IBM Content Manager OnDemand for i

Postprocessor Parameters

For servers running IBM i v7.2 and higher

February 11, 2022

Using Postprocessor Parameters with Content Manager OnDemand for i

This document describes the steps necessary to enable the use of postprocessor parameters within a Content Manager OnDemand application definition. It is extremely important to understand that postprocessor parameters can ***modify*** the index data after it has been extracted from the pages of data to be archived, but before it is written to the Content Manager OnDemand database. You must use caution when using a postprocessor parameter, and test your parameters thoroughly before using them in a production environment.

The postprocessor parameters can be used to run different programming facilities of IBM i. These facilities are: an ILE program, a qshell command, or a Stream EDitor (SED) script. This document covers all three options.

In this document, a complete explanation of the setup and testing of postprocessor parameters is included under the *ILE Program* topic. Note that steps 1, 2, 3, 6 and 8 of the setup and testing also apply to qshell commands and SED scripts.

ILE Program

The ILE postprocessor program is a custom-written program that can be written in ILE RPG, ILE COBOL, or ILE C. The program runs against a tab-delimited stream file that contains the index data extracted from the documents that are to be archived. The ILE RPG and ILE COBOL programs call C-functions which allow access and updating of the stream file that contains the index data. You must have some knowledge of programming to successfully complete the ILE program implementation.

How do I begin?

First, be sure that the current PTFs for Content Manager OnDemand for i are installed on your system.

Then, determine the programming language you will use for your postprocessor program. There are three sample postprocessor programs to help you create your own program. More detail about these sample programs can be found later in this document under the topic entitled “Write your postprocessor program.”

Once you have determined which programming language you will use, a sample input file should be generated. This file is needed for program creation and debugging. Information on creating the sample file can be found under the topic entitled “Create your sample input file” later in this document.

After your sample input file has been created, you can write and compile your postprocessor program and create a symbolic link for it. Then test it, first as a standalone program, and then

as part of the Add Report (ADDRPTOND) command process. Additional information on all of these topics can be found later in this document.

Header Row

The stream file used for loading indexes into Content Manager OnDemand includes a header row containing the field names.

The stream file contents might look like this:

```
< WORKDATE BADGENO EMPLNAME EMPLNO TOTALTIME VER >
14483 5102 F MCNEICH E102 8:00 01 1FAAA 0 1798 0 40033 N O 0 5 0
14483 5104 J JANKOWSKI E104 8:00 01 1FAAA 1798 1798 0 40033 N O 0 5 0
14483 5157 D THOMPSON E157 :03 01 1FAAA 3596 2263 0 40033 N O 0 5 0
```

Setup instructions

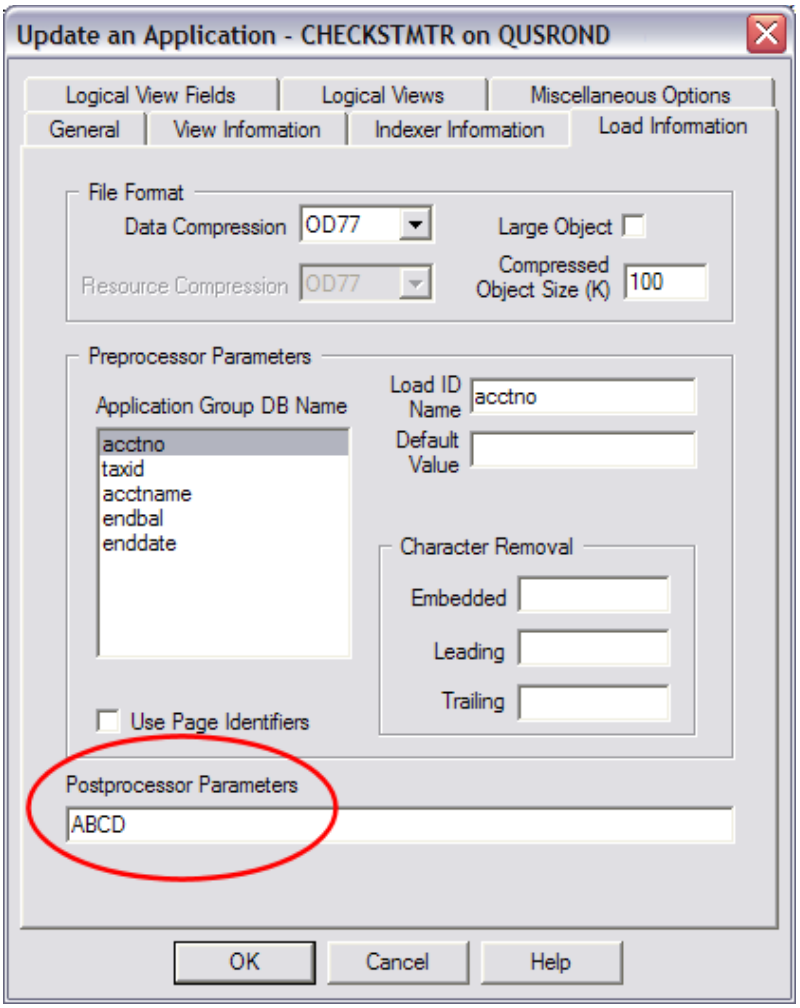
The steps required to implement a postprocessor program are listed below, followed by detailed instructions for each step:

