Chapter 24. December 2008

Welcome to the December 2008 edition of IBM InfoSphere Information Server Developer’s Notebook. This month we answer the question;

How do I perform looping, and before/after record processing within QualityStage custom rule sets?

Excellent question! This edition of (IBM) InfoSphere Information Server Developer’s Notebook (IISDN) is the last in the current series on the topic of QualityStage custom rules sets, and covers advanced topics, including; looping, before and after row processing, unhandled data, null record output, and more.

In the example put forth in this document, we process a large structured and custom record format, much as we would the PDF and HTTP file read examples from months prior. We use the looping and before/after record processing capabilities of QualityStage rule sets, to read data structures of these types.

Software versions

All of these solutions were *developed and tested* on (IBM) InfoSphere Information Server (IIS) version 8.1, using the Microsoft Windows XP/SP2 platform to support IIS client programs, and a RedHat Linux Advanced Server 4 (RHEL 4) FixPak U6 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server side components.

IBM InfoSphere Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM InfoSphere Information Server product contains the following major components;

WebSphere Business Glossary Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, Rational Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM InfoSphere Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

24.1 Terms and core concepts

In the two prior editions of (IBM) InfoSphere Information Server Developer's Notebook (IISDN), October/2008 and November/2008, we covered QualityStage related topics; more specifically, the Investigate and then Standardize stages. Also in the November/2008 edition, we covered QualityStage custom rule sets, including;

- How to create a new QualityStage custom rule set, how to provision same, and how to use this new custom rule set inside an Information Server DataStage/QualityStage component Job.
- How to work with 3 of the components to a QualityStage rule set, specifically; Classification files, Dictionary files, and Pattern Action files.
- We created two new QualityStage custom rule sets-
  - A simple rule set to standardize a company name.

    This was a simple rule, removing common words like (The, A, and An) from the leading portion of a name; this example being used to demonstrate all of the surrounding techniques of creating a new rule set, testing it, and making use of it within a Job.
  - A simple rule set to standardize a U.S. based telephone number.

    This example got more into the Pattern Action language; more pattern action language commands, as well as simple and user-defined pattern classes.

    By example, we also detailed the general scheme behind standardizing data; How to structure a custom rule, how to approach the tokenization of data and related.

In this edition of IISDN, we build the third and final custom rule set (for a while anyway). This rule set demonstrates looping, and before/after row processing. (Before and after each larger input record that is sent to the custom rule set.)

Note: This is one of the few editions of IISDN where you really must read and master a prior edition; specifically in this case, you must read and understand the November/2008 edition of IISDN, QualityStage Custom Rule Sets I.

Figure 24-1 displays the sample data and sample Job used in this example. Comments related to this sample data include;

- Figure 24-1 displays a subset, and the hardest part, of a production rule set we encountered. In effect, known column names are presented in the format;

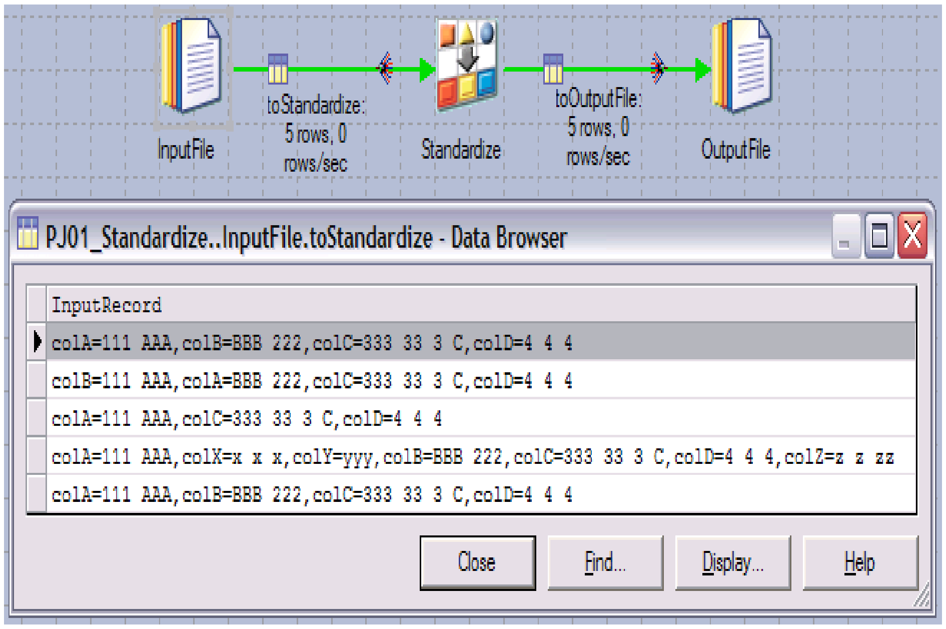
    colN=Some multi-word value

– The approved/accepted column names are from a list of known column names. The examples used in Figure 24-1 include colA, colB, colC, and colD. And these column name, column value pairs are separated by commas. (The last column name, column value pair ends with no comma.)

– Known column name, column value pairs may appear in any order. So data may appear colA, colB, colC, colD, or colD, colB, colA, colC, or whatever.

  And previously unknown column names must be sent to an "UnhandledData" column. Examples displayed in Figure 24-1, include; colX, colY, and colZ, but any unknown column name should produce this behavior.

