



**T O V E K**

---

## **Tovek Server: Categorisation**

---

**Version 5.30 and higher**

---

**Administrator Guide**

---





# Tovek Server

---

Categorisation

Version 5.30 and higher

# Copyright

---

Information contained in this document may change without notice. The software described herein is subject to and protected by copyright law and international agreements related to intellectual property and can be handled according to the licence agreement only.

© Copyright TOVEK. All rights reserved.

© Copyright Autonomy Corporation. All rights reserved.

## Registered Trademarks

Tovek Server, Tovek Tools and Tovek Agent are registered trademarks of TOVEK.

## Other Trademarks Recognition

All other companies and product names are trademarks or registered trademarks of their respective owners.

**TOVEK**

<http://www.tovek.com>

---

# Content

---

<b>Introduction .....</b>	<b>6</b>
<b>Introduction to Categorisation.....</b>	<b>7</b>
Using Categorisation in Client Applications.....	7
Categories.....	7
Sets of Categories.....	7
Document Categorisation.....	8
Manual Categorisation .....	8
One-off Categorisation .....	9
New Document Categorisation.....	9
<b>Creating Categories and their Description .....</b>	<b>11</b>
General Category Properties.....	11
Properties of in Advance Defined Categories.....	11
Properties of Automatically Created Categories.....	12
Sets of Categories.....	12
Defining Rules for Category Sets .....	13
Examples of Rules for Category Sets.....	22
<b>The ts_makecat.exe Tool .....</b>	<b>25</b>
Parameters for Start up.....	25
The List Action.....	25
The import Action .....	26
The export Action .....	28
The categorize Action.....	29
The history Action.....	29
Application Configuration .....	30
<b>Categorisation Configuration .....</b>	<b>33</b>
Modules for Document evaluation.....	33
TS_CatEvalFields.....	33
The TS_CatEvalProfiles Module .....	34
Modules for Categorisation Results Recording .....	35
Category Management and Results Evaluation Module.....	36
<b>Appendix: Syntax of Perl Regular Expressions .....</b>	<b>39</b>

# Introduction

---

This manual contains information about implementation of document categorisation within Tovek Server.

## Manual Structure

This manual contains the following chapters:

**Introduction to Categorisation** chapter describes basic categorisation terms.

**Creating Categories and their Description** chapter deals with category creation, their properties and description.

**The ts\_makecat.exe Tool** chapter describes ts\_makecat.exe application which makes it possible to work with categories in Tovek Server in bulk.

**Categorisation Configuration** chapter describes the technical part of categorisation implementation in Tovek Server and its configuration.

## Introduction to Categorisation

---

Document categorisation in the Tovek Server conception is a process in which the documents are assigned to the categories according to rules given in advance. During the evaluation of individual documents a score determining whether the document should be classified in this category is calculated for each category. Documents therefore can be included in several categories simultaneously, or they do not have to be part of any category.

### Using Categorisation in Client Applications

Categorisation results can be used for search simplification and refining.

The categorisation module allows a user to restrict the search only to a set of documents which are classified in some categories or are not classified in any categories.

The search results can now be presented not only as list of documents and their fields, but also the number of these documents in individual categories. Results withdrawal can then also be restricted to documents from selected categories.

### Categories

There are two basic category types in Tovek Server according to the way in which they are created:

1. Categories defined in advance
2. Categories created automatically

Categories defined in advance have given rules fixed in advance based on fulltext queries. The rules for these categories need to be prepared manually, and the individual categories need to be inserted into the whole structure.

Automatically created categories are categories which are created during the document evaluation (their field values). How the exact categories are derived is once again defined by rules. However, the rules do not apply to categories, but to the whole *category sets*.

### Sets of Categories

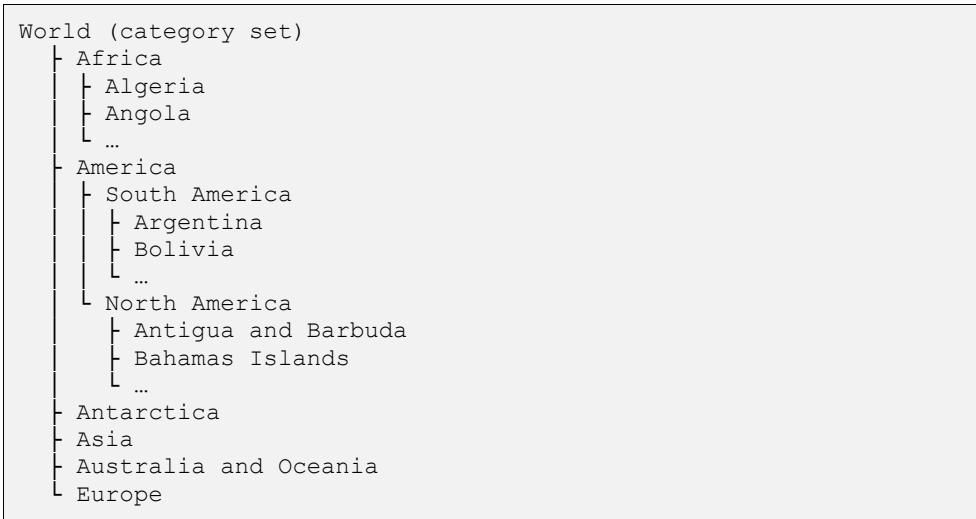
Category sets are designed to divide the categories into groups according to their subjects. Each category is always inserted into one set only. Each set can contain categories of one type only; all categories are defined in advance or they are all created automatically.

The categories can be organised within their set into a hierarchical structure in which each category has at most one parent category and an unlimited number of child categories. All categories without parent categories form category structure roots. If none

of the categories in the set has a parent category, then this set constitutes a simple list of various categories.

Inserting categories into hierarchical structure is important not only for their arranging and presentation, but primarily for the subsequent work with categorised documents, such as search results evaluation, restricting the set of searched documents and navigating above documents. It generally applies that each document belonging to the category also belongs to its parent category (with the exception of parent category with own definition, see Creating Categories and their Description chapter on page 11).