1. Update your Content Manager OnDemand application definition in preparation for the creation of your sample input file
2. Create your sample input file
3. Write your postprocessor program
4. Create (compile) your postprocessor program
5. Create a symbolic link for your postprocessor program
6. Test your postprocessor program by using your sample input file
7. Update your Content Manager OnDemand application definition to contain your postprocessor program's symbolic link
8. Test your postprocessor program by using the Add Report (ADDRPTOND) command

Step 1: Update your Content Manager OnDemand application definition in preparation for the creation of your sample input file

Using the OnDemand Administrator client, update the application definition for which you are writing your postprocessor program. Be careful not to update an application definition that is already in production, since the change you are about to make will affect the storing of data for this application. Export your definition to a test instance or to a test system to avoid any impact to production.

Locate the Postprocessor Parameters field on the Load Information tab of your Content Manager OnDemand application definition. Enter the characters ABCD. This is a meaningless value that will force the load process (using the ADDRPTOND command) to fail, which will cause a sample input file to be created for testing. See the following step for more details.



Step 2: Create your sample input file

Once the Postprocessor Parameters field of the application definition has been updated, run the Add Report (ADDRPTOND) command using an appropriate spooled file, database file, or stream file as input. The ADDRPTOND command will fail because of the meaningless value (ABCD) you have entered in the Postprocessor Parameters. This process will leave a sample input file in the IFS directory of the user's home directory (see Note 1).

Note 1: The default (home) directory can be determined by entering the Work with Object Links (WRKLNK) command (with no parameters specified) on an IBM i command line while signed on with the user profile that runs the ADDRPTOND command.

The default directory is also shown in the Home directory (HOMEDIR) field of the user profile, which can be determined using the Display User Profile (DSPUSRPRF) command. This directory is not automatically created when the user profile is created. If you need to create this directory, use the Create Directory (CRTDIR or MKDIR) command. If the default directory needs to be changed, use the Change User Profile (CHGUSRPRF) command to change the default directory.

Below are the naming rules for the file that is created when you run the ADDRPTOND command that fails. The file that is created is the index file that would be posted to Content Manager OnDemand if the postprocessor program was not run. It is also the sample input file for your postprocessor program.

