erwin by Quest

erwin Data Modeler

Template Language and Macro Reference

Release 12.5

Legal Notices

Contact erwin

Understanding your Support

Review support maintenance programs and offerings.

Registering for Support

Access the erwin support site and register for product support.

Accessing Technical Support

For your convenience, erwin provides easy access to "One Stop" support for all editions of erwin Data Modeler, and includes the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- erwin Support policies and guidelines
- Other helpful resources appropriate for your product

For information about other erwin products, visit http://erwin.com/products.

Provide Feedback

If you have comments or questions, or feedback about erwin product documentation, you can send a message to techpubs@erwin.com.

News and Events

Visit News and Events to get up-to-date news, announcements, and events. View video demos and read up on customer success stories and articles by industry experts.

Contents

Template Language and Macro Reference

This section contains the following topics

Introduction

Template Language (TLX) is the template or macro language publicly exposed with erwin Data Modeler. It has been present in the product since erwin Data Modeler r7.0.

TLX is employed in a number of places in the product using expansion. This includes templates used for forward engineering, metamodel dumps, and Model Explorer and Complete Compare object naming.

Metadata Names

A number of macros make use of class names   the names of metadata objects. The section below describes a few notes on the use of class names; the full listing of all object classes and property classes is located in the document erwin Metamodel Reference bookshelf.

- Class names are always case-sensitive.
- Some object types represent an object in both the logical and the physical model. This is legacy behavior when erwin Data Modeler had only logical/physical models. For these types of objects, display names seen in erwin Data Modeler's user interface change based upon whether you are looking at the logical or physical side. However, class names do not change in a similar manner. Therefore, an entity in the logical model and a table in the physical model both have a class name of Entity.

  The following list describes the class names that are included in this category:

  - Attribute
  - Default
  - Domain
  - Entity

  - Key_Group
  - Relationship
  - Validation_Rule

- Local User-Defined Properties (UDPs) have three part class names. All UDPs available in the current release are classified as Local UDPs. erwin Data Modeler allows a given UDP name to be used on a variety of object types, and in both the logical and physical model, with differing data types, defaults, descriptions, and so on. For example, the UDP named "Color" might be a color name on the logical side of an entity, a red-green-blue integer on the physical side, and a Boolean value indicating if a view should be colored on reports. To enable erwin Data Modeler to understand exactly what is being referenced, the actual class name is comprised of: <object type>.<model side>.<name>.

For example, the three "Color" values mentioned would be: Entity.Logical.Color, Entity.Physical.Color and View.Physical.Color. Future releases of erwin Data Modeler may also support Global User-Defined Properties where the property has a single data type, default, and so on, no matter where it occurs. Those UDPs will follow the normal naming convention of built-in properties, having a single-part name.

TLX Syntax

The following sections provide a brief review of the TLX syntax.

This section contains the following topics

Literals

Literal text is enclosed in double quotation marks. This includes literals intended for the final expanded output and literals used as parameters to macros. For example,

```
"This is a text literal", Property("Name").
```

The valid characters for a literal include the printing characters (those higher than 32 decimal). Escape sequences are used for some characters. The escape sequences that are currently supported are as follows:

\n

Carriage return

\t

Horizontal tab character

\

Back slash

\"

Double quote

One advantage to explicitly-delimited literals is that the formatting of the source and the output can vary. Source text can be formatted exactly as needed because the output text is formatted based upon tabs, spaces, and carriage returns embedded in literals.

Macros

Any text not enclosed in double quote marks, and that does not start with an ampersand, is a macro name. For example,

```
MyMacro
```

If parameters are being passed to a macro, they will follow the macro name enclosed in parentheses. If more than one parameter is supplied, they are separated by commas. All parameters are string values.

```
MyMacro("Parameter1", "Parameter2")
```

There are two general classes of macros: substitution and iteration macros.

Substitution Macros

Substitution macros evaluate to a string value. A substitution macro takes the form described in the previous section: a macro name and possibly a parameter list. For example,

```
Property("Name").
```

The resulting string may be an empty value if the actual operations performed by the macro are the purpose of the macro. For example, a macro that writes data out to a file might evaluate to an empty string.

Iteration Macros

Iteration macros (usually) loop through objects in the model. Iteration macros do not evaluate to string values themselves.

An iteration macro may or may not have parameters, just like a substitution macro, but they are followed by curly braces denoting an iteration block.

```
ForEachOwnee("Attribute")
{
Property("Name")
}
```

The source text inside the iteration block is evaluated once for each time that the iterator increments.

Some iterator macros do not actually traverse objects. For example, they might traverse values on the object. Generally, this type of iterator macro will have special accessor macros that allow these values to be retrieved.

The form of this type of iterator macro is identical. For example:

```
ForEachProperty
{
ForEachProperty.Value
}
```

Macro Return Codes

When a macro is evaluated, it returns a success code to the TLX template processor. If the macro succeeds, it returns 'true'; if it fails, it returns 'false'.

For a substitution macro, the return code and the string value it evaluates into are not the same thing   the macro may evaluate to an empty string and return 'true' to indicate a successful evaluation.

Iterator macros are invoked once before the iteration begins, and then once on each iteration. If the initial invocation fails, no iterations are performed. The looping continues until the macro returns 'false' from a request to increment.

Keywords

Unquoted text that starts with an ampersand indicates a keyword. Currently, TLX supports four keywords for conditional control: @if, @ifnot, @elseif and @else. These are not case-sensitive.

These conditional keywords are followed by a block enclosed with curly braces, much like an iteration macro. The code within the block is evaluated only if the condition is true.

```
@if ( Equal( Property("Name"), "Delaware" ) )
{
"The first state."
}
@elseif( Equal( Property("Name"), "Hawaii" ) )
{
"The last state."
}
@else
{
"A state somewhere in the middle."
}
```

Conditional Blocks

The @if/@elseif keywords provide conditional evaluation. An alternate mechanism for conditional behavior is the conditional block. Source text enclosed in square brackets is in a conditional block. A conditional block is emitted only if all macros within it succeed. The block stops evaluating when it encounters a failure.

The choice of whether to use conditional blocks or @if/@elseif statements is up to the user- conditional blocks can provide a format that is easier to read than an embedded @if block in some situations. For example:

```
"CREATE TABLE " [Property("DB Owner" "."] Property("Physical
Name")
```

Conditional blocks have no effect on enclosing blocks.

Propagating Blocks

Source text enclosed in angle brackets is in a propagating block. Propagating blocks essentially have a return code just like a macro does. They succeed if the text within them after evaluation is not empty; they fail otherwise.

In the following example, if we can find a first name or last name on the object, text will emit. If we can find neither, nothing will emit because the propagating block will cause a failure in the outer conditional block. If the propagating block was not used, the literal "My name is" would be emitted even if no name was present.

```
[ "My name is"
< [ " " Property("Name")] [ " " Property("Last Name") ] >
]
```

Comments

Text that is enclosed in C-style comments is ignored by the TLX parser and does not become part of the final output.

```
/* This is a comment */
```

TLX Expansion

This section contains the following topics

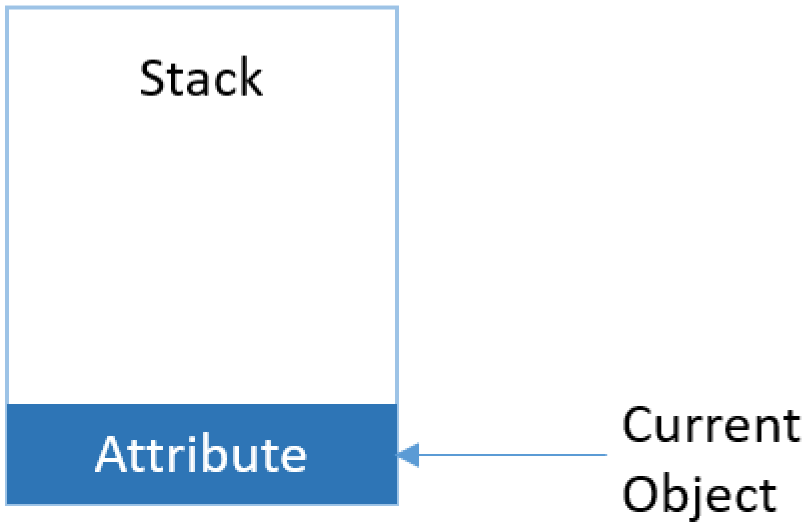
Phantom Objects

When macros are evaluating, they are generally operating on objects in the model. For example, the Property macro reads the value from a property on the "current" object. This current object is referred to as the context object. Objects become the current object by placing them on the context stack. This is an ordered list that operates in a Last In/First Out manner   objects are added to the list by pushing and removed from the list by popping them.

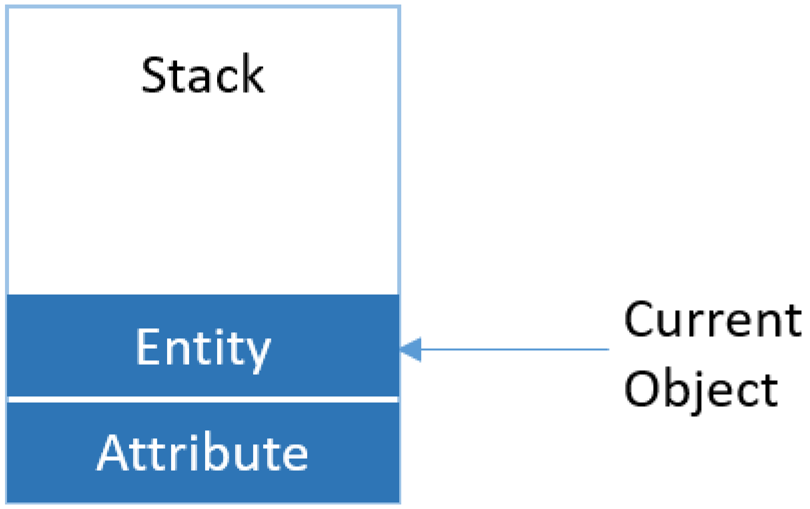
If you are familiar with erwin Data Modeler's Macro Language, this concept has been present since the earliest days. For example, in erwin Data Modeler r7.2 you could write %ForEachEntity(E_1). This would establish 'E_1' as the current object for subsequent macro calls. Additionally, many macros such as %JoinFKPK or %RelRI would only operate inside certain iterators or editors. This was because they required a certain context stack to be in existence internally.

This concept is simply being exposed more fully now to provide you more control.

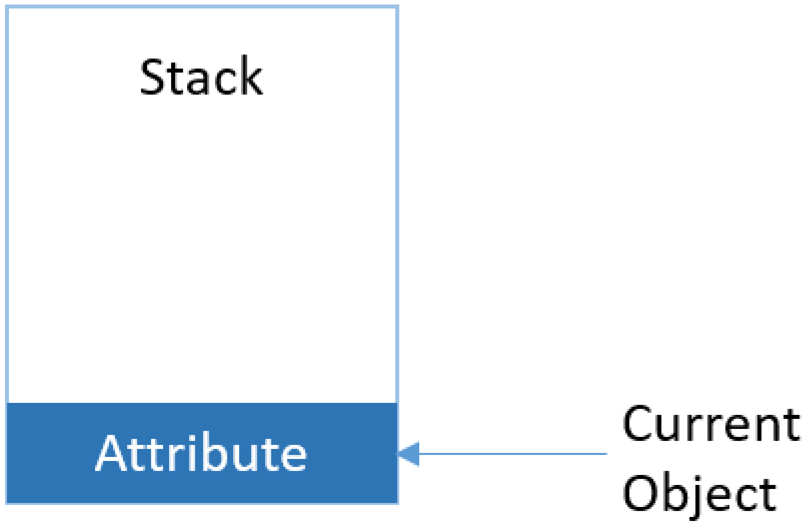
As an illustration, suppose the TLX parser is invoked with an Attribute as the starting object. The context stack would have one entry:



Now a macro is used that pushes the Attribute object's owning Entity onto the stack. The stack will now have two entries, with the Entity being the current object.



Now another macro is used that pops the last object from the stack. The stack now reverts to having only the Attribute in it and it is the current object.



Many macros operate upon the current context object. The Property macro is an example of this.

Other macros are capable of operating on objects deeper down the context stack. The PropertyFrom macro is an example of this type.

There are also a variety of macros that push objects onto the context stack and remove them. Macros with the words "Push" and "Pop" in their names are of this variety.

The bottommost (first) object on the stack is known as the anchor object and cannot be popped from the stack. This ensures that macros always have some object as the context. For example, assume that an Entity is the anchor object:

```
PushOwner /* pushes the model onto the stack */

Pop /* pops the model from the stack */

Pop /* fails - entity is the anchor object */
```

An object can appear in the stack more than once. If the object happens to be the anchor object, only its earliest appearance cannot be popped   other appearances can be popped at will. For example, assume that an Entity is the anchor object:

```
PushOwner /* pushes the model onto the stack */

Repush( "1" ) /* pushes entity a second time */

Pop /* pops second occurrence of the entity */

Pop /* pops the model from the stack */

Pop /* fails - entity is the anchor object */
```

Iteration macros that operate on objects implicitly push the current iteration object onto the stack. This frees you from having to write explicit push and pop macro calls. For example, assume that an Entity is the anchor object:

```
Property("Name") /* <- reads the entity name */

ForEachOwnee("Attribute")

{

Property("Name") /* <- reads an attribute's name */

}
```

```
Property("Name") /* <- back to reading the entity name */
```

Phantom Objects

In certain processes in erwin Data Modeler, objects that are not actually in the model are simulated so that they can be used by macros. The most notable example of this situation is when evaluating a TLX template for generating an Alter Script. These objects are referred to as phantom objects. Generally, they can be treated exactly like a regular object. However, certain macros behave differently or do not function when the context object is a phantom object. These situations are noted in the macros' descriptions.

Macro Overview

Each macro entry will contain the following sections describing the behavior of the macro and its usage.

This section contains the following topics

Name

Macro names are always case-sensitive.

The following are some tips on macro names that make it easier to understand their purpose at a glance.

- Macros containing the words "Push" and "Pop" in their names alter the contents of the context stack, such as *PushOwner*.
- Macros with names ending in "From" will, if necessary, reach beyond the first object in the context stack to find an object of the desired type, such as *PropertyFrom*.
- Macros with names ending in "Through" will traverse a reference property on the current context object to find the necessary context object, such as *PropertyThrough*.
- Macros starting with a DBMS name followed by an underscore are designed to work only if the target server matches the DBMS name, such as *Access_Datatype*.
- Macros with a period in the middle of their name will operate only within the context of a specific iterator, such as *ForEachProperty.IsInherited*.
- Macros with a double colon in their name will operate only in certain processes in erwin Data Modeler, such as *FE::Bucket*.

Definition

This section will provide a description of the macro's behavior, plus any particular notes on using it.

Prototype Specification

The prototype specification of each macro will show the full syntax available. The first term is always the macro's name.

```
MacroName
```

Terms in italics represent parameters to the macro. These parameters are described in the table following the prototype signature.

Parameters are always enclosed in parentheses. The parentheses can be omitted if parameters are optional and none are being supplied. Parameters are always separated by commas. Unless otherwise specified, parameters to macros, other than type names, are not case-sensitive.

```
MacroName( Parameter1, Parameter2 )
```

Parameters enclosed in square brackets are optional.

```
MacroName( [OptionalParameter1 [, OptionalParameter2]] )
```

If a macro can take a variable length list of parameters, it will be indicated by an ellipsis.

```
MacroName( Value1[, Value2 [, …]] )
```

Iterator macros are indicated by a trailing set of curly braces.

```
MacroName( Parameter1 ){}
```

Result Specification

This subsection will list the conditions under which the macro will fail.

As the software evolves, some macros may become deprecated. The deprecation specification indicates the level of deprecation at the time of documentation. The following describes the possible values and their meaning:

Active

The macro is not deprecated in any way.

Discouraged

The macro has been replaced by an alternate macro. You may switch to the new syntax when editing new code in order to take advantage of new capabilities or faster performance.

Macros in this status are fully supported and there is no intention to remove them from the product. However, they may not run as efficiently as other approaches and/or may not have the same features available.

Deprecated

The macro is deprecated and support for it will eventually be removed from the product.

Removed

The macro is no longer supported and will not function in the software.

Breaking Changes Specification

This section will describe ways in which the macro has changed from previous versions when that change would break existing code. Generally, breaking changes are adjusted by the post-load processors when a model is upgraded to a new version of the software. The description of the breaking change is provided so that you can adjust any new code you write.

Categories

This is the list of categories to which this macro is assigned by erwin Data Modeler. You can add the macro to more categories, if you choose, by using the Macro Categories editor.

Sample

An example of using the macro will be provided in this subsection.

Standard Macros

This set comprises the bulk of the new macros in erwin Data Modeler. They are usable in all, or almost all, processes in the product.

This section contains the following topics

Standard Macros

Standard Macros

Access_DatabaseName

Description

This macro evaluates to the database name for an Access target server.

Prototype

```
Access_DatabaseName
```

Result

This macro will fail in the following circumstances:

- The target database is not Access.

Deprecation Level

Active

Breaking Changes

None

Categories

- DBMS Specific Macros

Sample

Template

```
"Set ERwinDatabase = ERwinWorkspace.OpenDatabase("
Access_DatabaseName ")"
```

Result

```
Set ERwinDatabase = ERwinWorkspace.OpenDatabase("sERwinDatabase")
```

Access_Datatype

Description

This macro converts the erwin Data Modeler data type to a constant usable by an Access database.

Prototype

```
Access_Datatype
```

Result

This macro will fail in the following circumstances:

- The target database is not Access.

Deprecation Level

Active

Breaking Changes

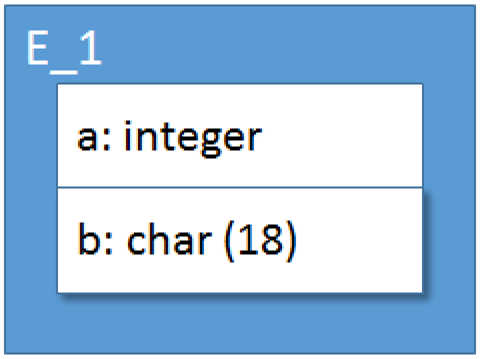
None

Categories

- DBMS Specific Macros

Sample

Assume the current context is the Attribute object E_1.a in the following illustration:



Template

```
"Set ERwinField = ERwinTableDef.CreateField("
"\"" Property("Physical_Name") "\" , " Access_Datatype ")"
```

Result

```
Set ERwinField = ERwinTableDef.CreateField("a", DB_INTEGER)
```

AllowAlterDatatype

Description

This macro determines if a change in an attribute's data type can be handled via an alter script.

Prototype

```
AllowAlterDatatype
```

Result

This macro will fail in the following circumstances:

- The database does not support alter for the data type change.

Deprecation Level

Active

Breaking Changes

None

Categories

Alter Macros

Sample

Assume the current context is an attribute whose datatype has been changed from smallint to integer.

Template

```
[ AllowAlterDatatype
"This can be altered"
]
```

Result

```
This can be altered
```

AreAllOwneesCreated

Description

This macro determines if all objects of the specified type that are owned by the current context object have been created in the current session.

Prototype

```
AreAllOwneesCreated( OwneeType [, Option1 [, OptionN [, …]]] )
```

Parameter	Status	Description
OwneeType	Req	The type of owned object to test.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

Option	Meaning
"reverse"	If this option is present, the macro will fail if all ownees of the specified type are created in the current session.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- There exists an object of the specified type owned by the context object that was not created in the current session.
- All ownees of the specified type were created in the current session and the re-

verse option was present.

- No objects of the specified type are owned by the context object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Assume the current context is a key group that had all members deleted.

Template

```
[
/* This makes sure at least one was present before */
AreAllOwneesCreated("Key Group Member", "reverse")
/* This makes sure none are left */
AreAllOwneesDeleted("Key Group Member")
"Emit a DROP statement"
```

Result

```
Emit a DROP statement
```

AreAllOwneesDeleted

Description

This macro determines if all objects of the specified type that are owned by the current model have been deleted in the current session.

Prototype

```
AreAllOwneesDeleted( OwneeType [, Option1 [, OptionN [,…]]] )
```

Parameter	Status	Description
OwneeType	Req	The type of owned object to test.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"reverse"

If this option is present, the macro will fail if all ownees of the specified type are created in the current session.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- There exists an object of the specified type owned by the context object.
- At least one object of the specified type owned by the context object does not appear in the list of objects deleted in the current session.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Assume the current context is a key group that had all members deleted.

Template

```
[
/* This makes sure at least one was present before */
AreAllOwneesCreated("Key Group Member", "reverse")
/* This makes sure none are left */
AreAllOwneesDeleted("Key Group Member")
"Emit a DROP statement"
]
```

Result

```
Emit a DROP statement
```

AreStringsEqual

Description

This determines if one string is equal to another.

Prototype

```
AreStringsEqual( LeftString, RightString [, Option] )
```

Parameter	Status	Description
LeftString	Req	The left string in the test.
RightString	Req	The right string in the test.
Option	Opt	By default the comparison is case-sensitive. If this para-meter is set to "NoCase" the comparison will be done in a case-insensitive manner. The word "NoCase" is not case sensitive.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The two strings are not equal.

Deprecation Level

Deprecated

The *Equal* macro should be used.

