Give Your MVS Users Equal Access
Scott G. Maurer, The Johns Hopkins University Applied Physics Laboratory

Abstract
The SAS® system provides many tools for users to access their data once the physical dataset is defined via the LIBNAME Statement. The LIBNAME and FNAME windows under DMS provide users access to "allocated" datasets. To allocate a new libref or fileref MVS users often find themselves exiting SAS to use utilities such as ISPF/PDF just to locate the name of the dataset. The SAS system under GUI platforms such as OS/2 provides tools for the user to scan devices and subdirectories for the desired file name.

This paper describes a tool that facilitates MVS catalog searches in a SAS full-screen environment. This tool, called MVSCT, uses SAS/C™ to interface with MVS routines. SAS/Toolkit™ is used to provide a SAS callable function capability and SAS/AF® to provide the user interface.

Introduction
The goal of this application was to provide MVS users a similar "look and feel" that GUI users, e.g., Windows, could obtain under that platform. Providing an MVS catalog search to a user involves the completion of three steps.

(1) An interface to the MVS catalog has to be developed. SAS/C was chosen because of its MVS access utilities and it is low level enough to provide quick response.
(2) A method to call this interface from a SAS session was needed. SAS/Toolkit provides the capability to pass parameters to a C function from a data step or SCL application.
(3) A method to display MVS datasets for user selection under a windowed environment was needed. Under SAS/AF, extended tables were used to accomplish this task.

The subsequent sections describe the methodology used to create this capability.

MVS Catalog Interface
Figure 1 shows the SAS/C program, written with the SAS/Toolkit interface, to perform the extraction of catalog information. "IDCAMS", an MVS Access Method Services utility, was used to extract MVS catalog information. The MVS Dynamic Allocation Supervisory Service, "SVC99", was used to allocate the required temporary files. Additionally, IDCAMS and SVC99 were used because they did not require a TSO environment. This would provide the flexibility to use this SAS callable function in non-TOO environments, such as batch.

The SAS/C System Interface Function "oslink" was used to call IDCAMS. Calling IDCAMS from a C program requires a parameter list to specify the processing options. The first and required parameter specifies any program options. Zero was used in this case. The second and optional parameter identifies the DDNAME of the file containing the search command and specifies the desired output file DDNAME. Unique DDNAME's were specified by this program to minimize side effects to other user applications. Three temporary files are allocated by this program. A search command file, "IDCAMSlN", a file for the "raw" output from IDCAMS, "IDCAMSOU" and a file (DDname specified by the AF application) for the "filtered" IDCAMS output.

Functions "svc99rl" and "svc99al", available in the SAS/C sample library, were expanded to have the ability to allocate new temporary files. The expanded code for SVC99 is not presented in this paper, but can be obtained from the author. Functions "chkvl", "bdl_flt" and "dsn_flt", shown in Figures 2 thru 4, provide error checking for the IDCAMS data set level parameter and additional filtering for IDCAMS raw output.

Figure 1. Function "mvset" and SAS/Toolkit Interface

```c
#include <stdio.h>
#include <tcl.h>
#include <os.h>
#include <uwproc>

#define NULL
#define nempty
#define nempty
#define memcmp
#define memcmp
#define memset
#define memset
#define strlen
#define strlen
#define strcat
#define strcat

typedef struct { /* Structure used for dataset filter */
```
Figure 2. Function "chkvlv"

Function "chkvlv", shown in Figure 2, performs a limited verification of the users node level request and generates a valid IDCAMS request. This program as well as IDCAMS accepts only one "wildcard" node, not in the first node. Checks are also made to insure that all node lengths are eight characters or less. If a wildcard is requested for the last node, the "chkvlv" function removes it for the IDCAMS input parameter. The function "dsn_flt" (Figure 4) will handle a wildcard character in the last node.

