Toad™ Data Modeler

internal scripting language syntax

document version: 0.7

Created: 07/25/2006

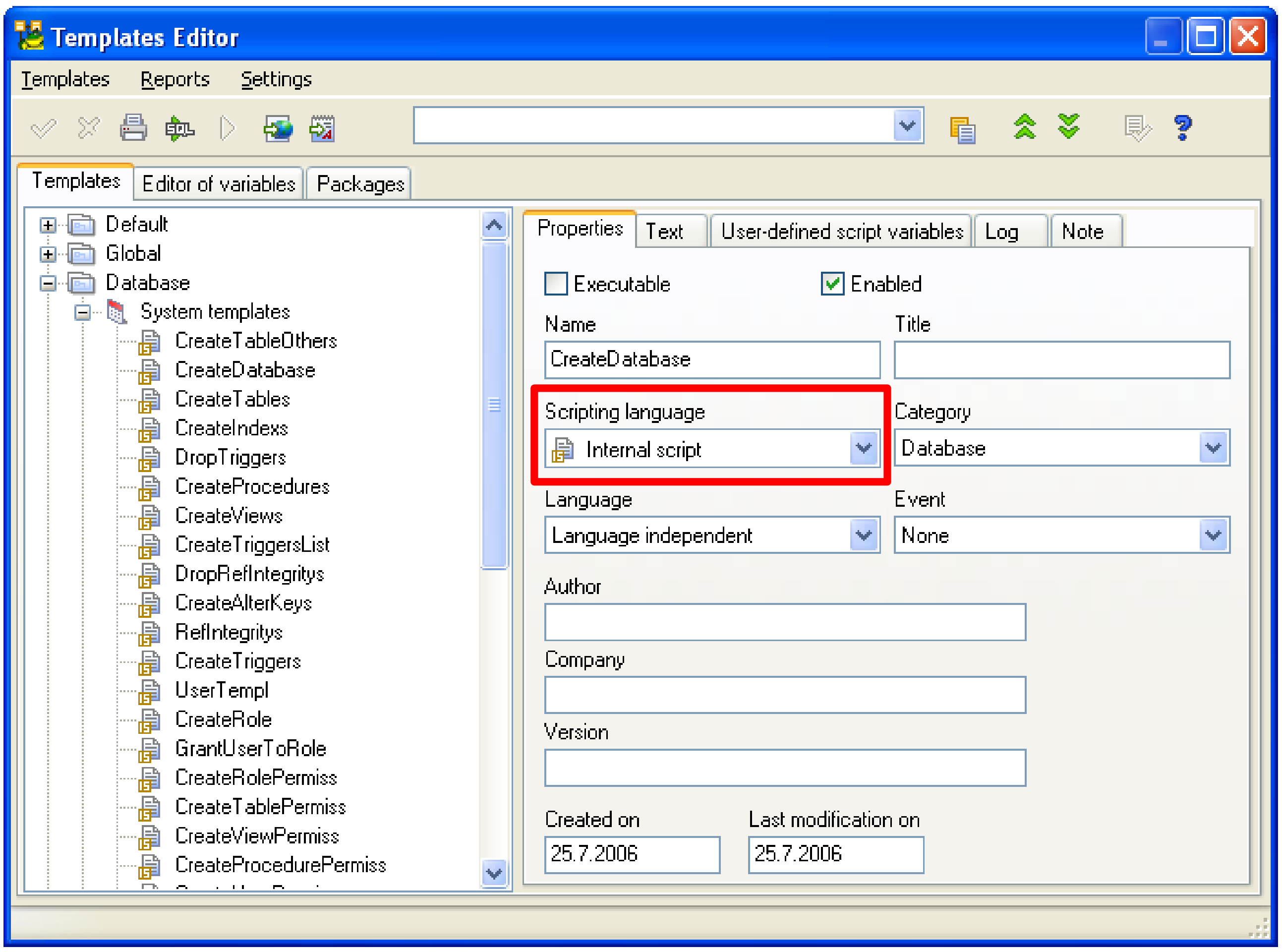
1. Templates editor and scripting

Would you like to extend the power of Toad™ Data Modeler? If yes, please try to use the Templates editor and take advantage of the Internal Scripting language in combination with JScript, VBScript, Perl or other imported language engines. Create new templates and new add-ins! Affect the final SQL/DDL script, customize HTML or RTF reports, create new support for currently unsupported database, make your own, useful and powerful templates and more. This document might help you to understand the internal scripting language syntax.

