CA ERwin Macro Code

By LeAnne Jergensen

Table of Contents

Purpose 3
Understanding CA ERwin Macro Commands 3
%<#><command> 3
%== 3
%AttFieldName 3
%Child 4
%ColName 4
%ColumnDataType 4
%ColumnNullOption 5
%DBMSDelim 5
%ForEachChildRel 5
%ForEachColumn 8
%ForEachIndex 11
%ForEachIndexMem 12
%ForEachTable 12
%If 14
%JoinFKPK 14
%JoinPKPK 15
%NK 15
%NKDecl 16
%Not 18
%Parent 18
%ParentCols 18
%PhysRelName 19
%PK 19
%PKDecl 19
%Switch 20
%Substr 21
%TableName 21
%VerbPhrase 22
Uses for ERwin Macros 22
Naming of indexes and constraints automated 22
Copy the records from one table to another for all the tables 23
Create Permissions 24

Purpose

The following document how to use the CA ERwin Macro commands as well as some examples of using Macros in code and in ERwin non-code locations.

Understanding CA ERwin Macro Commands

%<#><command>

Explanation: Used with certain commands that return a character value. By placing a number between the % sign and the command itself, the value returned normally from %command will be truncated to the number of characters specified between the % and the command name.

Syntax: %<number><commandname>

Example 1: Trucate the length of the table name returned to only the first 7 characters instead of returning the entire table name.
%7tablename - if table name is "financial" then only "financi" is returned
vs %table name which returns "financial"

You will have to experiment with commands to see if their value being returned can be truncated.

%==

Explanation: Used in %IF statement to check for equals

Syntax: %==

Example 1: Check if any of the columns in the table are an identity column – if so then put out the command to turn the identity_insert ON so inserts can be done to the table that contain the identity column.

```
%ForEachColumn(Parent, ) {%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert on
SET identity_insert %TableName ON
GO}}
```

Expansion 1: For tables that have an identity column the following will be produced

```
-- The table has an identity column so turn identity_insert on
SET identity_insert table_with_identity_column ON
GO
```

NOTE: For tables that do not have an identity column absolutely no commands will be produced not even the comment

%AttFieldName

Explanation: Same as %ColName

%Child

Explanation: This is used only in special macro commands that allow you the choice of referencing the %Parent table or the %Child table (ie: %JoinFKPK). It helps determine if the columns that are being referenced are from the Parent table or are from the Child table. See that command for example of it's use.

Syntax: %Child

Example 1: See %ForEachChildRel

%ColName

Explanation: References the column name in a table. Which column name is displayed is determined by the macro command it is used in.

Syntax: %ColName

Example 1: For every column in the table being processed show the column name

```
%ForEachColumn() {
   %ColName
}
```

Expansion 1:

```
ref_phone_type_id
phone_type_short_name
phone_type_desc
active_flag
display_sort_order_number
last_updated_by_user
last_updated_by_hostname
last_updated_datetime
```

Usage Context:
Can be used in %ForEachColumn, %ForEachAtt or %ForEachAttribute

%ColumnDataType

Explanation: References the column's data type in a table. The data type is shown for the column being referenced by the macro command. You could also use the combination of %PKDecl and %NKDecl to do the same thing.

Syntax: %ColumnDataType

Example 1: For every column in the table being processed show the column name and its data type

```
%ForEachColumn()
{%ColName   %ColumnDataType
}
```

Expansion 1:

ref_phone_type_id char(18)
phone_type_short_name char(18)
phone_type_desc char(18)
active_flag char(18)
display_sort_order_number char(18)
last_updated_by_user char(18)
last_updated_by_hostname char(18)
last_updated_datetime char(18)

Usage Context:
Can be used in %ForEachColumn, %ForEachAtt or %ForEachAttribute,

%ColumnNullOption

Explanation: In a macro command that loops through each column, it returns one of the following words; NOT NULL; NULL; IDENTITY for each column as it is processed. I found it useful for determining if the table contained any columns that are of the identity type. See example in %==.

Syntax: %ColumnNullOption

Example: See %ForEachTable

%DBMSDelim

Explanation: Different DBMS's end commands with different endings. In SQL Server they use GO. This command will cause whichever DBMS delimiter to be placed in the code.

Syntax: %DBMSDelim

Example in SQL Server: GO
Example in Oracle: ;

%ForEachChildRel

Explanation: This will do whatever you put in the <relationship code> for each of the current tables FK's (children relationships). Very handy to create selects with joins.

Syntax: %ForEachChildRel ([<separator>]) {<relationship code>}

<separator> - Optional, what character to put between each child relationship. If not specified then the ( ) is still required.
<relationship code> - The macro code that should be created for each child relationship that exist.