A typical category set containing hierarchically organised categories can describe geographical segmentation, for example:



## Document Categorisation

Document assignment to categories can be performed in Tovek Server in three different ways:

1. Manual assignment
2. One-off automatic assignment
3. Automatic new document assignment

## Manual Categorisation

Manual categorisation allows users to manually change assignment of individual documents to categories. The users can not only change the document score within the category to which it has already been assigned, but also assign documents to new categories, and possibly to delete existing assignment.

Tovek Server always saves the last change made by a user together with original automatically derived values, which makes it possible to annul this change. Moreover, on a server it is possible to save the full history of these changes for further processing. Document categorisation inserted by a user always has priority over calculated values in Tovek Server; therefore even a possible change in document assignment to categories does not overwrite user settings.

## One-off Categorisation

This categorisation type is performed using `ts_makecat.exe` tool (see The `ts_makecat.exe` Tool chapter on page 25) and is designed primarily for initial categorisation of documents which are present in the Tovek Server databases.

This document categorisation style is also good to use in the following cases:

1. New or changed category definitions - in this case it is possible to change document assignment to categories only within these categories.
2. New sets of categories - system makes it possible to run categorisation of current documents only with one category set.

One-off categorisation executed using `ts_makecat.exe` keeps values of all previous changes performed by users.

## New Document Categorisation

Categorisation of newly indexed documents is processed automatically in periodical time intervals depending on Tovek Server configuration (see Categorisation Configuration chapter on page 33).



---

## Creating Categories and their Description

---

There are two types of category sets in Tovek Server; sets with categories defined in advance and sets with automatically created categories.

This chapter describes how the rules for categories from each of these types of sets are defined; it introduces a tool for creating these definitions and presents how these definitions can be imported to Tovek Server.

### General Category Properties

The following parameters are related to all categories regardless of their type:

- *Id* – category id is assigned automatically during the first import of category definitions to Tovek Server. Category Id consists of a 4-byte number where the highest byte corresponds to set id to which this category belongs.
- *Name* – a category name can be any string with maximum length of 1024 characters. This name is designed only for client applications for presenting categories to the user, it does not have any impact on categorisation results or their evaluation and can be therefore changed anytime, if needed.
- *PSI* – PSI is any string with maximum length of 1024 characters which is designed as user defined identifier which can be used in client applications. This parameter is not used in Tovek Server; therefore the uniqueness of its values is not checked.
- *set Id* – identifies category set which the specified category belongs to.
- *Parent Category Id* determines the position of the current category within the hierarchical structure of all categories in one set; the parent category therefore has to be from the same category set as the current one. If the parent category id is not specified (0 value), the current category is placed on the highest level and thus forms one of the hierarchical structure roots. The category's position within the hierarchical structure plays an important role during the results evaluation. Each category without a definition determining which documents should be added to this category contains all documents from the categories which are its children.
- *User Name* – can contain the name of the user who last changed the category definition. The value is limited to 128 characters.
- *Latest Change Time* - contains time data when the category definition was changed last.
- *Status* - determines whether the category is active and should be included in the new document evaluation or if the category is still non-active.

### Properties of in Advance Defined Categories

The rules for evaluating documents according to categories of this type are defined as fulltext queries. The exact definition consists of the following parameters:

- *Query* – fulltext query the syntax of which is the same as the syntax of queries for searching, possibly profiling; and it depends on the chosen query parser. The score resulting from document evaluation against this query is the same as the score with which the document was classified in the category. The length of the fulltext query is limited only by the used fulltext technology (the maximum length is 32KB for Verity VDK).
- *Query locale* – defines with which locale the category query will be evaluated. The available locale differs according to Tovek Server licence conditions.
- *Query parser* – determines acceptable query syntax. The available parser types are the same as types for searching; the basic ones are the following:

```
o TS_QueryParser_Simple
o TS_QueryParser_Advanced
o TS_QueryParser_FreeText
o TS_QueryParser_Internet
o TS_QueryParser_Boolean
```

- *Score threshold* – defines the score which documents must exceed to be classified in the given category.

In most cases the definition contains only categories at the lowest level of the hierarchical structure. The other categories have an empty definition and act as if they have classified documents of all their child categories. This assignment is not saved anywhere and is implemented during the user requests evaluation. As a result of this it is possible to change the categories' structure without the necessity of performing document categorisation again.

Tovek Server of course also enables inner categories to have their own definition. The documents are then assigned to these categories in the standard way based on their definition regardless of which documents belong to their child categories.

## Properties of Automatically Created Categories

Automatically generated categories do not have their own definition and hence do not contain any parameters other than the following one:

- *Rule result* – contains the result of document evaluation against the rules specified at the relevant category set (**CATEGORY** variable value, see the following chapter).

## Sets of Categories

The categories are joined together into sets. Each set is characterised by the following parameters:

- *Id* – set id must be unique within the Tovek Server installation. Its value can be between 1 and 255.

- *Name* – the category set name can be any string with maximum length of 1024 characters which is used for viewing in client applications only.
- *FieldBased* – the value of this parameter determines whether the set contains in advance defined categories which have a definition based on a query (value `false`), or categories which can be created automatically based on categorised documents (value `true`).
- *User Name* – can contain the name of the user who last changed the category set definition. The value is limited to 128 characters.
- *Latest Change Time* - contains time data of when the category set definition was changed last.
- *Status* determines whether the category set is active and whether it should be involved in the new document evaluation.
- *AutoCreate* – this value is valid only for sets with flag *FieldBased* == `true`. Its value determines whether new categories will be created based on the *AutoCreateRule* rule (value `true`) or whether the documents not meeting any existing categories will not be classified in any set (value `false`).
- *AutoCreateRule* – is a rule defining how a category in which the document ought to be classified is derived from field values (see the following chapter).

## Defining Rules for Category Sets

The rules apply only to category sets which contain automatically created categories. Assigning documents to these categories is derived from their fields' values.