2. Template types

Toad™ Data Modeler supports several types of templates. Appropriate template type must be defined in the „Scripting language“ drop down menu of the „Properties“ tab. You can choose one of the following types:

- Internal script
- Jscript
- VBScript



Executable
[✓ Enabled]

Name	CreateDatabase
Title	
Scripting language	Internal script
Category	Database
Language	Language independent
Event	None
Author	
Company	
Version	
Created on	25.7.2006
Last modification on	25.7.2006

2.1 Internal script

The syntax of Internal scripts were created by Toad™ Data Modeler makers a long time ago. The internal scripting includes conditions, macros, commands and variables - and due to the evolution of Toad™ Data Modeler also few obsolete items. IScript has a great significance primarily for the SQL script generation.

- **Conditions** are defined by starting bracket { sign, condition name and closed by ending bracket } sign. For example:{lBeforeScript}
- **Macros** represent parts of scripts. You have to write a **macro** command to call a macro. For example: `macro(CreateAtrib)`. Inside a macro, every expression which is not written between quotation marks will be interpreted as a command or variable.
- **Commands** are represented by an ampresand @ sign followed by the name of the command. Any word which starts with the @ sign will be represented as a command.
- **Variables** are represented by starting **%** sign, variable name and ending **%** sign. For example: `%Version%`

See the starting part of the 'CreateDatabase' template.

/* Created %CreatedDate% Modified %ModifiedDate% Project %ProjectName% Model %ModelName% Company %Company% Author %AuthorName% Version %Version% Database %DatabaseType% */	The output file will include whole the text. All variables will contain previously defined values. This example shows values from the Toad™ Data Modeler Stamp.
`{lBeforeScript}`	The software will evaluate the value of lBeforeScript variable and return **true** or **false**. If the value is set to true, next command will be executed. Otherwise the software goes to another condition. Summary: Returns True or False
`%BeforeScript%`	The software will use the exact **value** of BeforeScript variable, which contains a text defined in the section BeforeScript. (See: Model -> Text objects -> Before script) Summary: Returns a value
`{lDropTriggerGener}`	The program evaluates the lDropTriggerGener variable. In case it returns false, the software goes to another condition automatically. (See:Model -> Script generation) Summary: Returns True or False
`@ShowMessage("Drop triggers")`	The software executes the ShowMessage command. (The text Drop triggers will be displayed in appropriate log window.) Summary: Executes a command (shows text)
`@Template(DropTriggers)`	The software executes the Template command and goes to the template DropTriggers. Summary: Executes a command (calls a template)

Internal script templates can't return values.

2.2 JScript

Toad™ Data Modeler supports MS Scripting engine and therefore you can take advantage of JScript. You can write templates in JScript, just remember to define appropriate template type in the Scripting language drop down menu of the Properties tab. For more information about JScript please see the official documentation.

2.3 VBScript

You can write your templates in VBScript also. Of course, you will have to define the template type in the Scripting language drop down menu of the Properties tab. For more information about VBScript please see the official documentation.

2.4 Calling templates

A template can be called in several different ways. The right method depends on:
– type of the template from which the other template is called (initial template)
– type of the called template (called template)

Initial template	***Command***	***Called template***
internal script	Template(*TemplName*)	internal script
internal script	ScriptProc(*TemplName, FunctName, params,...)*	MS Engine script
MS Engine script	Not available	internal script
MS Engine script	Scripting.*TemplName.FunctName(params,...)*	MS Engine script

TemplName – name of called template
FunctName – name of called function in TemplName

Template: the template command can be used only inside the Internal script to call another Internal Script template.

2.5 Other script types

- JScript (implemented by OS)
- VBScript (implemented by OS)
- Perl
- Python
- Tcl Script
- Haskell Script
- Ruby Script
- LuaScript and other imported Language Engines

You can add new script types into Toad™ Data Modeler by modifying the file „ScriptList.ini“ in your Bin directory. Open the file and follow the written instructions.

3. Internal scripts items

3.1 Conditions – IFF and IF

```
IFF(<cond>,<param1>,<param2>)
```

if the condition is true, the first parameter is evaluated and appropriate value is returned, if the condition is false, the second parameter is evaluated and appropriate value is returned. Only parameter corresponding to appropriate condition is evaluated.

```
IF(<cond>,<param1>,<param2>)
```

both parameters are always evaluated. If the condition is true, the value of the first parameter is returned, else the value of the second parameter is returned.

3.2 Macros - Using macros inside internal scripts

Specifics: If the Macro command is used, whole the text is interpreted as a macro.