- When the INPUT parameter of the ADDRPTOND command is specified as *FILE, then the resulting index file is named

  ```
  /DefaultDirectory/DB_Library_Filename_Member.ind.db
  ```

  where **DefaultDirectory** is the name of the home directory described above, **Library** is the name of the library that contains the input physical file, **Filename** is the name of the input physical file you entered as the FILE parameter, and **Member** is the file member used. For example:

  ```
  /home/myuserprf/DB_ONDSAMPLES_CHECKSTMTS_CHECKSTMTX.ind.db
  ```

- When the INPUT parameter of the ADDRPTOND command is specified as *SPLF, then the resulting index file is named

  ```
  /DefaultDirectory/SP_SplfName_JobName_UserName_JobNumber_SplfNumber_JobSysName_
     SplfDate_SplfTime.ind.db
  ```

  where **DefaultDirectory** is the name of the home directory described above, **SplfName** is the name of the spooled file you entered for the SPLF parameter, **JobName** is the name of the job, **UserName** is the name of the user associated with the spooled file, **JobNumber** is the number of the job you entered for the JOB parameter, **SplfNumber** is the spooled file number within the job you entered for the SPLNBR parameter, **JobSysName** is the system name where the spooled file was created that you entered for the JOBSYSNAME parameter, **SplfDate** is the creation date of the spooled file, and **SplfTime** is the creation time of the spooled file you entered as the CRTDATE parameter on the ADDRPTOND command. For example:

  ```
  /home/myuserprf/SP_CKSTMTPRTF_DSP02_MYUSERPRF_026768_000019_MYSYSTEM_1210811_10
     5538.ind.db
  ```

  Perhaps an easier way to relate the many parts of the SP_ name is to look at the ADDRPTOND parameters as they would appear on the screen as you enter them:

  ```
  Spooled file . . . . . . . . . . > CKSTMTPRTF
  Job name . . . . . . . . . . . . > DSP02
    User . . . . . . . . . . . . .   MYUSERPRF
    Number . . . . . . . . . . . .   026768
  Spooled file number  . . . . . . > 19
  Job system name  . . . . . . . . > MYSYSTEM
  Spooled file created:
    Creation date  . . . . . . . . > 08112021
    Creation time  . . . . . . . . > 105538
  ```

Note that the Creation Date value is changed slightly during processing. In the example, the Creation Date for the spooled file, which is entered on the ADDRPTOND command is 08112021, which is August 11, 2021. The Creation Date value that appears as part of the resulting index file is 1210811. The new format is CYYMMDD (century.year.month.day) where C equals 0 for 19xx years and C equals 1 for the year 2000 and greater.

- When the INPUT parameter of the ADDRPTOND command is specified as *STMF (such as a PDF file), then the resulting index file is named

  ```
  /DefaultDirectory/Filename.ind.db
  ```

  where **DefaultDirectory** is the name of the home directory described above, and **Filename** is the name of the input stream file you entered as the FILE parameter on the ADDRPTOND command. For example:

  ```
  /home/myuserprf/AP.invoices.pdf.ind.db
  ```

IMPORTANT: After it is created, you should make a backup copy of the sample input file (the file that ends in .ind.db), to assure that you do not lose the data prior to completing your testing.

Step 3: Write your postprocessor program

Sample programs are provided with Content Manager OnDemand and can be found in the QSAMPLES2 source file in the QRDARS library. Currently, the following samples are available:

```
CHECKSTMTB   CBLLE        SAMPLE POSTPROCESSING PROGRAM
CHECKSTMTC   C            SAMPLE POSTPROCESSING PROGRAM
CHECKSTMTR   RPGLE        SAMPLE POSTPROCESSING PROGRAM
```

The sample source code for the programming language you have chosen should be copied to the QSAMPLES2 source file in library QUSRRDARS. Modifications should not be made to the source file in the QRDARS library. The examples in the QRDARS library may be replaced in a future PTF or release.

When writing your program, it is extremely important that you understand the location and format of the data that will be modified. Use the sample input file created in a previous step for this purpose. The field positions within the record layout MUST NOT be modified. Modification of the position of field data may cause the load to fail or may cause invalid index data to be loaded into Content Manager OnDemand.

This is an example of the tab-delimited record used in the sample programs provided:

```
140172594→011-0794357→PIZZA PLACE→$     1,275.66→7658→CBL→1FAAA→0→8081→0→
   68379→N→O→0→1→0(CR)
```

Every position of the record must be accounted for in the file that is written out. In the above sample record, the → denotes the tab between each field, and (CR) denotes carriage return.