Example: This select is used to get not just the columns from the current table but also the columns from every Child table. Note that the use of %PhysRelName is needed to keep joins to the same table uniquely named. An example of this is applying_person which has three FK's to ref_plan each with a slightly different name. Also, strangely, sometimes you want double quotes around the separator and sometimes you do not. It sometimes causes the double quotes to show up with the separator. In this case the double quotes are left off the separator for the command %ForEachChildRel.

```
Select
  %ForEachColumn(, ", ") {
  %TableName.%ColName}
,
```

```
%ForEachChildRel(,) {
  %ParentCols(", ",,%PhysRelName.)
 }
FROM %tablename with (nolock)
%ForEachChildRel() {
LEFT JOIN  %Parent  %PhysRelName with (nolock)
ON %JoinFKPK(%Child,%Parent," = "," and")
}
WHERE %JoinPKPK(%Child,@)
```

Expansion: We have no control where the separator is put – on the front or on the end so you have to live with it being different for the different commands.

```
Select
  applying_person.person_id,
  applying_person.requested_health_plan_id,
  applying_person.requested_dental_plan_id,
  applying_person.requested_vision_plan_id,
  applying_person.requested_health_entity_code,
  applying_person.requested_health_entity_full_name,
  applying_person.requested_dental_entity_code,
  applying_person.requested_dental_entity_full_name,
  applying_person.requested_plan_combo_code,
  applying_person.previous_insurance_end_date,
  applying_person.current_insurance_flag,
  applying_person.current_health_insurance_flag,
  applying_person.current_dental_insurance_flag,
  applying_person.current_vision_insurance_flag,
  applying_person.pregnant_flag,
  applying_person.pregnancy_due_date,
  applying_person.unborn_flag,
  applying_person.notice_of_action_flag,
  applying_person.last_updated_by_user,
  applying_person.last_updated_by_hostname,
  applying_person.last_updated_datetime
,
fk_ref_plan_14.ref_plan_id,
  fk_ref_plan_14.ref_plan_type_id,
  fk_ref_plan_14.ref_program_id,
  fk_ref_plan_14.plan_short_name,
  fk_ref_plan_14.plan_code,
  fk_ref_plan_14.plan_desc,
  fk_ref_plan_14.plan_membership_limit_qty,
  fk_ref_plan_14.plan_membership_non_limit_qty,
  fk_ref_plan_14.include_elective_abortion_flag,
  fk_ref_plan_14.can_opt_out_of_elective_abortion_flag,
  fk_ref_plan_14.rural_combo_plan_flag,
  fk_ref_plan_14.active_flag,
  fk_ref_plan_14.display_sort_order_number,
  fk_ref_plan_14.last_updated_by_user,
  fk_ref_plan_14.last_updated_by_hostname,
  fk_ref_plan_14.last_updated_datetime
 ,
  fk_ref_plan_13.ref_plan_id,
  fk_ref_plan_13.ref_plan_type_id,
  fk_ref_plan_13.ref_program_id,
  fk_ref_plan_13.plan_short_name,
```

```
 fk_ref_plan_13.plan_code,
 fk_ref_plan_13.plan_desc,
 fk_ref_plan_13.plan_membership_limit_qty,
 fk_ref_plan_13.plan_membership_non_limit_qty,
 fk_ref_plan_13.include_elective_abortion_flag,
 fk_ref_plan_13.can_opt_out_of_elective_abortion_flag,
 fk_ref_plan_13.rural_combo_plan_flag,
 fk_ref_plan_13.active_flag,
 fk_ref_plan_13.display_sort_order_number,
 fk_ref_plan_13.last_updated_by_user,
 fk_ref_plan_13.last_updated_by_hostname,
 fk_ref_plan_13.last_updated_datetime
,
 fk_ref_plan_12.ref_plan_id,
 fk_ref_plan_12.ref_plan_type_id,
 fk_ref_plan_12.ref_program_id,
 fk_ref_plan_12.plan_short_name,
 fk_ref_plan_12.plan_code,
 fk_ref_plan_12.plan_desc,
 fk_ref_plan_12.plan_membership_limit_qty,
 fk_ref_plan_12.plan_membership_non_limit_qty,
 fk_ref_plan_12.include_elective_abortion_flag,
 fk_ref_plan_12.can_opt_out_of_elective_abortion_flag,
 fk_ref_plan_12.rural_combo_plan_flag,
 fk_ref_plan_12.active_flag,
 fk_ref_plan_12.display_sort_order_number,
 fk_ref_plan_12.last_updated_by_user,
 fk_ref_plan_12.last_updated_by_hostname,
 fk_ref_plan_12.last_updated_datetime
,

fk_person_data_23.person_id,
 fk_person_data_23.person_number,
 fk_person_data_23.ref_gender_id,
 fk_person_data_23.ssn,
 fk_person_data_23.first_name,
 fk_person_data_23.middle_name,
 fk_person_data_23.last_name,
 fk_person_data_23.mothers_maiden_name,
 fk_person_data_23.email_addr,
 fk_person_data_23.birth_date,
 fk_person_data_23.ref_marital_status_id,
 fk_person_data_23.ref_birth_location_id,
 fk_person_data_23.birth_state_id,
 fk_person_data_23.birth_county_id,
 fk_person_data_23.birth_country_id,
 fk_person_data_23.beneficary_identification_card_number,
 fk_person_data_23.ref_citizenship_status_id,
 fk_person_data_23.ref_lawful_permanent_resident_proof_id,
```

```
  fk_person_data_23.entered_usa_date,
  fk_person_data_23.ref_aian_proof_id,
  fk_person_data_23.disregard_person_flag,
  fk_person_data_23.applying_for_service_flag,
  fk_person_data_23.last_updated_by_user,
  fk_person_data_23.last_updated_by_hostname,
  fk_person_data_23.last_updated_datetime

FROM applying_person with (nolock)
LEFT JOIN  ref_plan  fk_ref_plan_14 with (nolock)
ON applying_person.requested_vision_plan_id = ref_plan.ref_plan_id
LEFT JOIN  ref_plan  fk_ref_plan_13 with (nolock)
ON applying_person.requested_dental_plan_id = ref_plan.ref_plan_id
LEFT JOIN  ref_plan  fk_ref_plan_12 with (nolock)
ON applying_person.requested_health_plan_id = ref_plan.ref_plan_id
LEFT JOIN  person_data  fk_person_data_23 with (nolock)
ON applying_person.person_id = person_data.person_id

WHERE applying_person.person_id = @person_id
```

