# Tamino

## Unicode and Text Retrieval

Version 10.1

April 2018

WEBMETHODS

This document applies to Tamino Version 10.1 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1999-2018 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

**Document ID: INS-UNICODE-101-20180413**

## Table of Contents

# 1 Unicode and Text Retrieval

This chapter describes various aspects of Tamino's text retrieval capabilities and how you, as an administrator, can customize the word recognition process. This has implications for query expressions that use full-text capabilities. In XQuery, this affects text retrieval functions such as `tf:containsText` or `tf:createNearTextReference`; in X-Query, it affects the `contains` operator ~=.

**Note:** You can customize word recognition only when using the default (white space-separated) tokenizer. The tokenizer for Japanese cannot be customized.

The representation of characters stored in Tamino is based on the Unicode standard. Unicode is more than a simple character code such as ASCII, ISO 646 or the ISO 8859 series, since it defines for each character not only its code position, but also a number of other characteristics, such as character class, bidirectionality, decomposability, etc. In Unicode, a character can also have a mapping defined for lower case, upper case, and title case. For example, the character `LATIN CAPITAL U WITH DIAERESIS`, which is "Ü", is treated as "ü" in lower case, and can be decomposed into "u" and the diaeresis sign. This, among other criteria, is used for determining character equivalence, which is important for comparison operations.

A Tamino user might expect "Ü" in German data to be treated as "ue", so that the query expression `/patient[name ~= "mueller"]` would return all name elements that contain a word "Müller", "Mueller" or "mueller", but not "Muller". This behavior is not necessarily desired by all users. Others might expect "Ü" to be treated as its base character "U", so the query above would yield a different result set. In Tamino, for each character a *replacement character* is defined, which is used when performing text retrieval using the `contains` operator. You can therefore model either of the aforementioned behaviors by changing the default replacement character for each defined character.

The *character class* is another important piece of information in the definition of a Unicode character. In Unicode, this is the “general category” that determines whether a character is, for example, a left-to-right character, a right-to-left character, a number, or punctuation mark, to name only a few. In Tamino, the character class also controls word separation. The XQuery functions `tf:containsText` and friends as well as the `contains` operator ~= in X-Query operate on words and rely on the character class definition.

The following sections explain how the above is defined in Tamino, and how you can customize the default behavior.

# The Unicode Character Database

The Unicode Character Database (UCD), as defined by the Unicode Consortium, is the basis for representing and handling characters in Tamino. The UCD is briefly introduced and its mapping onto Tamino's default character table is explained.

## Structure of the UCD Data

Since the default character base in Tamino uses the Unicode character database, it makes sense to have a look at the *properties* that are stored with each character. A record in this database consists of:

| Field Name | Example | Description |
|---|---|---|
| Code Value | U+00FC | The code value that uniquely identifies a character in Unicode |
| Character Name | LATIN SMALL LETTER U WITH DIAERESIS | The reference name of the character |
| General Category | Ll | The character class |
| Canonical Combining Classes | | Used for canonical ordering algorithm |
| BIDI Category | L | Determines role in bidirectional text |
| Character Decomposition Mapping | 0075 0308 | Maximal decomposition of combined characters; first value is base character |
| Decimal Digit Value | 3 for U+0033 | Decimal value of a digit |
| Digit Value | 4 for U+2074 (superscript four) | Value of a digit, not necessarily decimal |
| Numeric Value | 1/5 for the fraction U+2155 | Character with numeric property |
| Mirrored | Y for "[" | Boolean; if true, the character will be mirrored in text that is laid out from right to left |
| Unicode 1.0 Name | | Informative field for old 1.0 name |
| 10646 Comment Field | | Informative field for name in ISO 10646 |
| Uppercase Mapping | U+00DC ("Ü") for U+00FC ("ü") | Upper case equivalent |
| Lowercase Mapping | U+00FC ("ü") for U+00DC ("Ü") | Lower case equivalent |
| Titlecase Mapping | U+01F2 ("Dz") for U+01F3 ("dz") | Title case equivalent |

For a detailed explanation, please refer to the Unicode documentation, which is available at **http://www.unicode.org/**. You can also find the Unicode Character Database itself here.

