LIST AND REPRO

Programs are available to list or punch one or more copies of a card deck. The format of the EXEC card is:

// EXEC LIST[(options)]
or
// EXEC REPRO[(options)]

where any combination of the following options may be specified.

BCD - the input deck was punched on an IBM 026 keypunch and is to be translated.

n - multiple copies of the deck are to be produced (n may be any integer).

LIST - may be used only as option to REPRO. Causes a list to be produced simultaneously.

SEQ - output is to be sequenced. The beginning sequence field must immediately follow SEQ as an option. This field [length 8] will be reproduced in columns 73-80 of each card and will be incremented by 10 for each card.

For example, the following will reproduce and list a deck and translate the characters from old format to new.

// EXEC REPRO(BCD,LIST)

The following will print 10 copies of the input deck.

// EXEC LIST(10)

The following will reproduce and sequence a deck. The first card will contain PROG0010 in columns 73-80, the second will have PROG0020, etc.

// EXEC REPRO(SEQ,PROG0010)

The deck to be copied should not contain any Job Control cards. It should be placed immediately after the EXEC card and be followed /$ and /& cards.

Appropriate limits should be punched on the JOB card. No special punch key is required to use REPRO.
PROTECTION OF LIBRARIES

A user may assign a 'security code' to his library to protect it from destruction by or use by other users. This is done by a special form of the PROTECT statement (see article on 'Dataset Security' in this Newsletter). To assign a security code to a library use the following job set up:

```
//any JOB yourname
//   CHARGE projectname
//   ACCESS USERLIB(libname)
//   PROTECT USERLIB(libname)[,code]
&
```

where 'code' is as described in the above mentioned article.

DELETION OF UNWANTED LIBRARIES

If a library is no longer needed or is to be completely recreated it can be deleted by the following job set up:

```
//any JOB yourname
//   CHARGE projectname
//   ACCESS USERLIB(libname)
//   DELETE USERLIB(libname)
&
```

USING OTHER USER'S LIBRARIES

Since the user id is used internally as an extension to the user's library name, one user's library cannot be accessed by another user by methods thus far described. This can be accomplished by explicitly specifying the user id as part of the libname field on the ACCESS statement. For example to access a library named SPLIB belonging to user Z21, the following statement may be used:

```
//SYS004 ACCESS USERLIB(SPLIB|Z21)
```

Note the use of the vertical bar ('|') as a delimiter. Use of a library in this manner is of course controlled by any security code which the owner has placed on it.
To add a subprogram to an existing relocatable library, use the above job set up without statements (1) and (2).

To obtain a list of subprogram names within a single library use the following job set up:

```
//any JOB yourname
// CHARGE projectname
//SYS002 ACCESS USERLIB(libname)
// EXEC DIRECTRY
&
```

**SOURCE LIBRARIES**

A source library consists of a single stream of 80-byte card-image source statements. Since the library is kept in compressed form it cannot be directly operated upon by any program except COPYLIB. When creating a 'NEW' source library COPYLIB will operate in the same way as the system ASSEMBLER UPDATE2 feature. When operating on an existing library in the 'USE' mode COPYLIB will operate in the same way as the system ASSEMBLER UPDATE1 feature (under control of an 'edit' deck). Refer to IBM form number GC28-6811 '44PS PROGRAMMING SYSTEM ASSEMBLER LANGUAGE', pages 50-56 for a description of the UPDATE1 and UPDATE2 features.

To create a source library use the following deck set up:

```
//any JOB yourname
// CHARGE projectname
//SYS004 ACCESS USERLIB(libname),NEW
// EXEC COPYLIB(NEW,SRC)
  :
  source statements and any 'NUM' cards desired
  :
ENDUP
&
```

To use an existing source library containing, for example, a FORTRAN source program, use the following job set up:
/EXEC COPYLIB(NEW,ABS)
/

To use a program on an absolute library use the following job set up:

//any JOB yourname
//  CHARGE projectname
//SYSAB2 ACCESS USERLIB(libname)
// EXEC name

  ...

  data for program
  ...

/
/

To modify a program on an existing absolute library (note: if there is only one program on the library use the DELETE procedure followed by the NEW procedure) use the following job set up:

//any JOB yourname
//  CHARGE projectname
//SYS004 ACCESS USERLIB(libname)
// EXEC COPYLIB(USE,ABS)
// ACCESS SDSAB2
// DELETE SDSAB2(name)
//name EXEC FORTRAN

  ...

  source program and subprograms
  ...

/
/

// EXEC LNKEDIT(MAP)
/

// ERRLEV 0,EQ
// DELETE USERLIB(libname)
//SYS004 ACCESS USERLIB(libname),NEW
// EXEC COPYLIB(NEW,ABS)
/

To add a phase to an existing absolute library, use the above job set up without statements (1) and (2).

To obtain a list of program names within a single library use the following job set up:

//any JOB yourname
//  CHARGE projectname
//SYS002 ACCESS USERLIB(libname)
// EXEC DIRECTRY
/

The security code system only provides for protection from other users. It at times becomes desirable for a user to write protect a dataset from himself. Until now this was accomplished by the WRITE=NO parameter on the ACCESS or CATLG statements. This parameter is being abolished and a new parameter, "READONLY" being established for the ACCESS statement. The WRITE= parameter will be permitted but ignored. Default cataloging will no longer be supported due to its ambiguous definition. A dataset ACCESS will always be assumed writable to the owning user unless the READONLY parameter is specified explicitly in the ACCESS statement.

The format is:

`//[SYSxxx] ACCESS dsname[,vctyp=valid],READONLY`

for example to access a cataloged dataset named "MYSET" on a read-only basis use the statement

`//SYS007 ACCESS MYSET,READONLY`

or an uncataloged tape named "MYTP" on volume CU8731:

`//SYS007 ACCESS MYTP,2400=CU8731,READONLY`
A new security system for user and system datasets is being installed. It will be available for general use on or before January 14, 1974. It allows the owner of a tape or disk dataset to specify a "security code" to be associated with his dataset. A "USER" is identified by the USER ID. associated with his project name. The available security codes and their meanings are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No protection</td>
</tr>
<tr>
<td>1</td>
<td>This is a read-only dataset to anyone except the owner.</td>
</tr>
<tr>
<td>2</td>
<td>This dataset is not accessible to anyone except the owner</td>
</tr>
</tbody>
</table>

The security code for a dataset is changeable by anyone allowed to write on the dataset. If NO security code is explicitly assigned, 0 will be used.

The security code for a dataset is assigned via the PROTECT job-control statement. The format of the statement is:

```plaintext
// PROTECT  dsname[,code]
```

dsname is the name of the dataset to be protected.
(Note: an ACCESS or ALLOC for this dsname must appear somewhere in the job before the PROTECT statement).

code is the one-digit security code to be given to the dataset. If omitted, it is assumed to be "1".

If a PROTECT statement is executed for a disk dataset, the security code for that dataset is changed immediately. If a PROTECT statement is executed for a tape dataset, the security code is not assigned until the dataset is opened for output. If the tape dataset is never opened for output in the same job, the PROTECT statement is ignored.