Figure 2. Function "chkvlv"
The Dataset Filter

The IDCAMS utility has only limited dataset name filtering capability. The functions "bld_flt" and "dsn_flt", shown in Figures 3 and 4 respectively, were written to provide expanded dataset filtering. Under IDCMAS rules, the wildcard character "*" can only be specified within a node level by itself and must not be in the last node. If IDCAMS executes successfully, the "bld_flt" routine builds dataset filtering parameters. Once the main program filters the dataset names from the IDCAMS "raw output", routine "dsn_flt" checks each dataset name with the filtering parameters generated by the "bld_flt" routine.

Figure 3. Function "bld_flt"

```c
void bld_flt(char *in_str, FSEG *fseg1, FSEG *fseg2)
```

The Dataset Filter

The IDCAMS utility has only limited dataset name filtering capability. The functions "bld_flt" and "dsn_flt", shown in Figures 3 and 4 respectively, were written to provide expanded dataset filtering. Under IDCMAS rules, the wildcard character "*" can only be specified within a node level by itself and must not be in the last node. If IDCAMS executes successfully, the "bld_flt" routine builds dataset filtering parameters. Once the main program filters the dataset names from the IDCAMS "raw output", routine "dsn_flt" checks each dataset name with the filtering parameters generated by the "bld_flt" routine.

Figure 3. Function "bld_flt"

```c
void bld_flt(char *in_str, FSEG *fseg1, FSEG *fseg2)
```
**Figure 4. Function "dsnflt"**

```c
int dsnflt(char *dsn, FSEG *fsegl, FSEG *fseg2) {
    char *nodeaddr, *nextnode;
    segstr[35], tempdsn[44] ;
    int j, strmatch=0, result=0;
    nodeaddr = dsn ;
    memset(segstr, '\0', 35);
    if (strlen(segstr) > strlen(fsegl->str)) strmatch = 1 ;
    else {
        /* str longer than level spec, next chr must be "." */
        if (fsegl->endnode != 0) strmatch = 1 ;
    }
    if (strmpseg(str, tempdsn, fsegl->str)) result += 1 ;
    if (fsegl->nodeaddr != NULL) nodeaddr = fsegl->nodeaddr;
    for (j=0; fseg2->node > j; ++j) /* find starting node */
        nextnode = strchr(nodeaddr, '\1') ;
    if (nextnode != NULL) nodeaddr = nextnode;
    else nodeaddr = strchr(nodeaddr, '\1') ;
    nodeaddr++; /* move ptr back to period or blank*/
    if (fseg2->beg > 0) for (j=0; fseg2->beg <= j; ++j) nodeaddr++ ;
    else if (fseg2->beg < 0) for (j=0; fseg2->beg < j; ++j) nodeaddr++ ;
    result += 1 ;
    if (strftime(nodeaddr, fseg2->len) == 0) result = 1 ;
    else if (strmatch == 1) result = 1 ;
    return result ;
}
```

**User Interface for the MVS Catalog**

A sample application SAS/AF program window, displayed in Figure 5, prompts the user to specify a physical dataset name. If the dataset name is not known the user can use the "select button" (shown next to the MVS dataset name field). Upon "pressing" the "select button", a screen is presented for the MVS catalog search (Figure 6).

The user enters a desired dataset level to search for dataset names from the MVS catalog. After pressing enter, a list of dataset names will then appear. The user can identify the desired dataset by putting the cursor or mouse pointer on that dataset and pressing enter. The dataset name is passed back to the SAS/AF application so a libname statement can be generated.

**Figure 5. MVS Dataset Specification Window**

![MVS Dataset Specification Window](image)

**Figure 6. MVS Catalog Search Screen**

![MVS Catalog Search Screen](image)

The SCL code used for the MVS Catalog Search AF program is shown in Figure 7. Basically, a dataset level and output filename is passed to function "mvsct". The file of extracted dataset names are then used to populate an extended table for user selection. The following explains this in more detail.