%ForEachColumn

Explanation: For each column in the table create the macro code.

Syntax: %ForEachColumn ([<table>] ,[<separator>] ,[<sort order>]) { <macro code> }

<table> - Optional, can be the words Parent or Child (if a relationship is in scope – ie: This command is nested in some other command), or empty. If no relationship is in scope then if you specify the word Parent or Child it is just ignore and the current tables columns are only considered.
<separator> - Optional, the value here is put after each of whatever you generate in the macro code for each column. The value is not put after the last code generated though.
<sort order> - Optional, no idea what goes here
<macro code> - Lots of different things can go here
NOTE: If the <macro code> is on the same line as the {} brackets then all of the generated code stays on the same line. Otherwise there is a new line for each column. Also, spaces between { and } will show up in the generated code.

Example 1: Puts all of the column names on the same line separated by commas

```
%ForEachColumn(,",") {%AttFieldName}
```

Expansion 1:

```
ref_phone_type_id,phone_type_short_name,phone_type_desc,active_flag,display_sort_order_numb
er,last_updated_by_user,last_updated_by_hostname,last_updated_datetime
```

Example 2: Puts each column name on a separate line with commas before the next column name

```
%ForEachColumn(,",") {   %ColName
}
```

Expansion 2:

```
ref_phone_type_id
```

, phone_type_short_name
, phone_type_desc
, active_flag
, display_sort_order_number
, last_updated_by_user
, last_updated_by_hostname
, last_updated_datetime

Example 3: Creates a list of all columns and their datatypes with @ sign at the front – good for parameter list or declaration statements

```
%ForEachColumn(Parent, ", ") {@%ColName      %ColumnDataType
}
```

NOTE: This is similar to using the following except this puts the comma on the right side not left:

```
%PKDecl(,@,", ",),
%NKDecl(,@,", ",)
```

Expansion 3:

```
@ref_phone_type_id      char(18)
, @phone_type_short_name      char(18)
, @phone_type_desc      char(18)
, @active_flag      char(18)
, @display_sort_order_number      char(18)
, @last_updated_by_user      char(18)
, @last_updated_by_hostname      char(18)
, @last_updated_datetime      char(18)
```

Example 4: Creates a comma separated list of columns in the table along with a column named the same except it has _changed_flag at the end of it. The columns that start with last_updated_ are excluded.

```
  %ForEachColumn(Parent, ", ") {%If(%Not(%==(%Substr(%ColName,1,13),"last_updated_")))
{%ColName
  , %ColName_changed_flag}
  }
```

Expansion 4: Notice that none of the columns that start with last_updated show up in the list

```
  ref_phone_type_id
  , ref_phone_type_id_changed_flag
  , phone_type_short_name
  , phone_type_short_name_changed_flag
  , phone_type_desc
  , phone_type_desc_changed_flag
  , active_flag
  , active_flag_changed_flag
  , display_sort_order_number
  , display_sort_order_number_changed_flag
```

Example 5: Create an if statement for a tables update trigger that checks every column to see if they changed. If they did, then set a flag to TRUE else set it to FALSE

```
%ForEachColumn(Parent,) {%If(%Not(%==(%Substr(%ColName,1,13),"last_updated_"))) {
  IF @inserted_%ColName <> @deleted_%ColName
     SET @%ColName_changed_flag = @TRUE
  ELSE
     SET @%ColName_changed_flag = @FALSE}
  }
```

Expansion 5: See the update trigger for tables that also have an _archive table for how the flags are used.

```
IF @inserted_ref_phone_type_id <> @deleted_ref_phone_type_id
     SET @ref_phone_type_id_changed_flag = @TRUE
  ELSE
     SET @ref_phone_type_id_changed_flag = @FALSE
  IF @inserted_phone_type_short_name <> @deleted_phone_type_short_name
     SET @phone_type_short_name_changed_flag = @TRUE
  ELSE
     SET @phone_type_short_name_changed_flag = @FALSE
  IF @inserted_phone_type_desc <> @deleted_phone_type_desc
     SET @phone_type_desc_changed_flag = @TRUE
  ELSE
     SET @phone_type_desc_changed_flag = @FALSE
  IF @inserted_active_flag <> @deleted_active_flag
     SET @active_flag_changed_flag = @TRUE
  ELSE
     SET @active_flag_changed_flag = @FALSE
  IF @inserted_display_sort_order_number <> @deleted_display_sort_order_number
     SET @display_sort_order_number_changed_flag = @TRUE
  ELSE
     SET @display_sort_order_number_changed_flag = @FALSE
```

%ForEachIndex

Explanation: For each index that exists for the table specified create the macro code.