Step 4: Create (compile) your postprocessor program

To compile your postprocessor program, modify one of the commands shown below for the programming language you have chosen. For example, use the Create Bound RPG Program (CRTBNDRPG) command and parameters if you are creating an ILE RPG postprocessor program. Be sure to specify all parameters as shown, modifying only the program name/library and source member/file/library as needed.

For CBLLE:

```
CRTBNDCBL PGM(CHECKSTMTR)        <substitute your program name here>
          ACTGRP(*CALLER)
          BNDDIR(QC2LE)
          REPLACE(*YES)
          DBGVIEW(*ALL)          <use if running debug>
```

For C:

```
CRTBNDC PGM(CHECKSTMTR)          <substitute your program name here>
        SYSIFCOPT(*IFSIO)
        OUTPUT(*YES)
        REPLACE(*YES)
        DBGVIEW(*ALL)            <use if running debug>
```

For RPGLE:

```
CRTBNDRPG PGM(CHECKSTMTR)        <substitute your program name here>
          DFTACTGRP(*NO))
          ACTGRP(*CALLER)
          BNDDIR(QC2LE)
          REPLACE(*YES)
          DBGVIEW(*ALL)          <use if running debug>
```

Step 5: Create a symbolic link for your postprocessor program

A symbolic link must be created in IFS on your IBM i system to point to your postprocessor program. The name of this symbolic link is the value that will be entered (to replace the ABCD value entered earlier) in the Postprocessor Parameters field on the Load Information tab of your Content Manager OnDemand application definition. Use the Add Link (ADDLNK) command to create this symbolic link. The format of the ADDLNK command is:

```
ADDLNK OBJ('/QSYS.LIB/yourlib.LIB/yourpgm.PGM') NEWLNK('/usr/bin/yourpgm')
```

For example:

```
ADDLNK OBJ('/QSYS.LIB/QGPL.LIB/CHECKSTMTR.PGM') NEWLNK('/usr/bin/CHECKSTMTR')
```

In this example, the name you would enter in the Postprocessor Parameters field of your application definition is CHECKSTMTR.

Step 6: Test your postprocessor program by using your sample input file

To test your postprocessor program, you must use the QSHELL environment on your IBM i system. To start the QSHELL environment, enter the Start QSH (STRQSH or QSH) command from an IBM i command line. Once in the QSHELL environment, enter the following command:

```
/usr/bin/pgmname </DefaultDirectory/inputfilename
   >/DefaultDirectory/outputfilename
```

For example, for *FILE processing:

```
/usr/bin/CHECKSTMTR </home/myuserprf/DB_ONDSAMPLES_CHECKSTMTT_CHECKSTMTS.ind.db
   >/home/myuserprf/DB_ONDSAMPLES_CHECKSTMTT_CHECKSTMTS.out.db
```

For example, for *SPLF processing:

```
/usr/bin/CHECKSTMTR
   </home/myuserprf/SP_CKSTMTPRTF_DSP02_DBRYANT_026768_000019_MYSYSTEM_1060811_
   105538.ind.db
   >/home/myuserprf/SP_CKSTMTPRTF_DSP02_DBRYANT_026768_000019_MYSYSTEM_1060811_
   105538.out.db
```

