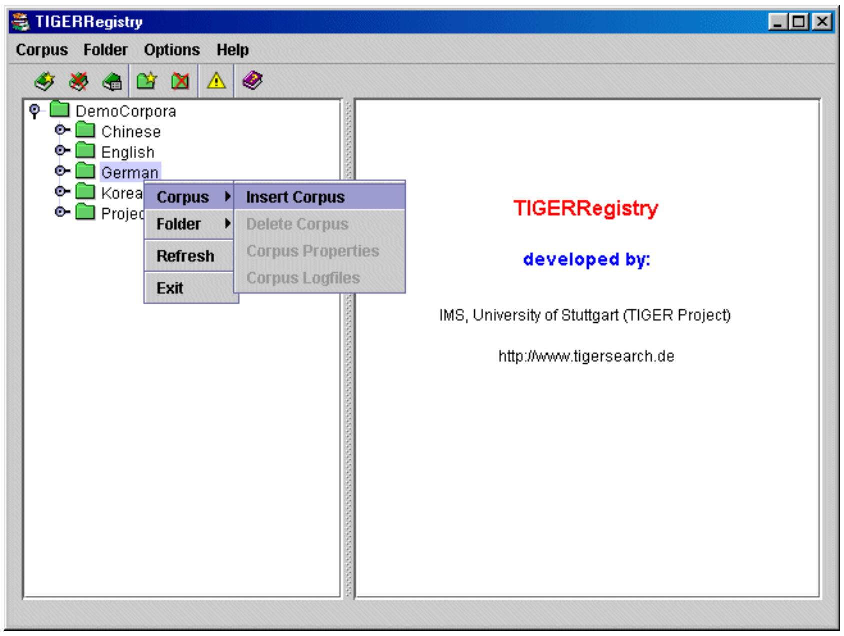
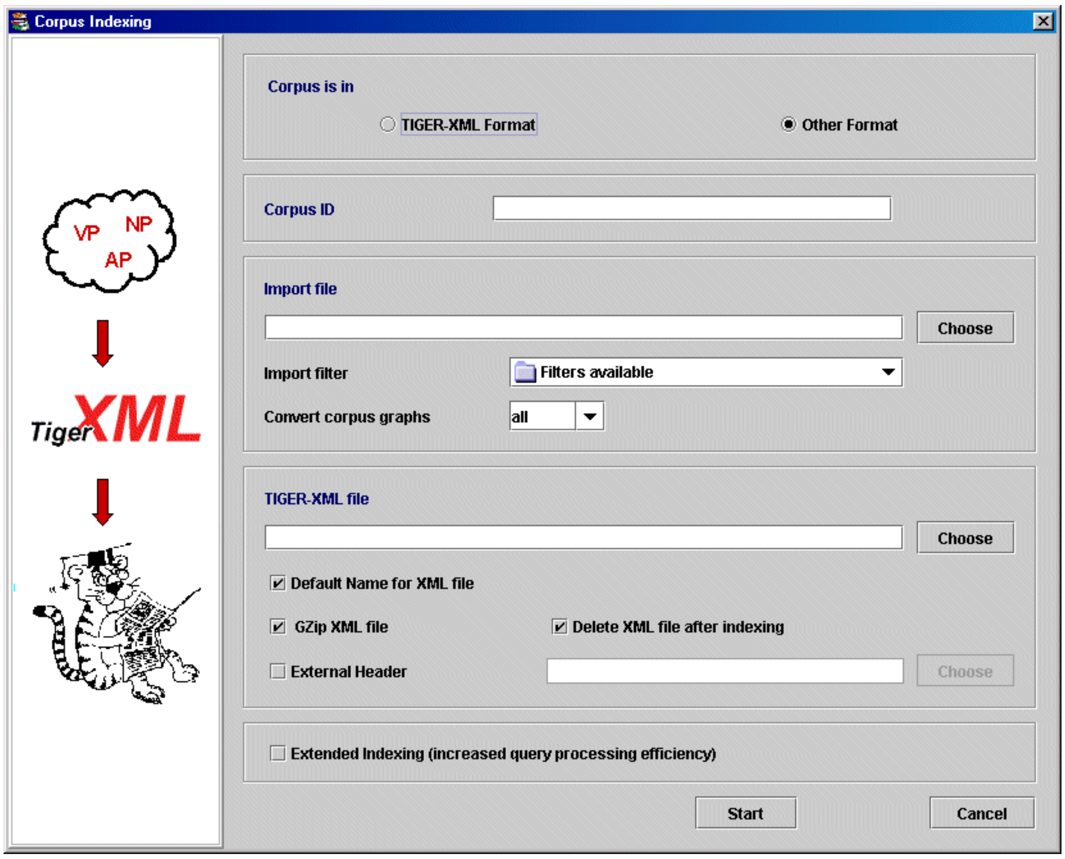


Figure: Inserting a new corpus

Next the corpus indexing window pops up. First of all, you have to specify the corpus input format: *TIGER-XML Format* or *Other Format*:


Corpus Indexing

Corpus is in

( ) TIGER-XML Format (●) Other Format

Corpus ID

Import file

Import filter: Filters available

Convert corpus graphs: all

TIGER-XML file

[✓] Default Name for XML file

[✓] GZip XML file [✓] Delete XML file after indexing

[ ] External Header

[ ] Extended Indexing (increased query processing efficiency)