As mentioned in the previous chapter, each automatically created category has a valid value in the *rule result* field. This value can be any string. During a new document categorisation based on the rule specified at a category set and its fields' values, a rule result is calculated which presents one concrete value or value list. The document is sequentially classified into categories which contain in the *rule result* field a value identical to the calculated value possibly contained in the calculated list. If no such category exists and creating of new categories within the given set is permitted, a new category with corresponding value in *rule result* field is created, and the document is included in it. The rules for the category set can also contain a part defining how the name of the newly created category should be derived. If this definition does not exist, the category name will be identical to a value in *rule result* field.

Definition of rules is based on regular expressions using which the field values are analysed. Accepted syntax of regular expressions corresponds to the syntax of regular expressions in Perl (see Appendix: Syntax of Perl Regular Expressions on page 39). Each rule expression is specified on a separate row.

Empty rows and rows starting with # character are considered to be comments and are ignored during the rule processing.

Warning: # character must be the first character of a row!

The rules can use variables for saving results arising in individual evaluation steps. A value can be assigned to each variable once only; sequentially it can be used repeatedly for this value reading. Results of expressions evaluated during the rule processing can be text, integer, decimal, date or binary values or their lists.

Saving `expression` expression result into `VARIABLE` variable will be recorded in the following way:

```
# It assigns expression value into the variable
VARIABLE=expression
```

### Predefined Variables

The final result of the category set rules evaluation determining in which categories the document should be classified must be saved in the `CATEGORY` predefined variable. In the result this variable can contain either a string value or list of string values. These values determine the categories in which the document is classified. The relevant categories are found according to their *rule result* field. Each rule must contain an expression with value assignment to this variable.

In addition to the `CATEGORY` variable there exist another three predefined variables which can be, but don't have to be, part of the rule definition. `CATEGORY_NAME` variable is used if the category in which the document should be classified does not yet exist and must be created. This variable value then defines the name of the newly created category. If `CATEGORY_NAME` is not defined in the rule, the name of the new category will be according to the `CATEGORY` variable.

```
# It sorts the documents according to the value of their SOURCE
field
CATEGORY=$SOURCE

# The categories have names "Source: SOURCE"
CATEGORY_NAME="Source: " + $SOURCE
```

The `HIERARCHY` predefined variable makes it possible to derive the hierarchical structure of categories from the `CATEGORY` variable value. The `HIERARCHY` variable simply defines a separating character of individual levels. In this way for categories it is possible to use field values which contain by default hierarchical information, e.g., path to file, but also other fields such as date, where the parent category can be year. Then it is possible to combine values of different fields and in this way create the hierarchical structure. During the creation of a new category, the full path of parent categories is always created also. The separator defined in the `HIERARCHY` variable also applies to the variable in `CATEGORY_NAME`, so it is possible to define simply the names of categories on all levels.

```
# Inserts documents to SOURCE/VOLUME category
CATEGORY=$SOURCE + "#" + $VOLUME
CATEGORY_NAME="Source: " + $SOURCE + "#Volume " + $VOLUME
```

```
HIERARCHY="#"
```

The previous example combines values of SOURCE and VOLUME fields and inserts like this the documents into the hierarchical structure of categories which has the following structure assuming that documents with the following field values were categorised:

- source=CTK, volume=2009
- source=CTK, volume=2008
- source=BBC, volume=2009

The structure of created categories is as follows:

```
The sources by volumes (categories set)
├─ Source: CTK           [rule result="CTK"]
│  └─ Volume 2008      [rule result="CTK#2008"]
│     └─ Volume 2009   [rule result="CTK#2009"]
└─ Source: BBC           [rule result="BBC"]
   └─ Volume 2009      [rule result="BBC#2009"]
```

The last predefined variable is LOG\_LEVEL variable. The LOG\_LEVEL variable does not have any impact on document evaluation, it only changes the behaviour manner of evaluation module if the given document was categorised. Permissible values for this variable are individual log levels: DEBUG, INFO, WARNING, ERROR and FATAL. The value of this variable determines how the information about not classifying a document in any category will be logged. The standard value for cases where this variable is not defined is determined by configuration of evaluation module.

## Automatic Variables

Individual expressions can refer not only to variables which are defined in the given rule, but also to automatic variables containing values of categorised document fields. These variables have the same names as document fields where all letters are switched to uppercase. The document fields are listed in the **TS\_SourceMonitor** module configuration.

For values extraction from FIELD document field according to the regular expression it is necessary to use the following syntax:

```
FILED%type1type2...typeN:regular expression
```

During the given expression evaluation, N of variables is created, which will be named as FIELD\_1 to FIELD\_N, each of the specified type, and will be initialised to values of individual regular expression submatch. New variable types are defined after the % separating character using the following characters:

- s – string type variable
- n – integer variable

- *f* – decimal value
- *d* – date value
- *b* – boolean value

Variables created in this way can be used in other expressions for final results derivation.

Warning: As the values in variables cannot be rewritten, any row adding a value to an automatic variable will be ignored!

## Variables and Constants in Expressions

The expressions serve for further possible calculations above already derived values. Each expression must be specified on a separate row. Within the expressions numerical or text constants are written directly, decimal points are used for variable values, text constants are specified in quotation marks:

```
NUMBER=3.14  
NAME="Ludolph number"
```

As a result of these expressions two variables will be created, `NUMBER` variable of decimal type with value 3.14 and `NAME` variable of string type with value „Ludolph number“.

The values in these variables can be used in other expressions using `$` character:

```
TITLE=$NAME
```

The string variable `TITLE` contains the value „Ludolph number“.

## Operators in Expressions

Variables and constants can serve in expressions as input parameters for the whole range of functions or can be connected using several operators.

There is support for standard arithmetical operations like addition, subtraction, multiplication and division. All these operations are defined for numerical parameters. The values of date type can be added and subtracted. For string values only addition is defined. If operator parameters do not have the same type, the value on the right side will be converted to a type which has the operand on the left site. The following default conversions are supported:

- *string* -> integer value, decimal value, date, boolean
- *integer value* -> decimal value, string
- *decimal value* -> integer value, string
- *date* -> string
- *boolean* -> string

If it is not possible to convert the right operand value, the calculation fails.