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Value1", "aaa")
Set("Value2", "bbb")
@if( AreStringsEqual( Value("Value1"), Value("Value2") ) )
{
"Yes"
}
%else
{
"No"
}
```

Result

No

Choose

Description

This macro, in conjunction with the *Switch* and *Default* macros, tests a predicate against a range of values and executes a block when a match is found.

The block associated with this macro will be evaluated if the predicate matches and no previous *Choose* or *Default* blocks have evaluated successfully.

Prototype

```
Choose( Value ) {}
```

Parameter	Status	Description
Value	Req	The predicate of the Switch macro will be tested against this value.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Assume the context is the primary key of an entity.

Template

```
Switch( Left( Property("Key_Group_Type"), "2" ) )
{
Choose( "PK" )
{
"This is a primary key."
}
Choose( "AK" )
{
"This is an alternate key."
}
Default
{
"This is an inversion entry."
}
Choose( "XX" )
{
/* This block will never execute, because a preceding block
will always evaluate successfully due to the presence of the
Default macro. */
}
}
```

Result

This is a primary key.

ClearAllGlobalFlags

Description

This clears all flags set with *SetGlobalFlag*.

Prototype

```
ClearAllGlobalFlags
```

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
SetGlobalFlag("Flag1")
SetGlobalFlag("Flag2")
@if( IsGlobalFlagSet("Flag1") )
{
"Flag1 is set.\n"
}
@if( IsGlobalFlagSet("Flag2") )
{
"Flag2 is set.\n"
}
ClearAllGlobalFlags
@if( IsGlobalFlagSet("Flag1") )
{
"Flag1 is still set.\n"
}
@if( IsGlobalFlagSet("Flag2") )
{
"Flag2 is still set."
}
```

Result

```
Flag1 is set.
Flag2 is set.
```

ClearGlobalFlag

Description

This clears the named global flag set with *SetGlobalFlag*.

Prototype

```
ClearGlobalFlag( Flag )
```

Parameter	Status	Description
Flag	Req	The name of the flag to clear.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
SetGlobalFlag("Flag1")
SetGlobalFlag("Flag2")
@if( IsGlobalFlagSet("Flag1") )
{
"Flag1 is set.\n"
}
@if( IsGlobalFlagSet("Flag2") )
{
"Flag2 is set.\n"
}
ClearGlobalFlag("Flag1")
@if( IsGlobalFlagSet("Flag1") )
{
"Flag1 is still set.\n"
}
@if( IsGlobalFlagSet("Flag2") )
{
"Flag2 is still set."
}
```

Result

```
Flag1 is set.
Flag2 is set.
Flag2 is still set.
```

ClearLocalFlag

Description

This clears the named global flag set with *SetLocalFlag*.

Prototype

```
ClearLocalFlag( Flag [, Depth] )
```

Parameter	Status	Description
Flag	Req	The name of the flag to clear.
Depth	Opt	This will clear the named local flag from the stack entry *Depth* levels above the current entry.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
SetLocalFlag("Flag1")
PushOwner
SetLocalFlag("Flag2 )
ClearLocalFlag("Flag1", "1")
ClearLocalFlag("Flag2")
@ifnot( IsLocalFlagSet("Flag2") )
{
"Flag2 was cleared.\n"
}
Pop
@ifnot( IsLocalFlagSet("Flag1") )
{
"Flag1 was cleared.\n"
}
```

Result

```
Flag2 was cleared.
Flag1 was cleared.
```

ConversionFunction

Description

This macro prompts you for a conversion function for a column data type change. This occurs when the Alter Script processing determines that it will drop and recreate the table.

Prototype

```
ConversionFunction
```

Result

This macro will fail in the following circumstances:

- An appropriate conversion function cannot be determined.

Deprecation Level

Active

Breaking Changes

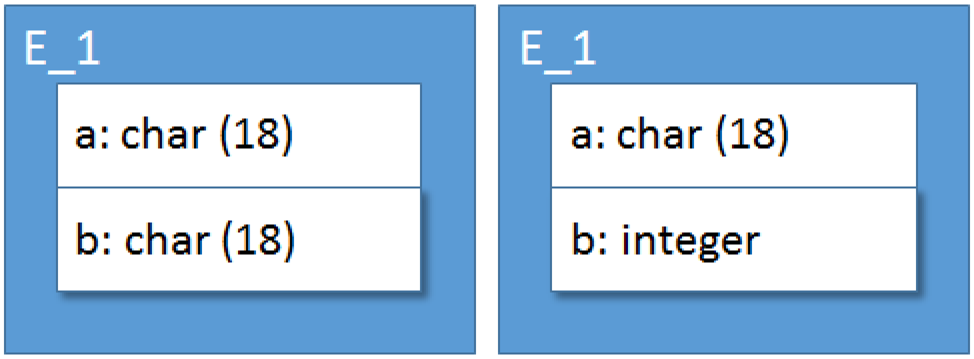
None

Categories

- Alter Macros

Sample

Assume that a table was modified from the left illustration to the right illustration.



If you ran the Alter Script process, pressed Preview and then Save Data, they would see a dialog something like the following.

DatabaseConnection

Description

This evaluates to one of the connection information strings

Prototype

```
DatabaseConnection( Option )
```

Parameter	Status	Description
Option	Req	A keyword indicating the desired string.

The options available are found in the following table. These terms are not case-sensitive.

Option	Meaning
"database"	The database name
"server"	The server name
"user"	The user name

Result

This macro will fail in the following circumstances:

- The required parameter is not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

None.

Sample

Template

```
DatabaseConnection( "user" )
```

Result

```
my_user_name
```

Date

Description

This evaluates to the current time stamp.

Prototype

```
Date( [Format] )
```

Parameter	Status	Description
Format	Opt	This may be either a format string as specified in the table below or the word "System". If the latter is supplied, the format string held by the operating system for the current locale is used. If no parameter is supplied, the default string used is: "%d %b %Y %I:%M:%S %p"

The format string consists of a series of specifiers that indicate a component of a full date/-time value, plus the formatting for that value. For full documentation of all of the formatting options available, the documentation for the C Runtime function strftime() function should be consulted. This documentation can be located on the Web at a number of sites, one example of which is: http://msdn2.microsoft.com/en-US/library/fe06s4ak.aspx.

The following provides an overview of some of the more common specifiers.

%a

Abbreviated weekday name.

%A

Full weekday name.

%b

Abbreviated month name.

%B

Full month name.

%c

Date and time representation appropriate for locale.

%#c

Long date and time representation, appropriate for current locale. For example: "Tuesday, March 14, 1995, 12:41:29".

%d

Day of month as a decimal number (01 31). If a # sign is inserted after the % sign, leading zeros are suppressed.

%H

Hour in 24-hour format (00 23). If a # sign is inserted after the % sign, leading zeros are suppressed.

%I

Hour in 12-hour format (01 12). If a # sign is inserted after the % sign, leading zeros are suppressed.

%j

Day of year as a decimal number (001 366). If a # sign is inserted after the % sign, leading zeros are suppressed.

%m

Month as a decimal number (01 12). If a # sign is inserted after the % sign, leading zeros are suppressed.

%M

Minute as a decimal number (00 59). If a # sign is inserted after the % sign, leading zeros are suppressed.

%p

Current locale's AM/PM indicator for a 12-hour clock.

%S

Second as decimal number (00 59). If a # sign is inserted after the % sign, leading zeros are suppressed.

%U

Week of year as decimal number, with Sunday as first day of week (00 53). If a # sign is inserted after the % sign, leading zeros are suppressed.

%w

Weekday as decimal number (0 6; Sunday is 0). If a # sign is inserted after the % sign, leading zeros are suppressed.

%W

Week of year as decimal number, with Monday as first day of week (00 53). If a # sign is inserted after the % sign, leading zeros are suppressed.

%x

Date representation for current locale.

%#x

Long date representation, appropriate to current locale. For example: "Tuesday, March 14, 1995".

%X

Time representation for current locale.

%y

Year without century as a decimal number (00 99). If a # sign is inserted after the % sign, leading zeros are suppressed.

%Y

Year with century as a decimal number. If a # sign is inserted after the % sign, leading zeros are suppressed.

%z, %Z

Either the time-zone name or time zone abbreviation, depending on registry settings. No characters if time zone is unknown.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Template

```
Date "\n"
Date("system") "\n"
Date("%b %d, %Y")
```

Result

```
03 Oct 2006 02:01:24 PM
Tuesday, October 03, 2006
Oct 03, 2006
```

DBMSVersion

Description

This macro determines if the version of the database specified for the current model falls into a specific range. Version numbers should be specified in the format:

<Major Version Number>.<Minor Version Number>

If no minor version is specified, then it is assumed to be zero.

Prototype

```
DBMSVersion( StartVersion [, StopVersion] )
```

Parameter	Status	Description
Start Version	Req	This identifies the lowest acceptable version number.
Stop Version	Opt	This identifies the highest acceptable version number. If this value is not supplied, only the lower version is tested.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The model has not target database set.
- The specified parameters cannot be parsed.
- The version of the database on the model is outside the specified range.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Assume the context is a deleted entity in Version 9:

Template

```
"DROP TABLE " Property("Physical_Name") " CASCADE CONSTRAINTS"
[DBMSVersion("10") " PURGE"]
```

Result

```
DROP TABLE customer CASCADE RESTRAINTS
```

Assume the context is a deleted entity in Version 10:

Template

```
"DROP TABLE " Property("Physical_Name") " CASCADE CONSTRAINTS"
[DBMSVersion("10") " PURGE"]
```

Result

```
DROP TABLE customer CASCADE RESTRAINTS PURGE
```

Decrement

Description

This macro decrements the value in the specified variable. The variable is assumed to contain an integer value. If it does not, the value is coerced to zero and the decrement occurs. If the variable does not exist, it is created first, with a value of zero.

Prototype

```
Decrement( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the variable.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Var1", "2")
Decrement("Var1") Value("Var1") "\n"
Decrement("Var1") Value("Var1") "\n"
```

```
/* Demonstrate data type coercion */
Set("Var2", "foo")
Decrement("Var2") Value("Var2") "\n"
/* Demonstrate auto-creation */
Decrement("Var3") Value("Var3")
```

Result

```
1
0
-1
-1
```

Default

Description

This macro, in conjunction with the Switch and Choose macros, tests a predicate against a range of values and executes a block when a match is found.

The block associated with this macro will be evaluated if no previous Choose blocks have evaluated successfully.

Prototype

```
Default {}
```

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Assume the context is an inversion entry of an entity.

Template

```
Switch( Left( Property("Key_Group_Type"), "2" ) )
{
Choose( "PK" )
{
"This is a primary key."
}
Choose( "AK" )
{
"This is an alternate key."
}
Default
{
"This is an inversion entry."
}
Choose( "XX" )
{
/* This block will never execute, because a preceding block
will always evaluate successfully */
}
}
```

Result

This is an inversion entry.

EnumProperty

Description

This is a lookup macro that evaluates to a specified string based upon a zero-based integer value contained in a property of the current context object. The integer values are assumed to be in a contiguous range starting at zero and positive in value.

Prototype

```
EnumProperty( Property, Value0 [, Value1 [, …]] )
```

Parameter	Status	Description
Property	Req	The property to examine.
Value0	Req	The value to return if the property has the value of '0'.
Value1 � ValueN	Opt	The value(s) to return if the property has the value of '1' to 'N'.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The specified property does not exist or it has a null value.
- The list of replacement strings is not large enough to accommodate the value found in the property.

Deprecation Level

Active

Breaking Changes

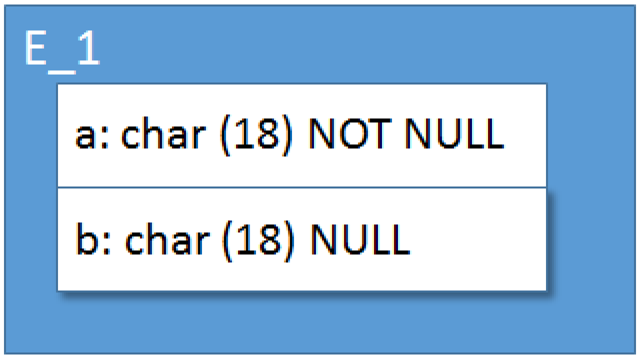
None

Categories

- Property Macros

Sample

Assume the context is the table in the following illustration and the target type is Access.



Template

```
ForEachOwnee("Attribute")
{
EnumProperty( "Null_Option_Type", "NULL", "NOT NULL" ) "\n"
}
```

Result

```
NOT NULL
NULL
```

EnumProperty2

Description

This is a lookup macro that evaluates to a specified string based upon a zero-based integer value contained in a property of the current context object. The values in the property are offset from zero by the amount specified in Offset. For example, if the value of Offset is '10', then the first string will correspond to a property value of '10', the second to the value '11' and so on. The integer values are assumed to be in a contiguous range and result in positive values once the offset is applied.

Prototype

```
EnumProperty2( Property, Offset, Value0 [, Value1 [, …]] )
```

Parameter	Status	Description

Property	Req	The property to examine.
Offset	Req	The amount to offset the value in the property.
Value0	Req	The value to return if the property has the adjusted value of '0'.
Value1 � ValueN	Opt	The value(s) to return if the property has the adjusted value of '1' to 'N'.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The specified property does not exist or it has a null value.
- The list of replacement strings is not large enough to accommodate the value found in the property.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the context is a logical/physical model.

Template

```
EnumProperty2( "Type", "1", "Logical", "Physical",
"Logical/Physical")
```

Result

```
Logical/Physical
```

Equal

Description

This determines if one string is equal to another.

Prototype

```
Equal( LeftString, RightString [, Option] )
```

Parameter	Status	Description
LeftString	Req	The left string in the test.
RightString	Req	The right string in the test.
Option	Opt	Option keyword

The options available are found in the following.

"no_case"

By default the comparison is case-sensitive. If this option is set the comparison will be done in a case-insensitive manner.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The two strings are not equal.

Deprecation Level

Active

Breaking Changes

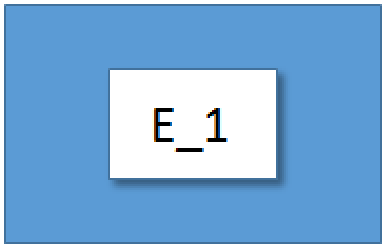
None

Categories

- String Macros

Sample

Assume the context is the following table:



Template

```
@if( Equal(Property("Name"), "e_1") )
{
"Name is exactly 'e_1'.\n"
}
@else
{
"Name is not exactly 'e_1'.\n"
}
@if( Equal(Property("Name"), "e_1", "no_case") )
{
"Name is case-insensitively like 'e_1'."
}
```

Result

```
Name is not exactly 'e_1'.
Name is case-insensitively like 'e_1'.
```

Execute

Description

This macro allows a piece of named template code to be reused. It will load a named template from a template file and expand it in the current context.

This macro will work only in processes that make use of named templates and template files. Currently, these are Schema Generation (Forward Engineering) and Alter Script Generation. More information on template files can be found in the document *Editing Forward Engineering Templates.pdf*.

If the *FileName* parameter is not supplied, the macro looks for the template in the current template file. The current template file is the initial file bound to the TLX engine by the erwin Data Modeler process being run.

If the *FileName* parameter is supplied, the macro looks for the template in the specified file.

Prototype

```
Execute( NamedTemplate [, FileName [, Param1 [, Param2 [, …]]]] )
```

Parameter	Status	Description
NamedTemplate	Req	This is the name of the template entry in the file.
FileName	Opt	This is the name of the template file containing the entry, if it is not the current file. An empty string also indicates the current file. If the specified file does not contain a path specification, the file is assumed to be in the same location as the current template file.
Param1 � *ParamN*	Opt	Parameters to the executed template. The template being invoked can contain replacement tokens�*Param1* will be substituted for �%1�, *Param2* will be substituted for �%2�, and so on.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The template cannot be located.
- The template cannot be evaluated.

Deprecation Level

Active

Breaking Changes

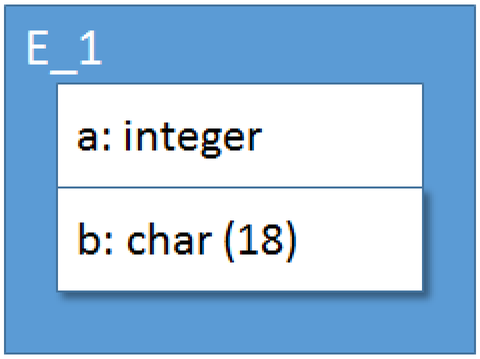
None

Categories

- Include Macros

Sample

The following illustrates a hypothetical template file for constructing very basic CREATE TABLE statements. Assume that the table in the following illustration is the current context the process is using:



Template

```
SPItemBegin = Create Entity
"CREATE TABLE " Property("Name") "\n"
"(\n"
ForEachOwnee("Attribute")
{
ListSeparator(",\n")
"\t" Execute("Clause: Column Properties")
}
")"
SPItemEnd
SPItemBegin = Clause: Column Properties
Property("Physical_Name") " " Property("Physical_Data_Type")
SPItemEnd
```

Result

```
CREATE TABLE E_1
(
a INTEGER,
b CHAR(18)
)
```

ExecuteTest

Description

This macro allows a piece of named template code to be reused. It will load a named template from a template file and expand it in the current context. If the result expands to the text Success , the macro will succeed. Otherwise, it will fail.

> This macro will work only in processes that make use of named templates and template files. Currently, these are Schema Generation (Forward Engineering) and Alter Script Generation. More information on template files can be found in the document *Editing Forward Engineering Templates.pdf.*

If the *FileName* parameter is not supplied, the macro looks for the template in the current template file. The current template file is the initial file bound to the TLX engine by the erwin Data Modeler process being run.

If the *FileName* parameter is supplied, the macro looks for the template in the specified file.

Prototype

```
ExecuteTest( NamedTemplate [, FileName [, Param1 [, Param2 [,
…]]]] )
```

Parameter	Status	Description
NamedTemplate	Req	This is the name of the template entry in the file.
FileName	Opt	This is the name of the template file containing the entry, if it is not the current file. An empty string also indicates the current file.

		If the specified file does not contain a path specification, the file is assumed to be in the same location as the current template file.
Param1 � *ParamN*	Opt	Parameters to the executed template. The template being invoked can contain replacement tokens�*Param1* will be substituted for �%1�, *Param2* will be substituted for �%2�, and so on.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The template cannot be located.
- The template cannot be evaluated.
- The template does not expand to the value Success .

Deprecation Level

Active

Breaking Changes

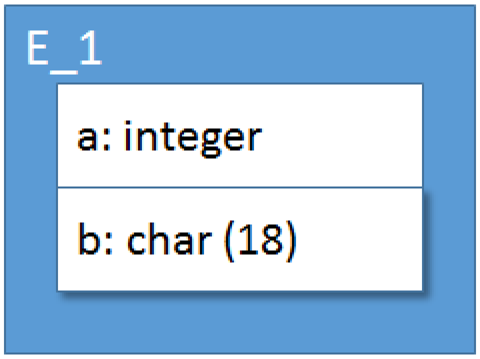
None

Categories

- Include Macros

Sample

The following illustrates a hypothetical template file for constructing very basic CREATE TABLE statements. Assume that the table in the following illustration is the current context the process is using.



Template

```
SPItemBegin = Create Entity
"CREATE TABLE " Property("Name") "\n"
"(\n"
ForEachOwnee("Attribute")
{
ListSeparator(",\n")
"\t" ExecuteTest("Ignore Logical Only")
}
")"
SPItemEnd
SPItemBegin = Ignore Logical Only
@ifnot( Property("Is_Logical_Only") ){ "Success" }
SPItemEnd
```

Result

```
CREATE TABLE E_1
(
a INTEGER,
b CHAR(18)
)
```

Fail

Description

This is a debugging macro that always fails. It can be used to debug conditional blocks.

Prototype

```
Fail
```

Result

This macro always fails.

Deprecation Level

Active

Breaking Changes

None

Categories

Sample

Template

```
[Fail "This should never emit"]
```

Result

ForEachFKColumn

Description

This iterates across the instances of a specific type of object.

Prototype

```
ForEachFKColumn{}
```

Result

This macro succeeds.

Deprecation Level

Active

Breaking Changes

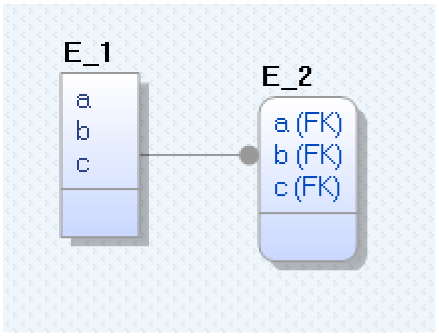
None

Categories

- Iterator Macros

Sample

Assume the following model and the foreign key in E_2 is the context object:



Template

```
ForEachFKColumn
{
PushOwner Property("Name") Pop "." Property("Name") "\n"
}
```

Result

```
E_2.a
E_2.b
E_2.c
```

ForEachMigratingColumn

Description

This iterates across the instances of a specific type of object.

Prototype

```
ForEachMigratingColumn{}
```

Result

This macro succeeds.

Deprecation Level

Active

Breaking Changes

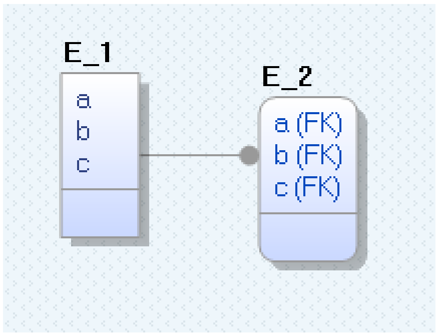
None

Categories

- Iterator Macros

Sample

Assume the following model and the foreign key in E_2 is the context object:



Template

```
ForEachMigratingColumn
{
PushOwner Property("Name") Pop "." Property("Name") "\n"
}
```

Result

```
E_1.a
E_1.b
E_1.c
```

ForEachOfType

Description

This iterates across the instances of a specific type of object.

Prototype

```
ForEachOfType( TypeName [, Category] ){}
```

Parameter	Status	Description
TypeName	Req	The type of object to traverse.
Category	Opt	A filter on what type of objects should be returned. If this value is not specified, "active" is assumed.

The *Category* values available are found in the following.

"active"

Only actual, non-phantom objects of the specified type are returned.

"deleted"

Only phantom objects representing deleted objects of the specified type are returned.