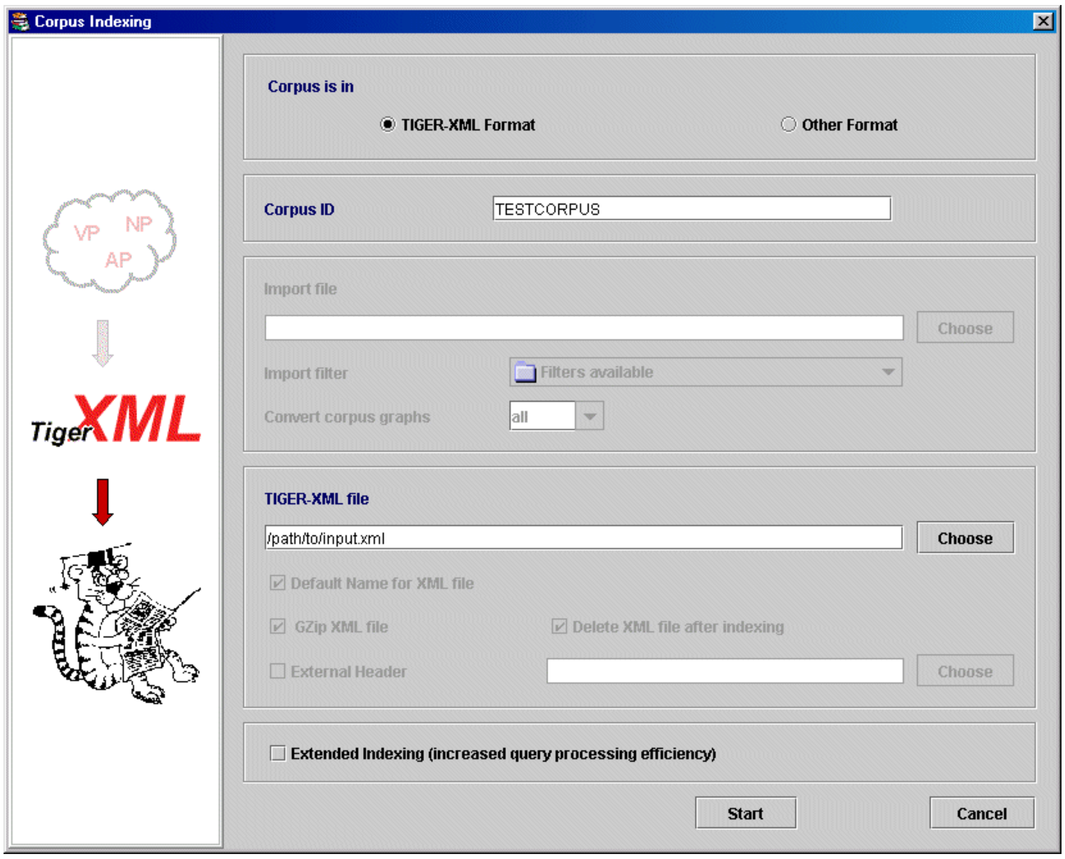
Figure: Corpus indexing window

The additional parameters of the indexing windows are explained in the following subsections (cf. subsection 3.2 and subsection 3.3).

⚠**Please note:** During the corpus indexing process a corpus directory, which comprises several corpus files, is generated. The directory and the files in it are created in a platform-independent way. So if you are working on a platform that allows for fine-grained user permissions (e.g. Unix), you should check the permissions of the new corpus directory right after the indexing process has finished in order to make sure that the desired group of TIGERSearch users will be able to access the newly created corpus.

3.2 Indexing of TIGER-XML files

If your corpus source is encoded in TIGER-XML format, please mark *Corpus is in TIGER-XML format* at the top of the window. Selecting this option deactivates some parameters of the window:



Corpus is in: [✓ TIGER-XML Format] Other Format

Corpus ID: TESTCORPUS

Import file:

Import filter: Filters available

Convert corpus graphs: all

TIGER-XML file: /path/to/input.xml

☑ Default Name for XML file

☑ GZip XML file

☑ Delete XML file after indexing

☐ External Header

☐ Extended Indexing (increased query processing efficiency)

Figure: Indexing parameters (TIGER-XML corpus input)

Now you have to specify the following parameters:

Corpus ID

The corpus ID is used by the TIGERSearch software suite to realize corpus-dependent configurations. The corpus ID must be unique with regard to all other indexed corpora. The uniqueness is checked before the indexing process is initiated. The ID has to start with a letter.

TIGER-XML file

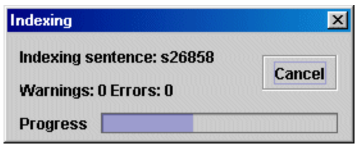
The source file (relative paths are evaluated with regard to the working directory) can be either an uncompressed `.xml` file, or a compressed `.xml.gz` file, or a `.zip` file that contains one source file only. Compressed files are automatically decompressed during the indexing process.

Extended indexing

If extended indexing is activated, additional corpus information is retreived during the indexing process. This information is used to improve corpus query processing efficiency. Effency will increase about 50% at the expense of main memory requirement which also increases about 50%.

⚠**Please note:** Default indexing requires a constant amount of main memory (about 128 MB). The main memory requirement of the extended indexing process will depend on corpus size. If an out of memory warning is displayed, please modify the main memory configuration of the TIGERRegistry tool (cf. section 4, chapter II).

After specifying the indexing parameters, you can start the indexing process by pressing the *Start* button. The corpus indexing can be stopped at any time. The current progress of the indexing is displayed by the indexing progress window:

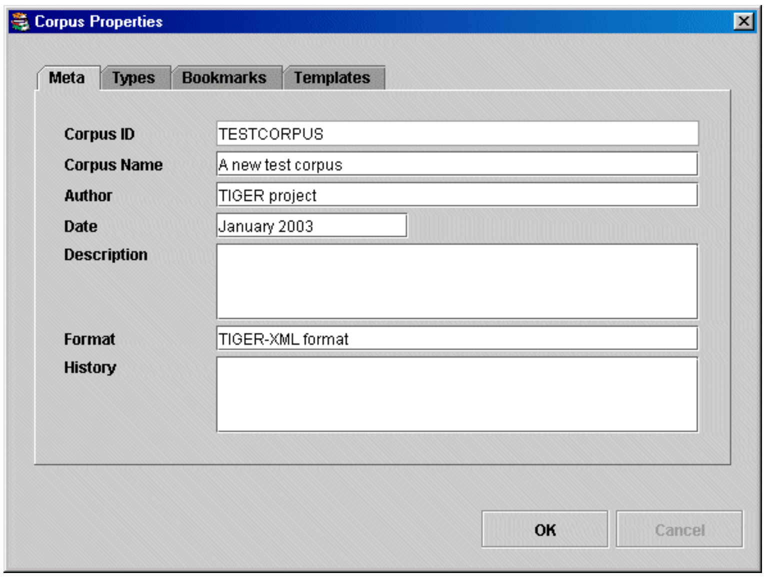

Indexing sentence: s26858

Warnings: 0 Errors: 0

Figure: Indexing progress window

The progress window also shows how many warnings and errors occured during corpus indexing. These messages are stored in the corpus log file `indexing.log` which is placed in the corpus directory. In subsection 3.4 we desribe how to view this log file within the TIGERRegistry application.