The following sample macro can be used for writing the „Create table“ command.

```
"Create table %TableName% "+
ForCol("(", "",cr+tb+macro(CreateAtrib),",",") ")+
TableStorage+term+cr+
ShowMessage("Table %TableName%")
```

This definition is also correct:

```
"Create table "+TableName+" "+
ForCol("(", "","%cr%%tb%@macro(CreateAtrib)",",",") ")+
"%TableStorage%%term%%cr%"+
ShowMessage("Table "+TableName)
```

You can write macros using both methods. The choice is up to you.

3.3 Commands

```
IncludeFromFile(<file_name>)
Inserts a file

Macro(<template_name>)
Calls macro <template_name>.

Script(<template_name>)
Executes the Main function from the template <template_name>.

ScriptProc(<template_name>,<function_name>)
Executes the <function_name> function from the template <template_name>.


ScriptProc(<template_name>,<function_name>,<param1>,<param2>,....)
Executes the <function_name> function from the template <template_name> with
parameters.

SetFlag(<flag_number>,<logic_value>)
Defines a logic variable.
Mathematical definition: 1 <= <flag_number>  <= 3
```

You can refer to the defined logic variables via the following variables: **Flag1, Flag2, Flag3.**

```
Setflag(1,true)+
if(flag1,"true","false")
...returns true
```

ShowMessage("text")
The text will be displayed in appropriate Log window.

Template(<template_name>)
Calls template <template_name>.

TemplateFromFile(<template_name>,<file_name>)
Calls template from defined file.

TemplateToFile(<template_name>,<file_name>)
Calls template <template_name> and generates the result into a file.

Variable("text")
Converts the text to a variable.

3.3.1 For commands hierarchy:*

```
Model
|- ForTable("","","","","") or ForTableR("","","","","")
| |
| |- ForAlterKey("","","","","")
| |  |- ForAlterKeyCol("","","","","")
| |
| |- ForChild("","","","","") and  ForParent("","","","","")
| |  |- ForRelPk("","","","","")
| |
| |- ForCol("","","","","")
| |- ForPFkCol("","","","","")
| |- ForPkCol("","","","","")
| |
| |- ForIndex("","","","","")
| |  |- ForIndexCol("","","","","")
| |
| |- ForTableTrigger("","","","","")
|
|- ForDict("","","","","")
|- ForProcedure("","","","","") or ForProcedureR("","","","","")
|- ForRole("","","","","")
|- ForUser("","","","","")
|- ForTextObject(type,"","","","","")
|- ForTrigger("","","","","") or ForTriggerR("","","","","")
|- ForView("","","","","") or ForViewR("","","","","")
```

3.3.2 For commands*

ForTable – the cycle goes through all tables of a model, sets appropriate table active and evaluates appropriate parameters. Properties of currently active table are accessible.

Example:
```
ForTable( "List of tables:"+cr+tb, tb, "%TableName%",
",""+cr, cr+"End of the list"+cr)
```

This example will create the following output.

```
Generated list of tables:
        Entity1,
        Entity2,
        Entity3
End of the list
```

As you can see, the first parameter writes the text „List of tables“ and adds a carriage return and a tabulator. The second parameter adds a tabulator before each entity name except the first one entity (the tabulator was already assigned to the first entity via the first parameter). The third parameter returns the Table Name. The fourth parameter adds a comma and a carriage return after every table, but the last one and the fifth parameter adds a carriage return and writes the text „End of the list“.

Parameter	***Parameter in example***	***Evaluated items***
First	"List of tables:"+cr+tb	First item of all iterated items
Second	tb	All iterated items except the first one
Third	`"%TableName%"`	All iterated items
Fourth	","+cr	All iterated items except the last one
Fifth	cr+"End of the list"+cr	Last item of all iterated items

The definition of parameters is valid for all For* commands.

ForTableR("","","","","")
this command is similar to the ForTable command, but the tables are evaluated in reversed order. Recursive commands should be used in case you need to erase objects.

3.3.3 For commands - selected tables:*

ForAlterKey("","","","","")
goes through all alternative keys in the active table.

ForChild("","","","","")
goes through all relations defined for Child tables. This command sets active relation and appropriate values.

ForCol("","","","","")
goes through all columns in the active table. This command sets values for columns.

ForPFkCol("","","","","")
goes through all columns in the active table, which are a part of the primary key or a part of any foreign key. This command sets values for columns.
ForPkCol("","","","","")
goes through all columns in the active table, which are a part of the primary key. This command sets values for columns.