"modified"

Only phantom objects representing the pre-image of modified objects of the specified type are returned.

"phantom"

Only phantom objects, both deleted and modified, of the specified type are returned.

"all"

All phantom and non-phantom objects of the specified type are returned.

Result

This macro succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

Iterator Macros

Sample

Assume a model that, when loaded, had three Entity objects: E_1, E_2 and E_3. Assume E_2 was deleted, E_3 was renamed to New_3, and E_4 was created.

Template

```
"Active entities: "
ForEachOfType("Entity")
{
ListSeparator(", ")
Property("Name")
}
"\nDeleted entities: "
ForEachOfType("Entity", "deleted")
{
ListSeparator(", ")
Property("Name")
}
"\nModified entities: "
ForEachOfType("Entity", "modified")
{
ListSeparator(", ")
Property("Name")
}
```

Result

```
Active entities: E_1, New_3, E_4
Deleted entities: E_2
Modified entities: E_3
```

ForEachOwnee

Description

This iterates across the ownee list of the current context object.

When the current context object is a phantom object representing the old state of a modified object, the ownees reported will be those present in the old state, not the current set of ownees.

Prototype

```
ForEachOwnee( [TypeName [, SortBy [, Option1 [, Option2 [,…]]]]] )
{}
```

Parameter	Status	Description
TypeName	Opt	If a type name is provided, the iteration will be filtered to pass only objects of that type. If this is not provided, all owned objects will be returned.
SortProperty	Opt	The property to use as the basis of sorting. If this is not provided or is an empty string, no sorting will take place and the objects will appear in whatever internal order is held in erwin Data Modeler.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following.

"owner_sort"

If this option is present, SortProperty is assumed to be found on the owning (context) object and to be in the form of a vector of ownee ids. If the owning object does not have the specified property, no sorting will take place. If this is not present, the property is assumed to be found on the ownee objects and they will be sorted based upon its value.

"reverse_sort"

If this option is present, the sorting will be reversed.

"require_one"

If this option is present, at least one iteration must occur or the macro will fail.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

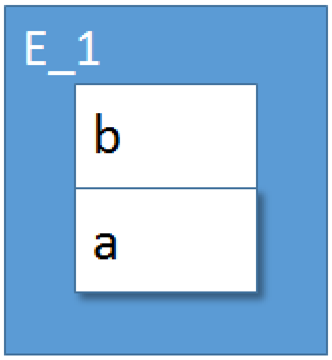
None

Categories

- Iterator Macros

Sample

Assume the context is the table in the following illustration, which is shown in the Physical Order display level:



Template

```
ForEachOwnee("Attribute", "Physical_Columns_Order_Ref", "owner_
sort")
{
Property("Physical_Name") "\n"
}
"\n"
ForEachOwnee("Attribute", "Name")
{
Property("Physical_Name") "\n")
}
```

Result

```
b
a
a
b
```

ForEachOwneeFrom

Description

This iterates across the ownee list on the first object in the context stack with the specified type.

Prototype

```
ForEachOwneeFrom( ContextType [, TypeName [, SortBy [, Option1
[, Option2 [,…]]]]] ){}
```

Parameter	Status	Description
ContextType	Req	The type of object to locate on the context stack.
TypeName	Opt	If a type name is provided, the iteration will be filtered to pass only objects of that type. If this is not provided, all owned objects will be returned.
SortProperty	Opt	The property to use as the basis of sorting. If this is not provided or is an empty string, no sorting will take place

		and the objects will appear in whatever internal order is held in erwin Data Modeler.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

Result

This macro will fail in the following circumstances:

- The current object is a phantom object.
- An object of that type does not exist on the context stack.

Deprecation Level

Active

Breaking Changes

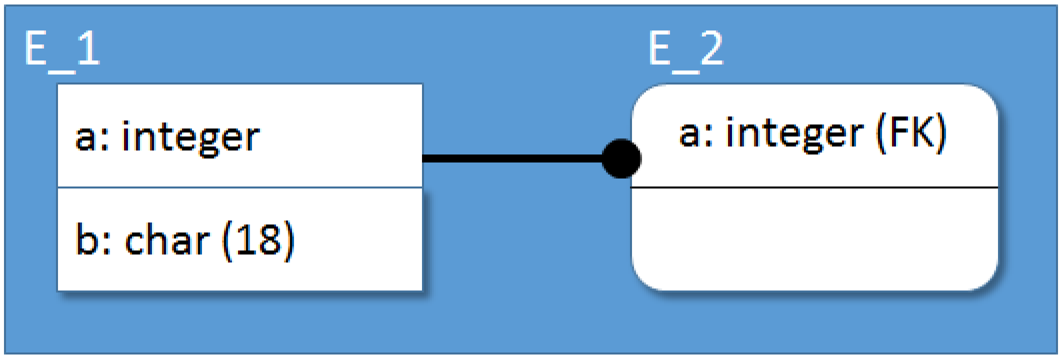
None

Categories

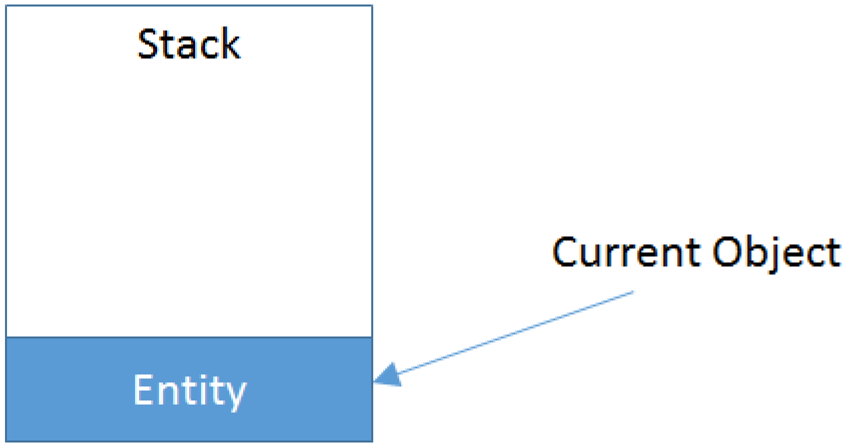
- Iterator Macros

Sample

To illustrate the difference between this and the *ForEachOwnee* macro, assume that context object is E_1 in the following illustration



When the following template code is expanded, both loops produce the same result because an Entity object is current on the stack.



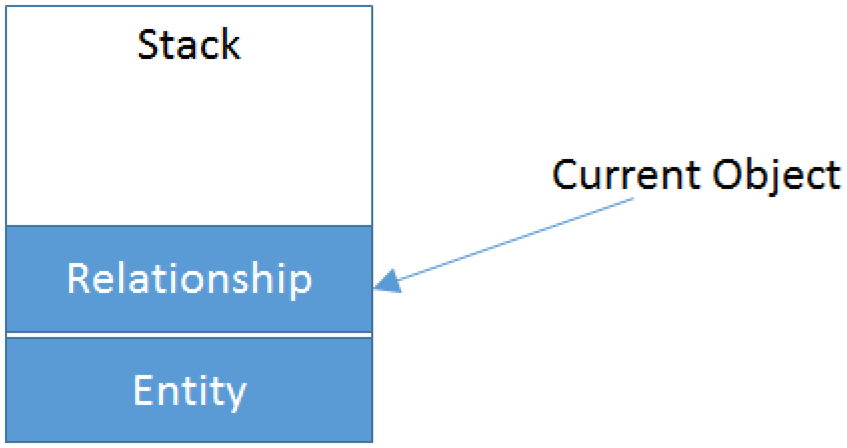
Template

```
ForEachOwnee("Attribute")
{
Property("Physical_Name") "\n"
}
"\n"
ForEachOwneeFrom("Entity", "Attribute")
{
Property("Physical_Name") "\n"
}
```

Result

```
a
b
a
b
```

However, if we introduce other objects onto the context stack, the behavior of the two iterators changes. For example, if we introduce Relationship onto the stack above the Entity, using the *ForEachReference* macro, the stack will appear as follows:



Since Relationship objects don t own Attribute objects, the first iterator will not traverse any objects. The second iterator will still locate the Entity object and traverse its ownees.

Template

```
ForEachReference("Parent_Relationships_Ref")
{
ForEachOwnee("Attribute")
{
Property("Physical_Name") "\n"
}
"\n"
ForEachOwneeFrom("Entity", "Attribute")
{
Property("Physical_Name") "\n"
}
}
```

Result

```
a
b
```

ForEachOwneeThrough

Description

This iterates across the ownee list on the object pointed to by the specified scalar reference property on the current context object.

Prototype

```
ForEachOwneeThrough( ReferenceProperty [, TypeName [, SortBy
[, Option1 [, Option2 [,…]]]]] ){}
```

Parameter	Status	Description
ReferenceProperty	Req	The reference property to use to locate the desired owning object.
TypeName	Opt	If a type name is provided, the iteration will be filtered to pass only objects of that type. If this is not provided, all owned objects will be returned.
SortProperty	Opt	The property to use as the basis of sorting. If this is not provided or is an empty string, no sorting will take place.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following.

"owner_sort"

If this option is present, SortProperty is assumed to be found on the owning (context) object and to be in the form of a vector of ownee ids. If the owning object does not have the specified property, no sorting will take place. If this is not present, the property is assumed to be found on the ownee objects and they will be sorted based upon its value.

"reverse_sort"

If this option is present, the sorting will be reversed.

"require_one"

If this option is present, at least one iteration must occur or the macro will fail.

Result

This macro will fail in the following circumstances:

- The current object is a phantom object.
- The reference property does not exist.

Deprecation Level

Active

Breaking Changes

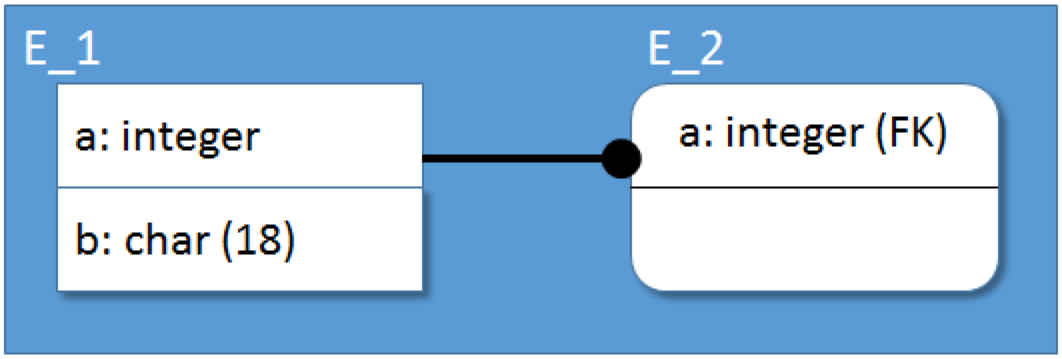
None

Categories

- Iterator Macros

Sample

Assume the current context object is the Relationship in the following illustration



Template

```
ForEachOwneeThrough("Parent_Entity_Ref", "Attribute")
{
Property("Physical_Name") "\n"
}
```

Result

```
a
b
```

ForEachProperty

Description

This is a special-purpose macro generally used in components that produce dumps of models. It will iterate across the properties of an object. Special sub-macros are then available to retrieve information about the property.

The context object is not changed by this iterator.

Prototype

```
ForEachProperty( [Option1 [, Option2 [, …]]] ){}
```

Parameter	Status	Description
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table. These terms are not case-sensitive.

"type_sort"	The properties will be sorted by their type.
"all"	Pass all types of properties. This is the default setting and will override conflicting options.
"user_defined"	Pass only user-defined properties.
"built_in"	Pass only built-in properties.

Result

This macro will fail in the following circumstances:

- The context object is a phantom object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

Assume that the current context is the M1 object Domain (the object defining the metadata for Domain instances in the model) and we want to see what tags are applied to it.

Template

```
/* List the tags, putting inherited ones in HTML italics */
ForEachProperty("user_defined", "types_sort")
{
[ForEachProperty.IsInherited "<i>"]
ForEachProperty.Type
[ForEachProperty.IsInherited "</i>"] "="
ForEachProperty.Value "\n"
}
```

Result

```
<i>HasUdpEditor</i>=true
<i>IsExplorerSuppressed</i>=true
IsLogical=true
Is_Physical=true
Legacy_MM_Order=7
Physical_Name=Domain
```

ForEachProperty.IsInherited

Description

This is a special-purpose macro is that is usable only inside of a *ForEachProperty* iterator, and then only when the context object is an M1 (metadata) object.

It will indicate if the property was defined on the context object or inherited from a parent M1 object.

Prototype

```
ForEachProperty.IsInherited
```

Result

This macro will fail in the following circumstances:

- It is not executed inside the proper containing iterator.
- The property is not inherited.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

See the example under the *ForEachProperty* macro.

ForEachProperty.Type

Description

This is a special-purpose macro is that is usable only inside of a *ForEachProperty* iterator, and then only when the context object is an M1 (metadata) object.

It will evaluate to the type name of the property.

Prototype

```
ForEachProperty.Type
```

Result

This macro will fail in the following circumstances:

- It is not executed inside the proper containing iterator.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

See the example under the *ForEachProperty* macro.

ForEachProperty.Value

Description

This is a special-purpose macro is that is usable only inside of a ForEachProperty iterator, and then only when the context object is an M1 (metadata) object.

It will evaluate to the value of the property.

This macro will fail if:

- It is not executed inside the proper containing iterator.

Prototype

```
ForEachProperty.Value( [NullString] )
```

Parameter	Status	Description

NullString	Opt	The string to emit if the property's value is a NULL. If this is not specified, an empty string will be emitted.

Result

This macro will fail in the following circumstances:

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

See the example under the *ForEachProperty* macro.

ForEachPropertyValue

Description

This iterates across the values present in the specified property on the current context object.

The context object is not changed by this iterator.

Prototype

```
ForEachProperty( Property [, Option1 [, …]]] ){}
```

Parameter	Status	Description
Property	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table. These terms are not case-sensitive.

"new_only"	On a modified object, only traverse the new values.
"old_only"	On a modified object, only traverse the old values.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The context object is a phantom object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

The following example traverses the old and new values of a *Partition* object, merging the old values and splitting out the new values.

Template

```
IsPropertyModified( "Partition_Values" )
[
ForEachPropertyValue( "Partition_Values", "old_only" )
{
"ALTER PARTITION FUNCTION " Property( "Name" ) "() "
"\nMERGE RANGE (" ForEachPropertyValue.Value ")"
FE::EndOfStatement
}
]
[
ForEachPropertyValue( "Partition_Values", "new_only" )
{
"ALTER PARTITION FUNCTION " Property( "Name" ) "() "
"\nSPLIT RANGE (" ForEachPropertyValue.Value ")"
FE::EndOfStatement
}
]
```

ForEachPropertyValue.Value

Description

This is a special-purpose macro is that is usable only inside of a ForEachProperty iterator, and then only when the context object is an M1 (metadata) object.

It will evaluate to the value of the property.

This macro will fail if:

- It is not executed inside the proper containing iterator.

Prototype

```
ForEachPropertyValue.Value
```

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

See the example under the *ForEachPropertyValue* macro.

ForEachReference

Description

This iterates across the objects listed in the specified property on the current context object.

Prototype

```
ForEachReference( Property, [Option1 [, Option2 [, …]]] ){}
```

Parameter	Status	Description
Property	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table.

"name_sort"

The objects will be sorted according to the 'Name' property in ascending order.

"flatten"

When the context objects exist in the M1 (metadata) layer of the model this option will cause the values on parent objects to be pushed down to the child objects. Do not use this when the macro is employed in the context of an M0 (data) model as unpredictable results will occur.

"exact"

This will cause the vector of references to be traversed exactly as they are found. Normally, duplicates are suppressed; this changes that behavior. This option will cause the "name_sort" and "flatten" options to be ignored, if they are specified.

Result

This macro will fail in the following circumstances:

- The specified parameters are not provided.
- The specified property is not found on the context object.

Deprecation Level

Active

Breaking Changes

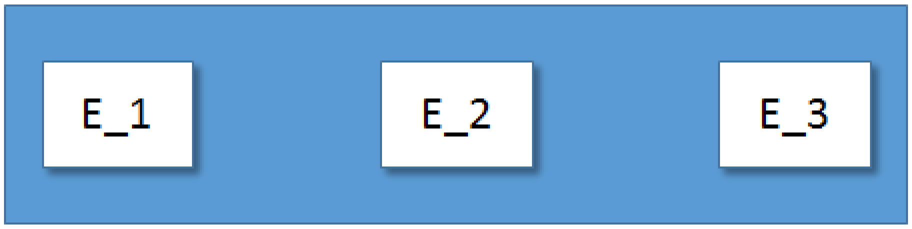
None

Categories

Iterator Macros

Sample

Assume the current context is a *Subject Area* object containing the tables shown in the following illustration:



Template

```
ForEachReference("Referenced_Entities_Ref")
{
Property("Name") "\n"
}
```

Result

```
E_1
E_2
E_3
```

ForEachReference.IsInherited

Description

This is a special-purpose macro is that is usable only inside of a ForEachReference iterator, and then only when the context object is an M1 (metadata) object.

It will indicate if the value in the reference property vector was defined on the context object or inherited from a parent M1 object.

Prototype

```
ForEachReference.IsInherited
```

Result

This macro will fail in the following circumstances:

- It is not executed inside the proper containing iterator.
- The value in the reference vector is not inherited.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

Assume that the current context is the M1 object Domain (the object defining the metadata for Domain instances in the model) and we want to see object types can own it.

Template

```
ForEachReference("Valid_Owners_Ref", "name_sort")
{
```

```
Property("Name") "\n"
}
```

Result

```
Model__owns__Domain
```

ForEachReferenceFrom

Description

This iterates across the objects listed in the specified property on the first object of the specified type found on the context stack.

Prototype

```
ForEachReferenceFrom( ContextType, Property, [Option1 [, Option2
[, …]]] ){}
```

Parameter	Status	Description
ContextType	Req	The type of object to locate on the context stack.
Property	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table.

"name_sort"

The objects will be sorted according to the 'Name' property in ascending order.

"flatten"

When the context objects exist in the M1 (metadata) layer of the model this option will cause the values on parent objects to be pushed down to the child objects. Do not use this when the macro is employed in the context of an M0 (data) model as unpredictable results will occur.

Result

This macro will fail in the following circumstances:

- The specified parameters are not provided.
- There is no context object of the specified type.
- The specified property is not found on the context object.

Deprecation Level

Active

Breaking Changes

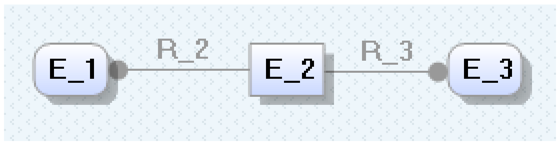
None

Categories

- Iterator Macros

Sample

Assume that the current context is E_2 in the following illustration, and that E_2 owns a trigger:



Template

```
ForEachOwnee("Trigger")
{
ForEachReferenceFrom("Entity", "Parent_Relationships_Ref")
{
ListSeparator("\n")
Property("Physical_Name")
}
}
```

Result

```
R_2
R_3
```

ForEachReferenceThrough

Description

This iterates across the objects listed in the specified property on the object pointed to by the specified scalar reference property on the current context object

Prototype

```
ForEachReferenceFrom( ReferenceProperty, Property, [Option1
[, Option2 [, …]]] ){}
```

Parameter	Status	Description
ReferenceProperty	Req	The reference property to use to locate the desired owning object.
Property	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table.

"name_sort"

The objects will be sorted according to the 'Name' property in ascending order.

"flatten"

When the context objects exist in the M1 (metadata) layer of the model this option will cause the values on parent objects to be pushed down to the child objects. Do not use this when the macro is employed in the context of an M0 (data) model as unpredictable results will occur.

Result

This macro will fail in the following circumstances:

- The specified parameters are not provided.
- There is no context object of the specified type.

- The specified property is not found on the context object.

Deprecation Level

Active

Breaking Changes

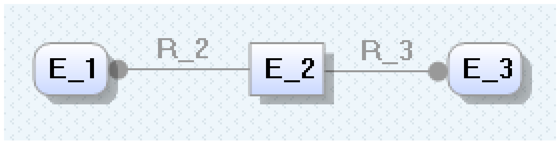
None

Categories

- Iterator Macros

Sample

Assume that the current context is R_2 in the following illustration:



Template

```
ForEachReferenceThrough("Parent_Entity_Ref",
"Parent_Relationships_Ref")
{
ListSeparator("\n")
Property("Physical_Name")
}
}
```

Result

```
R_2
R_3
```

ForEachReferencing

Description

This iterates across the referencing set of the current context object. The referencing set is all objects that hold a reference property to that object.

Prototype

```
ForEachReference{}
```

Result

This macro will fail in the following circumstances:

- The context object is a phantom object.

Deprecation Level

Active

Breaking Changes

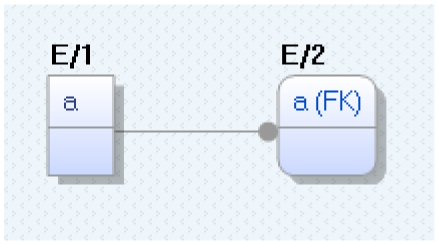
None

Categories

- Iterator Macros

Sample

Assume the current context is the Relationship object shown in the following illustration:



Template

```
ForEachReferencing
{
ObjectType " - " Property("Name") "\n"
}
```

Result

```
Entity - E_1
Entity - E_2
Drawing Object Relationship - R/1
Key Group - XIF1E/2
Attribute - a
```

ForEachUserDefinedProperty

Description

This will iterate across the user-defined properties of an object. Special sub-macros are then available to retrieve information about the property.

The context object is not changed by this iterator.

Prototype

```
ForEachUserProperty( [Option1 [, Option2 [, …]]] ){}
```

Parameter	Status	Description
Option1 � OptionN	Opt	One or more option keywords. These can appear in any order.

The options available are found in the following table. These terms are not case-sensitive.

"all"	Pass all user-defined properties. This is the same as passing no parameter.
"database"	Pass user-defined properties having the Is_Database_Property tag set.
"logical"	Pass user-defined properties having the Is_Logical tag set.
"physical"	Pass user-defined properties having the Is_Physical tag set.

Result

This macro will fail in the following circumstances:

The context object is a phantom object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

Assume that the current context is the Model with a UDP named "Foo" set on the logical side and a UDP named "Bar" set on the physical side.

Template

```
"All UDPs\n"
ForEachUserDefinedProperty
{
"\t" ForEachUserDefinedProperty.Type "\n"
}
"\nLogical UDPs\n"
ForEachUserDefinedProperty( "logical" )
{
"\t" ForEachUserDefinedProperty.Type "\n"
}
```

Result

```
All UDPs
Model.Logical.foo
Model.Physical.bar
Logical UDPs
Model.Logical.foo
```

FormatProperty

Description

This macro evaluates to a formatted representation of the specified property on the current context object. A property is identified by its class name as defined in the metadata for erwin Data Modeler.

Prototype

```
Property( PropertyName [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
PropertyName	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

-d:<value>

This specifies the delimiter that will be inserted between individual values in a vector property. If this option is not supplied, or <value> is empty, a comma will be used.

-b:<value>

This specifies the bracketing characters for each value in the property. If <value> is one character, it will be applied as an opening and closing bracket. If <value> is two characters, the first character will be applied as an opening bracket and the second character as a closing bracket.

-p:<value>

If the property is a reference property, this option will cause the property specified by <value> to be read on the referenced object. If this value is not supplied, or the property cannot be evaluated, the raw reference value will be used.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to the resource identifier string instead of the resource value.

"fail_if_empty"

If a parameter with the value of "fail_if_empty" is supplied, the macro will fail if the result is an empty string.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The "fail_if_empty" option is supplied and the result is an empty string.

Deprecation Level

Active

Breaking Changes

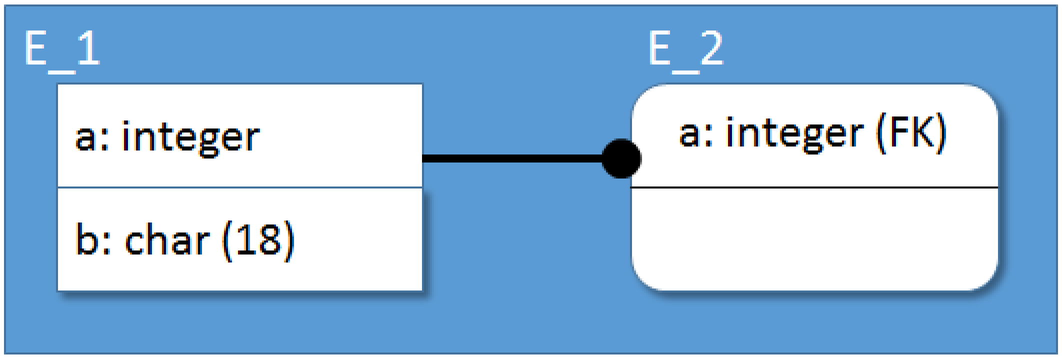
None.

Categories

- Property Macros

Sample

Assume the current context object is a subject area containing the tables in the following illustration:



Template

```
FormatProperty("Referenced_Entities_Ref", "d:,", "b:\"", "p:Name")
/* Which would be the same as… */ "\n\n"
ForEachReference("Referenced_Entities_Ref")
{
ListSeparator(",")
"\"" Property("Name") "\""
}
```

Result

```
"E_1","E_2"
"E_1","E_2"
```

Greater

Description

This determines if one value is greater than another.

If the values are being compared as numbers, it assumes that the parameters passed are numeric values. In these cases, the macro stops reading a parameter when it encounters a character it cannot convert. If no characters are converted, the value is assumed to be zero. For example (assuming "ascii" is not specified):

"123"

123

"123Foo"

123

"Foo"

0

Prototype

```
Greater( LeftString, RightString [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
LeftString	Req	The left value in the test.
RightString	Req	The right value in the test.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"ascii"

By default, the strings are compared as numeric values. For example, "11" would compare as greater than "9".

If this parameter is set to "ascii " the strings are interpreted as case-sensitive literals (ASCII sort). This would cause "11" to compare as less than "9".

"no_case"

If this is set to "no_case" and the values are being compared in an ASCII sort, the strings are compared as strings in a case-insensitive manner. The default behavior is a case-sensitive comparison

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The first string is not greater than the second.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Value1", "11")
Set("Value2", "9")
@if( Greater( Value("Value1"), Value("Value2") )
{
"Greater on a numeric sort\n"
}
@else
{
"Not greater on a numeric sort\n"
}
@if( Greater( Value("Value1"), Value("Value2"), "ascii" )
{
"Greater on an ASCII sort\n"
}
@else
{
"Not greater on an ASCII sort\n"
}
```

Result

```
Greater on a numeric sort
Not greater on an ASCII sort
```

GreaterOrEqual

Description

This determines if one value is greater than or equal to another.

If the values are being compared as numbers, it assumes that the parameters passed are numeric values. In these cases, the macro stops reading a parameter when it encounters a character it cannot convert. If no characters are converted, the value is assumed to be zero. For example (assuming "ascii" is not specified):

"123"

123

"123Foo"

123

"Foo"

0

Prototype

```
GreaterOrEqual( LeftString, RightString
[, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
LeftString	Req	The left value in the test.
RightString	Req	The right value in the test.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

ascii"

By default, the strings are compared as numeric values. For example, "11" would compare as greater than "9".

If this parameter is set to "ascii " the strings are interpreted as case-sensitive literals (ASCII sort). This would cause "11" to compare as less than "9".

"no_case"

If this is set to "no_case" and the values are being compared in an ASCII sort, the strings are compared as strings in a case-insensitive manner. The default behavior is a case-sensitive comparison

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The first string is not greater than or equal to the second.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Value1", "11")
Set("Value2", "9")
@if( GreaterOrEqual( Value("Value1"), Value("Value2") )
{
"Greater or equal on a numeric sort\n"
}
@else
{
"Less on a numeric sort\n"
}
@if( GreaterOrEqual( Value("Value1"), Value("Value2"), "ascii" )
{
"Greater or equal on an ASCII sort\n"
}
@else
{
"Less on an ASCII sort\n"
}
```

Result

```
Greater or equal on a numeric sort
Less on an ASCII sort
```

HasOwnees

Description

This will determine if the current context object has ownees of a given type.

Prototype

```
HasOwnees( [Type] )
```

Parameter	Status	Description
Type	Opt	The type of the ownee desired. If not type is specified, then the macro will succeed if the object has any ownees.

Result

This macro will fail in the following circumstances:

- There is no context object.
- There are no ownees of the specified type.

Deprecation Level

Active

Breaking Changes

None

Categories

- Object Macros

Sample

Assume the current context is a Key_Group.

Template

```
/* Only emit something if the key group has members. */
[HasOwnees(  Key_Group_Member  )
…
]
```

HasPropertyCharacteristic

Description

This determines if the specified property on the current context object has the specified characteristic.

If this macro is invoked against a phantom object, the results are undefined.

Prototype

```
HasPropertyCharacteristic( PropertyName, Characteristic )
```

Parameter	Status	Description
PropertyName	Req	The type name of the property.
Characteristic	Req	The name of the characteristic that is to be checked. Currently supported values are found below.

The characteristics available are found in the following table. The names are not case-sensitive.

"calculated" or "prefetch "

This characteristic is set when the property does not have a value of its own, but is calculated from other properties on the object.

"default "

This characteristic is set if the value in the property is a default value supplied by erwin Data Modeler.

"hardened"

This characteristic is set if the value in the property has been hardened against change. Currently, this is supported only for certain name properties on logical/physical objects, so this only has meaning when applied to the Name and Physical_ Name properties of an Attribute, Default, Domain, Entity, Key_Group, Relationship or Validation_Rule. Future releases of erwin Data Modeler may extend hardening to more property types.

"autocalculated"

This characteristic is set when a property is in an auto-calculate state. Currently, this is supported only for the Cardinality property on the Relationship object. Future releases may extend this to other properties.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The characteristic is not set.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
/* Indicate if the physical name is hardened. */
ForEachOwnee("Attribute")
{
[HasPropertyCharacteristic("Physical_Name", "hardened")
"Column " Property("Physical_Name") "is hardened\n"
]
}
```

IncludeFile

Description

This loads the contents of a text file, then parses and evaluates it.

> The file is expected to contain only TLX code, and the entire contents will be evaluated in one pass. The contents are expected to contain macro block delimiters. Contrast this with the *Execute* macro, which works on a file containing multiple TLX entries and where the entries do not contain macro block delimiters.

Prototype

```
IncludeFile( FileName )
```

Parameter	Status	Description
Fileame	Req	The name of the file. When specifying a path, remember that backslashes in literals must be escaped.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The specified source file is not found.

Deprecation Level

Active

Breaking Changes

None

Categories

- Include Macros

Sample

Assume a file, *reusable_template.txt*, had contents such as the following.

```
The entity's name is {# Property("Physical Name") #}.
```

Assume the current context is a table called E_1.

Template

```
IncludeFile( "c:\\reusable_template.txt" )
```

Result

```
The entity's name is E_1.
```

Increment

Description

This macro will increment the value in the specified variable. The variable is assumed to contain an integer value. If it does not, the value is coerced to zero and the increment occurs. If the variable does not exist, it is created first with a value of zero, then incremented.

Prototype

```
Increment( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the variable.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Var1", "0")
Increment("Var1") Value("Var1") "\n"
Increment("Var1") Value("Var1") "\n"
Set("Var2", "foo")
Increment("Var2") Value("Var2") "\n"
Increment("Var3") Value("Var3")
```

Result

```
1
2
1
1
```

Integer

Description

This retrieves the value in the specified variable previously set by SetInteger.

Prototype

```
Integer( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable is not found.

Deprecation Level

Deprecated

Use the *Set* and *Value* macros.

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetInteger("Counter", "0")
Integer("Counter")
```

Result

```
0
```

IsCreated

Description

This macro will test whether the context object was created during the current session.

Prototype

```
IsCreated
```

Result

This macro will fail in the following circumstances:

- The context object was not created during the current session.

Deprecation Level

Active

Breaking Changes

None

Categories

Alter Macros

Sample

Assume that there are two Entity objects in the model, E_1 was reverse-engineered from the database and E_2 was newly-created. The current context is the model.

Template

```
ForEachOfType("Entity")
{
ListSeparator("\n")
@if ( IsCreated )
{
"CREATE TABLE " Property("Physical_Name") …
}
@else
{
"ALTER TABLE " Property("Physical_Name") …
}
}
```

Result

```
ALTER TABLE E_1 …
CREATE TABLE E_2 …
```

IsDefaultRITrigger

Description

This macro will test whether the current Trigger object is a erwin Data Modeler-generated RI trigger.

Prototype

```
IsDefaultRITrigger
```

Result

This macro will fail in the following circumstances:

- The context object is not a erwin Data Modeler-generated RI Trigger object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Object Macros

Sample

The following will retrieve the body of the trigger from the object if it is a default RI trigger. Otherwise, it will use a template to create the body.

Template

```
<@if ( IsDefaultRITrigger )
{
Property( "Trigger_Body", "no_translate" )
}
@else
{
FE::ExpandErwinMacro( "Trigger_Body" )
}
>
```

IsDeleted

Description

This macro will test whether the context object was deleted during the current session.

Prototype

IsDeleted

Result

This macro will fail in the following circumstances:

- The context object was not deleted during the current session.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample-

Assume that table E_1 was deleted from the model.

Template

```
ForEachOfType("Entity", "all")
{
[ IsDeleted
"DROP TABLE " Property("Physical_Name")
]
}
```

Result

```
DROP TABLE E_1
```

IsGlobalFlagClear

Description

This macro will test whether a global flag is clear (not set). Global flags are set by the *SetG-lobalFlag* macro. Flags are not case-sensitive.

Prototype

```
IsGlobalFlagClear( Flag )
```

Parameter	Status	Description
Flag	Req	The name of the flag to test.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The global flag is set.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
/* Flag that the table is being recreated */
SetGlobalFlag( Property("Name") "created")
…
```

```
@if ( IsGlobalFlagClear( Property("Name") "created" ) )
{
"ALTER TABLE " …
}
@else
{
"/* ALTER not necessary */"
}
```

Result

```
/* ALTER not necessary */
```

IsGlobalFlagSet

Description

This macro will test whether a global flag has been set. Global flags are set by the *SetG-lobalFlag* macro. Flags are not case-sensitive.

Prototype

```
IsGlobalFlagSet( Flag )
```

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The global flag is not set.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
/* Flag that the table is being recreated */
SetGlobalFlag( Property("Name") "created")
…
@if ( IsGlobalFlagSet( Property("Name") "created" ) )
{
"/* ALTER not necessary */"
}
@else
{
"ALTER TABLE " …
}
```

Result

```
/* ALTER not necessary */
```

IsLocalFlagSet

Description

This macro will test whether a local flag has been set for the current context object. Local flags are set by the SetLocalFlag macro. Flags are not case-sensitive.

Prototype

```
IsLocalFlagSet( Flag )
```

Parameter	Status	Description
Flag	Req	The name of the flag to test.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The local flag is not set.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
SetLocalFlag( "MyFlag" )
[IsLocalFlagSet("MyFlag") "Flag was set #1.\n"]
PushOwner
[IsLocalFlagSet("MyFlag") "Flag was set #2.\n"]
Pop
[IsLocalFlagSet("MyFlag") "Flag was set #3."]
```

Result

```
Flag was set #1.
Flag was set #3.
```

IsMatch

Description

This macro will succeed if the specified value matches any item in a list. The comparisons are not case sensitive.

Prototype

```
IsMatch( Value, MatchValue0 [, MatchValue1 [, …] ] )
```

Parameter	Status	Description
Value	Req	The value to test for.
MatchValue	Req	The first value to compare against.
MatchValue1 � MatchValueN	Opt	Other values to compare against.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The Value parameter is not equal to any of the MatchValues.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Assume the current context is a View object.

Template

```
[IsMatch( ObjectType, "Entity", "View", "Cached_View" )
"It's an entity-like object."
]
```

Result

```
It's an entity-like object.
```

IsModified

Description

This macro will test whether the context object was modified during the current session.

If exception properties are supplied as parameters, they will be ignored in considering whether or not the object was modified.

Prototype

```
IsModified( [ExceptionProperty1 [, ExceptionProperty2 [, …]]] )
```

Parameter	Status	Description
ExceptionProperty1 � ExceptionPropertyN	Req	Properties to ignore when considering modification status.

Result

This macro will fail in the following circumstances:

- The context object was not modified during the current session.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Assume that table E_1 had its physical name modified.

Template

```
ForEachOfType("Entity")
{
@if( IsModified )
{
"Property("Physical_Name") " was modified.\n"
}
}
```

Result

```
E_1 was modified.
```

IsNotInheritedFromUDD

Description

This macro determines if domain (user-defined datatype) inheritance is present for a constraint. If Property is specified, the macro checks if the specified property is inherited. If Property is not specified, it checks to see if the Check_Constraint_Usage object is inherited.

Since inherited constraints and some properties are not present in the database, this macro allows determinations to be made if DROP statements should be created.

Prototype

```
IsNotInheritedFromUDD( [Property] )
```

Parameter	Status	Description
Property	Opt	The type name of the property.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The domain inheritance is not present..

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Template

```
/* If the column defines a default for itself, we need to do some-
thing. If it picks up the default from a user-defined datatype, we
don't. */
[ IsNotInheritedFromUDD
"exec sp_binddefault " …
]
```

Result

```
The sp_binddefault will be emitted for columns that define their
own defaults.
```

IsOwnerPropertyEqual

Description

This macro will succeed if the specified property has the specified value on the owning object of the current context object.

Prototype

```
IsOwnerPropertyEqual( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to 'true' or 'false' to indicate the

		desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The property is found and does not have the specified value.
- The property is not found and NotFoundValue is not set to "true".
- The current context object does not have an owner object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
ForEachOwnee("Attribute")
{
[IsOwnerPropertyEqual("Oracle_Is_Temporary_Table", "true")
…
]
}
I
```

IsOwnerPropertyFalse

Description

This macro will succeed if the value in the property is a Boolean value of 'false' on the owning object of the current context object. Missing Boolean properties are assumed to be 'false'; use the *NotFoundValue* if this behavior is not desired.

Prototype

```
IsOwnerPropertyFalse( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'false' and the property does not exist, the macro will fail.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property does not exist and NotFoundValue is set.
- The value in the property is not a Boolean.
- The value in the property is not 'false'.
- The current context object does not have an owner object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
ForEachOwnee("Attribute")
{
[IsOwnerPropertyFalse("Oracle_Is_Temporary_Table", "true")
…
]
}
```

IsOwnerPropertyNotEqual

Description

This macro will succeed if the specified property does not have the specified value on the owning object of the current context object.

Prototype

```
IsOwnerPropertyNotEqual( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to "true" or "false" to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property value equals Value.
- The property is not found and NotFoundValue is 'false' or not specified.
- The current context object does not have an owner object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
ForEachOwnee("Attribute")
{
[IsOwnerPropertyNotEqual("Oracle_Is_Temporary_Table", "true")
…
]
}
```

IsOwnerPropertyTrue

Description

This macro will succeed if the value in the property is a Boolean value of 'true' on the owning object of the current context object.

Prototype

```
IsOwnerPropertyTrue( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'true' and the property does not exist, the macro will succeed.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property does not exist and NotFoundValue is not set.
- The value in the property is not a Boolean.
- The value in the property is not 'true'.
- The current context object does not have an owner object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
ForEachOwnee("Attribute")
{
[IsOwnerPropertyTrue("Oracle_Is_Temporary_Table", "true")
…
]
}
```

IsPropertyEqual

Description

This macro will succeed if the specified property has the specified value.

Prototype

```
IsPropertyEqual( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to 'true' or 'false' to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The property is found and does not have the specified value.
- The property is not found and NotFoundValue is not set to "true".

Deprecation Level

Active

Breaking Changes

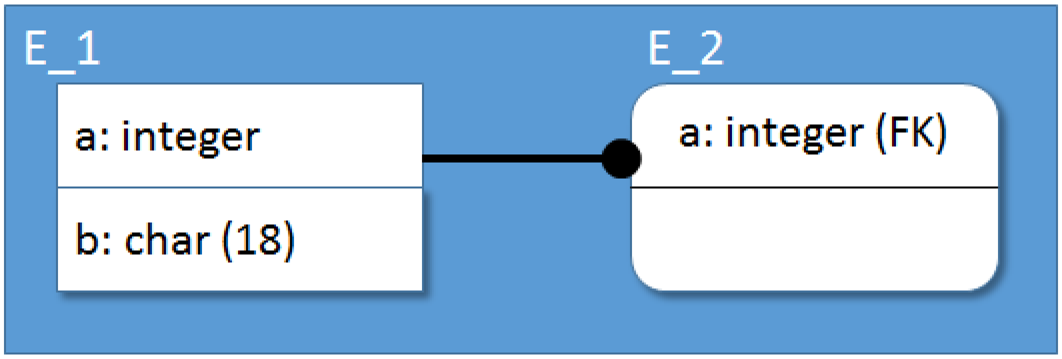
None

Categories

- Property Macros

Sample

Assume the context is the Subject_Area object containing the tables in the following illustration:



Template

```
ForEachReference("Referenced_Entities_Ref")
{
ListSeparator("\n")
Property("Physical_Name") " is "
[IsPropertyEqual("Type", "1") "Independent Entity"]
[IsPropertyEqual("Type", "6") "Dependent Entity"]
}
```

Result

```
E_1 is Independent Entity
E_2 is Dependent Entity
```

IsPropertyEqualFrom

Description

This macro will succeed if the specified property has the specified value on the first object in the context stack with the specified type.

Prototype

```
IsPropertyEqualFrom( ObjectType, Property, Value [, NotFoundValue]
)
```

Parameter	Status	Description

ObjectType	Req	The object type.
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to 'true' or 'false' to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The property is found and does not have the specified value.
- The property is not found and NotFoundValue is not set to "true".

Deprecation Level

Active

Breaking Changes

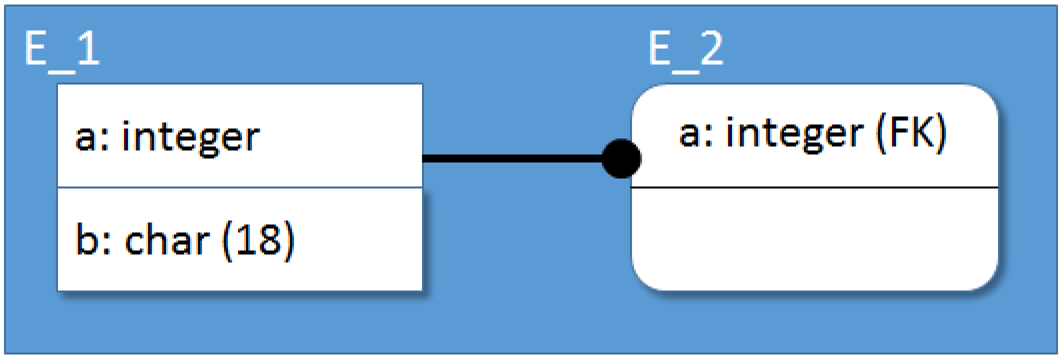
None

Categories

- Property Macros

Sample

Assume that the context stack has the Relationship as the first entry and E_2 as the second:



Template

```
/* Is the relationship identifying? */
@if( IsPropertyEqualFrom("Relationship", "Type", "2" )
{
"It's identifying."
}
```

Result

```
It's identifying.
```

IsPropertyEqualThrough

Description

This macro will succeed if the specified property has the specified value on the object pointed to by the specified scalar reference property on the current context object.