### Mapping of the UCD to Tamino

In Tamino, the following fields in the UCD are relevant:

Code Value
General Category
Character Decomposition Mapping
Uppercase Mapping

In addition to the identifying code value, the general category is important in Tamino. It defines the character class, which also determine a character's behavior in comparison operations and text retrieval. The following character classes are defined:

```
character
delimiter
embedded
ignore
number
single
```

This is sufficient for text retrieval purposes. However, the Unicode standard defines 30 general categories. They are mapped to Tamino's internal schema as follows:

| General Categories | Description | Character Class in Tamino |
|---|---|---|
| `Lu`, `Ll`, `Lt`, `Lm`, `Lo`, `Nd`, `Nl`, `No`, `Sm`, `Sc`, `Sk`, `So` | Letters (`L`*x*), Numbers (`N`*x*) and Symbols (`S`*x*) | `character` |
| `Mn`, `Mc`, `Me`, `Pc`, `Pd`, `Ps`, `Pe`, `Pi`, `Pf`, `Po`, `Zs`, `Zl`, `Zp`, `Cc`, `Cf`, `Cs`, `Co`, `Cn` | Mark (`M`*x*), Punctuation (`P`*x*), Separator (`Z`*x*) and Others (`C`*x*) | `delimiter` |

This also means that the Tamino character classes `embedded`, `ignored`, `number` and `single` are initially not used, since they do not have an equivalent in the UCD. The information is stored as a non-XML document in the doctype `ino:unicode` of the system collection `ino:vocabulary`. You can access it using the following URI: *http://*`<server>`*/*`<database>`*/ino:vocabulary/ino:unicode/@1*. See the section **Implications Concerning Text Retrieval** for information about these Tamino character classes.

**Note:** Tamino XML Server supports customization of properties only for the BMP (basic multilingual plane), i.e. the characters U+0000 to U+FFFF.

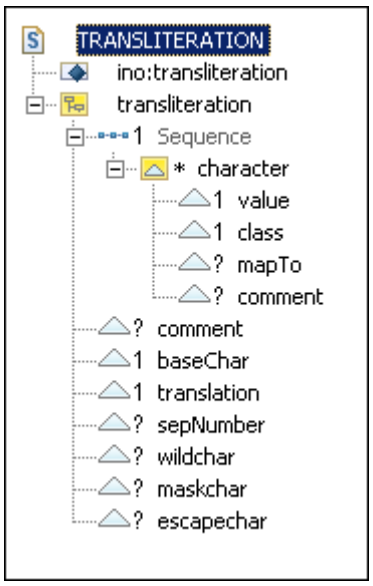
## Representation and Handling of Characters

As described in the previous section, Tamino uses the UCD as the basis for its built-in character database. You cannot alter this database, but you can create a modified version on the level of a Tamino database using the schema element `ino:transliteration`. Tamino itself has defined such a transliteration, which is used to determine the default character handling. You can override the default handling by defining your own transliteration.

The schema element `ino:transliteration` is introduced along with the default transliteration.

### ino:transliteration

Tamino uses the Unicode database for its own character mapping system. This information forms part of the schema element `transliteration` in the Tamino namespace *http://namespaces.software-ag.com/tamino/response2* that is prefixed by `ino`. It has the following structure:




The element `ino:translation` can have several attributes and contains a sequence of `ino:character` elements with the basic information for any Unicode character. The required attributes of `ino:character` define for each contained character a replacement to be used in text retrieval operations.

Note the following attributes of `ino:translation`:

1. `ino:baseChar` **(required)**
   It is of the XML Schema type `boolean` so "true" and "false" are the only allowed values.

   "**true**"
   Each character is replaced by its base character, which is the first character of the Unicode property “Character Decomposition Mapping”.

   Example: The Unicode character U+00E9 ("é") is replaced by "e", since the character decomposition mapping field is `0065 0301`. These code values represent the `LATIN SMALL LETTER E` and the `COMBINING ACUTE ACCENT`.

   "**false**"
   All characters are left unchanged.

2. `ino:translation` **(required)**
   It is of the XML Schema type `boolean`, so "true" and "false" are the only allowed values.

   "**true**"
   Each character is replaced by its uppercase equivalent, which is the value of the Unicode property “Uppercase Mapping”.

   Example: The Unicode character U+00E9 ("é") is replaced by "É", since the uppercase mapping field is `00C9`. This code value represents the `LATIN CAPITAL LETTER E WITH ACUTE`.

   "**false**"
   All characters are left unchanged.

3. `ino:comment` **(optional)**
   A comment describing the transliteration as a whole.

**Note:** If both attributes `ino:baseChar` and `ino:translation` are set to "true", then the replacement defined by `ino:baseChar` is performed first. For example, the character "é" is replaced by "E", because "é" is replaced by its base character "e", which in turn is replaced by its uppercase equivalent "E".

The element `ino:character` defines the properties for a single character. It has no contents. The information is contained in its attributes:

`ino:value` **(required)**
The code value; this corresponds to the code value in the Unicode database. You can use any notation allowed in XML such as:

- the character itself: for example `ino:value="ü";`
- a numerical character reference, either decimal (`ino:value="&#252;"`) or hexadecimal (`ino:value="&#xFC;"`);
- a predefined entity reference (one of `&amp;`, `&lt;`, `&gt;`, `&apos;`, or `&quot;`).

`ino:class` **(required)**
The character class as defined for Tamino; this is one of `character`, `delimiter`, `embedded`, `ignore`, `number`, or `single`; it determines how this character behaves when performing text

retrieval. Initially, the Unicode character property “General Category” is mapped to the `class` attribute.

`ino:mapTo` **(optional)**
The code values of a character or sequence of characters that replaces the `value` when performing text retrieval. Again, you can use any XML-conformant notation. Initially, the Unicode character property “Uppercase Mapping” is used.

Please note that you override the global settings `ino:baseChar` and `ino:translation` if you use `ino:mapTo`.

`ino:comment` **(optional)**
A comment to describe the definition of this single character. Initially, this attribute is empty.

### The Default Transliteration

The following default transliteration is defined on top of the initial built-in character set:

```
<ino:transliteration ino:baseChar="true" ino:translation="true">
  <ino:character ino:value="&amp;" ino:class="character" />
  <ino:character ino:value="/" ino:class="character" />
  <ino:character ino:value="_" ino:class="character" />
  <ino:character ino:value="@" ino:class="character" />
  <ino:character ino:value="*" ino:class="character" />
  <ino:character ino:value=":" ino:class="character" />
  <ino:character ino:value="." ino:class="embedded" />
  <ino:character ino:value="-" ino:class="ignore" />
  <ino:character ino:value="=" ino:class="delimiter" />
  <ino:character ino:value="&lt;" ino:class="delimiter" />
  <ino:character ino:value="&gt;" ino:class="delimiter" />
  <ino:character ino:value="~" ino:class="delimiter" />
  <ino:character ino:value="ä" ino:class="character" ino:mapTo="AE" />
  <ino:character ino:value="Ä" ino:class="character" ino:mapTo="AE" />
  <ino:character ino:value="ö" ino:class="character" ino:mapTo="OE" />
  <ino:character ino:value="Ö" ino:class="character" ino:mapTo="OE" />
  <ino:character ino:value="ü" ino:class="character" ino:mapTo="UE" />
  <ino:character ino:value="Ü" ino:class="character" ino:mapTo="UE" />
  <ino:character ino:value="ß" ino:class="character" ino:mapTo="SS" />
</ino:transliteration>
```

This transliteration is used when no other user-defined transliteration has been defined for the current database. It is stored in the doctype `ino:transliteration` of the system collection `ino:vocabulary`. You can retrieve this information from a running database by querying `ino:transliteration` against the collection `ino:vocabulary`. The result document from Tamino looks like the one shown above with one difference:

```
<ino:transliteration ino:id="1" ino:docname="default" ino:baseChar="true" ↩
ino:translation="true">
```

For a given database, only one transliteration can be active at a time. The currently active transliteration carries the `ino:id` 1.