- The screen program initializes the "DSNAME Level" field with the last value specified in a previous user session. This previous dataset level is stored in a catalog entry of the user's SASUSER file. When the user exits the SCL program, the current dataset level is written to the SAS catalog entry.
- The user specified dataset level and the output DDName are passed when the SAS/C function "mvsct" is called.
The output DDName is dynamically allocated by the C program.

- Upon successful completion of the function, the output stream is copied into a SCL list with the SCL function "fillist". The "fillist" first parameter specifies the source object for the SCL list, in this case, "fileref" is specified to identify an external file. The second "fillist" parameter specifies the fileref name and the third parameter specifies the SCL list to be filled.

- The "listlen" function is used to determine if any datasets were found by the function. If the list length is greater than zero, the extended table function "setrow" will define the table size with the list length. Otherwise, the table length will be set to zero so the dataset list from the previous request will be removed.

- The return code from the "mvsct" function is used to generate a message for the user if any problems were detected. The SCL variable "msg" will display a string when control is returned to the program.

- Once the SCL list is built an extended table is populated, using the "getitem" function, in the getrow section of the SCL program.

Figure 7. SCL Code for MVS Catalog Search Screen

```
entry seldsn $44 cancel $1 :
  label ; /* when screen field changes, execute that label */
  dsid = open('sesuser.awssprof', 'u') ; /* Store the ds for */
  nodnum = varnum(dsid, 'varnode') ; /* next session use */
  rc = fetch(dsid) ; /* read the list obs */
  call putvarc(dsid, nodnum, node) ; /* save perma ds name */
  rc = update(dsid) ;
  rc = close(dsid) ;
  return :

getrow:
  dsid = getitemc(cat_list, currow) ;
  return :

putrow:
  seldsn = dsid :
  return :

node:
  mvsctrc = mvsct(node, "trcatout") ; /* P5 C/toolkit function*/
  rc = fillist("fileref", "trcatout", cat_list) ;
  if (listlen(cat_list) > 0) then call setrow(listlen(cat_list), 1, 'A') ;
  else call setrow(0, 1, 'A') ;
  select(mvsctrc) :
    when(0) msg = 'No Data Sets found for specified level by ' ||
      'IDCAMS,' :
    when(5) msg = 'No Datasets found for specified level.' ;
    when(12) msg = 'ERROR: Invalid level specification.' ;
    when(16) msg = 'ERROR: IDCAMS failure.' :
    when(30) msg = 'Ending wildcard ignored.' :
    when(50) msg = 'ERROR: Illegal wildcard in first node.' :
    when(51) msg = 'ERROR: Illegal multiple wildcard characters.' :
    when(52) msg = 'ERROR: Level spec. contains node > 8 characters.' :
  otherwise msg = 'ERROR: unknown return code from MVSCT,' :
  end /
end:

Summary

This SAS callable function used with a SAS/AF program, helps fill the MVS dataset catalog access void. The catalog list produced by this function can be used to pass a dataset name to applications that need to issue libname or filename statements. Since an interactive environment is not required, e.g. TSO, batch mode could be used to perform tasks such as catalog reporting. This function could be expanded to extract additional catalog information such as reference dates and volume id.'s. Primarily, this function gives the application programmer the capability to provide the user easy access to dataset lists for resource allocation.

References


**Trademarks**

SAS, SAS/C, SAS/AF and SAS/Toolkit are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. MVS is a registered trademark of International Business Machines Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks of their respective companies.

**Acknowledgements**

The author thanks Marty Lurz of J&S and David Wawro of E&IS for help in developing the interface with IDCAMS and SVC99.

**Author Contact**

If you have questions or comments contact:

Scott G. Maurer  
JHU/APL  
Building 8-226  
Johns Hopkins Road  
Laurel, MD 20723-6099  
(410)792-6000 x4998  
Internet: scott.maurer@jhuapl.edu