– Lastly, any output column values should be defaulted to some value when NULL.


InputFile
toStandardize: 5 rows, 0 rows/sec
Standardize
toOutputFile: 5 rows, 0 rows/sec
OutputFile
PJ01_Standardize..InputFile.toStandardize - Data Browser
Close
Find...
Display...
Help

```
InputRecord
colA=111 AAA,colB=BBB 222,colC=333 33 3 C,colD=4 4 4
colB=111 AAA,colA=BBB 222,colC=333 33 3 C,colD=4 4 4
colA=111 AAA,colC=333 33 3 C,colD=4 4 4
colA=111 AAA,colX=x x x,colY=yyy,colB=BBB 222,colC=333 33 3 C,colD=4 4 4,colZ=z z zz
colA=111 AAA,colB=BBB 222,colC=333 33 3 C,colD=4 4 4
```

Figure 24-1 Sample data set used in this example, sample Job.

24.2 Create this example

In order to create the example QualityStage custom rule set detailed above, complete the following steps;

Remember to use consult the prior month's edition of IISDN as a guide.

1. Using your favorite editor, create a sample data set with the conditions as displayed in Figure 24-1, and accompanying text.
2. Log on to the DataStage/QualityStage Designer program.

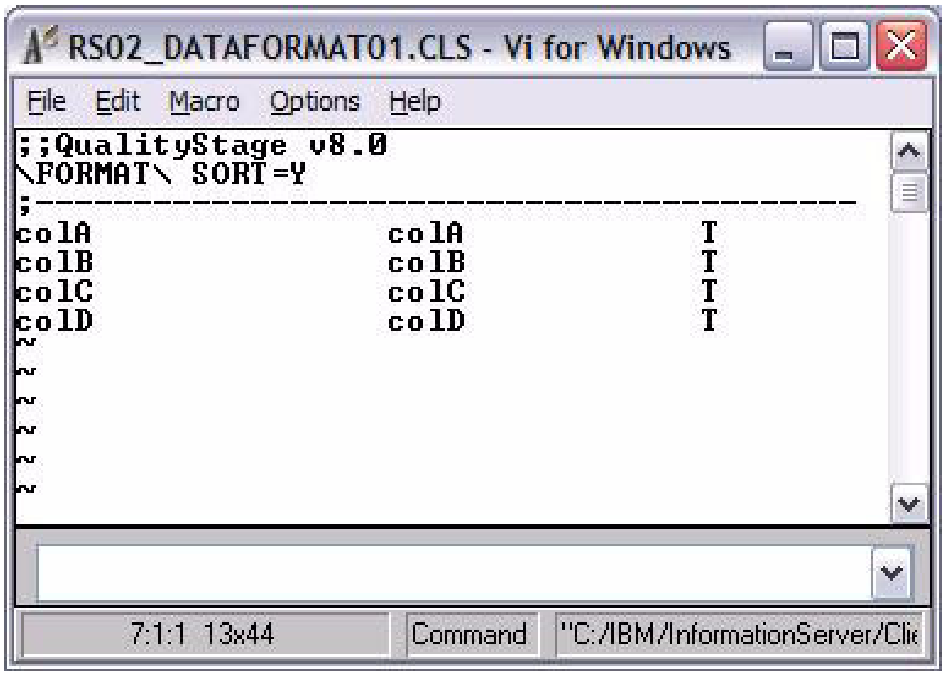
   Create a new custom rule set, and then a new Parallel Job that uses this new custom rule set. The input data source should output one column, which is processed by the Standardize stage displayed in Figure 24-1. To accomplish this, the Format TAB of the Sequential File stage should include;

   – Field defaults -> Quote -> None.

   – Field defaults -> Delimiter -> None.

   – Record level -> Final delimiter -> End.

   – Record Level -> Record delimiter -> UNIX newline.
3. Alter the contents of the Classification file to equal that as displayed in Figure 24-2. A code review follows.


RS02_DATAFORMAT01.CLS - Vi for Windows
File Edit Macro Options Help
7:1:1 13x44
Command
"C:/IBM/InformationServer/Cli

```
;;QualityStage v8.0
\FORMAT\ SORT=Y
;----------------------------------------
colA                    colA            T
colB                    colB            T
colC                    colC            T
colD                    colD            T
~
~
~
~
~
~
```

Figure 24-2 Classification File for this example.

A code review for Figure 24-2 includes;

– Basically we enter each known column name value in this file, and give it the user-defined pattern value of "T".