Syntax: %ForEachIndex ([<table>],[<type>],[<name>],[<separator>]) {<macro code>}

<table> - Optional, by default, this macro is applied to the table in the current scope. This argument can be used to name another table to loop through (ex. %Parent, MOVIE_COPY, etc...)
<type> - Optional, filter on type of index (ex. AK,IE,IF,PK,AK1,IE2,etc...), default is all index types.
<name> - Optional, name of a specific index
<separator> - Optional, string to be placed between the table's indexes
<macro code> - The code to be created for each index

Example 1: Very useful for a where clause, in this case use to check if an record exist based on the alternate key. Notice it defaults to AK1. If more than one alternate key exist then the modeler would have to make sure the more common alternate key index is the AK1 not the AK2.

```
SELECT %PK(,,@) = %PK(,,)
FROM   %TableName with (nolock)
```

```
WHERE  %ForEachIndex(%Parent,AK1) {%ForEachIndexMem(,and) {   %ColName=@%ColName
}}
```

Expansion 1:

```
SELECT @address_id = address_id
FROM   address_data with (nolock)
WHERE     line_1_addr=@line_1_addr
and   line_2_addr=@line_2_addr
and   apartment_number_code=@apartment_number_code
and   city_name=@city_name
and   ref_county_id=@ref_county_id
and   ref_state_id=@ref_state_id
and   ref_zipcode_id=@ref_zipcode_id
```

%ForEachIndexMem

Explanation: Used within the scope of an index macro (ex: %ForEachIndex). This macro loops through
the members of the index and creates the specified macro code.

Syntax: %ForEachIndexMem ([<sequence>],[<separator>]) {<macro code>}
<sequence> - Optional, number of a specific index member - default is all members
<separator> - Optional, string to be placed between the index's members
<macro code> - The code that will be generated for each index member

Example 1: See example and expansion in %ForEachIndex

%ForEachTable

Explanation: Loops through all tables in the current Subject Area, creating the macro code.

Syntax: %ForEachTable ([<table name>]) {<macro code>}

<table name> - Optional, this is the name of a table you wish to create the macro code for, or if not specified it defaults to all tables in the current Subject Area.
<macro code> - The code that will be generated for each table

Example 1: Create a script that will insert records into current tables from an old copy of the table. Note: If the table has an identity column then include code for that table that turns on and off the identity_insert attribute.

```
%ForEachTable(){
 %ForEachColumn(Parent, ) {%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert on
SET identity_insert %TableName ON
GO}
}
INSERT into %TableName
(
  %ForEachColumn(Parent, ", ") {%ColName
}
)
```

```
select
  %ForEachColumn(Parent, ", ") {%ColName
}
from old_%TableName
go
 %ForEachColumn(Parent, ) {%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert off
SET identity_insert %TableName OFF
GO}
}
}
```

Expansion 1: Only two tables are shown

```
-- The table has an identity column so turn identity_insert on
SET identity_insert aaps_invoice ON
GO

INSERT into aaps_invoice
(
  aaps_invoice_id
, request_payment_check_amount
, invoice_number_code
, payment_check_written_date
, payment_check_number
, payment_check_amount
, eds_vendor_remit_id
, last_updated_by_user
, last_updated_by_hostname
, last_updated_datetime

)
select
  aaps_invoice_id
, request_payment_check_amount
, invoice_number_code
, payment_check_written_date
, payment_check_number
, payment_check_amount
, eds_vendor_remit_id
, last_updated_by_user
, last_updated_by_hostname
, last_updated_datetime

from old_aaps_invoice
go
 -- The table has an identity column so turn identity_insert off
SET identity_insert aaps_invoice OFF
GO
```

```
INSERT into add_a_person
(
  document_header_id
, last_updated_by_user
, last_updated_by_hostname
, last_updated_datetime

)
select
  document_header_id
, last_updated_by_user
, last_updated_by_hostname
, last_updated_datetime

from old_add_a_person
go
```

%If

Explanation: Allows create macro code based on what value is found in the predicate.

Syntax: %If (<predicate>) {<macro code>} [%Else {<macro code>}]

<predicate> - Code that is check to see if it is TRUE or FALSE
<macro code> - The code that is generated, the first if the predicate check is TRUE, the second if the predicate check is FALSE

Example 1: Check if the characters 1 thru 8 are equal to "IDENTITY". If yes then create the macro code. Here the else is do nothing because it is not included.

```
%ForEachColumn(Parent, ) {%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert off
SET identity_insert %TableName OFF
GO}}
```

Expansion 1:

```
-- The table has an identity column so turn identity_insert off
SET identity_insert aaps_invoice OFF
GO
```

%JoinFKPK

Explanation: Allows you to build the ON section of a JOIN clause. It handles the name of the columns being different that are foreign keyed. Ex: applying_person has health_plan_id column that is foreign keyed to plan_data table using plan_id column. This command handles the column names being different spellings.