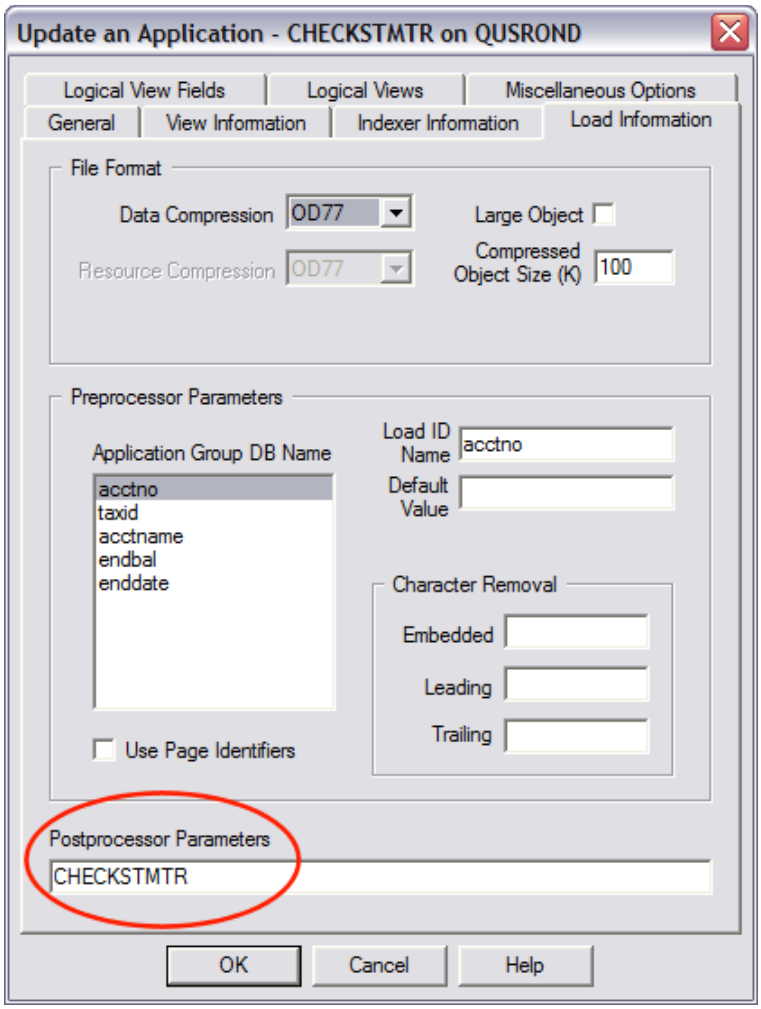
For example, for *STMF processing:

```
/usr/bin/CHECKSTMTR </home/myuserprf/AP.invoices.pdf.ind.db
   >/home/myuserprf/AP.invoices.pdf.out.db
```

You should display the output file to see the results of your program. If the results are not what you expect, modify the postprocessor program and rerun your test. Look very carefully at the results of your test. Errors in your postprocessor program will result in incorrect index data being loaded into Content Manager OnDemand when the postprocessor program is put into production.

Step 7: Update your Content Manager OnDemand application definition to contain your postprocessor program's symbolic link

Using the OnDemand Administrator client, update the Postprocessor Parameters field on the Load Information tab of your Content Manager OnDemand application definition to name the symbolic link that points to your postprocessor program. This will replace the meaningless ABCD value you entered earlier.



Step 8: Test your postprocessor program by using the Add Report (ADDRPTOND) command

The postprocessor program must now be tested with Content Manager OnDemand. Run the ADDRPTOND command to load your data into Content Manager OnDemand. After the load completes successfully, view the data using the OnDemand end-user client and validate the changes that your postprocessor program made to the index data. If the index data has been modified correctly, you have successfully implemented your postprocessor program. If the modified index data is not correct, you should delete the incorrect data you just stored in Content Manager OnDemand (by using the Remove Report (RMVRPTOND) command) and then return to the step entitled “Test your postprocessor program using your sample input file” in these instructions to retest your postprocessor program outside of Content Manager OnDemand using the QSHELL environment and your sample input file.

Optional: Pass the index file name to the postprocessor program

The postprocessor program can be modified to accept a parameter containing the name of the file being processed by Content Manager OnDemand.

To pass the parameter, create data area named ARSLOADEXT in library QUSRRDARS. The data area can be any length up to 500 bytes. The contents of the data area are not meaningful; in fact, it can be empty. The data area is global in scope, which means that once it is created, it affects all instances. For example:

```
CRTDTAARA DTAARA(QUSRRDARS/ARSLOADEXT) TYPE(*CHAR) LEN(10) TEXT('Pass file name
   to postprocessor')
```

Existing RPG and C programs tolerate the parameter and will simply ignore it if it is passed along with the program call without the program being coded to expect it. Existing COBOL programs will fail if they are not changed to accept the parameter.