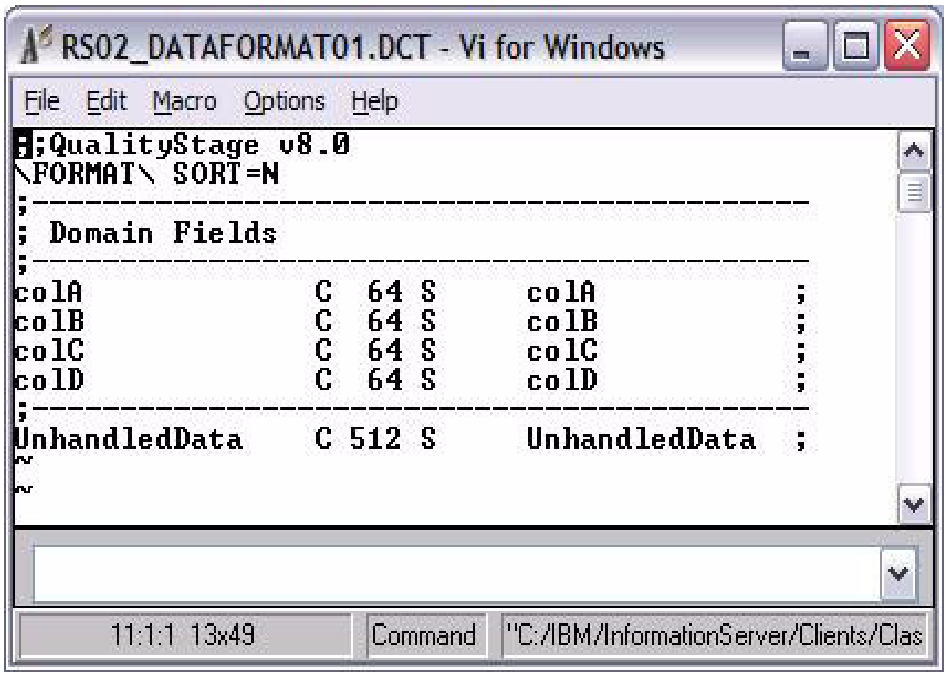
– The single advantage of this step is that we may later test for any known column name by looking for "T". This as opposed to saying something like,

  {psuedo code}

  if ((col = "colA") or (col = "colB") or (col = "colC") ... etcetera

  We say single advantage, because the Classification file can provide other services, just not any we need for this example.

4. Alter the contents of the Dictionary file to equal that as displayed in Figure 24-3. A code review follows.


RS02_DATAFORMAT01.DCT - Vi for Windows
File Edit Macro Options Help
11:1:1 13x49
Command
"C:/IBM/InformationServer/Clients/Clas

```
;;QualityStage v8.0
\FORMAT\ SORT=N
;--------------------------------------------
; Domain Fields
;--------------------------------------------
colA                C  64 S     colA         ;
colB                C  64 S     colB         ;
colC                C  64 S     colC         ;
colD                C  64 S     colD         ;
;--------------------------------------------
UnhandledData       C 512 S     UnhandledData  ;
~
~
```

Figure 24-3 Dictionary File for this example.

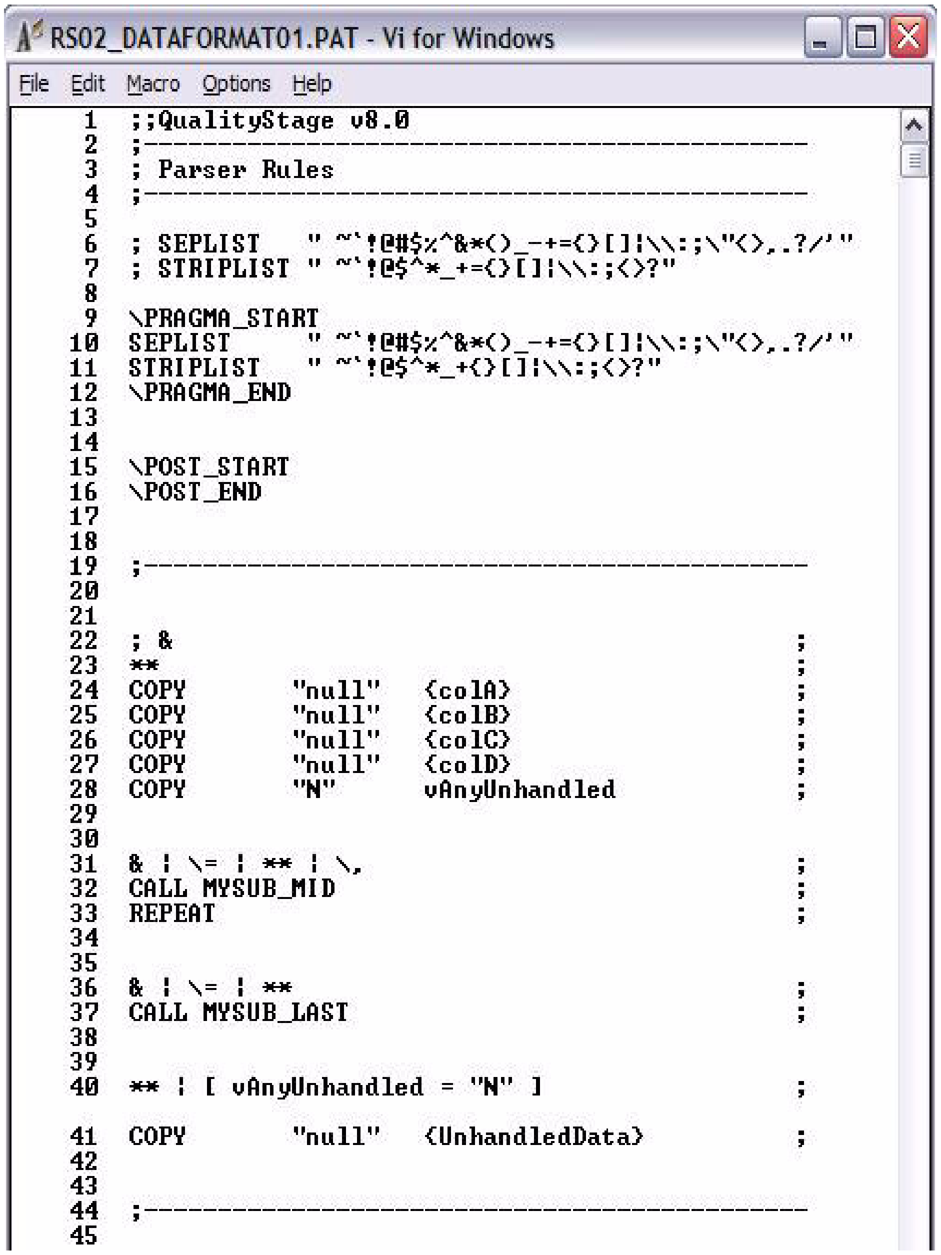
A code review for Figure 24-3 includes;