Prototype

```
IsPropertyEqualThrough( Reference, Property, Value [,
NotFoundValue] )
```

Parameter	Status	Description
Reference	Req	The reference property.
Property	Req	The type name of the property to query.

Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to 'true' or 'false' to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The property is found and does not have the specified value.
- The property is not found and NotFoundValue is not set to "true".

Deprecation Level

Active

Breaking Changes

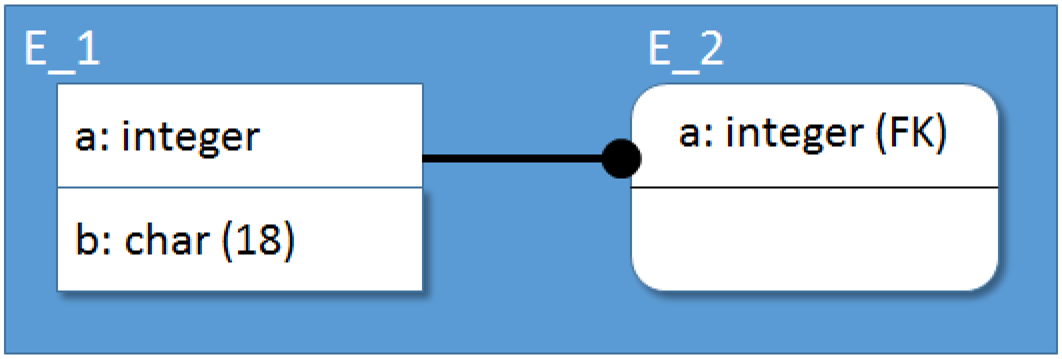
None

Categories

- Property Macros

Sample

Assume that the context is the relationship:



Template

```
/* Is the parent entity an independent entity? */
@if ( IsPropertyEqualThrough("Parent_Entity_Ref", "Type", "1" ) )
{
"It's independent."
}
```

Result

```
It's independent
```

IsPropertyFalse

Description

This macro will succeed if the value in the property is a Boolean value of 'false'. Missing Boolean properties are assumed to be 'false'; use the NotFoundValue if this behavior is not desired.

Prototype

```
IsPropertyFalse( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'false' and the property does not exist, the macro will fail.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property does not exist and NotFoundValue is set.
- The value in the property is not a Boolean.
- The value in the property is not 'false'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the model contains three entities: E_1 is logical only, E_2 is physical only and E_3 is neither.

Template

```
ForEachOfType("Entity")
{
Property("Name") " is "
@if ( IsPropertyFalse("Physical Only") )
{
" present in the logical model."
}
@else
{
" not present in the logical model."
}
}
```

Result

```
E_1 is present in the logical model.
E_2 is not present in the logical model.
E_3 is present in the logical model.
```

IsPropertyFalseFrom

Description

This macro will succeed if the value in the property is a Boolean value of 'false' on the first object in the context stack with the specified type. Missing Boolean properties are assumed to be 'false'; use the NotFoundValue if this behavior is not desired.

Prototype

```
IsPropertyFalseFrom( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'false' and the property does not exist, the macro will fail.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property does not exist and NotFoundValue is set.
- The value in the property is not a Boolean.
- The value in the property is not 'false'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume that the context stack has the Model on it somewhere.

Template

```
/* Are special characters disallowed? */
@if( IsPropertyFalseFrom("Model", "Allow_Special_Characters") )
{
…
}
```

IsPropertyFalseThrough

Description

This macro will succeed if the value in the property is a Boolean value of 'false' on the object pointed to by the specified scalar reference property on the current context object. Missing Boolean properties are assumed to be 'false'; use the NotFoundValue if this behavior is not desired.

Prototype

```
IsPropertyFalseThrough( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'false' and the property does not exist, the macro will fail.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property does not exist and NotFoundValue is set.
- The value in the property is not a Boolean.
- The value in the property is not 'false'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume that the context is a Relationship.

Template

```
/* Is the child entity physical only? */
@if ( IsPropertyFalseThrough("Child_Entity_Ref",
"Is_Physical_Only") )
{
"It's not physical only."
}
```

IsPropertyModified

Description

This macro will test whether any of the specified properties on the context object were modified during the current session.

Prototype

IsPropertyModified(Property1 [, Property2 [, ...]])

Parameter	Status	Description
Property1 � PropertyN	Req	One or more property names.

Result

This macro will fail in the following circumstances:

- None of the specified properties on the context object were modified during the current session.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Template

```
[IsPropertyModified("Parent_Relations_Ref")
Execute("Drop And Create View")
]
```

Result

Statements will be executed for each view that has new parent tables.

IsPropertyNotEqual

Description

This macro will succeed if the specified property does not have the specified value.

Prototype

```
IsPropertyNotEqual( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.

Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to "true" or "false" to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property value equals Value.
- The property is not found and NotFoundValue is 'false' or not specified.

Deprecation Level

Active

Breaking Changes

None

Categories

Property Macros

Sample

Template

```
[IsPropertyNotEqual( "Null Option", "0" ) "not null"]
```

IsPropertyNotEqualFrom

Description

This macro will succeed if the specified property does not have the specified value on the first object in the context stack with the specified type.

Prototype

```
IsPropertyNotEqualFrom( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to "true" or "false" to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property value equals Value.
- The property is not found and NotFoundValue is 'false' or not specified.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Template

```
[IsPropertyNotEqualFrom("Attribute", "Null Option", "0") "not
null"]
```

IsPropertyNotEqualThrough

Description

This macro will succeed if the specified property does not have the specified value on the object pointed to by the specified scalar reference property on the current context object.

Prototype

```
IsPropertyNotEqualThrough( Property, Value [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Value	Req	The value to test for.
NotFoundValue	Opt	This should be set to "true" or "false" to indicate the desired return value if the property is not found. If this is not supplied, the macro will fail in this situation.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property value equals Value.
- The property is not found and NotFoundValue is 'false' or not specified.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the context object is a Key_Group_Member.

Template

```
[IsPropertyNotEqualThrough("Attribute_Ref", "Null Option", "0")
"not null"]
```

IsPropertyNotNull

Description

This macro will succeed or fail based upon the value in the specified property.

Prototype

```
IsPropertyNotNull( Property [, CheckCount] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
CheckCount	Opt	If this is set to 'true', the macro will fail if the property does not have at least one value.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The value in the property is NULL.
- The property has no values and CheckCount is set to 'true'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the current context is a Subject_Area object with at least one member.

Template

```
[IsPropertyNotNull( "Referenced Entities" ) "has members"]
```

Result

```
has members
```

IsPropertyNull

Description

This macro will succeed or fail based upon the value in the specified property.

Prototype

```
IsPropertyNull( Property [, MissingValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
MissingValue	Opt	By default, the macro will succeed if the property does not exist. If this parameter is specified as "false" the macro will fail if the property does not exist.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The value in the property is not NULL.

- The property does not exist and MissingValue is "false".

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the current context is an Attribute object created with in-place editing on the diagram (its Null_Option_Type property has not been set explicitly).

Template

```
[IsPropertyNull( "Null_Option_Type" ) "null"]
```

Result

```
null
```

IsPropertyReordered

Description

This macro will succeed determine if the specified property has been modified such that its values are reordered.

NB: This macro will returns undefined results when either the old or the new copy of the property contains duplicate values.

Prototype

```
IsPropertyReordered( Property [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"existing_only"

By default, the presence of a new value or the deletion of an old value will cause the macro to fail. If this option is specified, then the macro will only check to make sure that the remaining elements are in the same order relative to each other.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The value of the property has not been modified.
- The value of the property has had new elements added or existing elements removed and "existing_only" is not specified.

Deprecation Level

Active

Breaking Changes

None

Categories

- Alter Macros

Sample

Template

```
[IsPropertyReordered("Physical_Columns_Order_Ref")
Execute("Create Entity")
]
```

Result

If the order of the columns in a table has changed, the table will be recreated.

IsPropertyTrue

Description

This macro will succeed if the value in the property is a Boolean value of 'true'.

Prototype

```
IsPropertyTrue( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'true' and the property does not exist, the macro will succeed.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The property does not exist and NotFoundValue is not set.
- The value in the property is not a Boolean.
- The value in the property is not 'true'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume the model contains three entities: E_1 is logical only, E_2 is physical only and E_3 is neither.

Template

```
ForEachOfType("Entity")
{
Property("Name") " is "
@if ( IsPropertyFalse("Is_Logical_Only") )
{
" present in the physical model."
}
@else
{
" not present in the physical model."
}
}
```

Result

```
E_1 is not present in the physical model.
E_2 is present in the physical model.
E_3 is present in the physical model.
```

IsPropertyTrueFrom

Description

This macro will succeed if the value in the property is a Boolean value of 'true' on the first object in the context stack with the specified type.

Prototype

```
IsPropertyTrueFrom( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'true' and the property does not exist, the macro will succeed.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property does not exist and NotFoundValue is not set.
- The value in the property is not a Boolean.
- The value in the property is not 'true'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume that the context stack has the Model on it somewhere.

Template

```
/* Are special characters allowed? */
@if( IsPropertyTrueFrom("Model", "Allow_Special_Characters" ) )
{
…
}
```

IsPropertyTrueThrough

Description

This macro will succeed if the value in the property is a Boolean value of 'true' on the object pointed to by the specified scalar reference property on the current context object.

Prototype

```
IsPropertyTrueThrough( Property [, NotFoundValue] )
```

Parameter	Status	Description
Property	Req	The type name of the property.
NotFoundValue	Opt	If this is set to 'true' and the property does not exist, the macro will succeed.

Result

This macro will fail in the following circumstances:

- The referenced object is not found.
- The required parameters are not supplied.
- The property does not exist and NotFoundValue is not set.
- The value in the property is not a Boolean.
- The value in the property is not 'true'.

Deprecation Level

Active

Breaking Changes

None

Categories

- Property Macros

Sample

Assume that the context is a Relationship.

Template

```
/* Is the child entity physical only? */
@if ( IsPropertyTrueThrough("Child_Entity_Ref",
"Is_Physical_Only") )
{
"It's physical only."
}
```

IterationCount

Description

This evaluates to a count of the iterations performed by the first iterator on the iterator stack.

Prototype

```
IterationCount( [SuccessOnly] )
```

Parameter	Status	Description
SuccessOnly	Opt	By default, the absolute number of iterations is returned. If this is set to 'true' then a count of the iterations where the body was successfully expanded will returned.

Result

This macro will fail in the following circumstances:

- There is no iterator on the iterator stack.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

Assume a model that has two Entity objects in it: E_1 and E_2.

Template

```
ForEachOfType("Entity")
{
ListSeparator("\n")
IterationCount " - " Property("Name")
}
```

Result

```
1 - E_1
2 - E_2
```

Left

Description

This macro evaluates to a substring comprised of the leftmost characters of the specified source string.

Prototype

```
Left( SourceString, Length )
```

Parameter	Status	Description
SourceString	Req	The source string.
Length	Req	The length of the desired substring.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The Length parameter cannot be evaluated to a number greater than or equal to '1'.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Assume the context is the primary key of an Entity object.

Template

```
Switch( Left( Property("Key_Group_Type"), "2" ) )
{
Choose( "PK" )
{
"This is a primary key."
}
Choose( "AK" )
{
"This is an alternate key."
}
Default
{
"This is an inversion entry."
}
Choose( "XX" )
{
/* This block will never execute, because a preceding block
will always evaluate successfully */
}
}
```

Result

```
This is a primary key.
```

Less

Description

This determines if one value is less than another.

If the values are being compared as numbers, it assumes that the parameters passed are numeric values. In these cases, the macro stops reading a parameter when it encounters a character it cannot convert. If no characters are converted, the value is assumed to be zero. For example (assuming "ascii" is not specified):

"123"	123
"123Foo"	123
"Foo"	0

Prototype

```
Less( LeftString, RightString [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
LeftString	Req	The left value in the test.
RightString	Req	The right value in the test.
Option1 � *OptionN*	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"ascii"

By default, the strings are compared as numeric values. For example, "11" would compare as greater than "9".

If this parameter is set to "ascii " the strings are interpreted as case-sensitive literals (ASCII sort). This would cause "11" to compare as less than "9".

"no_case"

If this is set to "no_case" and the values are being compared in an ASCII sort, the strings are compared as strings in a case-insensitive manner. The default behavior is a case-sensitive comparison

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The first string is not less than the second.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Value1", "11")
Set("Value2", "9")
@if( Less( Value("Value1"), Value("Value2") )
{
"Less on a numeric sort\n"
```

```
}
@else
{
"Not less on a numeric sort\n"
}
@if( Less( Value("Value1"), Value("Value2"), "ascii" )
{
"Less on an ASCII sort\n"
}
@else
{
"Not less on an ASCII sort\n"
}
```

Result

```
Not less on a numeric sort
Less on an ASCII sort
```

LessOrEqual

Description

This determines if one value is less than or equal to another.

If the values are being compared as numbers, it assumes that the parameters passed are numeric values. In these cases, the macro stops reading a parameter when it encounters a character it cannot convert. If no characters are converted, the value is assumed to be zero. For example (assuming "ascii" is not specified):

"123"

123

"123Foo"

123

"Foo"

0

Prototype

```
LessOrEqual( LeftString, RightString [, Option1 [, Option2 [, …]]]
)
```

Parameter	Status	Description
LeftString	Req	The left value in the test.
RightString	Req	The right value in the test.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"ascii"

By default, the strings are compared as numeric values. For example, "11" would compare as greater than "9".

If this parameter is set to "ascii " the strings are interpreted as case-sensitive literals (ASCII sort). This would cause "11" to compare as less than "9".

"no_case"

If this is set to "no_case" and the values are being compared in an ASCII sort, the strings are compared as strings in a case-insensitive manner. The default behavior is a case-sensitive comparison

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The first string is not less than or equal to the second.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("Value1", "11")
Set("Value2", "9")
@if( LessOrEqual( Value("Value1"), Value("Value2") )
{
"Less or equal on a numeric sort\n"
}
@else
{
"Greater on a numeric sort\n"
}
@if( Less( Value("Value1"), Value("Value2"), "ascii" )
{
"Less or equal on an ASCII sort\n"
}
@else
{
"Greater on an ASCII sort\n"
}
```

Result

```
Greater on a numeric sort
Less or equal on an ASCII sort
```

ListSeparator

Description

This macro is used inside of an iterator block. It evaluates to an empty string on the first loop of iteration. It evaluates to the value specified in String on subsequent loops. Loops are counted only if the loop block evaluates successfully.

Prototype

```
ListSeparator( String )
```

Parameter	Status	Description
String	Req	The separator string.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

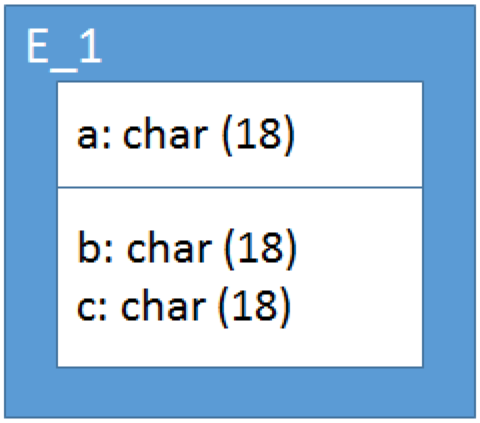
None

Categories

- Iterator Macros

Sample

Assume the current context is the Entity object in the following illustration:



Template

```
ForEachOwnee("Attribute")
{
ListSeparator( ",\n" )
Property( "Physical_Name" )
}
```

Result

```
a,
b,
c
```

Lookup

Description

This is a lookup macro that evaluates to a specified string based upon the value specified.

Replacement mappings are specified as value pairs in the parameter list. Values are compared case-insensitively.

Prototype

```
Lookup( Value [, Source0, Target0
[, Source1, Target1 [,…]]] [, Default] )
```

Parameter	Status	Description
Value	Req	The value to examine.
Source0	Opt	The first actual value to compare.
Target0	Opt	The value to return if Source0 is matched.
Source1 � SourceN	Opt	Subsequent values to compare.
Target1 � TargetN	Opt	The values to return if Source1 through SourceN are matched.
Default	Opt	The value to return if no other match is found.

Result

This macro will fail in the following circumstances:

- Required parameters are not supplied.
- The value is not found in the list of values and no default value is provided.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Lookup(Property("Null_Option_Type"), "0", "Null", "1", "Not Null")
```

LookupProperty

Description

This is a lookup macro that evaluates to a specified string based upon the value actually held in the specified property of the current context object.

Replacement mappings are specified as value pairs in the parameter list. Values are compared case-insensitively.

Prototype

```
LookupProperty( Property [, Source0, Target0
[, Source1, Target1 [,…]]] [, Default] )
```

Parameter	Status	Description
Value	Req	The value to examine.

Source0	Opt	The first actual value to compare.
Target0	Opt	The value to return if Source0 is matched.
Source1 � SourceN	Opt	Subsequent values to compare.
Target1 � TargetN	Opt	The values to return if Source1 through SourceN are matched.
Default	Opt	The value to return if no other match is found.

Result

This macro will fail in the following circumstances:

- Required parameters are not supplied.
- The specified property does not exist and no default value is provided.
- The actual value of the property is not found in the list of values and no default value is provided.

Deprecation Level

Active

Breaking Changes

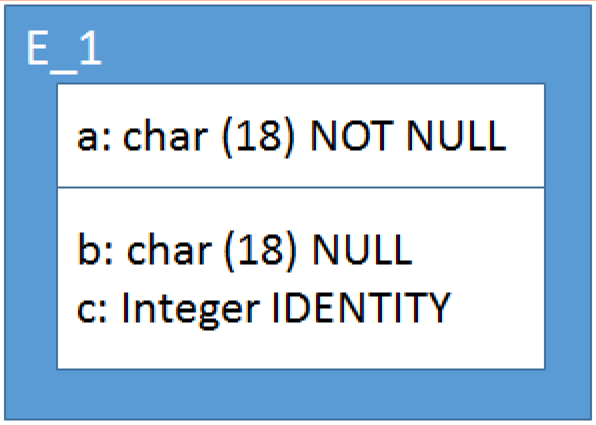
None

Categories

- Property Macros

Sample

Assume the current context is the Entity object in the following illustration:



Template

```
ForEachOwnee("Attribute")
{
ListSeparator( ",\n" )
Property("Physical_Name") " is "
LookupProperty("Null_Option_Type", "0", "null", "1", "not null",
"8", "identity")
}
```

Result

```
a is not null
b is null
c is identity
```

Loop

Description

This iterator loops until a global flag is cleared.

Prototype

```
Loop( Flag )
```

Parameter	Status	Description
Flag	Req	The name of the global flag.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Iterator Macros

Sample

Template

```
SetGlobalFlag("Continue")
Loop("Continue")
{
ListSeparator("\n")
@if ( Greater(IterationCount, "5") )
{
ClearGlobalFlag("Continue")
}
@else
{
IterationCount
}
}
```

Result

```
1
2
3
4
5
```

LowerCase

Description

This macro evaluates to the lower case version of a string.

Prototype

```
LowerCase( SourceString )
```

Parameter	Status	Description
SourceString	Req	The source string.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

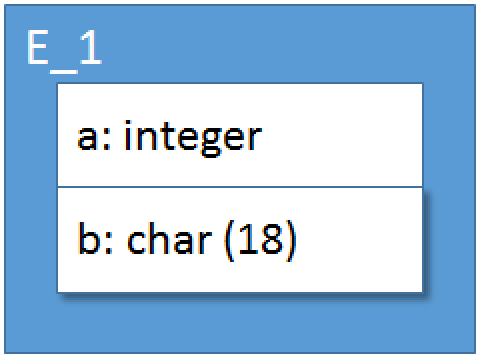
None

Categories

- String Macros

Sample

Assume the context object is E_1 in the following illustration:



Template

```
LowerCase( Property("Name") )
```

Result

```
e_1
```

Mid

Description

This macro evaluates to a substring of the specified source string.

Prototype

```
Mid( SourceString, Start, Length )
```

Parameter	Status	Description
SourceString	Req	The source string.
Start	Req	The zero-based starting position for the substring.
Length	Req	The length of the desired substring. If there are not enough characters in the source string to fulfill the Length specification, the macro will return the characters available. In non-Unicode versions of the product, the Length specification represents bytes. In other words, a character

		represented by a lead-byte and trail-byte would count as two.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The Start parameter is not '0' or greater, or the Length parameter is not '1' or greater.

Deprecation Level

Active

Breaking Changes

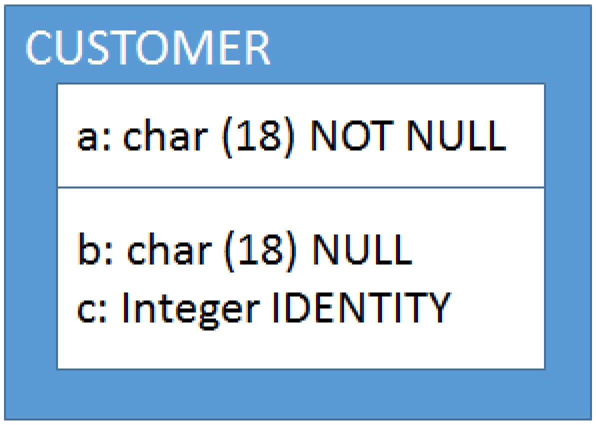
None

Categories

- String Macros

Sample

Assume the context object is the Entity in the following illustration:



Template

```
Mid(Property("Physical_Name", "0", "3")
```

Result

```
CUS
```

Modulo

Description

This predicate tests the modulo value of two numbers. The modulo is the remainder when Left is divided by Right.

Prototype