Syntax: %JoinFKPK (<child table>,<parent table>,[<comparison op>],[<separator>])
<child table> - Usually specify %Child
<parent table> - Usually specify %Parent

<comparison operator> - Optional, put the operator you wish to be placed between the two column names; ex: "=". If not specified then it defaults to "=".
<separator> - Optional, usually it is a "," or an " and"

Example 1: See %ForEachChildRel

%JoinPKPK

Explanation: Allows you to build the where clause for comparing the primary key columns to something else.

Syntax: %JoinPKPK(<table>,[<correlation>],[<comparison op]>,[<separator>])

<table> - Usually specify %Child
<correlation> - Optional, what should be put on the front of the name of each of the primary key columns on the right hand side of the comparison operator. If it is an @ then no period separator is inserted. If it is the word inserted then a period separator is inserted because it assumes this is the alias you want to use.
<comparison operator> - Optional, put the operator you wish to be placed between the two column names; ex: "=". It defaults to an "="
<separator> - Optional, usually it is a "," or an " and". It defaults to an "and"

Example 1: Good for checking that the primary key is equal to what was passed in to the stored procedure.

```
DELETE from %TableName
WHERE %JoinPKPK(%Child,@)
```

Expansion 1:

```
DELETE from address_data
WHERE address_data.address_id = @address_id
```

Example 2: Good for insert triggers to update the last_updated_datetime column

```
update %TableName
set last_updated_datetime = GETDATE()
from %TableName
       ,inserted
where %JoinPKPK(%Child,inserted)
```

Expansion 2:

```
update address_data
set last_updated_datetime = GETDATE()
from address_data
      ,inserted
where address_data.address_id = inserted.address_id
```

%NK

Explanation: Allows you to specify that all non primary key columns should be displayed. The ( ) is required even if none of the options are used.

Syntax: %NK ([<separator>],[<function>],[<prefix>])

<separator> - Optional, usually it is a ",” or an " and”.
<function> - Optional, whatever word you put here is put before each column name and also forces there to be ( ) around each column name. I'm have not found a use for this yet.
Ex: <prefix><function>(<columnname>)<separator> if function is specified and
<prefix><columnname><separator> if function is not specified.
<prefix> - Optional, this is what you what to go first in the result line. If you specify a function it goes before the function. If you do not specify a function then it goes before the column name.

Example 1:
%NK('',',,@)

Expansion 1:
@line_1_addr,
@line_2_addr,
@apartment_number_code,
@city_name,
@ref_county_id,
@ref_state_id,
@ref_zipcode_id,
@zip_code4,
@last_updated_by_user,
@last_updated_by_hostname,
@last_updated_datetime

%NKDecl

Explanation: Display the non primary key column names and their declaration. Very handy for declaring what parameters should be passed into a stored procedure.

Syntax: %NKDecl ([<old prefix>],[<new prefix>],[<separator>],[<attribute/type separator>])

<old prefix> - Optional, if this is used then all columns and their declaration is written out with the separators specified with whatever is in this place. If it is used it will usually have old. or old_ so there is a separation between the prefix and the column name.
<new prefix> - Optional, if this is used then all columns and their declaration is written out with the separators specified with whatever is in this place. If it is used it will usually have new. or new_ so there is a separation between the prefix and the column name.
<separator> - Optional, this is put at the end of each line until the last line where there is no separator placed.
<attribute/type separator> - Optional, this is the word that is put between the name of the column and the declare type and size.
NOTE: If both old and new prefix is specified then the columns are displayed twice, once with the old prefix and once with the new prefix.

Example 1: Put out just a list of the non primary key columns for a table for parameters to a stored procedure
%NKDecl(, @,", ",)

Expansion 1:
@line_1_addr varchar(60),
@line_2_addr varchar(60),
@apartment_number_code varchar(20),

```
@city_name varchar(40),
@ref_county_id int,
@ref_state_id int,
@ref_zipcode_id int,
@zip_code4 char(4),
@last_updated_by_user varchar(30),
@last_updated_by_hostname varchar(30),
@last_updated_datetime datetime
```

Example 2: Not sure why you would use this but thought I would show it as an example anyway
%NKDecl(in_,out_,",")

Expansion 2:
in_line_1_addr varchar(60),
in_line_2_addr varchar(60),
in_apartment_number_code varchar(20),
in_city_name varchar(40),
in_ref_county_id int,
in_ref_state_id int,
in_ref_zipcode_id int,
in_zip_code4 char(4),
in_last_updated_by_user varchar(30),
in_last_updated_by_hostname varchar(30),
in_last_updated_datetime datetime,out_line_1_addr varchar(60),
out_line_2_addr varchar(60),
out_apartment_number_code varchar(20),
out_city_name varchar(40),
out_ref_county_id int,
out_ref_state_id int,
out_ref_zipcode_id int,
out_zip_code4 char(4),
out_last_updated_by_user varchar(30),
out_last_updated_by_hostname varchar(30),
out_last_updated_datetime datetime

%Not

Explanation: Used to negate the answer in an %If statement.