– Basically this dictionary outputs 5 columns; colA, colB, colC, colD, and UnhandledData.

– UnhandledData is not a keyword, but is an output column name that is commonly used.

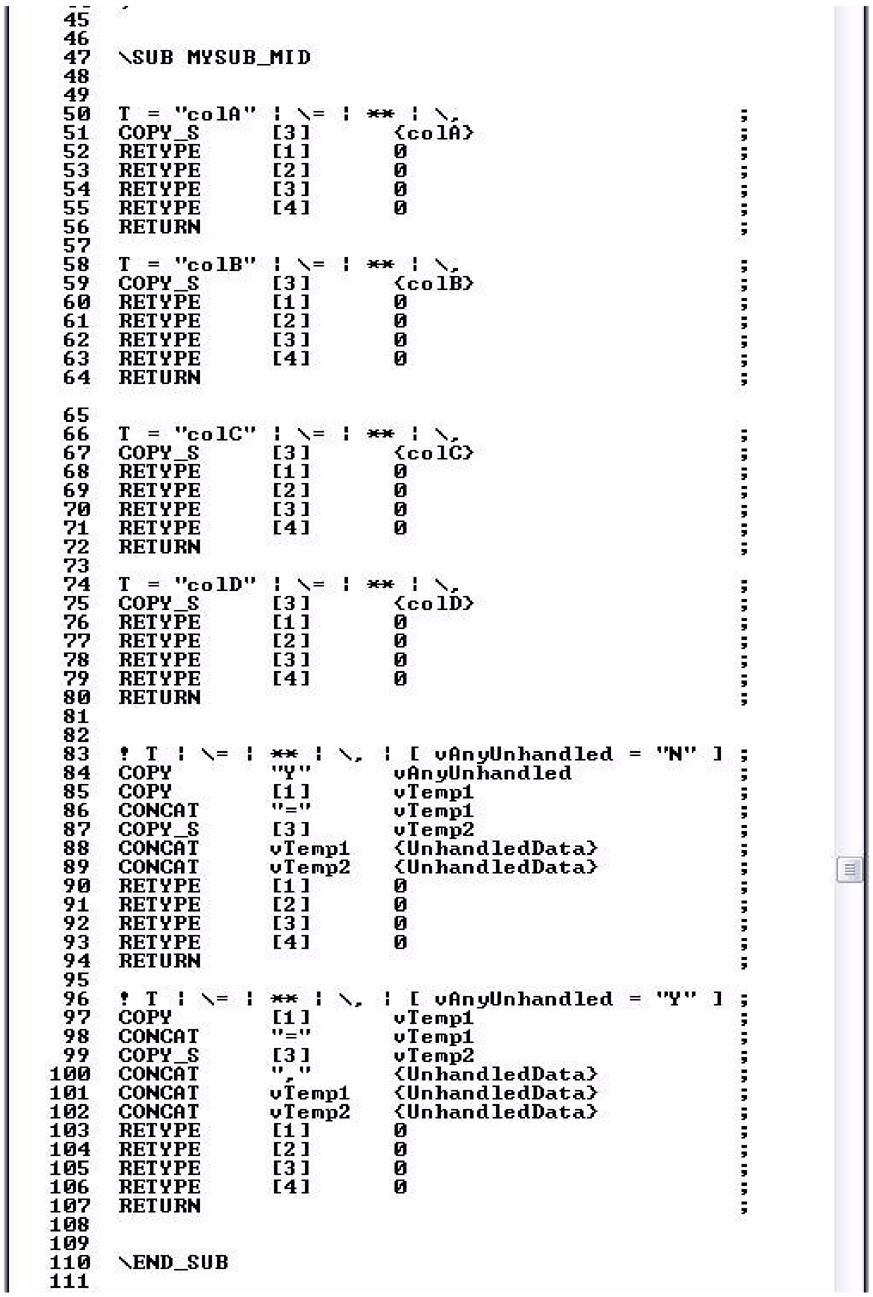
5. Alter the contents of the Pattern Action file to equal that as displayed in Figure 24-4, Figure 24-5, and Figure 24-6. A code review follows.

   Do not enter the line numbers; line numbers appear for the benefit of the code review that follows.


