CC

Automated List Manipulation in ISPF

V3.1

by Gil Segal, Sapiens

Table of Contents

1 Introduction

CC is an easy-to-use ISPF edit macro that allows you to easily duplicate selected sections of a skeleton file with data from another file.

Perhaps the easiest way to understand what CC does is to look at a sample skeleton and a sample data file and then to look at the results.

```
//JOBNAME  JOB
<START>
//STEP<N> EXEC ANYPROC,PARM1=<1>,PARM2=<2>
<END>
//LIST    EXEC PGM=ANYPGM
//SYSIN     DD *
<START>
<1>
```

Figure 1 Sample skeleton file

```
John Doe
Jane Doe
Jim Smith
```

Figure 2 Sample data file

Figure 1 contains a sample skeleton file, in this case a job. As you examine the skeleton, you will notice special tokens enclosed in angle brackets (<>). These tokens tell CC how to process the skeleton and where to substitute data. For example, the <1> token instructs CC to plug in the value of the first column from each row of data. The sample data file shown in Figure 2 consists of two columns delimited by one or more spaces. The result of running CC with these two sample files is shown in Figure 3 below.

```
//JOBNAME  JOB
//STEP1 EXEC ANYPROC,PARM1=John,PARM2=Doe
//STEP2 EXEC ANYPROC,PARM1=Jane,PARM2=Doe
//STEP3 EXEC ANYPROC,PARM1=Jim,PARM2=Smith
//LIST    EXEC PGM=ANYPGM
//SYSIN     DD *
John
Jane
Jim
```

Figure 3 Results of applying the sample data file to the sample skeleton file

2 Installation

1. Transfer the cc.xmit file to a temporary file on the mainframe. Use binary transfer with the following options:
   - FIXED
   - LRECL 80
   - TSO allocation parameters:
     - Primary allocation: 1
     - Secondary allocation: 1
     - Allocation units: Cylinders
     - Block size: 3120
2. In TSO, use the following command to copy the temporary file to the target library. The target library should be a REXX library that is part of your SYSPROC or SYSEXEC concatenation.

   ```
   TSO RECEIVE INDSN('<temporary-file>')
   ```

   At the prompt, enter:

   ```
   DATASET('<target-library>')
   ```

   Note that any member that previously existed in the target library with one of the names CC, CCSKEL, CCEND, or CCBATCH will be overwritten.

3 Basic Use

3.1 Setting up a data file

1. A CC data file is a text file stored in a sequential dataset or as a member of a partitioned dataset (PDS).
2. The data file consists of up to 25 columns of data, delimited by one or more spaces. If you would like to use an alternate delimiter, see the DELIM keyword on page 8.
3. Quotes (single and double) have no special meaning. A space between quotes is handled as a column delimiter like any other space.
4. The spaces are parsed within each row separately. The columns do not have to be lined up.

3.2 Setting up a skeleton file

1. A CC skeleton file is a text file which contains special CC tokens. The term “skeleton” as it used here does not refer to a valid ISPF skeleton used by the file tailoring services, but to any textual file containing CC tokens.
2. CC tokens:
   a. <START>
      The <START> token indicates the beginning of a section to be duplicated. More than one <START> token can appear in the skeleton however they cannot be nested. Each <START> token must be followed by an <END> token. If no <START> token is found in the skeleton or the first <END> token is not preceded by a <START> token, a <START> token is assumed at the top. The <START> token must be the only text on the row in which it appears.
   b. <END>
      The <END> token indicates the end of a section to be duplicated. Each <START> token must be followed by an <END> token. If no <END> token is found in the skeleton or the last <START> token is not followed by an <END> token, an <END> token is assumed at the bottom of the skeleton. The <END> token must be the only text on the row in which it appears.
   c. <1>... <25>
      These tokens are placeholders which indicate the place in the output file where data from the column corresponding to the token (i.e., <1> would be replaced by data from column 1, <2> by data from column 2, etc.) is substituted. Each <START>-<END> section of the skeleton file is duplicated for each row in the data file. The <1>...<25> tokens are substituted within each duplication using the values of the current row in the data file.
   d. <N>
      This token is replaced by the number of the current row in the data file for each duplication of a <START>-<END> section in the skeleton file.
   e. <N-1>
      This token is replaced by the number of the previous row in the data file for each duplication of a <START>-<END> section in the skeleton file. Note that the first

duplication (where N=1 and N-1=0) occurrence may need to be adjusted manually after running CC.