Warning: Because of default conversion the operand rank is important!

```
# The VAR1 result is number 10
VAR1=5 + "5"

# The VAR2 result is string "55"
VAR1="5" + 5
```

For value comparison, <, >, == and != operators are defined. These operators accept all supported types, except boolean type values, which cannot be compared using < and >. As well as arithmetic operators, these ones also support default conversion of values to the right operator type and therefore even here the operand order is important:

```
# Both of the following conditions are met
11 < "100"
"100" < 11
```

The last set of operators which are supported are logical operators && (logical and), || (logical or) and ! (logical not).

In addition to the above mentioned operators it is possible to use during the definition of individual rules a wide range of other functions which are described in the following part.

Warning: During their usage within the rules it is important to adhere to the writing of spaces between individual parts of expressions!

## Conditions in Expressions

### IF ( condition , expression1 , expression2 )

If the condition value is true, the function IF returns the value of expression1, otherwise it returns the value of expression2.

```
# It sorts according to price into two categories: cheap and
expensive goods
CATEGORY=IF ( $PRICE < 1000 , "cheap goods" , "expensive goods" )
```

## Functions for Work with Lists

### LIST ( expression1 , expression2 , ... , , expressionN )

It creates a list of values which correspond to individual expression values. The lists contain only values of basic types. If one of the parameters is a list type, then the resulting list will contain all its items.

```
# Both variables contain the same list of three numbers on order
1, 2, 3
LIST1=LIST ( 1 , 2 , 3 )
```

```
LIST2=LIST ( LIST ( 1 , 2 ) , 3 )
```

### **GETSIZE ( expressionList )**

This function returns the number of items in the list. If `expressionList` in the parameter does not return the list of items, the returning value of `GETSIZE` function is equal to one.

```
# All three variables have value 3
LENTH1=GETSIZE ( $LIST1 )
LENTH2=GETSIZE ( $LIST2 )
LENTH3=GETSIZE ( LIST ( 1 , 2 , 3 ) )

# Both LENTH4 and LENTH5 have value 1
LENTH4=GETSIZE ( "text" )
LENTH5=GETSIZE ( $LENTH4 )
```

### **MIN ( expressionList )**

`MIN` function returns the smallest item in the list. If the list items are not comparable, the function fails.

### **MAX ( expressionList )**

`MAX` function returns the biggest item in the list. If the list items are not comparable, the function fails.

```
# ITEMMIN variable contains value 1, ITEMMAX 3
ITEMMIN=MIN ( $LIST1 )
ITEMMAX=MAX ( $LIST1 )

# The list contains incomparable items, the calculation fails
ITEM=MIN ( LIST ( 1 , "text" , 3 ) )
```

### **GETELEMENT ( index , expressionList )**

This function returns the list item on given position. The items are numbered in the list from 0 to `GETSIZE( expressionList ) - 1`. If `index` is out of the right range, the `GETELEMENT` function returns the value `empty`, which acts in a similar way as an empty string, possibly 0 depending on expression in which it occurs.

### **CONCAT ( expressionList )**

`CONCAT` returns a string created by joining all list values. If the input list contains values of an other than string type, they are all first converted to string and sequentially joined into a result.

```
# TEXT contains "text123" string
```

```
TEXT=CONCAT ( LIST ( "text" , $LIST1 ) )
```

### **SPLIT ( separator , expression)** **SPLIT ( separator , expressionList )**

SPLIT function returns a value list which arose by splitting the expression value according to specified separators; separators are then part of the resulting list. In the case of the second variant of this function, all values in the list are divided and the results are joined into one resulting list.

```
# LIST3 as well as LIST4 contains LIST ( "1" , "2" , "3" ) value
LIST3=SPLIT ( ":" , "1:2:3" )
LIST4=SPLIT ( ":" , LIST ( "1:2" , 3 ) )

# LIST5 is LIST ( "1" , "2" , "3" , empty , "4" )
LIST5=SPLIT ( ";;" , "1:2;3;;4" )
```

### **FULLTRIM ( expressionList )**

This function converts all list items into a string and removes empty characters at their beginning and end. In the end all empty strings and empty values are excluded from the resulting list.

```
# LIST6 is LIST ( "1" , "2" , "3" , "4" )
LIST6=FULLTRIM ( $LIST5 )

# LIST7 is LIST ( "text" , "1" )
LIST7=FULLTRIM ( " text " , " " , 1 )
```

### **FILTER ( expression , filter )**

The input parameter `expression1` defines the list of values from which those meeting the specified filter should be chosen. If the expression is not a list but the basic value, then a temporary list containing this value will be created during the evaluation. The `filter` parameter defines a simple list of all acceptable values - as standard it is a list type or it is internally converted to it. The result of this function is then a list of all expression values which are present in the filter in the unchanged order. The comparison is performed using `==` operator, which supports the default conversion to string.

```
# LIST8 is LIST ( 1 , "1" )
LIST8=FILTER ( LIST ( 1 , 2 , "1" , "2" ) , LIST ( 1 ) )
```

## **Functions for Explicit Type Conversion**

All these functions convert values of one type to another one. If the conversion fails, evaluation of the converse expression fails.

**NUMBER ( expressionEx )**

It converts the `expressionEx` value to an integer. Conversion of string and decimal number type value is supported.

**FLOAT ( expressionEx )**

It converts the `expressionEx` value to decimal number. Conversion of string and integer type value is supported.

**STRING ( expressionEx )**

It converts `expressionEx` value to string.

**DATE ( expressionEx )**

It converts the `expressionEx` value to date with time information. Only a conversion from text value of YYYY-MM-DD or YYYY-MM-DD hh:mm:ss formats is supported.

**BOOLEAN ( expressionEx )**

It converts string value to boolean. Acceptable values are 0,1, TRUE and FALSE.

```
# NUMBER1 as same as NUMBER2 is 10000
NUMBER1=NUMBER ( "10 000" )
NUMBER2=NUMBER ( "10000" )

# DATE has value of September 4, 2009 0 hours 0 minutes 0 seconds
DATE1=DATE ( "2009-09-04" )

# DATE2 has value of September 4, 2009 12 hours 11 minutes 10
seconds
DATE2=DATE ( "2009-09-04 12:11:10" )

# DATETXT has value of "2009-09-04"
DATETXT=STRING ( $DATE2 )
```

**Functions for Work with data**

**YEAR ( expressionEx )**

It extracts year as a number from `expressionEx` value which must be of date type.