RS02_DATAFORMAT01.PAT - Vi for Windows
File Edit Macro Options Help

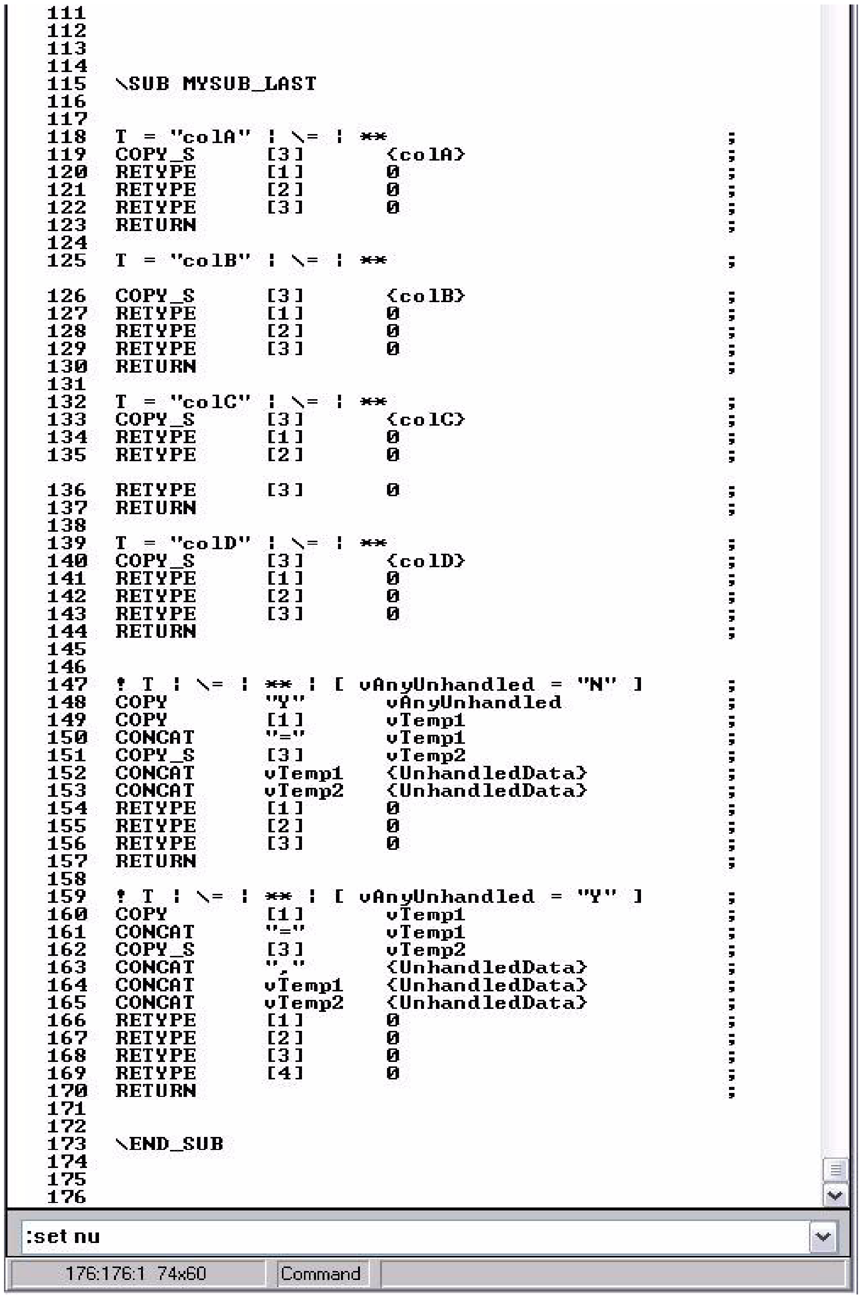
```
;;QualityStage v8.0
;---------------------------------------------
; Parser Rules
;---------------------------------------------

; SEPLIST   " ~`!@#$%^&*()_-+={}[]|\\:;\"<>,.?/'"
; STRIPLIST " ~`!@$^*_+={}[]|\\:;<>?"

\PRAGMA_START
SEPLIST     " ~`!@#$%^&*()_-+={}[]|\\:;\"<>,.?/'"
STRIPLIST   " ~`!@$^*_+{}[]|\\:;<>?"
\PRAGMA_END


\POST_START
\POST_END


;---------------------------------------------


; &                                          ;
**                                           ;
COPY        "null"   {colA}                  ;
COPY        "null"   {colB}                  ;
COPY        "null"   {colC}                  ;
COPY        "null"   {colD}                  ;
COPY        "N"      vAnyUnhandled           ;


& | \= | ** | \,                             ;
CALL MYSUB_MID                               ;
REPEAT                                       ;


& | \= | **                                  ;
CALL MYSUB_LAST                              ;


** | [ vAnyUnhandled = "N" ]                 ;

COPY        "null"   {UnhandledData}         ;


;---------------------------------------------