ForIndex("","","","","")
goes through all indexes in the active table. This command sets values for indexes.

ForParent("","","","","")
goes through all relations defined for Paren tables. This command sets active relation and appropriate values.

ForTableTrigger("","","","","")
goes through all user defined triggers which belong to the active table. This command defines active trigger and sets appropriate values.

3.3.4 For commands - indexes:*

ForIndexCol("","","","","")
goes through all columns in the active index.

3.3.5 For commands – alternate key:*

ForAlterKeyCol("","","","","")
goes through all columns in the active alternate key.

3.3.6 For commands – relation:*

ForRelPk("","","","","")
goes through all columns, which are a part of the primary key of the active relation. This command sets values for columns.

3.3.7 For commands - model:*

ForDict("","","","","")
goes through all items in the Dictionary (user defined data types).

ForProcedure("","","","","")
goes through all stored procedures. This command defines active procedure and appropriate values.

ForProcedureR("","","","","")
goes through all stored procedures, but in reversed order.

ForRole("","","","","")
goes through all user roles, defines active role.

ForUser("","","","","")
goes through all users, defines active user.

ForTextObject(type,““,"","","","")
goes through all text objects. The „type“ is an identification number of the text object)
ForTrigger("","","","","")
goes through all user defined triggers. This command defines active trigger and appropriate values.

ForTriggerR("","","","","")
goes through all user defined triggers, but in reversed order.

ForView("","","","","")
goes through all views. This command defines active view.

ForViewR("","","","","")
goes through all views, but in reversed order.

3.4 Variables

3.4.1 Entity

TableName
EntName
ConstraintPkName
ExistPk
EntityOthers
IIndexExist
EntDescription
TableStorage

3.4.2 Attribute

AttrName
AttrDescription
AttrIsDict(arg1,arg2)
ColName
DictName
Type
TypSQL
UserDataType
Length
Decimal
NotNULL
CheckValue - synonym **Check**
EvalCheck
UniqueAtr
DefExist
DefValue - synonym **Def**
Def2Exist
Def2Value - synonym **Def2**
ConstraintExist
ConstraintCheck - synonym **Constraint**
ConstraintDefault
ConstraintAtrUnique
CheckExist
DefaultExist
DefaultValue - synonym **Default**

3.4.3 Index

IndexName
Unique
Clustered
Desc
IndexColDesc
lIndexExpr
IndexExpr
IndexFilter
IndexFilterExist
IndexStorage

3.4.4 Alternate key

AlterKeyName
AlterKeyConstraintName
AlterKeyKeys
AlterKeyColDesc

3.4.5 Dictionary types

DictDescription
DefDict
Def2Dict

3.4.6 Permission

PermissTableUser("")
Is used to Set up permissions for actual entity and user.
@ForTable("","",ForUser("","",PermissTableUser(Macro(CreateTablePermiss)),"",""),"","")

PermissTableRole("")
PermissProcedureRole("")
PermissProcedureUser("")
PermissViewRole("")
PermissViewUser("")
UserRoleUser("")
lRole
lUser
RoleName
UserName
UserOrRoleName
lPermissSelect
lPermissInsert
lPermissUpdate
lPermissDelete
lPermissExec
lPermissDRI
lUserRoleUser

3.4.7 Relationship

PkChildName
PkChildDefaultValue
PkParentName
lRelParNone
Returns True if the referential integrity rules for Parent Update and Parent Delete are set to None. The referential integrity rules are defined in the Referential integrity tab of the Edit relationship dialog.

lRelParUpdDekl
Returns True if the generating of RI for (at least one) Parent-Update is set to Declarative. The settings are defined in the "How to generate" tab of the "Script Generating" dialog.

lRelParDelDekl
IsParent
IsChild
FRelName
RelName
RelDescription
ParUpdNone
Returns True if the referential integrity rule for Parent-Update is set to None. The referential integrity rules are defined in the Referential integrity tab of the Edit relationship dialog.