## Implications Concerning Text Retrieval

A transliteration has implications for the way text retrieval operations are performed. The class definition of a character determines whether it is recognized as part of a word, as a single word, as a delimiter, or is ignored. The following list contains information for each character class:

`character`
: This character is always recognized as part of a word. It must be a non-numeric character.

`delimiter`
: This character delimits a word; this also means that it is never part of a word. For example, the character sequence "a b" is treated as two adjacent words "a" and "b", since the space character is defined as a delimiter.

  You cannot search for this character.

`embedded`
: It depends on the position: This character is ignored at the beginning and end of a word, but is otherwise recognized as part of a word if it is surrounded by two characters of the same class. For example, the character sequence ".a.b." is replaced by "a.b".

`ignore`
: This character is always ignored. For example, "a-b" is replaced by "ab".

  You cannot search for this character.

`number`
: This character is always recognized as a digit and is part of a number, but not part of a word. It must be a numeric character.

`single`
: This character is treated as a word by itself. It effectively also separates words. For example, "a&b" is replaced by the three adjacent words "a", "&" and "b".

### Word-Separating Characters

From the UCD and the defined transliteration, you can determine the characters that separate words from each other. Since all punctuation characters, separators, marks and those classified in the UCD as “Other” are mapped to the Tamino character class `delimiter`, there are 1,128 characters that act as delimiters in a text retrieval operation. Instead of listing all of them, we show in the following table the delimiting characters of the two character blocks “Basic Latin” (character range U+0000 to U+007F) and “Latin-1 Supplement” (character range U+0080 to U+00FF):

| Code Value(s) | Character |
|---|---|
| 0000 - 001F | ASCII control codes |
| 0020 | SPACE |
| 0021 - 002A | ! " # $ % & ' ( ) * |
| 002C - 002F | , - . / |
| 003A - 003B | : ; |
| 003F - 0040 | ? @ |
| 005B - 005D | [ \ ] |
| 005F | _ |
| 007B | { |
| 007D | } |
| 007F | DELETE |
| 0080 - 00A0 | control codes |
| 00A1 | ¡ |
| 00AB | « |
| 00AD | - (soft hyphen) |
| 00B7 | · |
| 00BB | » |
| 00BF | ¿ |

In a default environment, you have to take the **default transliteration** into account so that `&`, `/`, `_`, `@`, `*`, and `:` are treated as characters, while `<`, `>`, `=` and `~` are additionally treated as delimiters. In addition, `.` is treated as an embedded character and `-` belongs to the character class `ignore`.

**Tip:** For a running database, you can query the collection `ino:vocabulary` to retrieve the current list of delimiters:

- **XQuery**

```
declare namespace ino="http://namespaces.softwareag.com/tamino/response2"
input()/ino:transliteration/ino:character[@ino:class="delimiter"]
```

- **X-Query**

```
/ino:transliteration/ino:character[@ino:class="delimiter"]
```

Furthermore, there is the server parameter "markup as delimiter" that specifies how markup is treated. By default, markup is not treated as word delimiter.

# Customizing Transliterations

You can use `ino:transliteration` to define your own transliteration, which is then used instead of the default transliteration described in the previous section. After definition, you can use the Tamino Interactive Interface to process this transliteration. The default transliteration implements the character handling used in German telephone books by mapping the umlaut characters. An example demonstrates how to customize this translation and define it in Tamino.

## Defining a Custom Transliteration in Tamino

There are two ways of activating a new transliteration in a Tamino database. You can either write an external transliteration document that conforms to the `ino:transliteration` schema introduced above and load it into Tamino, or you can directly use an XQuery expression.

### To Override the Default Transliteration Using External Documents

Make sure that the Tamino database server is running.

1 In the Tamino Interactive Interface, choose the tab **Load**.

2 In the input field **Database URL**, enter the name of the Tamino database.

3 In the **Into collection** input field, enter "ino:vocabulary/ino:transliteration/default" to overwrite the default transliteration.

