m_utl_hash_define.sas File Reference

Utilities

Utility macro to define a SAS dataset as hash table in memory

---

Description

The macro can be used to create a hash table in memory based on a list selection of columns from an input SAS dataset or database table. Only the columns listed in the KEYS= and COLS= parameters will be added to the hash object. The macro can be used anywhere in a SAS program specially within SAS data step. This macro is based on the ut_hash_define.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

2021-03-27 00:00:00

Version

21.1.03

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	hashname	Specifies the name of the result hash object. HASHNAME has a maximum length of 30 characters. The default value for HASHNAME is: _HASH_.
Input	name	Alias of the HASHNAME= parameter.
Input	hiter_flg	Boolean [Y|N] parameter to determine whether an hash iterartive object is to be created by the program. The hash iterator object can be used to store and search data based on lookup keys. The hash iterator object enables you to retrieve the hash object data in either forward or reverse key order. The default value for HITER_FLG is: N.
Input	hitername	Specifies the name of the result hiter object. HITERNAME has a maximum length of 30 characters. The default value for HITERNAME is: _HITER_.
Input	keys	Specifies the input key variables for the hash object. Key variables are separated by a blank character. If the optional parameter KEYS_ORIG is set, the original names of the key variables are renamed by the variables listed by the KEYS parameter.
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 equal between both parameters.
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	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	duplicate	Indicator [E|ERROR|R|REPLACE] to determine whether to ignore duplicate keys when loading a data set into the hash object. The default value _REPLACE_ is to store the first key and ignore all subsequent duplicates. Optionnaly the parameter value can be set to _ERROR_ to throw an error when a duplicate key is found. This parameter depends on wether the MULTIDATA= parameter is set to Y. If this is the case, the DUPLICATE= parameter is set to inactive. The default value for DUPLICATE is: R.
Input	multidata	Boolean [Y|N] parameter to determine if multiple data items are allowed for each key. The default value for MULTIDATA is: N.

Input	ordered	Indicator [ASCENDING|DESCENDING|NO|YES] parameter to specify if the ouput to the result hash object is to be ordered by the key variables. With _NO_ data is returned in some undefined order. When _YES_ data is returned in ascending value order. Specifying _ASCENDING_ is the same as specifying _YES_. When _DESCENDING_ is specified data is returned in descending key-value order. The default value for ORDERED is: N.
Input	rowcount	Optional. To specify the estimated number of records to be loaded into the hash object.
Input	debug	Boolean [Y|N] parameter to provide verbose mode information. The default value is: N.

Returns

- A hash table defined in memory

Calls

- m_utl_chg_delimiter.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_define(?)
```

Example 2: Create a hash object from an encrypted SAS dataset:

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

data _null_;
   %m_utl_hash_define(
      table = WORK.class
    , creds = %str(encryptkey=aespasskey)
    , name  = class
    , keys  = Name
    , cols  = _ALL_
    , debug = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;
```

Example 3: Create an ordered hash object:

```
data _null_;
   %m_utl_hash_define(
      table   = SASHELP.class
    , name    = class
    , keys    = Name
    , cols    = _ALL_
    , ordered = Y
    , debug   = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;
```

Example 4: Create an (A)scending ordered hash object:

```
data _null_;
   %m_utl_hash_define(
      table   = SASHELP.class
    , name    = class
    , keys    = Name
    , cols    = _ALL_
    , ordered = A
    , debug   = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;
```

Example 5: Create an (D)escending ordered hash object:

```
data _null_;
   %m_utl_hash_define(
      table   = SASHELP.class
    , name    = class
    , keys    = Name
    , cols    = _ALL_
    , ordered = D
    , debug   = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;
```

Example 6: Create a hash object with a selection of columns (keep=):

```
data WORK.result;
   %m_utl_hash_define(
      table = SASHELP.class(keep=Name Sex Age)
    , name  = class
    , keys  = Name
    , cols  = Sex Age
    , debug = Y
      );
   set SASHELP.class(keep=Name);
   rc = class.find();
   drop rc;
run;
```

Example 7: Create a hash object with a selection of columns (drop=):

```
data WORK.result;
   %m_utl_hash_define(
      table = SASHELP.class(drop=Height Weight)
    , name  = class
    , keys  = Name
    , cols  = _ALL_
    , debug = Y
      );
   set SASHELP.class(keep=Name);
   rc = class.find();
   drop rc;
run;
```

Example 8: Create a hash object with a selection and renaming of columns:

```
data WORK.result;
   %m_utl_hash_define(
      table     = SASHELP.class
    , name      = class
    , keys      = Name
    , cols      = Gender Age
    , cols_orig = Sex Age
    , debug     = Y
      );
   set SASHELP.class(keep=Name);
   rc = class.find();
   drop rc;
run;

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

Example 9: When loading duplicate key data items, only the last known value is saved:

```
data WORK.class;
   set SASHELP.class;
   if Name eq 'Alice' then do;
      output; Age=16; output;
   end;
run;

proc print data=WORK.class;
run;

data _null_;
   %m_utl_hash_define(
      table = WORK.class
    , name  = class
    , keys  = Name
    , cols  = _ALL_
    , debug = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;

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

Example 10: Step 1 - Load duplicate key data item pairs into the hash object (multidata):

```
data WORK.class;
   set SASHELP.class;
   if Name eq 'Alice' then do;
      output; Age=16; output;
   end;
run;

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

Example 10: Step 2 - Use FIND with key to get data items from hash object (returns first key):

```
data Work.result;
   %m_utl_hash_define(
      table     = WORK.class
    , name      = class
    , keys      = Name
    , cols      = _ALL_
    , multidata = Y
    , debug     = Y
      );
   rc = class.find(key: 'Alice');
run;

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

Example 10: Step 3 - Use OUTPUT to get all data items from hash object (returns all keys):

```
data _null_;
   %m_utl_hash_define(
      table     = WORK.class
    , name      = class
    , keys      = Name
    , cols      = _ALL_
    , multidata = Y
    , debug     = Y
      );
   rc = class.output(dataset: 'WORK.result');
run;

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

Example 11: Create hash iterative object using the FIRST and NEXT methods to obtain data items from hash object:

```
data Work.result;
   %m_utl_hash_define(
      table     = SASHELP.class
    , name      = class
    , hiter_flg = Y
    , hitername = hclass
    , keys      = Name
    , ordered   = Y
    , debug     = Y
      );
   %m_utl_hash_define(
      table     = SASHELP.classfit
    , name      = classfit
    , keys      = Name
    , ordered   = Y
    , debug     = Y
      );
   rc = hclass.first();
   do while (rc eq 0);
      if Sex eq 'F' then do;
         rc = classfit.find();
         output;
      end;
      rc = hclass.next();
   end;
run;

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

Copyright