```

Figure 24-4   Pattern Action file, page 1 of 3.



```
\SUB MYSUB_MID


T = "colA" | \= | ** | \,                          ;
COPY_S      [3]       {colA}                        ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;

T = "colB" | \= | ** | \,                          ;
COPY_S      [3]       {colB}                        ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;

T = "colC" | \= | ** | \,                          ;
COPY_S      [3]       {colC}                        ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;

T = "colD" | \= | ** | \,                          ;
COPY_S      [3]       {colD}                        ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;


! T | \= | ** | \, | [ vAnyUnhandled = "N" ] ;
COPY        "Y"       vAnyUnhandled                 ;
COPY        [1]       vTemp1                        ;
CONCAT      "="       vTemp1                        ;
COPY_S      [3]       vTemp2                        ;
CONCAT      vTemp1    {UnhandledData}               ;
CONCAT      vTemp2    {UnhandledData}               ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;

! T | \= | ** | \, | [ vAnyUnhandled = "Y" ] ;
COPY        [1]       vTemp1                        ;
CONCAT      "="       vTemp1                        ;
COPY_S      [3]       vTemp2                        ;
CONCAT      ","       {UnhandledData}               ;
CONCAT      vTemp1    {UnhandledData}               ;
CONCAT      vTemp2    {UnhandledData}               ;
RETYPE      [1]       0                             ;
RETYPE      [2]       0                             ;
RETYPE      [3]       0                             ;
RETYPE      [4]       0                             ;
RETURN                                              ;


\END_SUB

```

Figure 24-5   Pattern Action file, page 2 of 3.



```




\SUB MYSUB_LAST


T = "colA" | \= | **                                    ;
COPY_S      [3]     {colA}                              ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETURN                                                  ;

T = "colB" | \= | **                                    ;
COPY_S      [3]     {colB}                              ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETURN                                                  ;

T = "colC" | \= | **                                    ;
COPY_S      [3]     {colC}                              ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETURN                                                  ;

T = "colD" | \= | **                                    ;
COPY_S      [3]     {colD}                              ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETURN                                                  ;


! T | \= | ** | [ vAnyUnhandled = "N" ]                 ;
COPY        "Y"     vAnyUnhandled                       ;
COPY        [1]     vTemp1                              ;
CONCAT      "="     vTemp1                              ;
COPY_S      [3]     vTemp2                              ;
CONCAT      vTemp1  {UnhandledData}                     ;
CONCAT      vTemp2  {UnhandledData}                     ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETURN                                                  ;

! T | \= | ** | [ vAnyUnhandled = "Y" ]                 ;
COPY        [1]     vTemp1                              ;
CONCAT      "="     vTemp1                              ;
COPY_S      [3]     vTemp2                              ;
CONCAT      ","     {UnhandledData}                     ;
CONCAT      vTemp1  {UnhandledData}                     ;
CONCAT      vTemp2  {UnhandledData}                     ;
RETYPE      [1]     0                                   ;
RETYPE      [2]     0                                   ;
RETYPE      [3]     0                                   ;
RETYPE      [4]     0                                   ;
RETURN                                                  ;


\END_SUB



:set nu
```

Figure 24-6 Pattern Action File, page 3 of 3.

A code review for Figure 24-4, Figure 24-5, and Figure 24-6 includes;

– Lines 10-11, specify that we field separate (token separate) on commas and the equal symbol, but do not strip them from the patterns that we evaluate later.

  This is key, because our basic pattern is,

  ColumnName=Column value of 1 or more words, [comma]

  We will use the hard presence of the equal symbol, and then comma to delimit our unknown and multi-word value for (column value).

  We also separate and strip spaces, to avoid receiving extra spaces in our outputted values. Stripping spaces, but then also copying with spaces (COPY_S) allows for this to occur.

Note: Not certain what we are discussing here? As a test after you complete this example, call to strip spaces from the SEPLIST and/or STRIPLIST, and see what your outputted data looks like.

– Lines 15-16 are included for discussion, but are non-functional in this example.

  These lines allow for an after record processing capability, each individually received record after another. In other words, if you wanted to check what you produced for each row, before actually outputting data, you would put code between Line 15 and Line 16. If you have 99 input records, code contained between Line 15 and Line 16 would execute 99 times.

  However, this is a special code construct within QualityStage, and may only be used for features like SOUNDEX, NYSIIS, and related.

  You may have your own custom after row processing block, but you do not accomplish this need via Line 15 and Line 16; you accomplish it via a technique similar to Line 40, discussed below.

– Lines 22-28 form our before record processing block.

  In effect, Lines 22-28 are executed for/before every new input record we receive.

  Line 22 is actually commented out, but represents the common use case. The "&" simple pattern class matches any single token, and would thus cause the lines that follow to execute for every new input record we receive.

  We present an alternate solution in Line 23, with the simple pattern class of "**". The simple pattern class "**", matches zero or more tokens, in the event we wished to also process entirely NULL input records, a topic which is not pursued further in this document. ("&" does not match entirely NULL input records, "**" does.)

While Line 23 is our actual pattern line, Lines 24-28 are the commands that execute when this pattern is matched.

Note: Recall that this pattern is matched on every new input record. Because this pattern is also listed first, it serves as our *before record pattern block*.

Huh? If you want to perform any kind of processing before each input row that is received, this is the technique that you use. Here we use this technique to set a default value for our output columns. Only if these output columns are overwritten later as part of normal processing will their default values be changed.

Lines 24-27 copy the hard coded value of "null" to our output columns; colA, colB, colC, and colD.

Line 28 copies the value of "null" to a user defined variable with the name, vAnyUnhandledData. vAnyUnhandledData is not a keyword; we could of just as easily called this variable, Bob.

– Lines 31-33, and then Lines 36-37 are where the main processing is performed.