4 Use the **Browse** button right to the input field **Load file** button to locate the document with the custom transliteration.

5 Choose the **Load** button. Tamino overwrites its default transliteration. You get a response document similar to the following:

```
<ino:response xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
  xmlns:xql="http://metalab.unc.edu/xql/">
- <ino:message ino:returnvalue="0">
    <ino:messageline>document processing started</ino:messageline>
  </ino:message>
  <ino:object ino:collection="ino:vocabulary" ino:doctype="ino:transliteration" ino:id="2" />
- <ino:message ino:returnvalue="0">
    <ino:messageline>document processing ended</ino:messageline>
  </ino:message>
</ino:response>
```

6 In the Tamino Manager, restart the database server so that the changes take effect.

**Note:** The attribute `ino:docname` specifies the transliteration that is active for the current Tamino database. Only the transliteration that has the attribute element

`ino:docname="default"` is used by Tamino. Therefore you must store any custom transliteration into the doctype `ino:transliteration/default`.

You can check your changes by querying `ino:transliteration` against the collection `ino:vocabulary`.

## Examples

- Customizations for German
- Separating Numbers from Words

### Customizations for German

In contrast to the default transliteration, the German umlauts "ä", "ö", "ü" and their uppercase equivalents should be mapped to their respective base characters. However, the sharp s ("ß") should still be mapped to the character sequence "SS". In addition, expressions such as "Laurel&Hardy" should be treated as if they are three words ("Laurel and Hardy") instead of two words separated by a punctuation character (which would result from the Unicode database settings) or a single word "Laurel&Hardy" as specified by the default for Tamino's `ino:transliteration`.

The following transliteration is therefore required:

```
<?xml version="1.0"?>
  <ino:transliteration xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
    ino:comment="Custom Character Handling"
    ino:baseChar="true" ino:translation="true">
  <ino:character ino:value="&amp;" ino:class="single" ino:comment="read it as 'and'"/>
  <ino:character ino:value="/" ino:class="character" />
  <ino:character ino:value="_" ino:class="character" />
  <ino:character ino:value="@" ino:class="character" />
  <ino:character ino:value="*" ino:class="character" />
  <ino:character ino:value=":" ino:class="character" />
  <ino:character ino:value="." ino:class="embedded" />
  <ino:character ino:value="-" ino:class="ignore" />
  <ino:character ino:value="=" ino:class="delimiter" />
  <ino:character ino:value="&lt;" ino:class="delimiter" />
  <ino:character ino:value="&gt;" ino:class="delimiter" />
  <ino:character ino:value="~" ino:class="delimiter" />
  <ino:character ino:value="ß" ino:class="character" ino:mapTo="SS" />
</ino:transliteration>
```

Since the mapping to base characters is the Unicode default, there is no need to include extra definitions for the six umlaut characters. The mapping definition for "ß" is not present in the UCD. The ampersand, originally classified as a delimiter, now belongs to the class `single` so that it is treated as a single word. You could also define it differently so that it really reads as the character sequence "and", as the comment suggests using:

```
<ino:character ino:value="&amp;" ino:class="single" ino:mapTo="UND" />
```

Please note that upper case is used here for the value of `ino:mapTo` so that it is consistent with the behavior specified by the global `ino:transliteration`. Also note that literal text is suitable only for German data.

If you define this custom transliteration as the default for this Tamino database, you can check this by querying `ino:transliteration[@ino:docname="default"]` against the collection `ino:vocabulary`.