**MONTH ( expressionEx )**

It extracts month as a number (1-12) from `expressionEx` value which must be of date type.

**DAY ( expressionEx )**

It extracts day as a number (1-31) from `expressionEx` value which must be of date type.

**DAYS ( expressionEx )**

It returns the number of days from January 1, 0001. The `expressionEx` parameter must be of date type.

```
# YEAR is 2009, MONTH is 9, DAY is 4
YEAR=YEAR ( $DATE1 )
MONTH=MONTH ( $DATE1 )
DAY=DAY ( $DATE1 )
```

**Functions for Work with Strings****TRIM ( expressionEx )**

It removes empty characters from the beginning and the end of the `expressionEx` string.

**Shortened Format of Functions**

The majority of these functions can be used in a shortened format.

The first variant of shortened format makes it possible to leave out the `LIST` function while defining input parameter. This variant can be used for all functions which in the preceding description have the `expressionList` parameter. The following formats are equivalent:

- `FUNCTION( LIST ( expression1, ... , expressionN ) )`
- `FUNCTION( expression1, ... , expressionN )`

```
# The following rows are equipment:
ITEMMIN=MIN ( LIST ( 1 , LIST ( 2 , 3 ) ) )
ITEMMIN=MIN ( 1 , LIST ( 2 , 3 ) )
ITEMMIN=MIN ( 1 , 2 , 3 )
```

The second variant of the shortened format is related mainly to converse functions, but also to other functions which work over individual values, and allow these functions to apply to the value list. These functions then return result list, as if they had been applied to individual items of input list. The `LIST` function can once again be left out of the list format. This shortened format can apply to all parameters named in the previous description as `expressionEx`. The following formats are equivalent:

- `LIST ( FUNCTION( expression1 ) , ... , FUNCTION( expressionN ) )`

## Creating Categories and their Description

---

- FUNCTION( LIST ( expression1 , ... , expressionN ) )
- FUNCTION( expression1, ... , expressionN )

```
# The following rows are equivalent:
LIST_OF_NUMBERS=LIST ( STRING ( 1 ), STRING ( 2 ) )
LIST_OF_NUMBERS=STRING ( LIST ( 1 , 2 ) )
LIST_OF_NUMBERS=STRING ( 1 , 2 )
```

## Examples of Rules for Category Sets

1. In the simplest case the document field value directly determines the category and its name in which the document should be classified.

```
# It categorises documents according to COLUMN field, individual
categories
# have names identical to this field value
CATEGORY=$COLUMN
```

2. If the kind of category names generating needs to be changed, the CATEGORY\_NAME variable will be defined.

```
# It categorises documents according to COLUMN field, individual
categories
# have names "Column: " + the value of this field
CATEGORY=$COLUMN
CATEGORY_NAME="Column: " + $COLUMN
```

3. This example supposes that the values in the COLUMN field have universal form of „column: Name“, where in the names of categories the common prefix “column:” should be left out. A regular expression will be used for the column name extraction.

```
# It categorises documents according to COLUMN field, individual
categories
# have names "Column: " + the value of this field
COLUMN%s:^Column: (.*)$
CATEGORY=$COLUMN_1
```

4. If the FILE document field contains a path to the file, then it is possible to take the folder structure as the categories structure.

```
# It categorises documents according to the file path
CATEGORY=$FILE
HIERARCHY="//"
```

5. The following example categorises documents into the categories structure year/month. The date is taken from the DATE document field value. The YYYY-MM-DD hh:mm:ss standard format is assumed.

```
# It categorises documents according to year and month in the
DATE field
DATE%ss:^( [0-9]* ) - ( [0-9]* ) . * $
CATEGORY=$DATE_1 + "#" + $DATE_2
HIERARCHY="#"
```

6. This example substitutes month numbers in the category name with their names and the categories representing years will have names in form of „Year 2009“.

```
# It categorises documents according to year and month in the
DATE field
DATE%ss:^( [0-9]* ) - ( [0-9]* ) . * $
YEAR="Year " + $DATE_1
NAMES=LIST ( "Dummy" , "January" , "February" , "March" , "April"
, \
           "May" , "June" , "July" , "August" , \
           "September" , "October" , "November" , "December" )
MONTH=GETELEMENT ( NUMBER ( $DATE_2 ) , $NAMES )
CATEGORY_NAME=$YEAR + "#" + $MONTH
CATEGORY=$DATE_1 + "#" + $DATE_2
HIERARCHY="#"
```

7. The next example sorts documents according to authors listed in AUTHOR field, and the individual names are separated by semicolon or comma. Moreover this example contains a list of authors who should be taken into account - the rest of authors are ignored.

```
# It categorises documents according to authors listed in the
AUTHOR field
AUTHORS=FULLTRIM ( SPLIT ( ";," , $AUTHOR ) )
NAMES=LIST ( "Adam" , "Eve" )
CATEGORY=FILTER ( $AUTHORS , $NAMES )
```

8. This example categorises documents according to the FORTUNE field value into three categories - small firm, medium firm, large firm.

```
# It categorises documents according to the value in the FORTUNE
field
CATEGORY=IF ( 100 > $FORTUNE , "Small firm" , \
           IF ( 10000 > $FORTUNE , "Medium firm" , "Large firm" ) )
```



## The ts\_makecat.exe Tool

The ts\_makecat.exe program makes it possible to import and export category definitions and their sets and enables one-off categorisation of documents from selected collections against selected sets, possibly categories. This program is closely connected to Tovek Server and uses its modules and configuration.

### Parameters for Start up

When running the ts\_makecat.exe without parameters, it lists the help with program parameters description.