Syntax: %Not

Example 1: See example 4 in %ForEachColumn

%Parent

Explanation: This is used only in special macro commands that allow you the choice of referencing the %Parent table or the %Child table (ie: %JoinFKPK). It helps determine if the columns that are being referenced are from the Parent table or are from the Child table. See that command for example of it's use.

Syntax: %Parent

Example 1: See %ForEachChildRel, %JoinFKPK or %ForEachIndex.

%ParentCols

Explanation: Use it to loop through all of the columns in the Parent relationship in the commands: %ForEachParentRel and %ForEachChildRel.

Syntax: %ParentCols ([<separator>],[<function>],[<prefix>])

<separator> - Optional, usually it is a ",” or an " and”.
<function> - Optional, whatever word you put here is put before each column name and also forces there to be ( ) around each column name. I'm have not found a use for this yet.

Ex: <prefix><function>(<columnname>)<separator> if function is specified and
<prefix><columnname><separator> if function is not specified.
<prefix> - Optional, this is what you what to go first in the result line. If you specify a function it goes before the function. If you do not specify a function then it goes before the column name.

Example 1: See %ForEachChildRel

%PhysRelName

Explanation: Shows whatever the relationship name is. You can see this value using the GUI in the physical model by double clicking on any relationship line. The value in the Foreign Key Constraint Name is what will be displayed for %PhysRelName.

Syntax: %PhysRelName

Example 1: See %ForEachChildRel for an example of where it is handy to use. Needed there because it allows for a unique alias name in the joining of tables.

%PK

Explanation: Allows you to specify that all primary key columns should be displayed. The ( ) is required even if none of the options are used.

Syntax: %PK([<separator>],[<function>],[<prefix>])

<separator> - Optional, usually it is a ",” or an " and”.
<function> - Optional, whatever word you put here is put before each column name and also forces there to be ( ) around each column name. I'm have not found a use for this yet.
Ex: <prefix><function>(<columnname>)<separator> if function is specified and
<prefix><columnname><separator> if function is not specified.
<prefix> - Optional, this is what you what to go first in the result line. If you specify a function it goes before the function. If you do not specify a function then it goes before the column name.

Example 1:
%PK(',',,@)

Expansion 1:
@authorized_entity_id,
@person_id,
@ref_person_permission_id

%PKDecl

Explanation: Display the primary key column names and their declaration. Very handy for declaring what parameters should be passed into a stored procedure.

Syntax: %PKDecl ([<old prefix>],[<new prefix>],[<separator>],[<attribute/type separator>])

<old prefix> - Optional, if this is used then all columns and their declaration is written out with the separators specified with whatever is in this place. If it is used it will usually have old. or old_ so there is a separation between the prefix and the column name.
<new prefix> - Optional, if this is used then all columns and their declaration is written out with the separators specified with whatever is in this place. If it is used it will usually have new. or new_ so there is a separation between the prefix and the column name.
<separator> - Optional, this is put at the end of each line until the last line where there is no separator placed.
<attribute/type separator> - Optional, this is the word that is put between the name of the column and the declare type and size.
NOTE: If both old and new prefix is specified then the columns are displayed twice, once with the old prefix and once with the new prefix.

Example 1: If the primary keys of the table xref_authorized_entity_person needed to be passed back to the calling program from the stored procedure then you would also need to specify the word output. Notice the word OUTPUT outside the), it is outside the command so it only occurs on the last line. The other OUTPUT inside the ( )'s will be put on each line except the last line.

```
%PKDecl(,@," OUTPUT",) OUTPUT
```

Expansion 1:

```
@authorized_entity_id int OUTPUT,
@person_id int OUTPUT,
@ref_person_permission_id int OUTPUT
```

%Switch

Explanation: This command is similar to a case statement. It evaluates the switch argument and matches the result against the specified choices. If a match is found the choice's corresponding macro code is expanded. If no match is found, the default's macro code is expanded.

Syntax:

```
%Switch(<argument>) {
   %Choose(<choice 1>) {<macro code 1>}
   %Choose(<choice 2>) {<macro code 2>}
      <etc...>
   %Default {<default macro code>}
}
```

<argument> - This is some macro code that will result in a character string
<choice #> - The character string you wish to match up with
<macro code #> - The code to generate if the character string matches
<default macro code> - The code to generate if none of the character string choices match

Example 1: To use this in the naming window it must not have any end of line characters so it is not formatted very pretty. See Naming Indexes and Constraints for where you put this code.

```
%Switch(%Substr(%KeyType,1,2)) {%Choose(pk) {pk_%tablename} %Choose(ak)
{ak_%tablename_%substr(%keyType,3,2)} %Default {fk_%tablename_%substr(%keyType,3,2)}}
```

Expansion 1: %KeyType is only known for indexes so this is expanded based on the different keytypes for table person_data

Keytype	New Index Name
pk	pk_person_data
ak	ak_person_data_1
if1	fk_person_data_1
if2	fk_person_data_2

%Substr

Explanation: Allows you to strip characters from a character string and return just those.