Recall that the input data is in the general form,

colA=Word word,colB=Word 22,colC=More words still,colD=word

Line 31 will match the first 3 column name, column values pairs in the input data stream. A code review of Line 31 follows;

& is the built in simple pattern class for any single token, and will match colA, colB, or any unknown column name (value).

\= is the simple pattern class to match the equal symbol.

** is the simple pattern class to match zero or more (words).

\, is the simple pattern class to match the comma.

Line 36 differs from Line 31, in that Line 36 does not require the trailing comma, as is the case with the last column name, column value pair on any input record.

Line 32 calls to invoke a user defined pattern action language sub-routine with the name, MYSUB_MID.

Line 33 (REPEAT) will cause the sub-routine invoked on Line 32, to repeat until it finds no more pattern matches.

Note: In effect, this sub-routine (MYSUB_MID) will nibble off (look for, find, and then remove matching tokens off) of the leading portion of the input record, until there are no more pattern matches.

This is because this sub-routine invocation is followed by the REPEAT command. If the REPEAT command were not present on Line 33, then this sub-routine (MYSUB_MID) would only invoke one time, as is the case with the sub-routine invocation on Line 36 (MYSUB_LAST).

What these two user defined sub-routines do, is determined by their contents, listed below.

We say l*eading portion* above, because the default pattern matching process reads the input string from left to right. If or when necessary, you may choose to read input string left to right, right to left, or a mixture of both. You may even choose to just look for specific tokens found anywhere inside the larger input string.

– Lines 40-41 form our (user defined and configurable) after record processing block.

In effect, Lines 40-41 are executed for/after every new input record we receive.

Line 40 also offers a new and specific use case. While the "**" simple pattern matches every row, the next portion of this line displays how we can evaluate user defined variables.

[vAnyUnhandled = "N"]

Will be true of the user defined variable, vAnyUnhandled, is equal to "N". This clause is treated the same as any simple or user defined pattern class; if true, run the pattern action commands that follow, otherwise do not.

In effect, we use the block to set the output column UnHandledData to equal a default value, just as we did for output columns, colA, colB, etcetera. Logically we could have set and maintained the value in

UnHandledData in the same manner as colA, colB, etcetera, but this provided us an example to demonstrate after row processing.

– Lines 47-110 provide the contents of the user defined sub-routine with the name, MYSUB_MID.

The contents of any user defined sub-routine follow the same basic rules as the normal content areas of the Pattern Action file, with pattern match lines, and pattern action commands. User defined sub-routines may even call other user defined sub-routines.

The MYSUB_MID user defined sub-routine contains 6 (count) pattern match lines, located on; Lines 50, 58, 66, 74, 83 and 96. The pattern match blocks beginning on Lines 50, 58, 66 and 74, process a *known* input column names ("T"). The pattern match blocks beginning on Lines 83 and 96, process previously unknown column names("! T", not T).

The blocks starting on Lines 50, 58, 66, and 74 are similar to one another. For example; Line 51 copies the column value to a given/correct output column, and Lines 52-55 remove the received/identified tokens from further consideration. (These tokens are removed because we have fully processed them. We're done with them.)

Note: The RETURN command on Line 56, along with the REPEAT of the sub-routine on Line 33, is what forms our looping construct, allowing us to walk the larger input string.

The blocks starting on Lines 83 and 96 are similar to one another.

In effect, the block starting on Line 83 is executed upon receiving the first unknown column name. The block starting on Line 96 executes for any second and subsequent unknown column name. The only difference between these two blocks is how we format the output column, UnHandledData, which we append to consecutively in a very precise and well formatted manner.

Line 84 changes the values of the user defined variable named, vAnyUnhandled to "Y".

Lines 85-89 are just so that we may precisely control spacing and formatting of the value(s) in our output column.

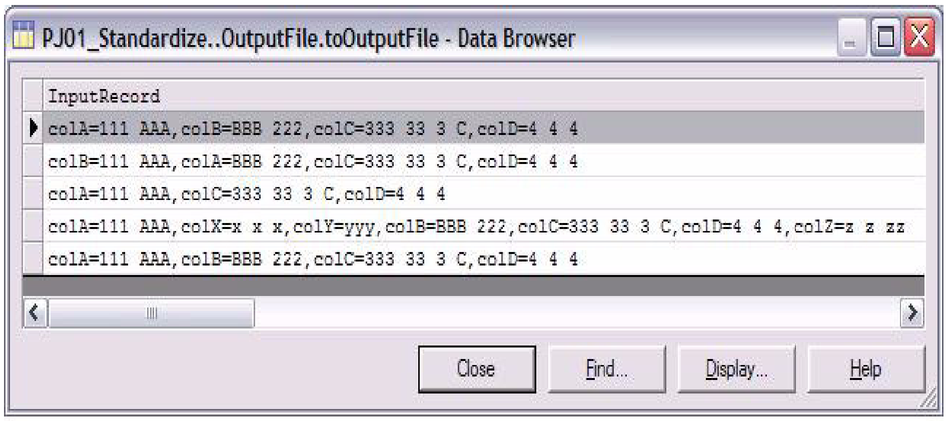
– Lines 115-173 provide the contents of the user defined sub-routine with the name, MYSUB_LAST.