The two basic parameters which must be specified with each program start up are:

- `-config configFile` – defines the path to the application configuration file.
- `-action actionName` – defines an action which should be carried out. The following values are valid names of actions:
  - o `list` – lists sets of categories
  - o `import` – imports or updates definitions of category sets, possibly categories alone
  - o `export` – exports definitions of category sets together with their categories
  - o `categorize` – categorises documents of selected collection against selected categories
  - o `history` – returns maximal or minimal value of a field in collections for use in **TS\_SourceMonitor** module history files.

Depending on the action, further parameters must be defined at the application start up on the command line.

### The List Action

```
ts_makecat.exe -config configFile -action list
```

The list action does not need any other parameters. The output of the program is a list of category sets and their properties, including category set names and ids or number of categories in the set.

```
1. CategorySet
-----
Name:          Tovek Topics
Id:            1
Changed by:    splichal
Changed Date:  2008-11-04 00:00:00
```

```
Category Count: 49
Status:         0
Type:          query based
```

## The import Action

```
ts_makecat.exe -config configFile -action import -file inputfile
-update updating
```

It serves to import or update category definitions, and possibly their sets. A new definition must be saved in an xml file the path of which is passed to the `ts_makecat` program using the `file` parameter. The `update` parameter determines whether the input file defines only new categories and sets, or whether their update will be performed. In the case of updating, new categories can of course be added.

```
<categorySets xmlns:ts="TovekServer" xmlns="TovekServer">

  <categorySetField name="Column" id="1" changedBy="author"
    status="0" autoCreate="true"
  >
    <autoCreateRule>CATEGORY=$COLUMN</autoCreateRule>
  </categorySetField>

  <categorySetQuery name="World" id="2" changedBy="author"
    status="0"
  >
    <category name="Africa" changedBy="author" status="0">
      <category name="Algeria" changedBy="author" status="0">
        <query locale="uni" threshold="0" parser="simple">
          Algeria
        </query>
      </category>
    </category>
    <category name="Angola" changedBy="author" status="0">
      <query locale="uni" threshold="0" parser="simple">
          Angola
        </query>
      </category>
    ...
  </category>

  <category name="America" changedBy="author" status="0">
    ...
  </category>
</categorySetQuery>
</categorySets>
```

The previous example shows a part of definitional file which contains the definition of two category sets, the first one is a set of automatically created categories, the second one is a set with predefined categories.

Category set definitions are included in the `categorySets` global section. Each set has its own element according to its type.

### The `categorySetQuery` Element

This element contains a set definition with predefined categories and has the following parameters:

- `id` – category set id (value from 1 to 255)
- `name` – category set name
- `changedBy` – name of the user who made the change
- `changedDate` – [optional] – date of last change
- `status` – set status, 0 is for active set

Individual categories contained in the set together with their structure are described using embedded elements of `category` type.

### The `categorySetField` Element

Using this element the sets of automatically created categories, i.e., categories which are derived from the field values, are defined. The `categorySetField` element has the following parameters:

- `id` – category set id (value from 1 to 255)
- `name` – category set name
- `changedBy` – name of the user, who made the change
- `changedDate` – [optional] – date of last change
- `status` – set status, 0 is for active set
- `autoCreate` – defines whether new categories should be established automatically. Accepted values are `true` and `false`.

In the same way as for the previous set type, the categories of the set are defined using embedded `category` elements. If the category is created automatically, no category needs to be specified.

The embedded `autoCreateRule` element then defines a rule according to which the documents are categorised. The rule definition is described in the previous chapter.

### The `category` Element

The `category` element defines the concrete category regardless of whether it is a predefined or automatically created category, and it contains the following parameters:

- o `id` – category id
- o `name` – category name
- o `PSI` – [optional] – category PSI
- o `changedBy` – name of the user who made the last change
- o `changedDate` - [optional] – date of last change
- o `status` – category status , 0 is for active

The `category` element can contain other embedded categories (elements of `category` type) and in this way describe the hierarchical structure of categories.

Based on the `category` type this element can contain an embedded `query` element defining a query of predefined categories or `expressionResult` element containing the rule result for automatically created categories.

### The query Element

This element defines a query of predefined categories together with all its attributes. Query text is specified as element `#CDATA` and is detailed by the following attributes:

- o `locale` – defines query locale which affects language side of query processing
- o `threshold` – defines minimal score of document in specified category.  
Permissible values are from 0 to 10000
- o `parser` – specifies query parser. Accepted values of this attribute are `simple`, `advanced`, `freetext`, `internet` and `boolean`

### The expressionResult Element

The `expressionResult` element contains the rule result as `#CDATA`.

## The export Action

```
ts_makecat.exe -config configFile -action export -set set_id -file outputfile
```

This action makes it possible to export definitions of categories and category sets into an xml file. The format of the created file is the same as the format of the files for import.

The destination file is entered using the `file` parameter.

If only the definition of one category set should be exported, it is necessary to specify this set using the `set` parameter. If this parameter is not specified on the command line, all category sets are exported.

---

## The categorize Action

```
ts_makecat.exe -config configFile -action categorize -set set_id  
-category category_id -source source_name -mode mode_name
```

The categorize action is designed for one-off document categorisation. Using parameters transferred on the command line, it can be specified which documents against which category set (possibly which categories) should be processed.

By default all documents from all databases which are specified in the Tovek Server configuration (in the `ts_makecat` configuration program) are categorised. The set of documents for processing can be limited by using the name of database or collection from which these documents should be withdrawn using the `source` parameter.

If neither `set` parameter nor `category` parameter is specified on the command line, the documents will be categorised against all category sets. Using the `set` parameter the process of categorisation can be limited to a concrete category set and the `category` parameter even makes it possible to specify the concrete category. This possibility is appropriate especially if the definition of one or several categories changes.

The last parameter of categorize action which can be specified is the `mode` parameter. Its `normal` or `fastinsert` value impacts the method of results recording in the repository. The standard value is `normal` and ensures errorless progress even if in the repository there are already some results from previous categorisations of the documents against the specified set. But this method of results recording is slower than the second variant. In cases where there are no previous results (e.g. by the first document categorisation; it always relates to the current document set and current category set), it is possible to set the repository `mode` to `fastinsert` and speed up the process of categorisation.