Syntax: %Substr(<macro code>,<initial pos>,[<length>])

<macro code> - Some macro command that will return a character string or a double quoted string.
<initial pos> - The starting position of the string to return (1 base – so the first character is considered position number 1).
<length> - Optional. Number of characters returned if there is that many characters left in the string. If it is not specified then all the rest of the characters in the string are returned.

Example 1: To have the relationships names follow the Healthy Families standards the following is used for constraint names. Using it does prevent you from putting explanations in the verb phrase or you still need to keep the r/# along with the words you add to display.
Example 1: See Naming of Indexes and Constraints for where to place this code.
fk_%Parent_%substr(%VerbPhrase,3,4)

Expansion 1: %VerbPhrase is only known for constraints so this is expanded based on the different verb phrases for table person_data. The r/234 and r/12 is the default value of a logical Parent-to-Child verb phrase. This only works well if the defaults are left in the verb phrase.

VerbPhrase	Constraint Name
r/234	fk_person_data_234
r/12	fk_person_data_12

Example 2:
%substr("mother",3,2)

Expansion 2:
th

%TableName

Explanation: The table name that is currently being worked with

Syntax: %TableName

Example 1:
%TableName

Expansion 1:
person_data

%VerbPhrase

Explanation: Shows whatever the verb phrase is defined for a relationship. You can see this value using the GUI in the logical model by double clicking on any relationship line. The value in the Parent-to_Child and Child-to_Parent names is what will be displayed for %VerbPhrase

Syntax: %VerbPhrase

Example 1: See Naming of Indexes and Constraints for a place to use this macro command.

Uses for ERwin Macros

CA ERwin Macros can be used to create stored procedures and specialized triggers. They can also be used to name different ERwin objects consistently.

Naming of indexes and constraints automated

Use the unique number assigned in the logical model to create an unique physical fk. As well, get the physical pk and ak to match your organization's naming standards.
Indexes and foreign keys defaults for names can be changed from the starting defaults but it is a pain to do it manually to match the Naming Standards for primary, foreign and alternate indexes and constraints. The following describes all of the ERwin changes that can be done so that the physical constraints and indexes do **NOT** need to be manually changed by the DBA. These use ERwin Macros command. If this method is used, you cannot change the verb phrase on the logical side of the model because the unique number that is automatically generated for the key group is used in the name of the constraint. Note: This still leaves the logical key group for the relationship not matching the physical side, but since it is never generated as code it should not matter.

- Click on Tools on the top line
- Choose Names
- Click on Model Naming Options
- Click on the Name Mapping tab
- Change the Relationships row – column ERwin Macro to:
    - Option 1: fk_%Parent_%P2CVerbPhrase
        - This will cause the foreign key relationship to be named fk_<tablename>_<words in the Parent-to-Child verbphrase>
        - This does limit what can be put into the logical model's verb phrase depending on the maximum length the constraints can be for the DBMS being used. See Option 2 second bullet for ideas of how to limit the size of this.
        - If the DBA chooses to change the constraint name in the physical model, the new name will be used and overrides the default name this gives it even if this is changed in the future. The only way the DBA would lose the overridden name is if the button "Reset Name" is chosen and constraint names are reset.
        - You can also choose to use the verb phrase in the Child to Parent instead using: fk_%Parent_%C2PVerbPhrase
    - Option 2: fk_%Parent_%Child
        - This will cause the foreign key relationship to be named fk_<tablename of parent>_<tablename of child>

        - Depending on the size of table names, this could cause problems with the length of the constraint name for the DBMS. One way around that is to put a limit on the number of characters both tables return with fk_%13Parent_%13Child which would give a maximum of 30 characters in the constraint name. Another option would be to truncate the entire name with %substr(fk_%parent_%child,1,31) which builds the complete name and then returns only the first 30 characters of the string as the name of the physical constraint.
    - Option 3: fk_%Parent_%Substr(%VerbPhrase,3,4)
        - This will cause the foreign key relationship to be named fk_<tablename>_<unique number on DEFAULT verbphrase>
        - This does limit what can be put into the logical model's verb phrase. By default ERwin puts r/### in it where the ### is unique for the model for all relationships. This does not work well at all if the Standard of filling in the verb phrases on the logical model is followed because that changes the default so that this formula does not have a number to pick up anymore.

- Change the Key Group To Index row – column ERwin Macro to:
    - %Switch(%Substr(%KeyType,1,2)) {%Choose(pk) {pk_%tablename} %Choose(ak) {ak_%tablename} %Default {fk_%tablename_%substr(%keyType,3,2)}}
        - This will make the PK name look like pk_<tablename>, the AK name look like ak_<tablename> and FK name look like fk_<tablename>_<#>. This keeps the index names unique without forcing the DBA to have to name each and everyone. If the DBA chooses to change the index name in the physical model, the new name will be used and overrides the default name this gives it even if this is changed in the future. The only way the DBA would loose the overridden name is if the button "Reset Name" is chosen and Index names are reset.