```
Modulo( Left, Right [, Remainder] )
```

Parameter	Status	Description
Left	Req	The numerator of the division.
Right	Req	The denominator of the division.
Remainder	Opt	By default, the macro succeeds if the remainder is '0'. If this value is specified, the macro succeeds if it matches the remainder.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

In this sample, we want to insert a carriage return every three values. Since we placed the ListSeparator call before the output of the actual value (to avoid a trailing separator), we need to test based upon a modulo value of '1'…for example, "insert the carriage return before the fourth, seventh, and so on, value."

Template

```
SetGlobalFlag("Continue")
Loop("Continue")
{
ListSeparator([Modulo(IterationCount, "3", "1") "\n"])
@if ( Greater(IterationCount, "6") )
{
ClearGlobalFlag("Continue")
}
@else
{
IterationCount
}
}
```

Result

```
123
456
```

NotEqual

Description

This determines if one string is not equal to another.

Prototype

```
NotEqual( LeftString, RightString [, Option] )
```

Parameter	Status	Description

LeftString	Req	The left string in the test.
RightString	Req	The right string in the test.
Option	Opt	By default the comparison is case-sensitive. If this parameter is set to "no_case" the comparison will be done in a case-insensitive manner.

The options available are found in the following table.

"no_case"

By default the comparison is case-sensitive. If this option is set the comparison will be done in a case-insensitive manner.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The two strings are equal.

Deprecation Level

Active

Breaking Changes

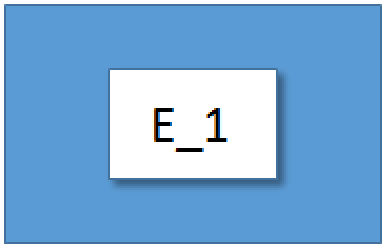
None

Categories

- String Macros

Sample

Assume the context is the following table:



Template

```
@if( NotEqual(Property("Name"), "e_1") )
{
"Name is not exactly 'e_1'.\n"
}
@else
{
"Name is exactly 'e_1'.\n"
}
@if( NotEqual(Property("Name"), "e_1", "no_case") )
{
"Name is not case-insensitively like 'e_1'."
}
```

Result

```
Name is not exactly 'e_1'.
```

ObjectId

Description

This evaluates to the id of the current context object.

For M0 objects (data), this is just an integer that identifies it uniquely.

For M1 objects (metadata), the integer is also uniquely-identifying, but it is a packed combination of an identifier for the source of the metadata and a unique value for the object. For example, metadata defined by erwin Data Modeler will generally have a value of '1' or '0', erwin Data Modeler NSM metadata will have a value of '4', while UDPs will have a value of '9'.

The values for object ids are stable for a given session of erwin Data Modeler. However, they can change between sessions. If you need an identifier that is stable across sessions, use the value retrieved by:

```
Property("Long Id")
```

Prototype

```
ObjectId( [Option] )
```

Parameter	Status	Description
Option	Req	By default the entire id is returned. A component of the id can be retrieved by using one of the values found below.

"product "

Retrieve just the product identifier portion of the id for an M1 object. This has no effect for M0 objects.

"identifier"

Retrieve just the object identifier portion of the id for an M1 object. This has no effect for M0 objects.

Result

This macro will fail if:

- There is no context object.

Deprecation Level

Active

Breaking Changes

None

Categories

Miscellaneous Macros

Sample

Assume the context object is the Model object.

Template

```
ObjectId "\n"
ObjectId("product") "\n"
ObjectId("identifier") "\n"
PushMetaObject
ObjectId "\n"
ObjectId("product") "\n"
ObjectId("identifier")
```

Result

```
1
1
1
1075838978
1
2
```

ObjectType

Description

This evaluates to the type code name of the current context object.

Prototype

```
ObjectType
```

Result

This macro will fail in the following circumstances:

- There is no context object.

Deprecation Level

Active

Breaking Changes

None

Categories

Sample

Assume the current context is an Entity.

Template

```
[Equal(ObjectType, "Entity")
"This is an entity"
]
```

Result

```
This is an entity
```

OnceForObject

Description

This macro will attempt to set a global flag that is a concatenation of the Label parameter and the context object   s id. The macro will fail if this flag is already set. This allows a template to test for a previous emission of a given template for the object.

Prototype

```
OnceForObject( Label [, Option] )
```

Parameter	Status	Description
Label	Req	A label that will be used to distinguish the particular flag being set.
Option	Opt	An option keyword. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"no_set"

If this is present, then only the test of the existence of the flag is performed, but the flag is not set if it is not present.

Result

This macro will fail in the following circumstances:

- There is no context object.
- The flag has been set previously.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros
- Object Macros

Sample

Assume the current context is an Entity.

Template

```
[OnceForObject("Create Table")
"First time"
]
[OnceForObject("Create Table")
"Second time"
]
```

Result

```
First time
```

OwnerProperty

Description

This macro evaluates to the string representation of the specified property on the object owning the current context object. A property is identified by its class name as defined in the metadata for erwin Data Modeler.

Prototype

```
OwnerProperty( PropertyName [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
PropertyName	Req	The type name of the property.
Option1 � *OptionN*	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to the resource identifier string instead of the resource value.

"no_translate"

By default, properties with a data type of 'Resource' will load and all other properties will be run through the default translator for their type. If this value is provided, the macro will evaluate to a raw value for the property. This is a superset of the "no_load" behavior.

"fail_if_empty"

If a parameter with the value of "fail_if_empty" is supplied, the macro will fail if the result is an empty string.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The "fail_if_empty" option is supplied and the result is an empty string.
- The current context object does not have an owning object.

Deprecation Level

Active

Breaking Changes

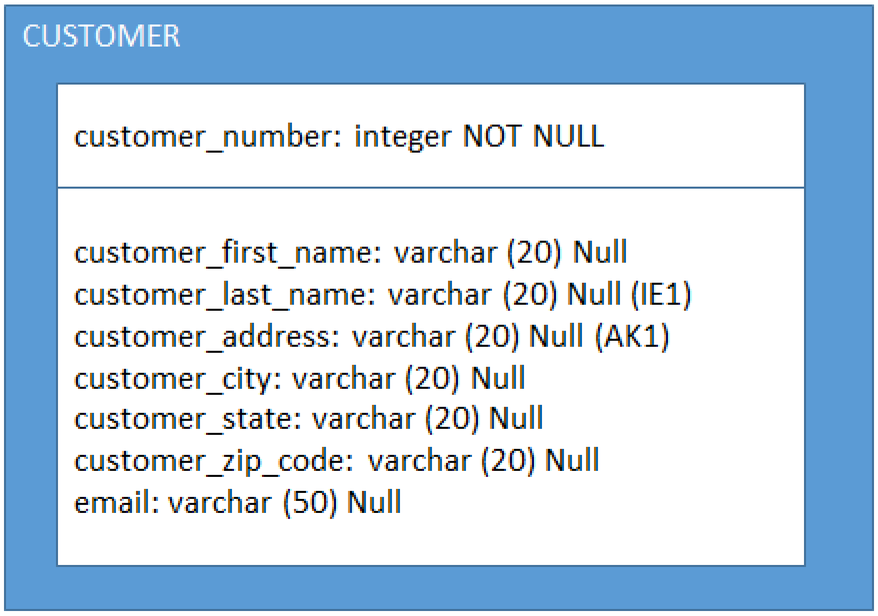
None

Categories

- Property Macros

Sample

Assume that the customer_number column in the following picture is the current context object:


CUSTOMER
customer_number: integer NOT NULL
customer_first_name: varchar (20) Null
customer_last_name: varchar (20) Null (IE1)
customer_address: varchar (20) Null (AK1)
customer_city: varchar (20) Null
customer_state: varchar (20) Null
customer_zip_code: varchar (20) Null
email: varchar (50) Null

Template

```
OwnerProperty( Physical_Name )
```

Result

```
CUSTOMER
```

OwnerQuotedName

Description

This macro evaluates to the string quoted name of the object owning the current context object.

Prototype

```
OwnerProperty( QuoteCharacter )
```

Parameter	Status	Description
QuoteCharacter	Opt	he quote character to be used. If this is not supplied, double quotes will be used..

Result

This macro will fail in the following circumstances:

- The current context object does not have an owning object.

Deprecation Level

Active

Breaking Changes

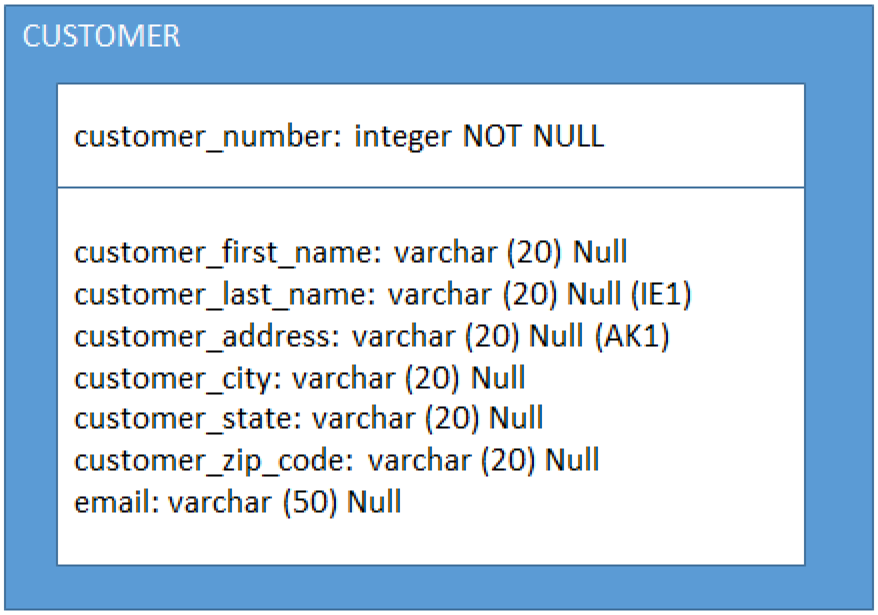
None

Categories

- Property Macros

Sample

Assume that the customer_number column in the following picture is the current context object:


CUSTOMER
customer_number: integer NOT NULL
customer_first_name: varchar (20) Null
customer_last_name: varchar (20) Null (IE1)
customer_address: varchar (20) Null (AK1)
customer_city: varchar (20) Null
customer_state: varchar (20) Null
customer_zip_code: varchar (20) Null
email: varchar (50) Null

Template

```
OwnerQuotedName
```

Result

```
"CUSTOMER"
```

Pad

Description

This macro evaluates to the source string padded to the specified length. The padding will be done by appending Character to the end of the string enough times to reach the desired length. If the source string is longer than the desired length, the entire source string will be returned.

Prototype

```
Pad( SourceString, Length[, Character] )
```

Parameter	Status	Description
SourceString	Req	The source string.
Length	Req	The desired length of the string.
Character	Opt	The character used for padding. If this is not specified, a space will be used.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Pad("Hello", "10") "<\n"
Pad("Hello", "10", "*") "<\n"
Pad("Hello", "3") "<"
```

Result

```
Hello <
Hello*****<
Hello<
```

Pop

Description

This pops the current object from the context stack.

Prototype

```
Pop
```

Result

This macro will fail in the following circumstances:

- Only the anchor object is left on the stack.

Deprecation Level

Active

Breaking Changes

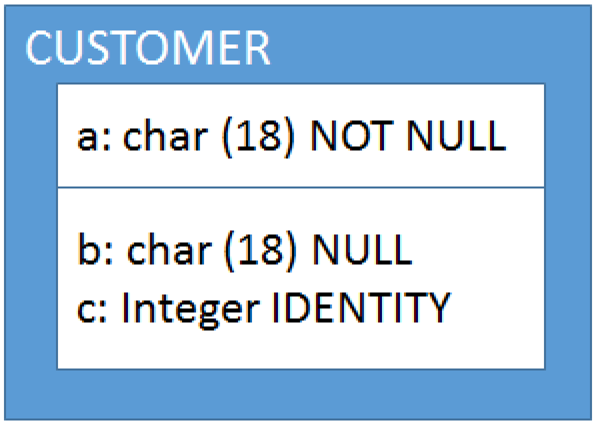
None

Categories

- Stack Macros

Sample

Assume the context object is the Entity in the following illustration:



Template

```
ForEachOwnee("Attribute")
{
ListSeparator("\n")
PushOwner Property("Name") Pop "." Property("Name")
}
```

Result

```
CUSTOMER.a
CUSTOMER.b
CUSTOMER.c
```

Progress_ColumnDecimals

Description

This evaluates to the decimal portion of the data type of the current object.

Prototype

```
Progress_ColumnDecimals
```

Result

This macro will fail in the following circumstances:

- The current context object does not have the Physical_Data_Type property.

Deprecation Level

Active

Breaking Changes

None

Categories

Sample

Assume the current context is an attribute with a data type of DECIMAL(5,3).

Template

```
Progress_ColumnDecimals
```

Result

```
"3"
```

Progress_ColumnFormat

Description

This checks if the column has a Display_Format object attached via the Display_Format_Ref property. If so, its Server_Value property is emitted. If not, and the data type of the column is "CHAR" the macro returns the precision.

Prototype

```
Progress_ColumnFormat
```

Result

This macro will fail in the following circumstances:

- There is no attached 'Display Format' object, the data type is not "CHAR" or there is no precision.

Deprecation Level

Active

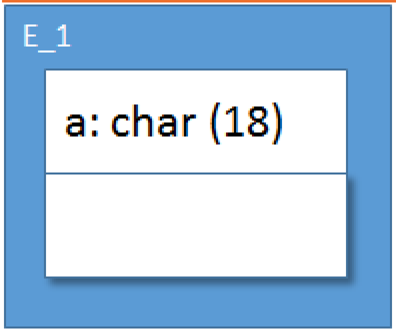
Breaking Changes

None

Categories

Sample

Assume the current context is E_1.a in the following illustration:



Template

```
Progress_ColumnFormat
```

Result

```
"X(18)" Page
```

ProperCase

Description

This macro evaluates to the proper case version of a string, where the initial letter and letters following a space are converted to upper case, and all other letters are converted to lower case.

Prototype

ProperCase(SourceString)

Parameter	Status	Description
SourceString	Req	The source string.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
ProperCase("CUSTOMER")
```

Result

```
Customer
```

Property

Description

This macro evaluates to the string representation of the specified property on the current context object. A property is identified by its class name as defined in the metadata for erwin Data Modeler.

Prototype

```
Property( PropertyName [, Option1 [, Option2 [, …]]] )
```

Parameter	Status	Description
PropertyName	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to the resource identifier string instead of the resource value.

"no_translate"

By default, properties with a data type of 'Resource' will load and all other properties will be run through the default translator for their type. If this value is provided, the macro will evaluate to a raw value for the property. This is a superset of the "no_load" behavior.

"fail_if_empty"

If a parameter with the value of "fail_if_empty" is supplied, the macro will fail if the result is an empty string.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The "fail_if_empty" option is supplied and the result is an empty string.

Deprecation Level

Active

Breaking Changes

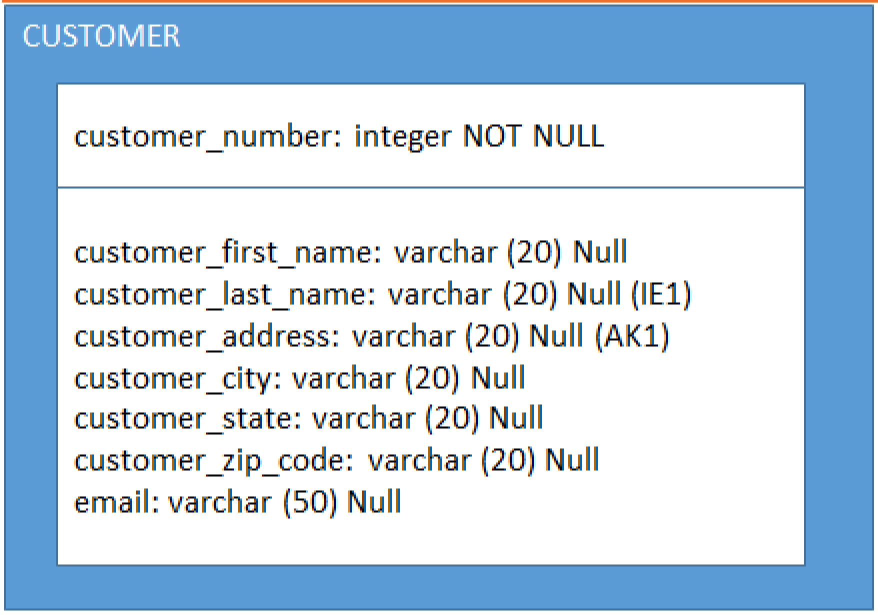
The default behavior of this macro has changed since the erwin Data Modeler 7.1 release. The earlier version defaulted to 'no_translate' and 'no_load'.

Categories

- Property Macros

Sample

The difference in result when using the "no_translate" option is shown in the two example templates below. Assume that the CUSTOMER table in the following picture is the current context object.


CUSTOMER
customer_number: integer NOT NULL
customer_first_name: varchar (20) Null
customer_last_name: varchar (20) Null (IE1)
customer_address: varchar (20) Null (AK1)
customer_city: varchar (20) Null
customer_state: varchar (20) Null
customer_zip_code: varchar (20) Null
email: varchar (50) Null

Template

```
ForEachOwnee("Attribute")
{
Property("Name") " is "
Property("Null_Option_Type", "no_translate") "\n"
}
Result
customer_number is 1
customer_first_name is 0
customer_last_name is 0
…etc.
Template
ForEachOwnee("Attribute")
{
Property("Name") " is "
Property("Null_Option_Type") "\n"
}
```

Result

```
customer_number is NOT NULL
customer_first_name is NULL
```

```
customer_last_name is NULL
…etc.
```

PropertyFrom

Description

This macro evaluates to the string representation of the specified property on the first object in the context stack with the specified type. An object and a property are identified by their class names as defined in the metadata for erwin Data Modeler.

Prototype

```
PropertyFrom( ObjectType, PropertyName [, Option1 [, Option2 [,
…]]] )
```

Parameter	Status	Description
ObjectType	Req	The type of the object desired.
PropertyName	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to the resource identifier string instead of the resource value.

"no_translate"

By default, properties with a data type of 'Resource' will load and all other properties will be run through the default translator for their type. If this value is provided, the macro will evaluate to a raw value for the property. This is a superset of the "no_load" behavior.

"fail_if_empty"

If a parameter with the value of "fail_if_empty" is supplied, the macro will fail if the result is an empty string.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- An object of the specified type is not found on the context stack.
- The "fail_if_empty" option is supplied and the result is an empty string.

Deprecation Level

Active

Breaking Changes

None

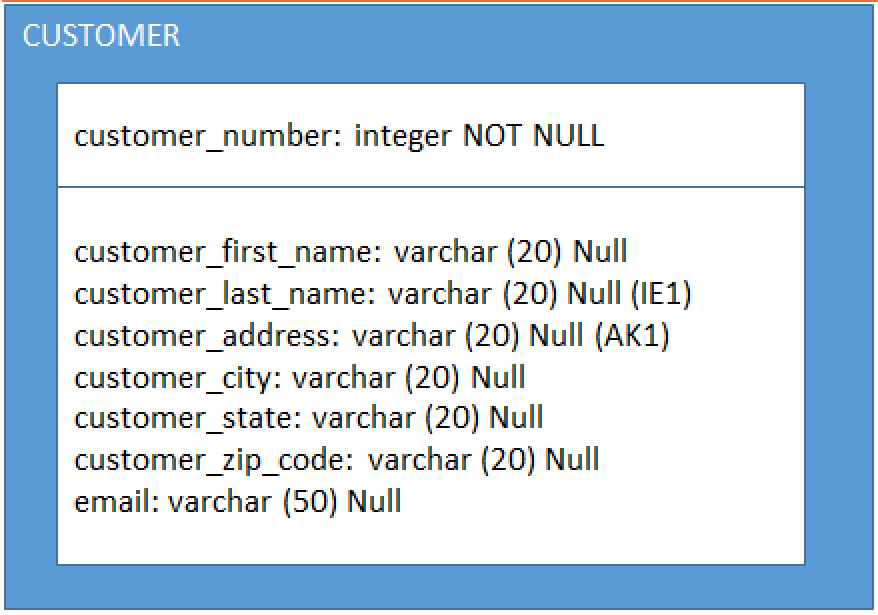
Categories

- Property Macros

Sample

See the sample under the Property macro for an example of the use of the "no_translate" option.

Assume that the CUSTOMER table in the following picture is the current context object:


CUSTOMER
customer_number: integer NOT NULL
customer_first_name: varchar (20) Null
customer_last_name: varchar (20) Null (IE1)
customer_address: varchar (20) Null (AK1)
customer_city: varchar (20) Null
customer_state: varchar (20) Null
customer_zip_code: varchar (20) Null
email: varchar (50) Null

Template

```
ForEachOwnee("Attribute")
{
PropertyFrom("Entity", "Name") "." Property("Name") "\n"
}
```

Result

```
CUSTOMER.customer_number
CUSTOMER.customer_first_name
CUSTOMER.customer_last_name
…etc.
```

PropertyThrough

Description

This macro evaluates to the string representation of the specified property on the object pointed to by the specified scalar reference property on the current context object. An object and a property are identified by their class names as defined in the metadata for erwin Data Modeler.

Prototype

```
PropertyThrough( ReferenceProperty, PropertyName [, Option1 [,
Option2 [, …]]] )
```

Parameter	Status	Description
ReferenceProperty	Req	The reference property to use to locate the desired owning object.
PropertyName	Req	The type name of the property.
Option1 � OptionN	Opt	One or more option keywords. They can appear in any order and are not case-sensitive.

The options available are found in the following table.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to the resource identifier string instead of the resource value.

"no_translate"

By default, properties with a data type of 'Resource' will load and all other properties will be run through the default translator for their type. If this value is provided, the macro will evaluate to a raw value for the property. This is a superset of the "no_load" behavior.