When the indexing process is finished, the *Corpus properties* window pops up (see screenshot below). Here you can specify meta information about the corpus such as the corpus name. The corpus properties window is explained in detail in section 4.



Corpus ID	TESTCORPUS
Corpus Name	A new test corpus
Author	TIGER project
Date	January 2003
Description	
Format	TIGER-XML format
History	

Figure: Corpus properties window

After the corpus properties specification, just press the *OK* button to finish corpus indexing. Now the new corpus can be found in the corpus tree.

3.3 Indexing of non TIGER-XML files

Indexing of non TIGER-XML files consists of two steps: First of all, the corpus is converted to TIGER-XML. Afterwards the generated TIGER-XML corpus is indexed. Thus, you have to choose the *Corpus is in Other Format* option at the top of the window and specify the following parameters (cf. screenshot):

Corpus ID

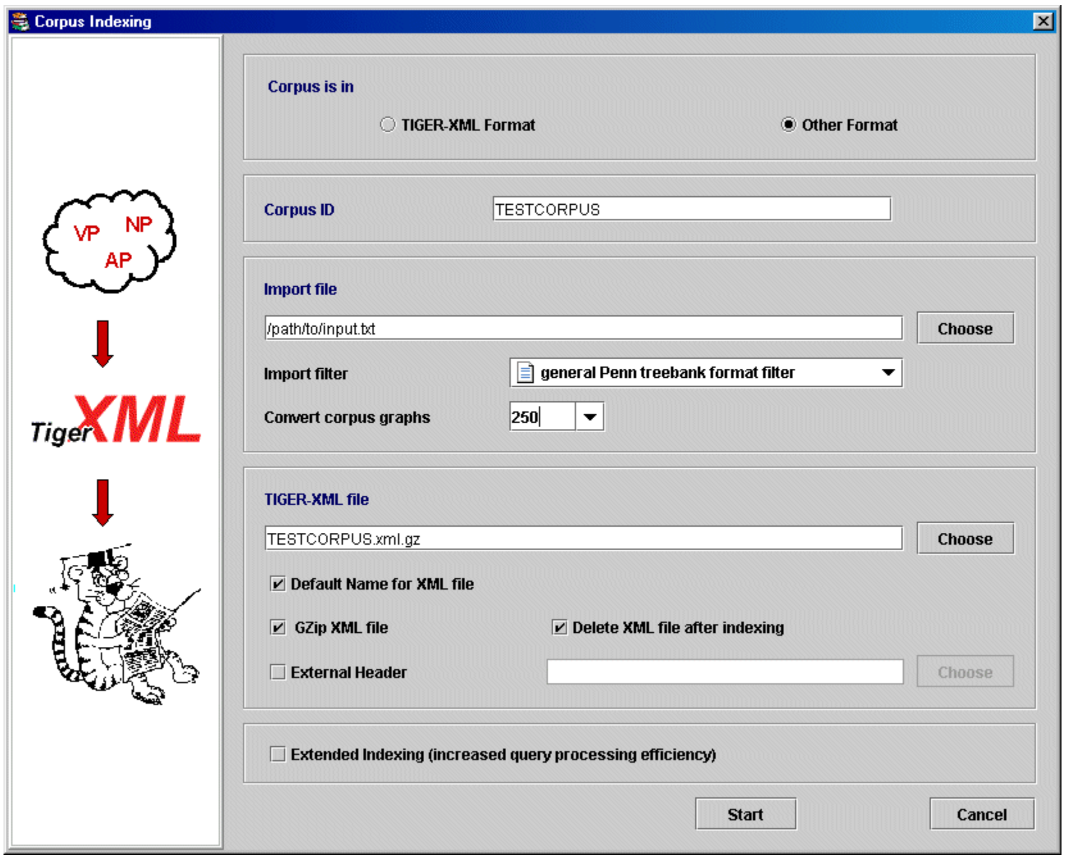
The corpus ID is used by the TIGERSearch software to realize corpus-dependent configurations. The corpus ID must be unique with regard to all indexed corpora. The uniqueness is checked before the indexing process is initiated. The ID has to start with a letter.

Import file

The source file can either be an uncompressed file, or a compressed `.gz` file, or a `zip` file that contains one source file only (relative paths are evaluated with regard to the working directory). Compressed files are automatically decompressed during the indexing process.

Import filter

Select one of the corpus format filters (cf. screenshot above). For a list of all implemented filters see subsection 3.5.


Corpus Indexing

Corpus is in

( ) TIGER-XML Format (●) Other Format

Corpus ID: TESTCORPUS

Import file: /path/to/input.txt

Import filter: general Penn treebank format filter

Convert corpus graphs: 250

TIGER-XML file: TESTCORPUS.xml.gz

☑ Default Name for XML file

☑ GZip XML file ☑ Delete XML file after indexing

☐ External Header

☐ Extended Indexing (increased query processing efficiency)

Convert corpus graphs

You can either convert and index the whole corpus or the first n graphs of the corpus.

Temporary TIGER-XML file

The first step of the indexing is the conversion to TIGER-XML. Thus, a temporary TIGER-XML file is needed. When typing in the corpus ID, a file name is automatically generated by the system (cf. checkbox *Default name for XML file*). Of course, you can also specify a different file name. Please note that relative paths are evaluated with regard to the working directory.

TIGER-XML file parameters

As the TIGER-XML file is a temporary file, it makes sense to compress it. You can enforce GZIP compression by checking the *GZip XML file* box. When the indexing progress is finished, the temporary file is automatically deleted. If you want to save the file (e.g. for debugging purposes) just uncheck the *Delete after indexing* box.