Copy the records from one table to another for all the tables

This code generates SQL that will copy all of the records from one table to another. The code has been tested in SQL Server 2000 but could easily be converted to other DBMSs.

```
%ForEachTable(){
 %ForEachColumn(Parent, )
{%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert on
SET identity_insert %TableName ON
GO}
}
INSERT into %TableName
(
  %ForEachColumn(Parent, ", ") {%ColName
}
)
select
  %ForEachColumn(Parent, ", ") {%ColName
}
from old_%TableName
```

```
go
 %ForEachColumn(Parent, )
{%If(%==(%Substr(%columnnulloption,1,8),"IDENTITY"))
{-- The table has an identity column so turn identity_insert off
SET identity_insert %TableName OFF
GO}
}
}
```

Create Permissions

This code generates SQL that makes sure all stored procedures are granted access to a specified database role. The code has been tested in SQL Server 2000 but could easily be converted to other DBMSs.

```
--Misc DBA SQL Code for SQL Server 2000
--Create Permissions Macro Code
-- Make sure all stored procs are granted access to the r_hf_system role -
change this to whatever role name you are using in SQL Server.

IF NOT exists (select * from dbo.sysusers where name = N'r_hf_system' and uid >
16399)
   EXEC sp_addrole N'r_hf_system'
%DBMSDelim

IF NOT exists (select * from dbo.sysusers where name = N'hf_user' and uid <
16382)
   EXEC sp_grantdbaccess N'hfamxform01\hf_user', N'hf_user'
%DBMSDelim

-- NOW MAKE SURE THAT THE TWO ROLES THIS ACCESSES HAVE AT LEAST ONE USER IN
THEM

-- Add the new login ID for accessing the database using VB.NET  This
-- will change in the near future to hf_db_user when the system moves
-- to the new servers

EXEC sp_addrolemember N'r_hf_system', N'hf_user'
%DBMSDelim

-- Grant execute to all stored procs to r_hf_system role

DECLARE csr_grants_hf_system_role CURSOR FAST_FORWARD FOR
SELECT 'GRANT EXECUTE ON [' + su.name + '].[' + so.name + '] TO [r_hf_system]'
FROM   sysobjects so
JOIN   sysusers su
ON     su.uid = so.uid
WHERE  (xtype = 'P')
AND    so.name LIKE 'usp%'

OPEN csr_grants_hf_system_role
DECLARE @grant_command_hf_system_role as varchar(500)

FETCH NEXT FROM csr_grants_hf_system_role
INTO  @grant_command_hf_system_role
WHILE @@FETCH_STATUS = 0
BEGIN
    EXECUTE (@grant_command_hf_system_role)
```

```
    FETCH NEXT
    FROM  csr_grants_hf_system_role
    INTO  @grant_command_hf_system_role
END

CLOSE csr_grants_hf_system_role
DEALLOCATE csr_grants_hf_system_role

-- Sets up all of the r_testing_stored_procedures grants

IF NOT exists (select * from dbo.sysusers where name =
N'r_testing_stored_procedure' and uid > 16399)
   EXEC sp_addrole N'r_testing_stored_procedure'
%DBMSDelim

DECLARE csr_SEs CURSOR FAST_FORWARD FOR
SELECT 'exec sp_addrolemember N' + '''' + 'r_testing_stored_procedure' + '''' +
', N' + '''' + u2.name + ''''
FROM   sysusers as u1
JOIN   sysmembers as m
ON     u1.uid = m.groupuid
JOIN   sysusers as u2
ON     m.memberuid = u2.uid
where  u1.name='r_hf_operational_prod_support';

OPEN csr_SEs
DECLARE @add_SE as varchar(500)

FETCH NEXT
FROM  csr_SEs
INTO  @add_SE
WHILE @@FETCH_STATUS = 0
BEGIN
    EXECUTE (@add_SE)
    FETCH NEXT
    FROM  csr_SEs
    INTO  @add_SE
END

CLOSE csr_SEs
DEALLOCATE csr_SEs

DECLARE csr_grants CURSOR FAST_FORWARD FOR
SELECT 'GRANT EXECUTE ON [' + su.name + '].[' + so.name + '] TO
[r_testing_stored_procedure]'
FROM   sysobjects so
JOIN   sysusers su
ON     su.uid = so.uid
WHERE  (xtype = 'P')
AND    so.name like 'usp%';

OPEN csr_grants
declare @grant_command as varchar(500)

FETCH NEXT
FROM  csr_grants
INTO  @grant_command
WHILE @@FETCH_STATUS = 0
BEGIN
    EXECUTE (@grant_command)
    FETCH NEXT
```

```
    FROM  csr_grants
    INTO  @grant_command
END

CLOSE csr_grants
DEALLOCATE csr_grants

%ForEachTable(){
-- Grant all access to the table to the SE

  GRANT  SELECT  ON [dbo].[%TableName]  TO [r_hf_operational_prod_support]
%DBMSDelim
  GRANT  UPDATE ON [dbo].[%TableName]  TO [r_hf_operational_prod_support]
%DBMSDelim
  GRANT  INSERT ON [dbo].[%TableName]  TO [r_hf_operational_prod_support]
%DBMSDelim
  GRANT  DELETE ON [dbo].[%TableName]  TO [r_hf_operational_prod_support]
%DBMSDelim
}
```