"fail_if_empty"

If a parameter with the value of "fail_if_empty" is supplied, the macro will fail if the result is an empty string.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The referenced object is not found.
- The "fail_if_empty" option is supplied and the result is an empty string.

Deprecation Level

Active

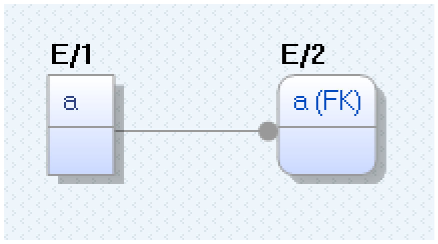
Breaking Changes

None

Categories

Sample

Assume the context object is the Relationship in the following illustration:



Template

```
PropertyThrough("Parent_Entity_Ref", "Name" )
```

Result

```
E_1
```

PropertyValueCount

Description

This macro evaluates to the string representation of number of values in the property, zero if the property is null or missing.

Prototype

```
PropertyValueCount( Property )
```

Parameter	Status	Description
Property	Req	The type name of the property.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

Sample

Assume the context object is a Subject_Area with three Entity objects in it.

Template

```
PropertyValueCount("Referenced_Entities_Ref")
```

Result

```
3
```

PropertyWithDefault

Description

This macro evaluates to the string representation of the specified property on the current context object. If the property does not evaluate successfully, a default value is returned.

Prototype

```
PropertyWithDefault( Property, Default [, Option1[, Option2 [,
…]]] )
```

Parameter	Status	Description

Property	Req	The type name of the property.
Default	Req	The default value.
Option1- OptionN	Opt	One or more option keywords. They can appear in any order.

The options available are found in the following table.

"no_load"

By default, properties with the data type of 'Resource' will load and the macro will evaluate to the loaded value. If this value is provided, properties with this data type will evaluate to a resource identifier string.

"no_translate"

By default, properties with a data type of 'Resource' will load and all other properties will be run through the default translator for their type. If this value is provided, the macro will evaluate to a raw value for the property. This overrides the "no_load" option.

Result

This macro will fail in the following circumstances:

The required parameters are not supplied.

Deprecation Level

Active

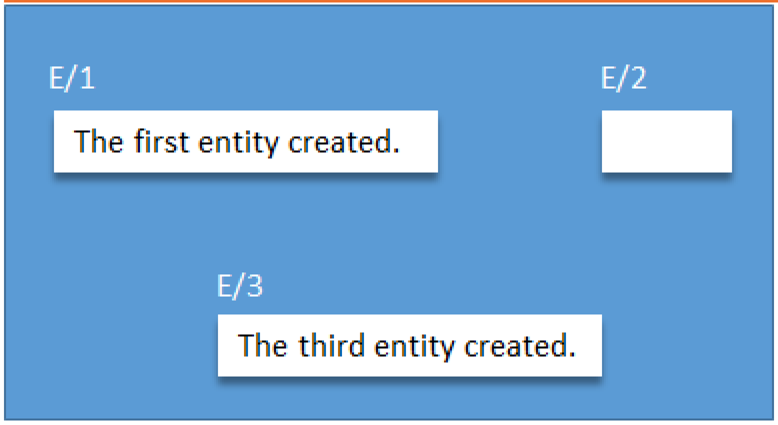
Breaking Changes

The default behavior of this macro has changed since the erwin Data Modeler r7.1 release. The earlier version defaulted to 'no_translate' and 'no_load'.

Categories

Sample

Assume the model in the following illustration, which is shown in Definition Display Level:



Template

```
ForEachOfType("Entity")
{
ListSeparator("\n")
Property("Name") "\t"
PropertyWithDefault("Definition", "<No definition>")
}
```

Result

```
E/1 The first entity created.
E/2 <No definition>
E/3 The third entity created.
```

PushFKViewRelationship

Description

This macro will push the contributing Relationship object if the current context object is a Key_Group on a View object.

Prototype

```
PushFKViewRelationship
```

Result

This macro will fail in the following circumstances:

- The current object is not a foreign key Key_Group on a View.

Deprecation Level

Active

Breaking Changes

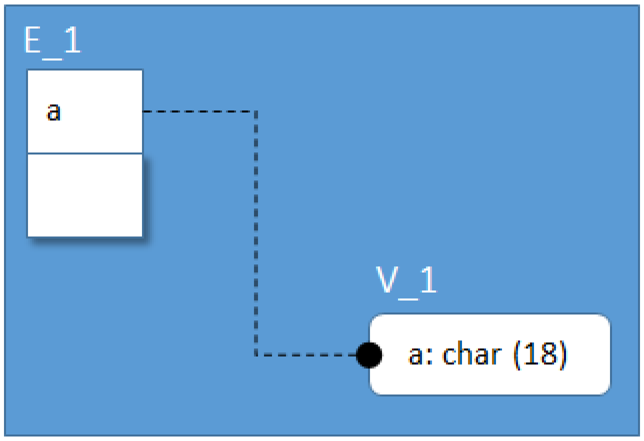
None

Categories

- Stack Macros

Sample

Assume that the current context is the foreign key group created in V_1 in the following illustration:



Template

```
PushFKViewRelationship
Property("Name")
```

Result

```
"E_1 R_1 V_1"
```

PushNewImage

Description

This macro is used when processing objects modified in the model, for example, during Alter Script processing in erwin Data Modeler. If the current object is a phantom object representing a previous state of the object, this macro will push the actual, current image of the object onto the context stack.

Prototype

```
PushNewImage
```

Result

This macro will fail in the following circumstances:

- The current process does not supply the template engine a previous image of the model.
- The current context object is not a phantom object representing a previous image of the object.
- The object no longer exists in the actual model.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Assume that the Entity object CUSTOMER was renamed to CUST during the current session.

Template

```
/* With these parameters, the ForEachOfType iterator will push the
OLD image onto the context stack */
ForEachOfType("Entity", "modified")
{
"execute sp_rename '" Property("Physical_Name") "', '"
PushNewImage Property("Physical_Name") Pop "', 'OBJECT'\ngo"
}
```

Result

```
execute sp_rename 'CUSTOMER', 'CUST', 'OBJECT'
go
```

PushOldImage

Description

This macro is used when processing objects modified in the model, for example, during Alter Script processing in erwin Data Modeler. If the current object is a real object in the model (i.e., not a phantom object) this macro will push the previous image of the object onto the context stack.

Prototype

```
PushOldImage
```

Result

This macro will fail in the following circumstances:

- The current process does not supply the template engine a previous image of the model.
- The current context object is not a real object.
- The previous image of the model does not contain a previous image of the object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Assume the current context is an Entity object that was renamed from CUSTOMER to CUST during the session.

Template

```
"execute sp_rename '" PushOldImage Property("Physical_Name") Pop
"', '" Property("Physical_Name") "', 'OBJECT'\ngo"
```

Result

```
execute sp_rename 'CUSTOMER', 'CUST', 'OBJECT'
go
```

PushOwner

Description

This pushes the owner of the current context object onto the context stack.

> this refers to the owning object, not the value of the DB Owner property.

Prototype

```
PushOwner
```

Result

This macro will fail in the following circumstances:

- The current context object has no owner.

Deprecation Level

Active

Breaking Changes

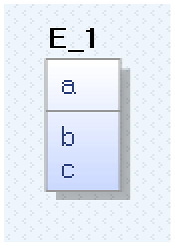
None

Categories

- Stack Macros

Sample

Assume that the context object is E_1 in the following illustration:



```
PushOwner Property( "Name" ) Pop
```

Template

```
ForEachOwnee("Attribute")
{
ListSeparator("\n")
[PushOwner Property("Name") "." Pop]Property("Name")
}
```

Result

```
E_1.a
E_1.b
E_1.c
```

PushReference

Description

This pushes the object referenced by the specified property of the current context object onto the context stack.

Prototype

```
PushReference( Property )
```

Parameter	Status	Description
Property	Req	The type name of the property.

Result

This macro will fail in the following circumstances:

- The reference property does not exist.
- The referenced object does not exist.

Deprecation Level

Active

Breaking Changes

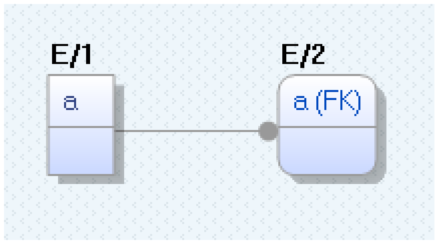
None

Categories

- Stack Macros

Sample

Assume the context object is the Relationship in the following illustration:



Template

```
PushReference("Parent_Entity_Ref")
Property("Name")
Pop
```

Result

```
E_1
```

PushTopLevelObject

Description

This macro will push the top level object for the current context object onto the stack. The top level object is defined as the object for which a CREATE statement would be executed in SQL.

Prototype

```
PushTopLevelObject
```

Result

This macro will fail in the following circumstances:

- The current context object is not represented in the database.

Deprecation Level

Active

Breaking Changes

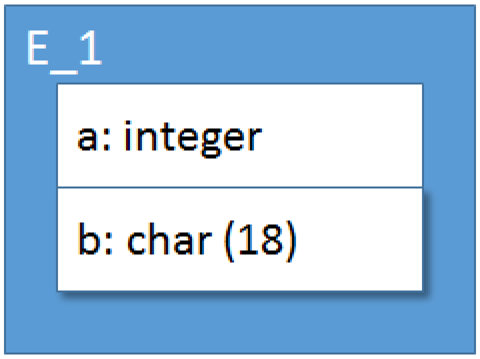
None

Categories

- Stack Macros

Sample

Assume that the Attribute object 'a' is the current context object.



Template

```
PushTopLevelObject
Property("Name")
```

Result

```
"E_1"
```

QuotedName

Description

This macro retrieves a name property from the current object and, based upon the current settings in the FE Option Set, quotes the name. For objects having both the 'Physical Name' and the 'Name' properties, the 'Physical Name' property will be read first. If that fails, the 'Name' property will be used. For all other objects, the 'Name' property will be used.

This macro is sensitive to the current FE Option Set. If quoting of names is disabled in the option set, the quotes will not be emitted.

Prototype

```
QuotedName( [QuoteCharacter] )
```

Parameter	Status	Description
QuoteCharacter	Opt	The quote character to be used. If this is not supplied, double quotes will be used.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

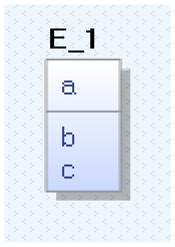
None

Categories

- Property Macros

Sample

Assume the context object is E_1 in the following illustration, and the current FE Option Set has quoted naming enabled:



Template

```
QuotedName
```

Result

```
"E_1"
```

QuotedNameThrough

Description

This macro retrieves a name property from the object pointed to by the specified scalar reference property on the current context object and, based upon the current settings in the

FE Option Set, quotes the name. For objects having both the 'Physical Name' and the 'Name' properties, the 'Physical Name' property will be read first. If that fails, the 'Name' property will be used. For all other objects, the 'Name' property will be used.

This macro is sensitive to the current FE Option Set. If quoting of names is disabled in the option set, the quotes will not be emitted.

Prototype

```
QuotedNameThrough( ReferenceProperty, [QuoteCharacter] )
```

Parameter	Status	Description
ReferenceProperty	Req	The reference property to use to locate the desired owning object.
QuoteCharacter	Opt	The quote character to be used. If this is not supplied, double quotes will be used.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

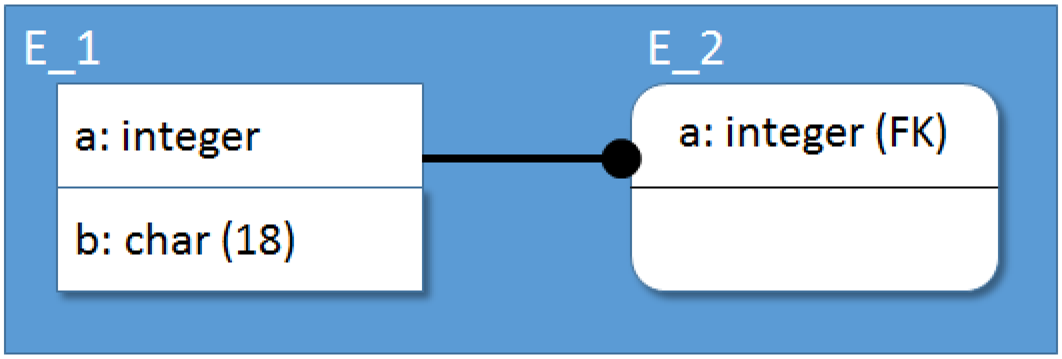
None

Categories

- Property Macros

Sample

Assume the context object is the Relationship object in the following illustration, and the current FE Option Set has quoted naming enabled:



Template

```
QuotedNameThrough("Parent_Entity_Ref")
```

Result

```
"E_1"
```

Remove

Description

This removes the specified variable.

Prototype

```
Remove( VariableName )
```

Parameter	Status	Description
ValriableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable does not exist.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("MyValue", "1")
["The value is now: " Value("MyValue")]
Remove("MyValue")
["The value is now: " Value("MyValue")]
```

Result

```
The value is now: 1
```

RemoveInteger

Description

This removes the specified variable.

Prototype

```
RemoveInteger( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable does not exist.

Deprecation Level

Deprecated

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetInteger("MyValue", "1")
["The value is now: " Integer("MyValue")]
RemoveInteger("MyValue")
["The value is now: " Integer("MyValue")]
```

Result

```
The value is now: 1
```

RemoveString

Description

This removes the specified variable.

Prototype

```
RemoveString( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable does not exist.

Deprecation Level

Deprecated

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetString("MyValue", "One")
["The value is now: " String("MyValue")]
RemoveInteger("MyValue")
["The value is now: " String("MyValue")]
```

Result

```
The value is now: One
```

Repush

Description

This pushes an object that already exists on the context stack onto the stack again.

Prototype

```
Repush( Depth )
```

Parameter	Status	Description
Depth	Req	The zero-based depth of the object in the context stack. The value provided must be greater than zero (zero indic-ates the current context object) and less than the size of the stack.

		The special value of "anchor" indicates that the anchor object should be pushed regardless of the stack's depth.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- The Depth parameter does not evaluate to a number greater than or equal to '1' and is not the special value of "anchor".
- The depth specified by Depth is greater than the current size of the context stack.

Deprecation Level

Active

Breaking Changes

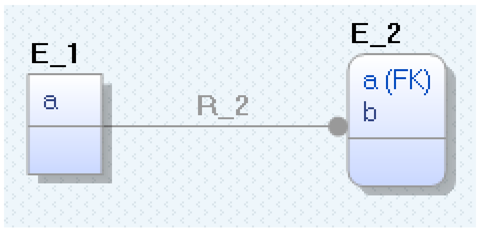
None

Categories

- Stack Macros

Sample

Assume the context object is the Relationship object in the following illustration. For sake of simplicity in the example, the template assumes that there is only one relation coming into E_2.



Template

```
PushReference("Child_Entity_Ref")
ForEachOwnee("Attribute")
{
ListSeparator("\n")
@if( IsPropertyNull("Parent_Attribute_Ref")
{
Property("Name") " is owned."
}
@else
{
Property("Name") " is migrated by "
Repush("2") Property("Name") Pop "."
}
}
```

Result

```
a is migrated by R_2.
b is owned.
```

RepushType

Description

This pushes an object that already exists on the context stack onto the stack again based upon the class type.

Prototype

```
Repush( Type )
```

Parameter	Status	Description
Type	Req	The type name of the object desired.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.
- An object of the specified type was not found on the context stack.

Deprecation Level

Active

Breaking Changes

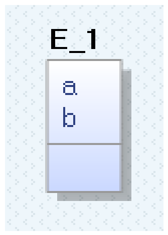
None

Categories

- Stack Macros

Sample

Assume the context object is E_1 in the following illustration:



Template

```
ForEachOwnee("Key_Group")
{
ListSeparator("\n")
ForEachOwnee("Key_Group_Member")
{
ListSeparator("\n")
Property("Name") " participates in a key group in "
RepushType("Entity") Property("Name") Pop
}
}
```

Result

```
a participates in a key group in E/1
b participates in a key group in E/1
```

Right

Description

This macro evaluates to a substring comprised of the rightmost characters of the specified source string.

Prototype

```
Right( SourceString, Length )
```

Parameter	Status	Description
SourceString	Req	The source string.
Length	Req	The length of the desired substring.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.
- The Length parameter cannot be evaluated to a number greater than '1'.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Right("CUSTOMER", "3")
```

Result

```
MER
```

Separator

Description

This inserts the specified separator between a set of values. The separator will be inserted between any two values that are not empty strings.

Prototype

```
Separator( Separator, Value1 [, Value2 [,…] ] )
```

Parameter	Status	Description
Separator	Req	The separator to insert.
Value1	Req	The first value in the set.
Value2 � ValueN	Opt	The second through Nth value in the set.

Result

This will fail if:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Separator(",", "", "A", "B", "", "C")
```

Result

```
A,B,C
```

Set

Description

This defines the specified variable and establishes its initial value. If the variable already exists, its value is modified.

Prototype

```
Set( VariableName, InitialValue )
```

Parameter	Status	Description
VariableName	Req	The name of the variable.
InitialValue	Req	The initial value of the variable.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("My Counter", "1")
Value("My Counter")
```

Result

```
1
```

SetGlobalFlag

Description

This macro will set a flag. Flags are identified by a name and are not case-sensitive. Global flags are set for the entire duration of the template evaluation unless cleared.

Prototype

```
SetGlobalFlag( Flag )
```

Parameter	Status	Description
Flag	Req	The name of the flag to set.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

Stack Macros

Sample

Template

```
/* Flag that the table is being recreated */
SetGlobalFlag( Property("Name") "created")
…
@if ( IsGlobalFlagSet( Property("Name") "created" ) )
{
"/* ALTER not necessary */"
}
@else
{
"ALTER TABLE " …
}
```

Result

```
/* ALTER not necessary */
```

SetInteger

Description

This defines the specified variable and establishes its initial value. If the variable already exists, its value is modified.

Prototype

```
SetInteger( VariableName, InitialValue )
```

Parameter	Status	Description
ValriableName	Req	The name of the variable.
InitialValue	Req	The initial value for the variable.

Result

This macro always succeeds.

Deprecation Level

Deprecated

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetInteger("My Counter", "1")
Integer("My Counter")
```

Result

```
1
```

SetLocalFlag

Description

This macro will set a flag tied to the current context object. Flags are identified by a name and are not case-sensitive. When a context object is pushed down on the stack, its flags are not longer visible by the IsLocalFlagSet macro. When a context object is popped off the stack, its flags are discarded.

Prototype

```
SetLocalFlag( Flag [, Depth] )
```

Parameter	Status	Description
Flag	Req	The name of the flag to set.
Depth	Opt	This will set the named local flag on the stack entry Depth levels above the current entry.

Result

This macro will fail in the following circumstances:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

- Stack Macros

Sample

Template

```
SetLocalFlag( "MyFlag" )
[IsLocalFlagSet("MyFlag") "Flag was set #1.\n"]
PushOwner
[IsLocalFlagSet("MyFlag") "Flag was set #2.\n"]
Pop
[IsLocalFlagSet("MyFlag") "Flag was set #3."]
```

Result

```
Flag was set #1.
Flag was set #3.
```

SetString

Description

This defines the specified variable and establishes its initial value. If the variable already exists, its value is modified.

Prototype

```
SetString( VariableName, InitialValue )
```

Parameter	Status	Description
ValriableName	Req	The name of the variable.
InitialValue	Req	The initial value of the variable.

Result

This macro always succeeds.

Deprecation Level

Deprecated

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetString("MyValue ", "One")
String("MyValue ")
```

Result

```
One
```

ShouldGenerate

Description

This macro tests whether the current context object should generate, based upon both its properties and the current FE Option Set.

Prototype

```
ShouldGenerate
```

Result

This macro will fail in the following circumstances:

- The Generate property is associated with the object type by the metadata, but not set.
- The Is_Logical_Only property is set on the object.
- The object is built-in   the Built_In_Id property is set on the object.
- The object is filtered out by the current FE Option Set.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Assume the model contains two Entity objects: E_1 is marked logical-only and E_2 is not.

Template

```
ForEachOfType("Entity")
{
ListSeparator("\n")
[ShouldGenerate
"Generate " Property("Name")
]
}
```

Result

```
E_2
```

String

Description

This retrieves the value in the specified variable previously set with SetString.

Prototype

```
String( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable is not found.

Deprecation Level

Deprecated

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
SetString("MyValue ", "One")
String("MyValue ")
```

Result

```
One
```

Substitute

Description

The macro evaluates to a string where one or more substrings are replaced with a new value.

Prototype

```
Substitute( SourceString, NewValue, OldValue1 [, OldValue2 [,…]] )
```

Parameter	Status	Description
SourceString	Req	The source string.
NewValue	Req	The new value to place into the string.
OldValue1	Req	The first substring to replace.
OldValue2- OldValueN	Opt	Additional substrings to replace.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

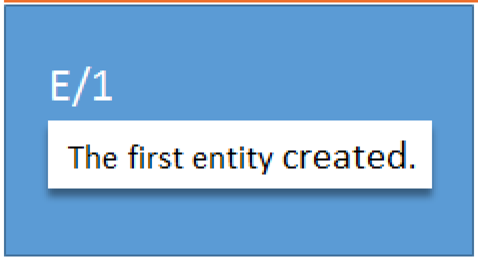
None

Categories

- String Macros

Sample

Assume the context object is E_1 in the following illustration:



Template

```
/* Replace all spaces and tabs in the decription with underscores
*/
Substitute( Property("Description"), "_", " ", "\t" )
```

Result

```
The_first_entity_created.
```

Switch

Description

This macro, in conjunction with the Choose and Default macros, tests a predicate against a range of values and executes a block when a match is found.