ParUpdRestrict
ParUpdCascade
ParUpdSetNull
ParUpdSetDefault
ParDelNone
ParDelRestrict
ParDelCascade
ParDelSetNull
ParDelSetDefault
ChildInsNone
ChildInsRestrict
ChildInsSetNull
ChildInsSetDefault
ChildUpdNone
ChildUpdRestrict
ChildUpdSetNull
ChildUpdSetDefault
ParentTableName
ChildTableName
ParentKeys
ChildKeys
lParUpdRestDekl
Returns True if the generating of RI for Parent-Update-Restrict is set to Declarative. The settings are defined in the "How to generate" tab of the "Script Generating" dialog.

lParUpdCascDekl
lParUpdSetNullDekl
lParUpdSetDefaultDekl
lParDelRestDekl

lParDelCascDekl
lParDelSetNullDekl
lParDelSetDefaultDekl
lChildInsRestDekl
lChildUpdRestDekl
lRelParUpdRestDekl
Returns True if the referential integrity rule for Parent-Update is set to Restrict and *lParUpdRestDekl* is true. The referential integrity rules are defined in the Referential integrity tab of the Edit relationship dialog. *Is used in cycle ForChild and ForParent.*

lRelParUpdCascDekl
lRelParUpdSetNullDekl
lRelParUpdSetDefaultDekl
lRelParDelRestDekl
lRelParDelCascDekl
lRelParDelSetNullDekl
lRelParDelSetDefaultDekl
lRelChildInsRestDekl
lRelChildUpdRestDekl
lRelAnyDekl
lRelAnyDekl := lRelParUpdRestDekl or lRelParUpdCascDekl or
lRelParUpdSetnullDekl or lRelParUpdSetDefaultDekl or
lRelParDelRestDekl or
lRelParDelCascDekl or lRelParDelSetnullDekl or
lRelParDelSetDefaultDekl or
lRelChildInsRestDekl or lRelChildUpdRestDekl

lRelUpdDekl
lRelUpdDekl := lRelParUpdRestDekl or lRelParUpdCascDekl or
lRelParUpdSetnullDekl or lRelParUpdSetDefaultDekl

lRelDelDekl
lRelDelDekl := lRelParDelRestDekl or lRelParDelCascDekl or
lRelParDelSetnullDekl or lRelParDelSetDefaultDekl

lUpdDekl
lUpdDekl := lParUpdRestDekl or lParUpdCascDekl or lParUpdSetnullDekl or
lParUpdSetDefaultDekl or lChildUpdRestDekl;

lDelDekl
lDelDekl := lParDelRestDekl or lParDelCascDekl or lParDelSetnullDekl or lParDelSetDefaultDekl;

lRelUpdTrig
lRelDelTrig
lRelInsTrig
lUpdTrig
lDelTrig
lInsTrig
lEntParUpdRest
lEntParUpdCasc
lEntParUpdSetNull
lEntParUpdSetDefault

lEntParDelRest
lEntParDelCasc
lEntParDelSetNull
lEntParDelSetDefault
lEntChildInsRest
lEntChildUpdRest
lEntParUpd
lEntParDel
lEntChildUpd
lEntChildIns
lEntParUpdTrig
lEntParDelTrig
lEntChildUpdTrig
lEntChildInsTrig
lRelChildUpdDekl
lRelChildInsDekl

3.4.8 Generating script checkboxes

lEntityOthers
lDropTableGener
lDropIndexGener
lDropTriggerGener
lDropTriggersListGener
lDropProceduresGener
lDropViewsGener
lTriggersGener
lTriggersUserGener
lBeforeScript
lAfterScript
lPkAsConstraint
lFkAsConstraint
lRefIntegGener
lIndexGener
lDropDomainGener
lDomainGener
lTableGener
lAlterKeysGener
lProceduresGener
lViewsGener
lPkGener
lRoleGener
lRolePermissGener
lUserPermissGener
lUserToRoleGener

3.4.9 Model

BefreScript
AfterScript
CreatedDate
ModifiedDate

ProjectName
ModelName
AuthorName
Version
Company
DatabaseType

3.4.10 Other

TriggerName
TextObject
TextObjectName
ProcedureName
ViewName
Date
Time
NOW

4. COM interface

If any object is set as active, the COM interface is being used in appropriate variables.

ITABLE or **TABLE**
Table

IMODEL or **MODEL**
Model

ICOLUMN or **COLUMN**
Column

IDICTTYPE or **DICTTYPE**
Dictionary type

IALTERKEY or **ALTERKEY**
Alternate key
IINDEX or **INDEX**
Index

IINDEXCOL or **INDEXCOL**
Column which refers to a column in the index.

IRELATION or **RELATION**
Relation.

ITRIGGER or **TRIGGER**
Trigger

IPROCEDURE or **PROCEDURE**
Interface to procedures

IVIEW or **VIEW**
View

5. Special characters

If you do not want to interpret the below mentioned special characters, you have to use special definitions. See:

C1 - @
C2 - %
C3 - {
C4 - }
C5 - "
C6 - '