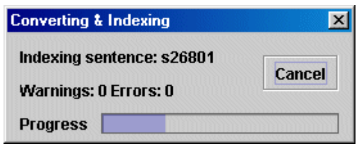
You can also make use of a so-called external header, i.e. a TIGER-XML document header which is stored in a separate file. To use this external header check the *External Header* box and type in the path of the header file (relative paths are evaluated with regard to the working directory).

Extended indexing

If extended indexing is activated, additional corpus information is retreived during the indexing process. This information is used to improve corpus query processing efficiency. Effiency will increase about 50% at the expense of main memory requirement which also increases about 50%.

⚠**Please note:** Default indexing requires a constant amount of main memory (about 128 MB). The main memory requirement of the extended indexing process will depend on corpus size. If an out of memory warning is displayed, please modify the main memory configuration of the TIGERRegistry tool (cf. section 4, chapter II).

To start the indexing process press the *Start* button. The corpus conversion and indexing can be stopped at any time. The current progress is displayed by the Converting & Indexing progress window:

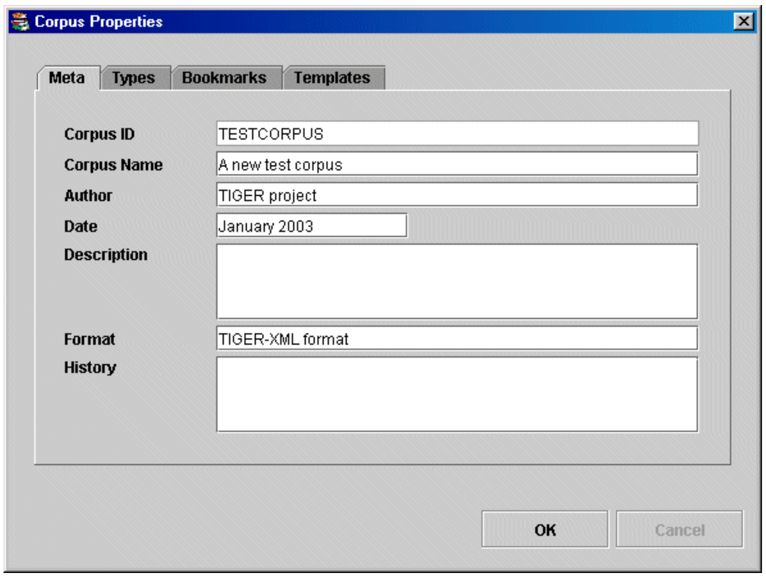


Indexing sentence: s26801
Warnings: 0 Errors: 0

Figure: Conversion and Indexing progress window

The progress window also shows how many warnings and errors occured during corpus conversion and indexing. These messages are stored in the corpus log files `conversion.log` and `indexing.log` which are placed in the corpus directory. In subsection 3.4 we desribe how to view these log files within the TIGERRegistry application.

When the indexing process is finished, a corpus properties window pops up (cf. screenshot below). Here you can fill in meta information about the corpus such as the corpus name. The corpus properties window is explained in section 4.



Corpus ID	TESTCORPUS
Corpus Name	A new test corpus
Author	TIGER project
Date	January 2003
Description	
Format	TIGER-XML format
History	

Figure: Corpus properties window

After specifying the corpus properties, just click the *OK* button to finish corpus indexing. Now the new corpus can be found in the corpus tree.

3.4 Viewing corpus log files

The conversion of corpora into the TIGER-XML format and the indexing of TIGER-XML corpora are both implemented in a robust way, i.e. both processes are also capable of handling corpus sentences that do not fulfill the syntactic and semantic restrictions in some minor points. However, warnings and error messages are produced in these cases. All these messages are collected in two corpus log files which are stored in the corpus directory of the new corpus:

- conversion.log

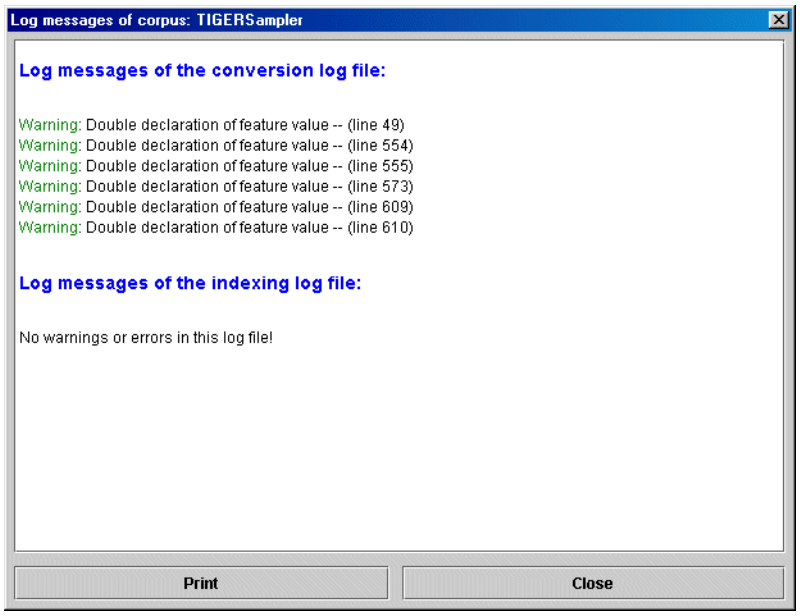
  Warnings and errors that have been produced during the corpus conversion process, i.e. during the conversion to TIGER-XML.

- indexing.log

  Warnings and errors that have been produced during the corpus indexing process.

After corpus indexing you may inspect these messages in order to modify your corpus. Of course, you might view these files using your favourite external editor. However, you can also have a look at these files within the TIGERRegistry window. Just mark the corpus your interested in and select the *Corpus Logfiles* item in the *Corpus* submenu of the context menu or select the corresponding item in the TIGERRegistry menu.