3. The skeleton file cannot contain any characters considered invalid by the ISPF editor.

3.3 Using CC online

1. Edit the skeleton file using the standard ISPF editor.
2. CC may be called by entering the following command on the COMMAND ==> line from within the ISPF editor:

   ```
   CC <data-file>
   ```

   where <data-file> is the full name of the data file. Do not place quotes around the data file name. If the skeleton file is a PDS member and the data file is another member in the same PDS, it is sufficient to specify the member name.
3. The resulting output file is displayed in edit mode. Upon exit from the editor, the output file is deleted and you are returned to the skeleton file edit session.

3.4 Using CC in batch

1. Call the CCBATCH REXX program using a batch ISPF job such as the following sample job:

```
//JOBNAME  JOB
//TSOISPF EXEC PGM=IKJEFT01,DYNAMNBR=50
//SYSEXEC   DD DISP=SHR,DSN=<CC-library>
//ISPMLIB   DD DISP=SHR,DSN=ISP.SISPMENU
//ISPPLIB   DD DISP=SHR,DSN=ISP.SISPPENU
//ISPTLIB   DD DISP=SHR,DSN=ISP.SISPTENU
//ISPSLIB   DD DISP=SHR,DSN=ISP.SISPSENU
//ISPPROF   DD DSN=&&PROF,UNIT=VIO,SPACE=(TRK,(1,1,1)),
//             LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB
//ISPLOG    DD DSN=&&LOG,UNIT=VIO,SPACE=(TRK,(1,1)),
//             LRECL=125,BLKSIZE=129,RECFM=VA
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
ISPSTART CMD(%CCBATCH <skeleton-file> <data-file>)
//
```

Figure 4 Sample job to run CC in batch

   where <CC-library> is the library where CC was installed.
2. The CCBATCH parameters are:
   a. <skeleton-file>
      <skeleton-file> is the full name of the skeleton file. Do not place quotes around the skeleton file name.

b. <data-file>
   <data-file> is the full name of the data file. Do not place quotes around the data file name. If the skeleton file is a PDS member and the data file is another member in the same PDS, it is sufficient to specify the member name.

3. The output file is placed in <userid>.CC.ISPFILE. If you would like to use a different output file name, see the OUT keyword on page 8.

4 Advanced Use

4.1 CC Parameters

1. Optional parameters can be given to CC by adding one or more

   ```
   <CCPARMS <keyword>=<value> <keyword>=<value>…>
   ```

   lines at the top of the skeleton file. CCPARMS lines elsewhere in the skeleton file are ignored.
2. CC parameter keywords:
   a. DELIM
      The character to be used as the column delimiter in the data file.
      Default value: ‘ ‘ (blank)
   b. GROUP
      A number or string to be used to break up the data file into groups. The skeleton file is processed in its entirety for each group. The output file is the concatenation of the output for all of the groups.
      When a number is given, the size of each group (the last one may be smaller) is that number of rows. If GROUP=0, groups are not being used and the entire data file is processed together.
      If a string is given (for example, GROUP=/*), then a group ends when a row in the data file equals the specified string. That row is not included in the output file.
      Default value: 0
   c. HDRROW
      Specifies whether the first row of the data file is a header row. If HDRROW=Y, the first row of the data file is not included in the output file.
      Valid values are Y or N.
      Default value: N
   d. LRECL
      The LRECL of the output file.
      Valid values are 80 – 255.
      Default value: 255
   e. OUT
      The full name of the output file.
      Default value: <userid>.CC.ISPFILE
   f. RESTART
      Specifies whether <N> and <N-1> restart for each group. The starting value is specified by the START keyword. If RESTART=Y, the numbering for each group starts from the starting value specified by the START keyword. If RESTART=N, then all of the rows are numbered consecutively regardless of their group.
      Valid values are Y or N.
      Default value: Y

g. START
   The starting value for <N>.
   Default value: 1
h. STRIP
   Specifies whether leading and trailing blanks are removed from the column data. If STRIP=Y, then all leading and trailing blanks are removed. If STRIP=N, then all leading and trailing blanks are preserved. When DELIM=' ' (blank), the value of STRIP is ignored and all blanks are removed.
   Valid values are Y or N.
   Default value: N

4.2 Advanced Tokens

1. <G>
   This token is replaced by the number of the current group in the data file. Groups are numbered consecutively from 1. If groups are not being used (i.e., GROUP=0), <G> will have the value 1.
2. <D>
   This token is replaced by the full dataset name of the data file. If the data file resides in a PDS, <D> will not include the member name.
3. <M>
   This token is replaced by the member name of the data file when the data file resides in a PDS. If the data file does not reside in a PDS, <M> will have a null value.
4. <G:n>, <N:n>, <N-1:n>
   When :n is specified as part of a <G>, <N>, or <N-1> token, the value of the token is padded or truncated to n digits as needed. Each <G>,<N>, or <N-1> token can have a different value for n.
   Valid values for n: 1 – 8.

4.3 Example

The following example demonstrates use of some of the advanced features discussed above.

Skeleton file:

```
<CCPARMS DELIM=~ GROUP=/* RESTART=N>
//JOB<G:5> JOB
<START>
//STEP<N:4> EXEC ANYPROC,PARM1='<1>',PARM2='<2>'
<END>
//
```

Figure 5 Advanced features example skeleton file

Data file:

```
Fred Flintsone~Bedrock~
Wilma Flintsone~Bedrock~
Barney Rubble~Bedrock~
Betty Rubble~Bedrock~
/*
Homer Simpson~   Springfield   ~
Marge Simpson~   Springfield   ~
Bart Simpson~   Springfield   ~
/*
Yogi Bear~Jellystone Park~
Ranger Smith~Jellystone Park~
```

Figure 6 Advanced features example data file

Output file:

```
//JOB00001 JOB
//STEP0001 EXEC ANYPROC,PARM1='Fred Flintsone',PARM2='Bedrock'
//STEP0002 EXEC ANYPROC,PARM1='Wilma Flintsone',PARM2='Bedrock'
//STEP0003 EXEC ANYPROC,PARM1='Barney Rubble',PARM2='Bedrock'
//STEP0004 EXEC ANYPROC,PARM1='Betty Rubble',PARM2='Bedrock'
//
//JOB00002 JOB
//STEP0005 EXEC ANYPROC,PARM1='Homer Simpson',PARM2='   Springfield   '
//STEP0006 EXEC ANYPROC,PARM1='Marge Simpson',PARM2='   Springfield   '
//STEP0007 EXEC ANYPROC,PARM1='Bart Simpson',PARM2='   Springfield   '
//
//JOB00003 JOB
//STEP0008 EXEC ANYPROC,PARM1='Yogi Bear',PARM2='Jellystone Park'
//STEP0009 EXEC ANYPROC,PARM1='Ranger Smith',PARM2='Jellystone Park'
//
```

Figure 7 Advanced features example output file

5 Usage Tips

1. <N:n> is useful for building job step names.
2. When <N:n> is used to build job step names, <N-1:n> can be used to do condition code checking on the results of the previous job step.
3. When using groups, <G:n> is useful to build unique job names for each group.

If you have any questions or comments, contact me at gil.s@sapiens.com.