#### Separating Numbers from Words

Normally, a number is regarded as part of a word, i.e. it is part of a word token. Data such as a street address or a serial number is therefore tokenized as follows:

| | |
|---|---|
| Beutelsendstraße 14b | "Beutelsendstraße" "14b" |
| XPR0746TU#2 | "XPR0746TU" "2" |

However, in cases like these you may want to separate numbers from words so that instead the data is tokenized as follows:

| | |
|---|---|
| Beutelsendstraße 14b | "Beutelsendstraße" "14" "b" |
| XPR0746TU#2 | "XPR" "0746" "TU" "2" |

You can achieve this by assigning all numbers to the character class "number". This has the effect that any successive digit forms part of a number token that you can search for in the same way as a word token: a number token is delimited by any character that belongs to one of the classes "character", "delimiter", or "single". Conversely, a word token is delimited by any character that belongs to one of the classes "number", "delimiter", or "single". The corresponding transliteration is as follows:

```
<?xml version="1.0"?>
  <ino:transliteration xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
    ino:comment="Separating Numbers from Words"
    ino:baseChar="true" ino:translation="true">
  <ino:character ino:value="0" ino:class="number"/>
  <ino:character ino:value="1" ino:class="number"/>
  <ino:character ino:value="2" ino:class="number"/>
  <ino:character ino:value="3" ino:class="number"/>
  <ino:character ino:value="4" ino:class="number"/>
  <ino:character ino:value="5" ino:class="number"/>
  <ino:character ino:value="6" ino:class="number"/>
  <ino:character ino:value="7" ino:class="number"/>
  <ino:character ino:value="8" ino:class="number"/>
  <ino:character ino:value="9" ino:class="number"/>
</ino:transliteration>
```

You can check your changes by querying `ino:transliteration` against the collection `ino:vocabulary`:

- **XQuery**

```
declare namespace ino="http://namespaces.softwareag.com/tamino/response2"
for    $a in input()/ino:transliteration/ino:character
where  $a/@ino:class eq "number"
return $a
```

- **X-Query**

```
/ino:transliteration/ino:character[@ino:class="number"]
```

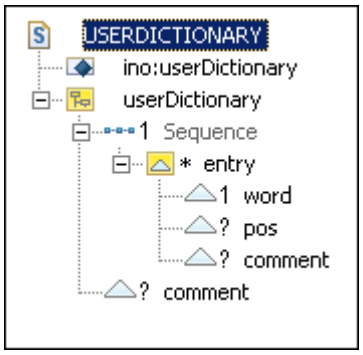
## User-Defined Dictionaries

The tokenizer for Japanese uses a built-in dictionary for word recognition. In addition, you can supply an optional user-defined dictionary. Entries are added to this dictionary by storing a document with all entries, the current and the new ones, in the collection `ino:vocabulary/ino:userDictionary`.

This section explains the schema and shows you how to store a dictionary file in Tamino.

### ino:userDictionary

The schema element `userDictionary` in the `ino` namespace is used to hold user-defined dictionaries for Japanese in Tamino. It has the following structure:




A `userDictionary` consists of a sequence of entries, along with an optional comment describing the dictionary. The element `entry` defines a single dictionary entry. It has no contents. The information about a single entry is contained in the following attributes:

`ino:word` **(required)**
One or more code values of Han characters for the respective language. This sequence constitutes a “word” that is not segmented by the tokenizer.

`ino:pos` **(optional)**
This attribute is reserved for future use.

`ino:comment` **(optional)**
A comment to describe the definition of this single dictionary entry.

The attribute `ino:docname` determines the dictionary to be used. It has the value "udict-jp".

As an example, you store a user-defined dictionary for Japanese in `ino:vocabulary/ino:userDictionary/udict-jp`.

### Defining a User Dictionary in Tamino

There are two steps for storing a user-defined dictionary in a Tamino database. First, you have to write a document that conforms to the `ino:userDictionary` schema described above. You can find an example below. In a second step, you can load this user dictionary into Tamino.

#### To Store a User Dictionary

Make sure that the Tamino database server is running.

1 In the Tamino Interactive Interface, enter the URL of the Tamino database in the field **Database URL**.

2 In the **Collection** input field, enter ino:vocabulary/ino:userDictionary/*<dictionary>*.

3 Use the **Browse** button right to the input field of the **Process** button to locate the dictionary file.

4 Choose the **Process** button. Tamino overwrites the current version of that user dictionary.

5 In the Tamino Manager, restart the database server so that the changes take effect.

You can check your changes by querying `ino:userDictionary` against the collection `ino:vocabulary`.

# Index