The value contained in the passed parameter varies based on the type of file being archived. See below for more information.

Spooled File

For spooled file input, the parameter value contains the following attributes of the spooled file: file name, job, system, date, and time. The elements of the parameter value are separated by the forward slash (/) character. Our example parameter value is:

```
*SPLFCKSTMTPRTF/DSP02.JLSHERR.050349/000018/RDR400C/1210511/113736
```

The first element, *SPLFCKSTMTPRTF, is the file type and spooled file name. The file type is *SPLF for a spooled file; the spooled file name is CKSTMTPRTF in our example.

The second element is the qualified job name. The three parts of the job are job name, user, and number, delimited by periods (.).

The third element is spooled file number.

The fourth element is system name.

The fifth element is date in CYYMMDD format.

The sixth element is time in HHMMSS format.

These elements can be used to access all the spooled file attributes using IBM i APIs.

Database File

For database file input, the parameter value passed contains the file, library, and member names. Our example parameter value is:

```
*DBFMQRDARS/CHECKSTMTS/CHECKSTMTS
```

The first element, *DBFMQRDARS, is the file type (*DBFM) and library name.

The second element is the file name.

The third element is the member name.

Stream File

For stream file input, the parameter value passed contains the file name and directory path. Our example parameter value is:

```
*STMF/mydir/myfile.file
```

The first element, *STMF, is the file type.

The second element is the directory. The directory may consist of multiple elements, for example /home/dbryant.

The third element is the file name.

Code Snippets

COBOL

```
WORKING-STORAGE SECTION.
01  INPUT_PARM                 PIC X(256) VALUE ' '.
LINKAGE SECTION.
01  INPARM                           PIC X(256).
PROCEDURE DIVISION USING INPARM.
MOVE INPARM TO INPUT_PARM.
```

RPG

```
C                   PARM                    INPARM          256
```

C

```
/* BEGIN MAIN PROGRAM LOGIC                          */
void main(int argc, char *argv[])
/* Enter work fields here                            */
char INPARM[256];
/* Begin standard input processing                   */
strncpy (INPARM, argv[1], 256);
```

Qshell commands

Qshell commands can be run directly as postprocessor parameters.

tr command

The tr command is a qsh command line utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.

Example:

An example of where to use the tr command is to remove an unknown number of extra blanks from unknown data. The tr -s option squeezes repeating characters down to a single character.

As an example, the names in a report are printed as follows:

```
BOB         TEJAS
DEKE T      SLAYTON
EDGAR D     ROSS
HAROLD      SPONSON
```

The postprocessor parameter to use tr to remove extra blanks is:

```
"tr -s ' '"
```

After processing the index values look like this:

```
BOB TEJAS
DEKE T SLAYTON
EDGAR D ROSS
HAROLD SPONSON
```

SED scripts

What is SED?

Stream **ED**itor - A Unix text editor that processes an entire file, a powerful but cryptic command language based on regular expressions.

What are regular expressions?

regexp, regex, RE - One of the wild card patterns used by Unix utilities such as grep, sed and awk and editors such as vi and Emacs. A regular expression is a sequence of characters. The exact details of how regex will work in a given application vary greatly from platform to platform. You will have to carefully test your scripts in qshell before using them in production.

A (very) little more on regular expressions

First, you need to consider carefully what you want to match and whether you can deal with false matches. Second, you need to test the regex on sample data. Deciding what to match is a trade-off between making false matches and missing valid matches. If your regex is too strict, it will miss valid matches. If it is too loose, it will generate false matches. Once the regex is released into live code, you probably will not notice either way.

If you have questions on SED or regular expressions, do NOT call the support line. You are responsible for understanding and correctly implementing postprocessor parameters, programs, commands, and scripts.

Specifying the SED Script

The postprocessor parameters to execute a sed script:

```
sed -f /path/script.sed
```

Examples

The following examples are provided to give you some ideas about how SED scripts can be used as a postprocessor. These examples are provided without any warranty or guarantee as to their fitness for any purpose.