  The user defined sub-routines MYSUB_MID and MYSUB_LAST are *very similar*. (The only difference is the presence of the check for the trailing and field delimiting comma.) We offer the current solution as is, preferring to leave a support legacy of a simpler and cleaner example, versus a more complex, exotic, and possibly harder to understand example for those who follow us.

  Because this sub-routine nearly mirrors the preceding one, no further code review is provided.

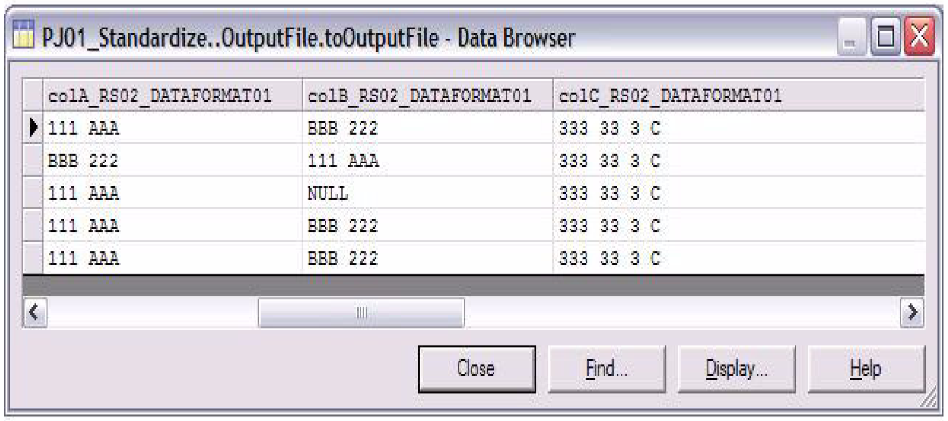
6. Save, compile, and test your work. A successful result appears similar to that as displayed in Figure 24-7.

Note: When creating and working with QualityStage custom rules sets, remember to save your work often.



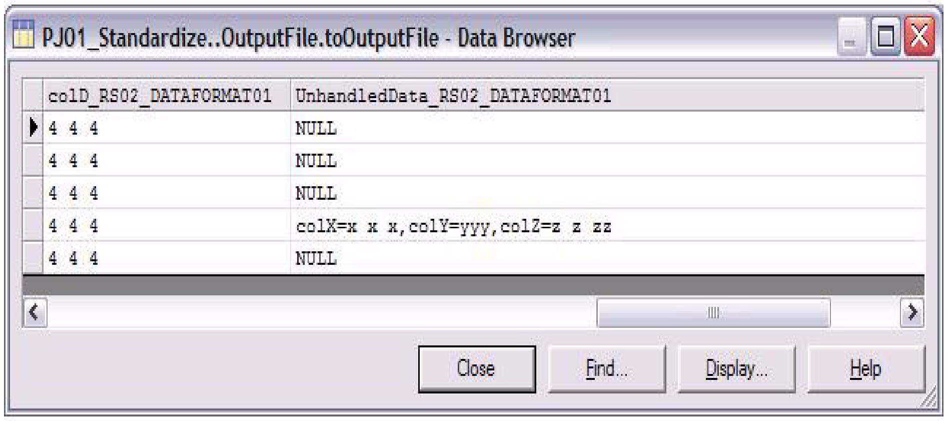
PJ01_Standardize..OutputFile.toOutputFile - Data Browser

InputRecord
colA=111 AAA,colB=BBB 222,colC=333 33 3 C,colD=4 4 4
colB=111 AAA,colA=BBB 222,colC=333 33 3 C,colD=4 4 4
colA=111 AAA,colC=333 33 3 C,colD=4 4 4
colA=111 AAA,colX=x x x,colY=yyy,colB=BBB 222,colC=333 33 3 C,colD=4 4 4,colZ=z z zz
colA=111 AAA,colB=BBB 222,colC=333 33 3 C,colD=4 4 4



PJ01_Standardize..OutputFile.toOutputFile - Data Browser

colA_RS02_DATAFORMAT01	colB_RS02_DATAFORMAT01	colC_RS02_DATAFORMAT01
111 AAA	BBB 222	333 33 3 C
BBB 222	111 AAA	333 33 3 C
111 AAA	NULL	333 33 3 C
111 AAA	BBB 222	333 33 3 C
111 AAA	BBB 222	333 33 3 C



PJ01_Standardize..OutputFile.toOutputFile - Data Browser

colD_RS02_DATAFORMAT01	UnhandledData_RS02_DATAFORMAT01
4 4 4	NULL
4 4 4	NULL
4 4 4	NULL
4 4 4	colX=x x x,colY=yyy,colZ=z z zz
4 4 4	NULL

Figure 24-7 Successful result from test.

24.3 In this document, we reviewed or created:

We examined the (IBM) InfoSphere Information Server (IIS), QualityStage component, Standardize stage. Specifically, we created and tested one more advanced QualityStage custom rule set; a very powerful technology with nearly endless applications to standardize and cleanse data of any source or purpose.

More specifically, we demonstrated looping, before and after row processing, and handling NULL and (variably user defined) UnHandled Data.

Persons who help this month.

Andy Wilson, and Robert Dickson.

Additional resources:

The IBM InfoSphere Information Server, QualityStage component, product tutorial.

Legal statements:

Special attributions: