m_utl_hash_lookup.sas File Reference

Utilities

Utility macro to define a hash table object and perform lookups

---

Description

The macro can be used to create a hash table in memory based on a list selection of columns from a transaction SAS dataset or database table. Only columns listed in the KEYS= and COLS= parameters will be added to the hash object. If the hash object already exist in the SAS data step context, it will be declared only once to avoid duplicate hash objects in memory. It can be used anywhere in a SAS program specially within SAS data step. This macro is based on the ut_hash_lookup.sas macro program from Dave Prinsloo (dave.prinsloo@yahoo.com).

Note

In case of encrypted SAS datasets, the ENCRYPTKEY= parameter must be provided as part of the CREDS credentials string.

Autors

Paul Alexander Canals y Trocha (paul.canals@gmail.com)

Date

2020-10-22 00:00:00

Version

20.1.10

Link

https://github.com/paul-canals/toolbox

Parameters

Input	help	Parameter, if set (Help or ?) to print the Help information in the log. In all other cases this parameter should be left out from the macro call.
Input	dataset	Full LIBNAME.TABLENAME name of the table or SAS dataset to be loaded into memory as a hash object. The parameter can contain a combination of SAS data step style statement between brackets like where=, keep=, drop= or rename=.
Input	table	Alias of the DATASET= parameter.
Input	data	Alias of the DATASET= parameter.
Input	creds	Optional. Specifies the ENCRYPTKEY= parameter value if there is an encrypted table involved.
Input	context	Optional. Specifies the name of the SAS data step context in which the hash object is created. The CONTEXT value is added as suffix to the HASHNAME value. Please do remember that the maximum value of HASHNAME is restricted to 30 characters.
Input	keys	Specifies the input key variables for the hash object. Key variables are separated by a blank character.
Input	keys_orig	Optional. Specifies the original names of the key variables in the base table. If this parameter is used do remember to keep the key variable list in exactly the same order as the KEYS parameter, and also the number of key variables should be the same between both parameters.
Input	key_values	Alias of the KEYS_ORIG= parameter.
Input	cols	Specifies the output data variables for the hash object. Data variables are separated by a blank character. If the optional parameter COLS_ORIG is set, the original names of the data variables are renamed by the variables listed by the COLS parameter. The default value for COLS is: _ALL_.
Input	cols_orig	Optional. Specifies the original names of the data variables in the base table. If this parameter is used do remember to keep the data variable list in exactly the same order as the COLS parameter, and also the number of data variables should be the same between both parameters.
Input	col_values	Alias of the COLS_ORIG= parameter.
Input	prefix	Optional. Character string to be added as to each hash object COLS output data column.
Input	quoted	Boolean [Y|N] parameter to specify wether to put named quotes around each variable name in the list. The default value for QUOTED is: N.
Input	rc_var	Character string to specify the return column for the hash find call. The default value is: rc_find.
Output	list_mvar	Character string to specify a SAS macro variable containing a list of hash objects separated by a blank character which are set into memory in the context of a single SAS data step. The default value for LIST_MVAR is: _list_.
Input	debug	Boolean [Y|N] parameter to provide verbose mode information. The default value is: N.

Returns

- A hash table defined in memory and lookups performed

Calls

- m_utl_chg_delimiter.sas
- m_utl_hash_define.sas
- m_utl_list_operation.sas
- m_utl_print_message.sas
- m_utl_print_mtrace.sas
- m_utl_varlist.sas

Usage

Example 1: Show help information:

```
%m_utl_hash_lookup(?)
```

Example 2: Create a new hash object and perform lookup for all columns:

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name
    , cols      = _ALL_
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 3: Create a new hash object and perform lookup for some columns:

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , cols      = Predict Lowermean Uppermean
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 4: Create new hash objects and perform lookups (will fail):

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , cols      = Predict
    , list_mvar = _LIST_
    , debug     = Y
      );
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , cols      = Lowermean Uppermean
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 5: Create new hash objects and perform lookups (successful):

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class1
    , keys      = Name Sex Age
    , cols      = Predict
    , list_mvar = _LIST_
    , debug     = Y
      );
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class2
    , keys      = Name Sex Age
    , cols      = Lowermean Uppermean
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 6: Create a hash object and perform lookups with changed key names:

```
%let _LIST_=;

data WORK.class(rename=(Sex=Gender));
   set SASHELP.class;
run;

data WORK.result;
   set WORK.class;
   %m_utl_hash_lookup(
      table      = SASHELP.classfit
    , context    = class
    , keys       = Name Sex Age
    , keys_orig  = Name Gender Age
    , cols       = _ALL_
    , list_mvar  = _LIST_
    , debug      = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 7: Create a hash object, perform lookups with changed column and key names:

```
%let _LIST_=;

data WORK.class(rename=(Sex=Gender));
   set SASHELP.class;
run;

data WORK.result;
   set WORK.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , keys_orig = Name Gender Age
    , cols      = Predicted Lower Upper
    , cols_orig = predict lowermean uppermean
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 8: Create a hash object, perform lookups with added prefix to all data columns:

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , cols      = _ALL_
    , prefix    = fit_
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 9: Create a hash object, perform lookups with added attribute Predicted_Weight format:

```
%let _LIST_=;

data WORK.result;
   set SASHELP.class;
   attrib Predicted_Weight length=8 format=8.3 label='blaat';
   %m_utl_hash_lookup(
      table     = SASHELP.classfit
    , context   = class
    , keys      = Name Sex Age
    , cols      = Predicted_Weight
    , cols_orig = predict
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 10: Create a hash object, perform lookups on an encrypted SAS dataset:

```
%let _LIST_=;

data WORK.classfit(encrypt=aes encryptkey=aespasskey);
   set SASHELP.classfit;
run;

data WORK.result;
   set SASHELP.class;
   %m_utl_hash_lookup(
      table     = WORK.classfit
    , creds     = %str(encryptkey=aespasskey)
    , context   = class
    , keys      = Name Sex Age
    , cols      = _ALL_
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Example 11: Create a hash object, and perform lookups using SAS invalid names:

```
%let _LIST_=;

data WORK.class;
   set SASHELP.class;
   rename
      name = '/bic/Name'n
      sex  = '/bic/Sex'n
      age  = '/bic/Age'n
      ;
run;

data WORK.classfit;
   set SASHELP.classfit;
   rename
      name    = '/bic/Name'n
      sex     = '/bic/Sex'n
      age     = '/bic/Age'n
      predict = '/bic/Predict'n
      upper   = '/bic/Upper'n
      ;
run;

data WORK.result;
   set WORK.class;
   %m_utl_hash_lookup(
      table     = WORK.classfit
    , context   = class
    , keys      = '/bic/Name'n '/bic/Sex'n '/bic/Age'n
    , cols      = '/bic/Predict'n '/bic/Upper'n
    , quoted    = Y
    , list_mvar = _LIST_
    , debug     = Y
      );
run;

%put &=_LIST_.;

proc print data=WORK.result;
run;
```

Copyright