## The history Action

```
ts_makecat.exe -config configFile -action history -source  
source_name -field field_name -value which_value
```

It is designed to withdraw the maximal or minimal value of a field in collections for use in **TS\_SourceMonitor** module history files. The field name the value of which should be ascertained is defined using the `field` parameter. If the maximum value of this field within individual collections should be ascertained, then the `value` parameter should be set to the `max` value. Failing that, if the minimal value is needed, the `value` parameter should be set to `min`.

Using another parameter - `source` you can limit the collections for which the minimal or maximum value of the specified fields is calculated.

## Application Configuration

The main configuration file of application is as standard named *ts\_makecat.xml*. The path to this file is specified as mandatory `config` parameter by each running.

Typically this file contains links to configuration files for Tovek Server modules which are used for document categorisation:

- **TS\_LicenceHolder** – (*ts\_licence.xml*) – configures the path to the Tovek Server licence file
- **TS\_LoggerWrapper** – (*ts\_logger\_makecat.xml*) – configures logging. In this case it is appropriate to use different configuration of the logging module so the logs of Tovek Server will not mix with the ones of the run application.
- **TS\_PluginFactoryRepository** – (*ts\_plugins.xml*) – defines fulltext plug-ins used for work with individual databases
- **TS\_DatabaseRepository** – (*ts\_databases.xml*) – configures list of connected fulltext databases and their collections. Only documents from these databases can be categorised.
- **TS\_EventDispatcher** – (*ts\_eventdispatcher.xml*) – configures module for events management
- **TS\_Categorizer, TS\_CategoryStorageODBC** – (*ts\_catengine.xml*) – defines central categorisation module and results database repository
- **TS\_CatEvalProfiles, TS\_CatEvalFields** – (*ts\_catevaluate.xml*) – describes modules for category evaluation

Moreover the *ts\_makecat.xml* file contains configuration of the application main module named **TS\_MakeCatConf** and contains only information about under which user and with which password the documents should be withdrawn from the collections.

The password is entered in an encrypted form which can be obtained using the `pwencode.exe` program.

For both parameters it is also possible to insert the `$prompt` value, which causes the user to be prompted to insert his/her name and password only after starting the application.

```
<?xml version="1.0"?>

<waspc-config xmlns:wasp="urn:WaspServer"
              xmlns:makecat="urn:TS_MakeCat"
>
  <!-- Wasp minimal client configuration -->
  <wasp:import ref="wasp/client-core.xml"/>

  <!-- Tovek Server modules -->
  <wasp:import ref="ts_licence.xml"/>
  <wasp:import ref="ts_logger_makecat.xml"/>
</waspc-config>
```

```
<wasp:import ref="ts_plugins.xml"/>
<wasp:import ref="ts_databases.xml"/>
<wasp:import ref="ts_eventdispatcher.xml"/>
<wasp:import ref="ts_catengine.xml"/>
<wasp:import ref="ts_catevaluate.xml"/>

<!-- Main ts_makecat.exe module -->
<wasp:module wasp:class="TS_MakeCatConf" />
<makecat:options>
  <option name="user" value="$prompt"/>
  <option name="password" value="$prompt"/>
</makecat:options>

</waspc-config>
```



# Categorisation Configuration

Categorisation implementation in Tovek Server is divided into the three module groups. The first group contains modules for document evaluation and their assignment to categories, the second group contains modules for results saving, and the last one contains a module which manages categories and their sets and is designed for search results evaluation.

## Modules for Document evaluation

This module group contains two concrete implementations:

- **TS\_CatEvalFields**
- **TS\_CatEvalProfiles**

By default the configuration of both modules is located in the same file - *ts\_catevaluate.xml*. The aspects of their configurations are described in the following chapters.

### TS\_CatEvalFields

This module sorts documents into automatically created categories based on their field values. Based on the rules defined in sets of these categories, it derives not only in which category the document should be classified, but also possibly which other categories need to be created, what the names of the new categories will be, and how they should be classified into their structure.

All parameters affecting this module behaviour are defined in the category sets rules, and therefore its configuration is relatively simple. The only parameter which should be set for this module is the standard logging level of status when the document couldn't be classified in any of the categories during the document evaluation:

- `logLevel` – permissible values are `DEBUG`, `INFO`, `WARNING`, `ERROR` and `FATAL`.

The following example shows the standard configuration of this module:

```
<!-- ##### -->
<!-- # TS_CatEvalFields -->
<!-- ##### -->
<wasp:externalLibrary>
  <wasp:path>ts_catevalfield.dll</wasp:path>
  <wasp:debugPath>ts_catevalfieldDbg.dll</wasp:debugPath>
</wasp:externalLibrary>

<wasp:module wasp:class="TS_CatEvalFields" />
<catfld:CatEvalField
  logLevel="DEBUG"
/>
```

## The TS\_CatEvalProfiles Module

The **TS\_CatEvalProfiles** module evaluates documents against predefined categories.

Internally this module uses the functionality of profiler for query evaluation defining the categories. The configuration of this module is therefore the same as the configuration of the **TS\_Profiler** module. Configuration details can be found in the Tovek Server manual; the following example shows only its basic variant:

```
<!-- ##### -->
<!-- #   TS_CatEvalProfiles   -->
<!-- ##### -->
<wasp:externalLibrary>
  <wasp:path>ts_catevalprf.dll</wasp:path>
  <wasp:debugPath>ts_catevalprfDbg.dll</wasp:debugPath>
</wasp:externalLibrary>

<wasp:module wasp:class="TS_CatEvalProfiles"/>
<cateval:options>
  <option name="MapSize" value="10000"/>
  <option name="PersistentStore" value="../data/catprf/prf.dat"/>
  <option name="Style" value="..\tdk\611\common\styles\prf"/>
  <option name="Locale" value="uni/cs"/>
  <option name="Charmap" value="utf-8"/>
  <option name="LoadInBackground" value="false"/>
  <option name="KBase" value="..\data\topicset "/>
</cateval:options>
```

Parameters used in the previous example have the following meaning:

- `MapSize` – initialize map size for storing category queries
- `PersistentStore` – file for storing binary form of category queries, the value should ideally be a little bit larger than the number of all categories with defined queries
- `Style` – folder with styles which manage the evaluation
- `Locale` – profiler locale
- `Charmap` – profiler code page
- `LoadInBackground` – if this parameter is set to `false`, the queries defining the categories will be loaded during the start of the Tovek Server module - failing that it will start another thread on the background which will load the specified definitions.
- `KBase` – makes it possible to connect knowledge base to module for query evaluation.

---

## Modules for Categorisation Results Recording

The module responsible for categorisation of results recording is **TS\_CatStorageODBC**. This module records the results to relational databases connected via ODBC and supports the following databases:

- Microsoft SQL Server
- PostgreSQL
- MySQL

During start this module controls the version of connected database as well as version of ODBC driver, and if one of them is not on the list of supported ones, the module will not start.

This module configuration is as standard saved in the *ts\_catengine.xml* file and contains the following parameters:

- `odbcSource` – name of ODBC source
- `userName` – name of the user under whose account the data in ODBC source is accessed. If the connected database does not contain the needed tables, this module will try to create these tables under the specified account.
- `password` – user password for access to database connected via ODBC entered in encrypted form, generated by `pwencode.exe` program
- `connectionPooling` – [optional] – activates (value `true`) or deactivates (value `false`) connection pooling for ODBC connection. The standard value is `false`.
- `localPoolSize` – [optional] – sets the number of connections to the database which are stored in the module for repeated use instead of releasing them. Permissible values are numbers larger than or equal to 0. The default value is 8.
- `documentsTable` – [optional] – name of database table for storing document keys
- `docSourcesTable` – [optional] – name of database table for storing sources of the documents
- `sqlQuoteChar` – [optional] – character starting swinged constants in SQL commands - the default value is apostrophe
- `sqlPutDataChunkSize` – [optional] – maximum size in bytes of buffer for storing data using `SqlPutData`. If the saved data is longer than the specified limit, this data is recorded in several sequential steps. The standard value is 32768, some ODBC drivers however will not necessarily support this buffer size. In that case the specified parameter need to be modified.
- `sqlGetDataBufferSize` – [optional] – analogous to the previous parameter, there is also a configuration of the buffer size for loading data using `SqlGetData`, which is as standard set to 32768 bytes.

- `userChangeHistory` – [optional] – turns on the recording of history of changes made in manual document categorisation. If this parameter is set to `false` value, only the last change for the document and the category made by any user are recorded in the repository. Failing that (value `true`), all changes ever made by users are recorded .
- `logLevel` – [optional] – sets the level of ODBC functions logging. Permissible values are `error`, `info` and `all`. The standard setting for this parameter is `info` value.

The following example shows the minimal configuration of this module:

```
<!-- ##### -->
<!-- #   TS_CatStorageODBC module           --
>
<!-- ##### -->
<wasp:module wasp:class="TS_CategoryStorageODBC" />
<tscat:odbcstorage
    odbcSource="TSCat_MSSQL"
    userName="cat"
    password="KjuXUpvAwCPRvdqWD8mgag=="
/>
```

## Category Management and Results Evaluation Module

The main task of the **TS\_Categorizer** module is management and evaluation of categorisation results, together with management of categories and their sets. This module makes it possible to:

- withdraw the list of all category sets and categories which they contain
- create and record new categories and new sets in its repository,
- specify in which categories the document was classified,
- change categorisation of document in categories by user or using evaluation modules,
- evaluate result of fulltext searching, i.e., specify number of result documents in each category,
- filter search result according to selected categories,
- specify documents complying with the fulltext query and set filter.

After installation, the configuration of the `TS_Categorizer` module is stored in the `ts_catengine.xml` file and sets the following parameters:

- `storageModuleClass` – defines module for recording categorisation results. The current version contains only one such module, and therefore this parameter is set as standard to the `TS_CategoryStorageODBC` value.

- `catSearchPoolSize` – [optional] – number of not-allocated structures for evaluation of search results which are not released. The default value of this parameter is 16.
- `maxThreads` – [optional] – evaluation of each search result runs in one thread. This parameter in fact determines the maximum number of simultaneous searches which could be served. Any further one will be blocked and will wait for a thread release. This parameter is as standard set to -1, which means unlimited number of threads.
- `maxSpareThreads` – [optional] – each time the search results evaluation finishes, its thread is closed and released or put into the queue of threads which could be used again for another calculation. This parameter restricts the maximum length of this queue. The default setting of its value to 8 makes it possible to keep 8 threads ready for faster processing of incoming requests.
- `resultWaitTimeLimit` – [optional] – if the categorisation is in use, the search result is withdrawn through the categorisation module, which filters it based on the selected categories. In the request for result withdrawal, there is a setting of whether to wait till the results will be available (blocked call). This parameter then defines the maximum waiting time for the result after which the calling of the client application finishes with an error. However, during the result calculation, it does not wait for the processing of the complete list of all documents. At the moment when the specified number of suitable documents is found, their list is returned to the client application. The value of this parameter is entered in milliseconds and the default setting is 5000 (5 seconds).
- `resultChunkSize` – [optional] – defines the number of documents which are withdrawn at once during the calculation. The default value is 500.

```

<!-- ##### -->
<!-- #   TS_Categorizer module           --
>
<!-- ##### -->
<wasp:module wasp:class="TS_Categorizer" />
<tscat:categorizer
    storageModuleClass="TS_CategoryStorageODBC"
    catSearchPoolSize="16"
    maxThreads="-1"
    maxSpareThreads="8"
    resultWaitTimeLimit="5000"
    resultChunkSize="500"
/>

```



## **Appendix: Syntax of Perl Regular Expressions**

---

The definition of regular expressions syntax can be found on the following page:

[http://www.boost.org/doc/libs/1\\_39\\_0/libs/regex/doc/html/boost\\_regex/syntax/perl\\_syntax.html](http://www.boost.org/doc/libs/1_39_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html)