Prototype

```
Switch( Predicate ) {}
```

Parameter	Status	Description
Predicate	Req	This parameter evaluates to the value that is to be tested.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- Miscellaneous Macros

Sample

Assume the context is the primary key of an Entity object.

Template

```
Switch( Left( Property("Key_Group_Type"), "2" ) )
{
Choose( "PK" )
{
"This is a primary key."
}
Choose( "AK" )
{
"This is an alternate key."
}
Default
{
"This is an inversion entry."
}
Choose( "XX" )
{
/* This block will never execute, because a preceding block
will always evaluate successfully */
}
}
```

Result

```
This is a primary key.
```

TableHasFilteredIndex

Description

This macro determines if the context Entity owns a KeyGroup with a WHERE clause marked to be generated.

Prototype

```
TableHasFilteredIndex
```

Result

This macro will fail in the following circumstances:

- The context object is not an Entity.
- The specified owned object cannot be found.

Deprecation Level

Active

Breaking Changes

None

Categories

None

Sample

Template

```
TableHasFilteredIndex
```

Result

Trim

Description

This macro evaluates to the source string trimmed of leading and trailing characters.

Prototype

```
Trim( SourceString [, TrimSet] )
```

Parameter	Status	Description
SourceString	Req	The source string.
TrimSet	Opt	The set of characters to trim. If this is not supplied, spaces and tabs are trimmed.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Trim(" A string with extra spaces. ") "\n"
Trim(">>>A string with unwanted characters.<<<","<>")
```

Result

```
A string with extra spaces.
A string with unwanted characters.
```

UpperCase

Description

This macro evaluates to the upper case version of a string.

Prototype

```
UpperCase( SourceString )
```

Parameter	Status	Description
SourceString	Req	The source string.

Result

This macro will fail in the following circumstances:

- The required parameters are not passed in.

Deprecation Level

Active

Breaking Changes

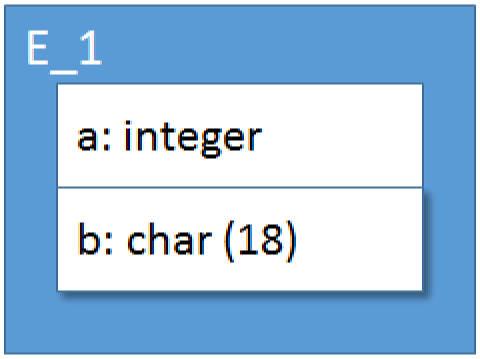
None

Categories

- String Macros

Sample

Assume the context object is E_1 in the following illustration:



Template

```
ForEachOwnee("Attribute")
{
ListSeparator(", ")
UpperCase(Property("Name"))
}
```

Result

```
A, B
```

Value

Description

This retrieves the value in the specified variable.

Prototype

```
Value( VariableName )
```

Parameter	Status	Description
VariableName	Req	The name of the previously defined variable.

Result

This macro will fail in the following circumstances:

- The specified variable is not found.

Deprecation Level

Active

Breaking Changes

None

Categories

- String Macros

Sample

Template

```
Set("My Counter", "1")
Value("My Counter")
```

Result

```
1
```

Forward Engineering Macros

These macros are usable only in the context of the script generation in Forward Engineering and Alter Script.

This section contains the following topics

ActivateDataPreservation

Description

This macro registers an Entity or Attribute object for the Data Preservation mechanism. The context object is added in the Data Preservation objects list. For the context object, data is prepared in the background to be displayed in the Data Preservation Options dialog.

Prototype

```
FE::ActivateDataPreservation
```

Result

The macro will fail if:

- The context object is not an Entity or Attribute.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
[IsAlterScriptX

/* Add this entity in the data preservation list. */

[ActivateDataPreservation]

…

]
```

Result

The entity's data will now be preserved across the alter process.

Description

This macro inserts a bucket identification token into the script. This token is interpreted by the script processing engine for sorting purposes and then removed. See the document, Editing Forward Engineering Templates.pdf, for a complete description of buckets.

Prototype

```
FE::Bucket( BucketNumber )
```

Parameter	Status	Description
BucketNumber	Req	The bucket number.

Result

This macro will fail if:

- The required parameters are not supplied.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
FE::Bucket("90")
```

…

Result

There will be no visible effect in the script generated by the FE engine, as the bucket tokens are stripped out of the final output. If a template is expanded outside of the FE engine, the output will appear as follows, where 90 is the bucket number.

```
@@*B=90B*@@
```

DataPreservationOption

Description

This macro is used to get a specific user selection from the Data Preservation Options dialog. The Data Preservation is enabled when there is a drop and re-create of an entity for the Alter Scripts.

The list of options that can be tested is found in the following table.

WhereClause

Evaluates to the state of the user-specified "Where Condition".

DropTempTable

Evaluates to the state of the "Drop Temp Table" check box.

PreserveData

Evaluates to the inverse value of Do NOT Preserve Data check box.

IsDropRecreate

Evaluates to the value of Force DROP/re-CREATE Table check box.

IsAlterRequired

Evaluates to the inverse value of Force DROP/re-CREATE Table check box.

RegisterEntity

This option sets the template flag indicating that the query is generated to transfer the data from the temp table to the new modified table.

Prototype

```
FE::DataPreservationOption( OptionName )
```

Parameter	Status	Description
OptionName	Req	The name of the option.

Result

See the above table for explanations of return values.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Description

This emits the erwin Data Modeler-generated triggers for the context entity.

Prototype

```
FE::EmitERwinGeneratedTriggers
```

Result

This macro will fail if:

- The erwin Data Modeler triggers cannot be emitted.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
FE::EmitERwinGeneratedTriggers
```

Result

```
create procedure erwin_raise_except(err int,msg varchar(255))

raise exception err,0,msg;

end procedure;

CREATE TRIGGER tD_E_1 DELETE ON E_1
```

```
…etc.
```

EndOfStatement

Description

This macro inserts an end-of-statement token into the script. This token is interpreted by the script processing engine and then removed. This should not be confused with macros like %DBMSDelim that insert end of statement delimiters interpreted by the database.

Prototype

```
FE::EndOfStatement
```

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
FE::EndOfStatement
```

Result

There will be no visible effect in the script generated by the FE engine, as the end of statement markers are stripped out of the final output. If a template is expanded outside of the FE engine, the output will appear as follows.

```
@@*EOS*@@
```

IsAlterScriptGeneration

Description

This macro succeeds if the process running the script is Alter Script Generation.

Prototype

```
FE::IsAlterScriptGeneration
```

Result

This macro will fail in the following circumstances:

- The template is not being expanded as part of Alter Script.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
[ FE:: IsAlterScriptGeneration

/* Generate alter script-specific stuff */

]
```

Result

The Alter Script-specific template will be emitted.

IsEntityInSubjectArea

Description

This macro succeeds if current context object is located in the Referenced_Entities_Ref property of the current Subject_Area.

Prototype

```
FE::IsAlterScriptGeneration
```

Result

This macro will fail in the following circumstances:

- The current context object is not located in the Referenced_Entities_Ref property of the current Subject_Area.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
[ FE::IsEntityInSubjectArea

/* Only do this if the entity is in the current subject area */

]
```

IsLastColumn

Description

This predicate is used by Alter Script processing to determine if the insertion of columns can be done with an ALTER statement, or if the table must be dropped and recreated.

It determines if new Attribute objects have been added to the Entity object owning the Attribute object that is the current context object during the current session.

If new Attribute objects were added, then the macro checks if subsequent Attribute objects were also newly added. The current FE Option Set is used to determine what ordering is being used.

If no Attribute objects were added, the macro returns tests if the current context is the last Attribute based on the settings of the current FE Option Set.

Prototype

```
FE::IsLastColumn
```

Result

This macro will fail in the following circumstances:

- Attribute objects were added, but there are Attribute objects later in the current sort order that were not newly added.
- No Attribute objects were added, but the current context object is not the last one.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
[FE::IsLastColumn

/* We can use ALTER, so emit construct the statement */
"ALTER TABLE " …

]
```

Result

This particular clause will be emitted.

IsModified

Description

This macro succeeds functions as does the IsModified macro, except that properties not represented in the database are not considered when testing for modifications.

Prototype

```
FE::IsModified
```

Result

This macro will fail in the following circumstances:

- The current context object is not modified, or is only modified in properties that are not represented in the database.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

The following block will only execute if the object is modified in a database property other than comment.

Template

```
FE::IsModified("Comment")
```

IsSchemaGeneration

Description

This macro succeeds if the process running the script is Schema Generation.

Prototype

```
FE::IsSchemaGeneration
```

Result

This macro will fail in the following circumstances:

- The template is not being expanded as part of Schema Generation.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
[ FE::IsSchemaGeneration

/* Generate forward engineering script-specific stuff */

]
```

Result

The Forward Engineering-specific template will be emitted.

NextExistingColumn

Description

This macro is used by the Alter Script mechanism for databases that allow the creation of a column in the middle of a table. It evaluates to the name of the next pre-existing Attribute object after a newly-created Attribute. The current FE Option Set is used to determine the sort order.

Prototype

```
FE::NextExistingColumn
```

Result

This macro will fail in the following circumstances:

- There is not a succeeding pre-existing Attribute object.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
…

"ALTER TABLE "

…

["\r\n" "BEFORE " FE::NextExistingColumn]

…
```

Result

This particular clause will be emitted.

Option

Description

This predicate tests whether the specified option is turned on in the current FE Option Set.

The list of options that can be tested is found in the following table. For a description of what these options control, consult the Help system for the Forward Engineering dialogs.

AKConstraintInAlter AKConstraintInCreate

AlterStatements CachedView

CachedViewCreateFunction CachedViewCreateFunctionSynonym

CachedViewCreateMacro CachedViewCreateOption

CachedViewCreateProcedure CachedViewCreateProcedureSynonym

CachedViewCreateSynonym CachedViewCreateTrigger

CachedViewDropFunction CachedViewDropFunctionSynonym

CachedViewDropMacro CachedViewDropOption

CachedViewDropProcedure CachedViewDropProcedureSynonym

CachedViewDropSynonym CachedViewDropTrigger

CachedViewLOBStorage CachedViewPartitions

CachedViewPhysicalStorage CachedViewPostScript

CachedViewPreScript CachedViewUsingIndexStorage

Column ColumnCheckConstraint

ColumnCompress ColumnCreatePrivilege

ColumnDefaultValue ColumnLabel

ColumnLOBStorage ColumnLogicalName

ColumnPhysicalOrder ColumnTitle

ColumnUseDomain Comments

ConstraintFormat ConstraintName

ConstraintState Create

Create CreateAggregate

CreateAKIndex CreateApplicationRole

CreateAssembly CreateAssemblySynonym

CreateAsymmetricKey CreateAuthorization

CreateBufferpool CreateCachedView

CreateCachedViewAlternateKeyIndex CreateCachedViewIndexOption

CreateCachedViewIndexPartitions CreateCachedViewIndexPhysicalStorage

CreateCachedViewInversionEntryIndex CreateCast

CreateCertificate CreateCluster

CreateClusterIndex CreateClusterIndexPhysicalStorage

CreateCollection CreateCredential

CreateDatabase CreateDatabaseLink

CreateDatabasePrivilege CreateDatabaseRole

CreateDatabaseTrigger CreateDBPrivilege

CreateDefault CreateDirectory

CreateDiskgroup CreateDomain

CreateFKConstraint CreateFKIndex

CreateFulltextCatalog CreateFulltextIndex

CreateFunctionSynonym CreateHashIndex

CreateIEIndex CreateIndex

CreateIndexOption CreateLibrary

CreateLocation CreateLogin

CreateMethod CreateNodegroup

CreateOrdering CreatePackage

CreatePackageContext CreatePackageSynonym

CreatePartitionFunction CreatePartitionScheme

CreatePKConstraint CreatePKIndex

CreateProfile CreateReplicationGroup

CreateRole CreateRollbackSeg

CreateRule CreateSchema

CreateSchemaPrivilege CreateSegment

CreateSequence CreateServerTrigger

CreateStogroup CreateSymmetricKey

CreateSynonym CreateTable

CreateTablespace CreateTransform

CreateUniqueConstraint CreateUserDefinedType

CreateUserId CreateView

CreateViewIndex CreateViewIndexAK

CreateViewIndexClustered CreateViewIndexIE

CreateViewIndexPhysicalStorage CreateViewPrivilege

CreateXMLIndex CreateXMLSchemaCollection

DeleteRelation DoNotUseODBC

Drop DropAggregate

DropAggregateSynonym DropAKIndex

DropApplicationRole DropAssembly

DropAssemblySynonym DropAsymmetricKey

DropAuthorization DropCachedView

DropCachedViewAlternateKeyIndex DropCachedViewIndex

DropCachedViewInversionEntryIndex DropCast

DropCertificate DropCluster

DropClusterIndex DropCredential

DropDatabase DropDatabaseLink

DropDatabaseRole DropDatabaseTrigger

DropDefault DropDirectory

DropDiskgroup DropFKIndex

DropFulltextCatalog DropFulltextIndex

DropFunctionSynonym DropHashIndex

DropIEIndex DropIndex

DropIndexOption DropLibrary

DropLogin DropMethod

DropOrdering DropPackage

DropPackageContext DropPackageSynonym

DropPartitionFunction DropPartitionScheme

DropPKIndex DropProfile

DropReplicationGroup DropRole

DropRollbackSeg DropRule

DropSchema DropSequence

DropServerTrigger DropSymmetricKey

DropSynonym DropTable

DropTablespace DropTransform

DropUserDefinedType DropUserId

DropView DropViewIndex

DropViewIndexAK DropViewIndexIE

DropXMLIndex DropXMLSchemaCollection

ErwinExceptions ERwinGeneratedTrigger

FKConstraintInAlter FKConstraintInCreate

GeneratedTriggerRelationshipOverride GeneratedTriggerRITypeOverride

GenerateRI GenerateUserDefinedTrigger

Include IncludeMDXIndex

IncludeNDXIndex Index

IndexClustered IndexPartitions

IndexPhysicalStorage LabelPKIndex

LastOption MaterializedViewLog

MaterializedViewLogCreateOption MaterializedViewLogDropOption

MaterializedViewLogPartitions MaterializedViewLogPhysicalStorage

ModelCreateFunction ModelCreateMacro

ModelCreateProcedure ModelCreateProcedureSynonym

ModelCreateSynonym ModelDropFunction

ModelDropMacro ModelDropProcedure

ModelDropProcedureSynonym ModelDropSynonym

ModelOption ModelPostScript

ModelPreScript NoCarraigeReturn

ODBC OnDeleteFKConstraint

OnUpdateFKConstraint OtherOptions

OtherOptionsDatabase OtherOptionsSchema

OtherOptionsUserDefinedProperties OverrideOwnerAll

OverrideOwnerAuthorization OverrideOwnerCachedView

OverrideOwnerDatabase OverrideOwnerDefault

OverrideOwnerDomain OverrideOwnerEntity

OverrideOwnerFunction OverrideOwnerKeyGroup

OverrideOwnerMacro OverrideOwnerOracleCluster

OverrideOwnerOracleClusterIndex OverrideOwnerOracleLibrary

OverrideOwnerOraclePackage OverrideOwnerSequence

OverrideOwnerSQLServerAggregate OverrideOwnerSQLServerApplicationRole

OverrideOwnerSQLServerXMLSchemaCollection OverrideOwnerStoredProcedure

OverrideOwnerSynonym OverrideOwnerTrigger

OverrideOwnerValidationRule OverrideOwnerView

PKConstraintInAlter PKConstraintInCreate

QuoteName RefrentialIntegrity

RunCheckModel SchemaCreateOption

SchemaDropOption Security

SequenceCreateSynonym SequenceDropSynonym

SPCreateFKConstraint SPCreatePKConstraint

SpecifyOwner Storage

StorageCreateOption StorageDropOption

StripDelimiter SuppressParameterNames

SuppressPrimaryIndexName SuppressSecondaryIndexNames

TabCheck Table

TableCheckConstraint TableCreateFunction

TableCreateFunctionSynonym TableCreateMacro

TableCreateMaterializedViewLog TableCreateOption

TableCreatePrivilege TableCreateProcedure

TableCreateProcedureSynonym TableCreateSynonym

TableCreateTrigger TableDropFunction

TableDropFunctionSynonym TableDropMacro

TableDropMaterializedViewLog TableDropOption

TableDropProcedure TableDropProcedureSynonym

TableDropSynonym TableDropTrigger

TablePartitions TablePhysicalStorage

TablePostScript TablePreScript

Trigger TriggerCreateOption

TriggerDropOption UseODBC

UserTriggerRelationshipOverride UserTriggerRITypeOverride

View ViewCreateFunction

ViewCreateFunctionSynonym ViewCreateMacro

ViewCreateOption ViewCreateProcedureSynonym

ViewCreateStoredProcedure ViewCreateSynonym

ViewCreateTrigger ViewDropFunction

ViewDropFunctionSynonym ViewDropMacro

ViewDropOption ViewDropProcedureSynonym

ViewDropStoredProcedure ViewDropSynonym

ViewDropTrigger ViewPostScript

ViewPreScript WrapText

Prototype

```
FE::Option( OptionName )
```

Parameter	Status	Description
OptionName	Req	The name of the option.

Result

This macro will fail in the following circumstances:

- he option is not turned on.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
/* Are we generating in physical order? */

[OptionX("ColumnPhysicalOrder")

ForEachReference("Physical_Columns_Order_Ref")

{

Execute("Column Properties")
}

]
```

Result

The code for generating the columns in physical order will be emitted if you have selected this option.

OwnerOverride

Description

This macro evaluates to the appropriate Owner Override value specified in the Forward Engineering dialog.

Prototype

```
FE::OwnerOverride( [DefaultToModel [, OwnerLevels]] )
```

DefaultToModel

Status: Opt

If this value is set to true and no owner override is specified in the Forward Engineering dialog, then the value found in the DB Owner property of the object will be used.

OwnerLevels

Status: Opt

If this is specified, the value will be determined for the owning object this number of levels above the current context object.

Result

This macro will fail in the following circumstances:

- There is no owner override specified and DefaultToModel is not present.
- There is no owner override, DefaultToModel is present, but there is no value specified for the Schema_Name property on the object in the model.

- OwnerLevels is specified and the current context object is not that deep in the model ownership tree.

Deprecation Level

Active

Breaking Changes

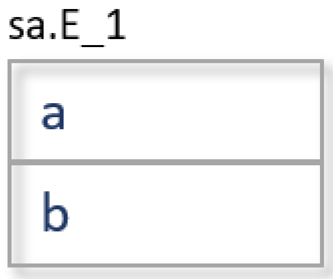
None

Categories

- Forward Engineering Macros

Sample

Assume the following table is the current context object. The first example results show what would emit if an owner override of sa2 was in place. The second example results show what would emit if no owner override was in place.



Template

```
[FE::OwnerOverride "."] Property("Physical_Name") "\n"

[FE::OwnerOverride("true") "."] Property("Physical_Name") "\n

ForEachOwnee("Attribute")

{

ListSeparator("\n")

FE::OwnerOverride("true , "1") "." OwnerProperty("Physical_Name")
"." Property("Physical_Name )

}
```

Result Example #1

```
sa2.E_1

sa2.E_1

sa2.E_1.a

sa2.E_1.b
```

Result Example #2

```
E_1

sa.E_1

sa.E_1.a

sa.E_1.b
```

RecordAlter

Description

This macro inserts an FE Record Alter token into the script. This token is interpreted by the script processing engine when creating Alter scripts. Some changes to a model object can be handled by an ALTER statement, while other changes require the corresponding database object to be dropped and recreated. If a change of the first type is processed before a change of the second type, a redundant ALTER statement is generated. To counter this, all ALTER statements are tagged using this macro. When the FE engine processes the script, it examines the entire state of the object and, if another change forces the drop/recreate, eliminates the ALTER statement. These tags are removed from the final script.

Prototype

```
FE::RecordAlter
```

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
FE::RecordAlter
```

…

Result

There will be no visible effect in the script generated by the FE engine, as the alter script tokens are stripped out of the final output. If a template is expanded outside of the FE engine, the output will appear as follows, where 189 is an object s id.

```
@@*A=189A*@@
```

RecordCreate

Description

This macro will attempt to set a global flag that is a concatenation of the context object s id, the label Create and the context object s type name. The macro will fail if this flag is already set. The flag indicates that the object has been created, which allows the post processing to test for and remove any alter statements which may have been generated for the object.

Prototype

```
FE::RecordCreate
```

Result

This macro will fail in the following circumstances:

- The flag is already set.

Deprecation Level

Active

Breaking Changes

None

Categories

Forward Engineering Macros

Sample

Template

```
FE::RecordCreate
```

…

Result

No visible result.

SchemaExecCommand

Description

This macro inserts an FE Execute Command token into the script. This token is interpreted by the script processing engine when working against a 4GL such as Access or FoxPro in order to construct the commands to modify the database objects via the 4GL's API.

Prototype

```
FE::SchemaExecCommand( CommandString )
```

Parameter	Status	Description
CommandString	Req	The command string to embed.

Result

This macro always succeeds.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
FE::SchemaExecCommand("Create Access Field Start")
```

Result

There will be no visible effect in the script generated by the FE engine, as the schema execution tokens are stripped out of the final output. If a template is expanded outside of the FE engine, the output will appear as follows.

```
@@*SECreate Access Field StartSE*@@
```

TempTable

Description

This evaluates to the name of the temporary table that the Data Preservation option of Alter Script has created.

Prototype

`FE::TempTable`

Result

This macro will fail in the following circumstances:

- Alter Script has not constructed a temporary table.

Deprecation Level

Active

Breaking Changes

None

Categories

- Forward Engineering Macros

Sample

Template

```
…

"INSERT INTO " …

…

" SELECT " …

…

" FROM " TempTable

[" WHERE " DataPreservationOptions("WhereClause")]
```