Now the corpus logging window pops up. It displays the content of the two log files. To keep track of all the messages, the keywords *Warning* and *Error* are displayed green-colored and red-colored, respectively.



```
Log messages of the conversion log file:

Warning: Double declaration of feature value -- (line 49)
Warning: Double declaration of feature value -- (line 554)
Warning: Double declaration of feature value -- (line 555)
Warning: Double declaration of feature value -- (line 573)
Warning: Double declaration of feature value -- (line 609)
Warning: Double declaration of feature value -- (line 610)

Log messages of the indexing log file:

No warnings or errors in this log file!
```

Figure: Viewing corpus log files

3.5 List of implemented import filters

⚠**Please note:** Corpora to be processed by the text-based import filters of TIGERRegistry (except some XML-based filters) have to be encoded in ISO-Latin-1. If characters outside the ISO-Latin-1 character set have to be used in a corpus, please use the following unicode encoding convention: Prefix the hexadecimal unicode number of your character by the string `\u`. For example, the unicode character corresponding to the hexadecimal number 03a9 (Greek capital letter Omega) has to be encoded as `\u03a9`.

The following import filters have been implemented:

General bracketing formats

- general () filter

  This filter should work with bracketing-style corpora that use braces for structuring. It generates `cat`, `pos` and `word` features.

- general [] filter

  This should work with bracketing-style corpora that use brackets for structuring. It generates `cat`, `pos` and `word` features.

PennTreebank formats

- general PennTreebank filter

  This filter should work with UPenn-style corpora. Syntactic functions are modelled as edge labels, traces are modelled as secondary edges.

  This filter has been tested for the Wall Street Journal and Brown Corpus (Penn Treebank - bracketing version in `mrg/` subdirectory), the Penn-Helsinki Parsed Corpus of Middle English, and the Chinese Treebank.

  The Chinese Treebank has to be converted to the mentioned Unicode encoding first. The command line tool `native2ascii` can be used for this purpose. It is included in Sun's Java Development Kit which you can download at *http://java.sun.com*. For the Chinese treebank, the command line is the following:

  ```
  native2ascii -encoding GB2312 chinese.txt unicodeoutput.txt
  ```

- ATIS corpus filter

  This is a special filter for the ATIS corpus format **only** (Penn Treebank - bracketing version in `mrg/` subdirectory). It handles the different `pos` and `word` notation and other corpus-specific differences.

- SWITCHBOARD corpus filter

  This is a special filter for the SWITCHBOARD corpus format **only** (Penn Treebank - bracketing version in `mrg/` subdirectory). It skips code and interjection sections.

- Korean treebank filter

  This is a special filter for the Korean treebank corpus format **only**. The Korean Treebank has to be converted to the mentioned Unicode encoding first. The command line tool `native2ascii` can be used for this purpose. It is included in Sun's Java Development Kit which you can download at *http://java.sun.com*. For the Korean treebank, the command line is the following:

  ```
  native2ascii -encoding KSC5601 korean.txt unicodeoutput.txt
  ```

- Treebanks converted by negra-topenn

  This is a special filter for corpora that have been generated using the `negra-topenn` command line tool. This tool is part of the *Negra Corpus* deliverable. It has been developed to linguistically transform treebanks which have been annotated according to the Negra annotation scheme to the UPenn style format.

Susanne and Christine

- Susanne corpus filter

  This is a special filter for the Susanne corpus format **only**.

- Christine corpus filter

  This is a special filter for the Christine corpus format **only**.

Negra format

- general Negra format filter

  This filter should work with any corpus encoded according to the Negra format, Version 3 or Version 4. It has been tested for the Negra Corpus, the Negra 2000 Corpus, the VerbMobil Treebank, and the TIGER Corpus Release 1.

IMS tools

- LoPar format filter

  LoPar is an implementation of a parser for head-lexicalised probabilistic context-free grammars. Grammars are currently available for German and English. LoPar has been developed at IMS, University of Stuttgart (cf. *http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/LoPar-en.html*).

  The LoPar format filter is able to process a special output format of LoPar. This output can be generated using the following LoPar command line:

  ```
  cat input.txt | lopar -in <model> -stems -heads -viterbi -
  viterbi-probs -tgrep > output.txt
  ```

  The input file must be a text file in one word per line format.

- TreeTagger chunking filter

  The TreeTagger is a tool for annotating text with part-of-speech and lemma information. It also comprises a chunker that is based on the tagging output. Chunking modules are currently available for German and English. The TreeTagger has been developed at IMS, University of Stuttgart (cf. *http://www.ims.uni-stuttgart.de/projekte/corplex/*). The TreeTagger chunking filter is able to process the XML output of the chunker.

- YAC format filter

  The chunker YAC (*Yet Another Chunker*) is a rule-based chunker for German. It has been developed at IMS, University of Stuttgart (cf. *http://www.ims.uni-stuttgart.de/projekte/corplex/*). The YAC format filter is able to process the XML-based YAC output.

Other formats

- DEREKO format filter

  This filter should work with corpora encoded according to the DEREKO corpus format. The DEREKO corpus format has been developed within the *DEREKO project*.

3.6 Corpus conversion only

If you just want to convert a corpus to the TIGER-XML format without subsequent corpus indexing, you can use the corpus conversion feature. Choose the *Convert Corpus* item in the *Corpus* menu. The corpus conversion window pops up. Specify the parameters which have been explained in the previous two subsections and press the *Start* button to start the conversion. The conversion process can be stopped at any time.

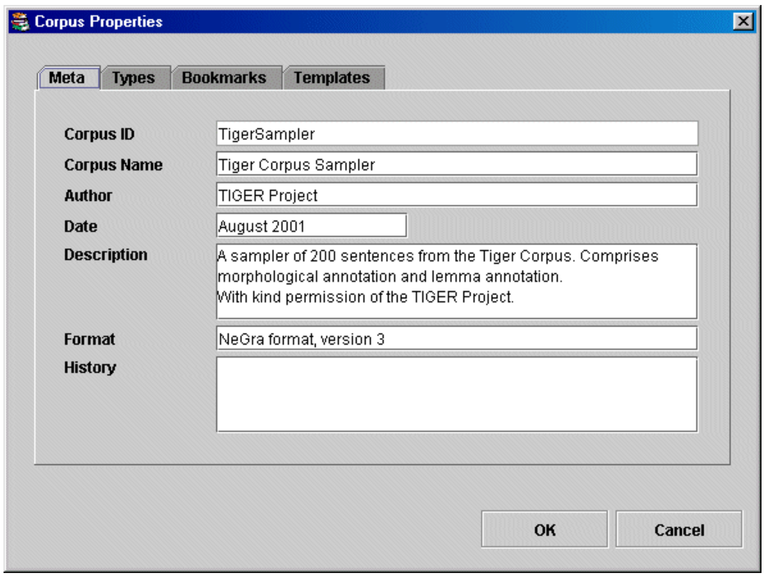
4. Changing corpus properties

4.1 Introduction

To change the properties of a corpus, please mark the corpus symbol and press the *Corpus Properties* button in the button toolbar. The corpus properties window pops up. Now you can change the corpus meta information (cf. subsection 4.2), specify a type system (cf. subsection 4.3), include corpus bookmarks (cf. subsection 4.4), or link predefined corpus templates (cf. subsection 4.5).

4.2 Corpus meta information

To edit the meta information of the corpus (which is displayed by the TIGERSearch and TIGERRegistry GUI), select the *Meta* tab in the corpus properties window (cf. screenshot below). Now you can edit the corpus meta information except the ID which cannot be changed after corpus creation:

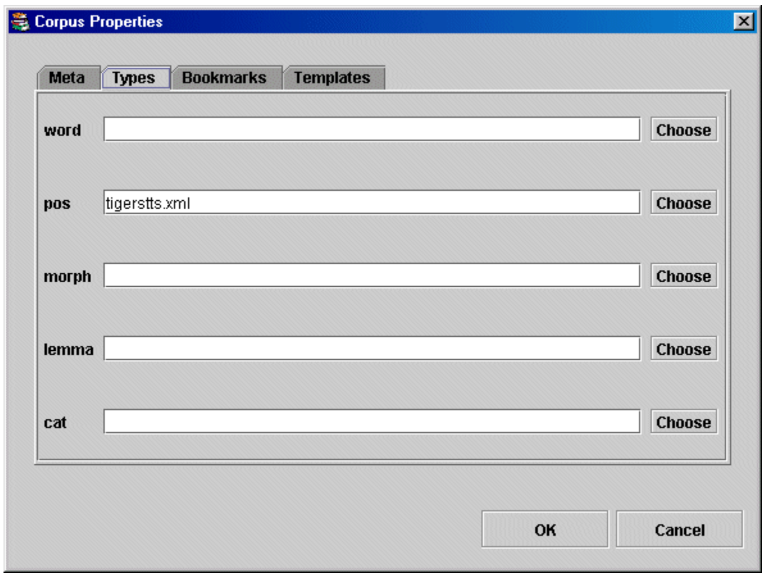


Corpus ID: TigerSampler
Corpus Name: Tiger Corpus Sampler
Author: TIGER Project
Date: August 2001
Description: A sampler of 200 sentences from the Tiger Corpus. Comprises morphological annotation and lemma annotation.
With kind permission of the TIGER Project.
Format: NeGra format, version 3
History:

Figure: Specifying corpus meta information

4.3 Feature types

To specify a type hierarchy for a feature, you must first create an XML file comprising the hierarchy definition. The concept of feature types and its XML representation is described in section 8, chapter III. After creating the XML file, you have to register it. Please select the *Types* tab in the corpus properties window (cf. screenshot above). Now select the feature and type in the path to your file (absolute or relative to the corpus path). In the example, a relative link to the file `tigerstts.xml` in the corpus directory is specified:



word
pos tigerstts.xml
morph
lemma
cat

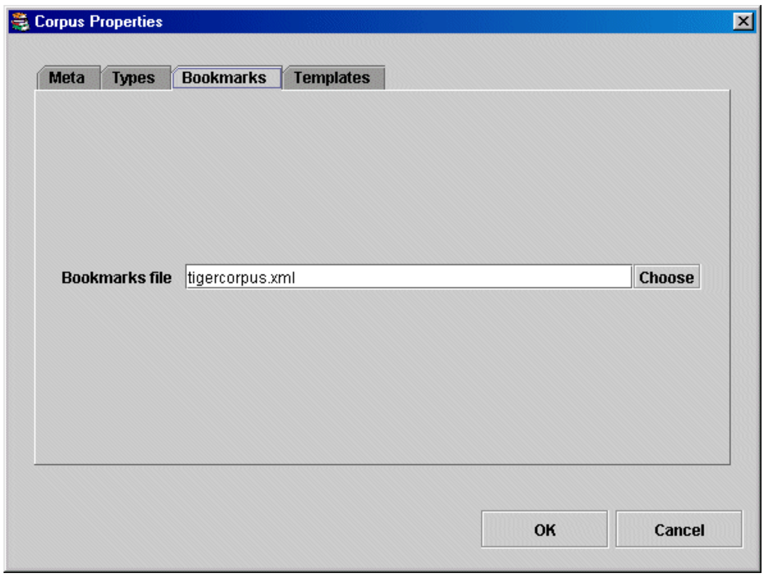
Figure: Specifying feature types

If there are any problems loading a type hierarchy in the TIGERSearch tool (e.g. if there is a feature value in the corpus that is not used in the hierarchy, or if there is a feature value used in the hierarchy that is unknown to the corpus), all warnings are collected and can be inspected within the TIGERSearch corpus documentation tab (cf. subsection 2.2, chapter IV).

4.4 Corpus bookmarks

One helpful feature of the TIGERSearch tool is the management of bookmarks (cf. subsection 2.3, chapter IV). Users can store their favourite bookmarks for later inspection or reuse. The so-called *corpus bookmarks* file can be linked to a corpus so that all users of the corpus have access to it.

To link a corpus bookmarks file to a corpus, select the *Bookmarks* tab in the corpus properties window. Type in the path to your bookmarks file (absolute or relative to the corpus path). In the example, a relative link to the file `tigercorpus.xml` in the corpus directory is specified:

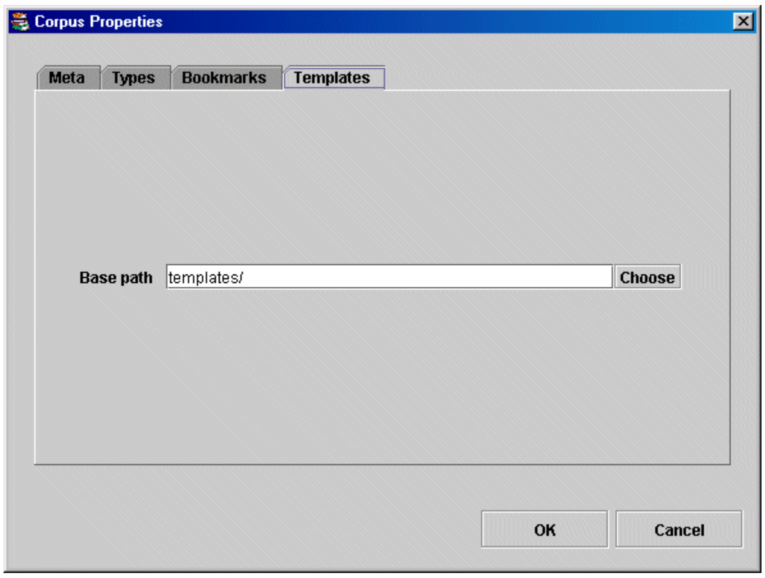


Bookmarks file: tigercorpus.xml

Figure: Specifying corpus bookmarks

4.5 Corpus templates

Templates definitions (cf. section 9, chapter III for an introduction) are stored in files, template files are organized in directories. In order to link a template collection to a corpus, you have to specify the root directory of your collection. Select the *Templates* tab in the corpus properties window and type in the path to the templates root directory (absolute or relative to the corpus path). In the example, a relative link to the directory `templates/` is specified:



Base path: templates/

Figure: Specifying corpus templates

If there are any problems loading the templates in the TIGERSearch tool (e.g. if one of the templates is not wellformed), all warnings are collected and can be inspected within the TIGERSearch corpus documentation tab (cf. subsection 2.2, chapter IV).

Chapter VII - Appendix

1. References

[Abeille2003] Abeillé, Anne; Clement, Lionel and Kinyon, Alexandra (2003): *Building and using syntactically annoted corpora*. Kluwer Academic Publishers, Dordrecht.

[BlackburnEtAl1993] Blackburn, Patrick; Gardent, Claire and Meyer-Viol, Wilfried (1993): Talking About Trees. In: *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 21-29, Utrecht.

[Christ1994] Christ, Oliver (1994): A modular and flexible architecture for an integrated corpus query system. In: *Proceedings of COMPLEX'94: 3rd Conference on Computational Lexicography and Text Research*, pp. 23-32, Budapest.

[ChristEtAl1999] Christ, Oliver; Schulze, Bruno M. and König, Esther (1999): *Corpus Query Processor (CQP). User's Manual*. Institut für Maschinelle Sprachverarbeitung, University of Stuttgart.

[Doerre1996] Dörre, Jochen (1996): *Feature-Logik und Semiunifikation*. Dissertationen zur Künstlichen Intelligenz, infix-Verlag, Sankt Augustin.

[DoerreDorna1993] Dörre, Jochen and Dorna, Michael (1993): *CUF - A Formalism for Linguistic Knowledge Representation*. Deliverable R.1.2A, DYANA2

[DoerreEtAl1996] Dörre, Jochen; Gabbay, Dov M. and König, Esther (1996): Fibred Semantics for Feature-based Grammar Logic. *Journal of Logic, Language, and Information. Special Issue on Language and Proof Theory*, vol. 5, pp. 387-422.

[DuchierNieren1999] Duchier, Denys and Niehren, Joachim (1999): *Solving Dominance Constraints with Finite Set Constraint Programming*. Technical report, Universität des Saarlandes, Programming Systems Lab.

[Emele1997] Emele, Martin (1997): *Der TFS-Repräsentationsformalismus*. Ph.D. thesis, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart. Arbeitspapiere des IMS, vol. 3, no. 6.

[EmeleZajac1990] Emele, Martin and Zajac, Rémi (1990): A Fixed-Point Semantics for Feature Type Systems. In: *Proceedings of the 2nd International Workshop on Conditional and Typed Rewriting Systems*, Montreal.

[HoehfeldSmolka1988] Höhfeld, Markus and Smolka, Gert (1988): *Definite Relations over Constraint Languages*. LILOG-Report 53, IBM Deutschland, Stuttgart.

[KoenigLezius2002] König, Esther and Lezius, Wolfgang (2003): *The TIGER language* -

A Description Language for Syntax Graphs. Formal Definition. Technical report, IMS, University of Stuttgart.

[Lezius2002] Lezius, Wolfgang (2002): *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Ph.D. thesis, IMS, University of Stuttgart. Arbeitspapiere des IMS, vol. 8, no. 4.
http://www.ims.uni-stuttgart.de/projekte/corplex/paper/lezius/diss/

[Lezius2002b] Lezius, Wolfgang (2002): *TIGERSearch - Ein Suchwerkzeug für Baumbanken*. In: Stephan Busemann, editor: Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache (KONVENS 2002), pp. 107-114, Saarbrücken.

[MarcusEtAl1993] Marcus, Mitchell; Santorini, Beatrice and Marcinkiewicz, Mary Ann (1993): Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, vol. 19, pp. 313-330.

[RogersEtAl1992] Rogers, James and Vijay-Shanker, K. (1992): Reasoning with Descriptions of Trees. In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. Newark, Delaware.

[Schieber1986] Shieber, Stuart M. (1986): *An Introduction to Unification-Based Approaches to Grammar*. Lecture Notes, Center for the Study of Language and Information, Stanford.

[Schmid1999] Schmid, Helmut (1999) *YAP: Parsing and Disambiguation With Feature-Based Grammar* Ph.D. thesis, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart.

[SkutEtAl1997] Skut, Wojciech; Krenn, Brigitte; Brants, Thorsten and Uszkoreit, Hans (1997): An Annotation Scheme for Free Word Order Languages. In: *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP)*, Washington, D.C.

[Smith2002] Smith, George (2002): *A brief introduction to the TIGER Corpus Sampler*. University of Potsdam.

[SteinerKallmeyer2002] Steiner, Ilona and Kallmeyer, Laura (2002) VIQTORYA - A Visual Query Tool for Syntactically Annotated Corpora. In: *Proceedings of LREC 2002*, Las Palmas, Gran Canaria.

[SterlingShapiro1986] Sterling, Leon and Shapiro, Ehud (1986): *The Art of Prolog: Advanced Programming Techniques*. MIT Press, Cambridge, Mass.

[WallEtAl1996] Wall, Larry; Christiansen, Tom; Schwartz, Randal L. and Potter, Stephen (1996): *Programming Perl*. O'Reilly, Cambrige.

[Voormann2002] Voormann, Holger (2002): *Grafische Eingabe von Suchanfragen in TIGERSearch*. Diploma thesis, Fakultät für Informatik, Universität Stuttgart.

2. Acknowledgements

2.1 Project background

The TIGERSearch software suite has been developed in the context of the following projects:

- *DEREKO project* funded by the *Land Baden-Württemberg*
- *TIGER project* funded by the *Deutsche Forschungsgemeinschaft*

- Esther König's post-doc research
- *Wolfgang Lezius'* PhD and post-doc research
- *Holger Voormann's* diploma thesis and PhD research

2.2 Third party software

The following software is part of the TIGERSearch distribution:

Software	**Organization**	**URL**
batik.jar	Apache Software Foundation, Apache XML Project	*http://xml.apache.org/batik/*
fop.jar	Apache Software Foundation, Apache XML Project	*http://xml.apache.org/fop/*
jakarta-oro.jar	Apache Software Foundation, Apache Jakarta Project	*http://jakarta.apache.org/oro/*
jh.jar	Sun Microsystems, Inc.	*http://java.sun.com/products/javahelp/*
jdom.jar	JDOM Project	*http://www.jdom.org*
log4j.jar	Apache Software Foundation, Apache Jakarta Project	*http://jakarta.apache.org/log4j/*
poi.jar	Apache Software Foundation, Apache Jakarta Project	*http://jakarta.apache.org/poi/*
xalan.jar	Apache Software Foundation, Apache XML Project	*http://xml.apache.org/xalan-j/*
xercesImpl.jar	Apache Software Foundation, Apache XML Project	*http://xml.apache.org/xerces-j/*
xml-apis.jar	Apache Software Foundation, Apache XML Project	*http://xml.apache.org/xerces-j/*
Java Runtime Environments	Sun Microsystems, Inc.	*http://java.sun.com/j2se/*

The following software has been used to generate parts of the functionality of TIGERSearch:

Software	**Organization**	**URL**
JavaCC	Sun Microsystems Laboratories	*http://www.experimentalstuff.com*

The following software has been used to develop the TIGERSearch software:

Software	**Organization**	**URL**
Ant	Apache Software Foundation	*http://ant.apache.org*
Eclipse	The Eclipse Consortium	*http://www.eclipse.org*

2.3 Third party copyright statements

- "This product includes software developed by the Apache Software Foundation (*http://www.apache.org/*)."
- "This product includes software developed by the JDOM Project (*http://www.jdom.org/*)."
- "This product includes code licensed from RSA Security, Inc." (JRE)
- "Some portions licensed from IBM are available at *http://oss.software.ibm.com/icu4j/*" (JRE)

2.4 Trademarks

- Adobe is a registered trademark of Adobe Systems, Inc.
- Excel is a registered trademark of Microsoft Corporation.
- InstallAnywhere is a registered trademark of Zero G Software, Inc.
- Mac OS is a registered trademark of Apple Computer, Inc.
- PowerPoint is a registered trademark of Microsoft Corporation.
- Solaris and Java are trademarks of Sun Microsystems, Inc.
- StuffIt Expander is a registered trademark of Aladdin Systems, Inc.
- Windows is a registered trademark of Microsoft Corporation.
- All other marks are properties of their respective owners.

2.5 Third party corpus samplers

A range of institutions have kindly agreed that excerpts from their text corpora may be distributed with TIGERSearch. The current version of TIGERSearch includes the following corpus samplers (in alphabetic order):

Chinese

- Chinese Treebank sampler

  105 corpus graphs, *University of Pennsylvania*, distributed by *LDC*

English

- Penn Treebank: Brown Corpus and Switchboard Corpus samplers

  200 sentences each, *University of Pennsylvania*, distributed by *LDC*
- Penn-Helsinki Parsed Corpus of Middle English (PPCME2 Corpus) sampler

  200 sentences, *University of Pennsylvania* / *PPCME2 Project*
- Susanne and Christine Corpus samplers

  200 sentences each, *Sussex University* / *Susanne and Christine projects*
- VerbMobil Corpus sampler

  250 sentences, see German VerbMobil sampler

German

- DEREKO Corpus sampler

  250 sentences, *SfS, University of Tübingen* and *IMS, University of Stuttgart* / *DEREKO project*
- IMS chunking and parsing tools

  The tools LoPar, TreeTagger, and YAC processed the same technical text (about 250 sentences). *IMS, University of Stuttgart*
- Negra Corpus sampler

  250 sentences, *Department of Computational Linguistics, Universität des Saarlandes* / *Negra project*
- TIGER Corpus sampler

  200 sentences, *Institut für Germanistik, University of Potsdam* / *Department of Computational Linguistics, Universität des Saarlandes* / *IMS, University of Stuttgart* / *TIGER project*
- VerbMobil Corpus sampler

  250 sentences, *SfS, University of Tübingen* / *VerbMobil Project*, distributed by *IPSK, Ludwig-Maximilian-Universität München*

Japanese

- VerbMobil Corpus sampler
  250 sentences, see German VerbMobil sampler

Korean

- Korean Treebank sampler
  125 corpus graphs, *University of Pennsylvania*, distributed by *LDC*

Index