SED – Example 1

In the report, an account number of 99999 is used to indicate the totals page, but the account name is left blank on that page.

The names in a report are printed as follows:

```
27    FRITZ WALTER BAD NAUHEIM HAUPTSTRASSE 11
219   SPARKASSE KIEL KIEL LORENTZENDAMM 22-30
1937  STROHKARK, ROLF WEDEL ROLANDSTRASSE 14
99999
```

The SED script locates the string 99999, then replaces the first consecutive 6 blanks with the word TOTALS.

```
/99999/s/      /TOTALS/
```

After processing with SED, the index values look like this:

```
27    FRITZ WALTER BAD NAUHEIM HAUPTSTRASSE 11
219   SPARKASSE KIEL KIEL LORENTZENDAMM 22-30
1937 STROHKARK, ROLF WEDEL ROLANDSTRASSE 14
99999 TOTALS
```

SED – Example 2

This example demonstrates how to remove an unknown number of extra blanks from unknown data. The names in a report are printed as follows:

```
BOB         TEJAS
DEKE T      SLAYTON
EDGAR D     ROSS
HAROLD      SPONSON
```

The SED script locates 2 spaces, replaces 2 spaces with 1 space, then repeats this until reaching the end of line, and does this multiple times for each line.

```
/  /s/  / /g
/  /s/  / /g
/  /s/  / /g
/  /s/  / /g
```

After processing with SED, the index values look like this:

```
BOB TEJAS
DEKE T SLAYTON
EDGAR D ROSS
HAROLD SPONSON
```

SED – Example 3

This example shows how to flip last names and first names. If present, a middle initial is handled. The names in a report are printed as follows:

```
BENNAT, BROOKE R
BLACK, TYLOR J
CARR, HAY T
CHEVEER, ANGEL
```

The SED script flips the names including a middle initial, then it flip the names without a middle initial. All names are in UPPER CASE. First and last names contain at least 2 characters.

```
s/\([A-Z][A-Z]*\), \([A-Z][A-Z]* [A-Z]\)/\2 \1/
s/\([A-Z][A-Z]*\), \([A-Z][A-Z]*\)/\2 \1/
```

After processing with SED, the index values look like this:

```
BROOKE R BENNAT
TYLOR J BLACK
HAY T CARR
ANGEL CHEVEER
```

SED – Example 4

In this example, a customer has multiple No Index reports. (A No Index report contains no index values except the name of the report and the date it was archived.) All these reports use QSYSPRT with form type *STD. The customer has no access to program source code. The report title is on the first page of the report and will be used as an index.

The report titles are printed as follows:

```
A D D R E S S   C O R R E C T I O N   R E P O R T
```

The customer wants to remove embedded spaces, but leave one space between words.

Any leading and trailing blanks can be removed in the Application > Load Information tab.

The SED script finds the 'letter space' patterns, then keeps the letter & discards the space. This is repeated to end of line. Next, the SED script finds 2 spaces, then replaces 2 spaces with 1 space, and repeats this to end of line. All names are in UPPER CASE.

```
s/\([A-Z]\)\([ ]\)/\1/g
/  /s/  / /g
```

After processing with SED, the report title index value looks like this:

```
ADDRESS CORRECTION REPORT
```

References

Some of the available references for SED and regular expressions are:

Books

Jeffrey E.F. Friedl, *"Mastering Regular Expressions"*, Third Edition, O'Reilly
https://www.oreilly.com/library/view/mastering-regular-expressions/0596528124/

Dale Dougherty, Arnold Robbins "*sed & awk*", Second Edition, O'Reilly
https://www.oreilly.com/library/view/sed-awk/1565922255/

On the Internet

https://www.regular-expressions.info/

https://regexlib.com/

http://sed.sourceforge.net/grabbag/

https://www.opengroup.org/onlinepubs/007908799/xbd/re.html

http://www.grymoire.com/Unix/Sed.html

https://cheatography.com/davechild/cheat-sheets/regular-expressions/

and many more...