# Table of Contents

## Chapter Basic Sample 1: Extracting Information from a Document and Creating a Flat File

- Overview .................................................. 12
- Files Used by Sample ................................. 12
- Business Scenario ...................................... 13
- Explanation of Sample ............................... 14
  - Input Tag Group ........................................... 14
  - Field Tag Group .......................................... 15
  - Output Tag Group ....................................... 16
  - Sidefile Tag Group .................................... 16
- Run Time .................................................. 17

## Chapter Basic Sample 2: Sorting and Outsorting Documents within a Single Print Stream

- Overview .................................................. 20
- Files Used .................................................. 20
- Business Scenario ..................................... 21
- Explanation of Sample ............................... 21
  - Input Tag Group ........................................... 22
  - Field Tag Group .......................................... 22
  - Sortmatch Tag Group .................................. 23
  - Output Tag Group ....................................... 23
  - Rules ..................................................... 24
- Run Time .................................................. 25

## Chapter Basic Sample 3: Modifying Document Contents

- Overview .................................................. 28
- Using Field Replacement to Add Information to a Page ........ 28
- Files Used .................................................. 28
- Business Problem ....................................... 29
- Explanation of Sample ............................... 30
  - Input Tag Group ........................................... 30
  - Field Tag Group .......................................... 31
  - Rules ..................................................... 32
  - Output Tag Group ....................................... 33
- Run Time .................................................. 33

## Chapter Basic Sample 4: Cleansing Addresses, Adding IMBs and Outsorting Bad Addresses

- Overview .................................................. 38
- Files Used .................................................. 38
- Business Scenario ..................................... 39
- Explanation of Sample ............................... 40
CHAPTER BASIC SAMPLE 5: ADDING OMR MARKS AND CODING 3OF9 BARCODES

Overview .............................................52
Files Used .............................................52
Business Scenario .................................53
Explanation of Sample ..........................55
Input Tag Group ....................................55
Field Tag Group ....................................55
Rules ..............................................57
Output Tag Group .................................58
Add Tag Group ....................................58
Run Time ...........................................59

CHAPTER BASIC SAMPLE 6: POSTAL PRESORTING

Overview .............................................62
Files Used .............................................62
Business Scenario .................................63
Explanation of Sample ..........................63
Input Tag Group ....................................64
Field Tag Group ....................................65
Rules ..............................................65
Output Tag Group .................................66
Presort Tag Group .................................66
Run Time ...........................................68

CHAPTER BASIC SAMPLE 7: ADDING BANNER PAGES AND MANIPULATING OUTPUT FILES

Overview .............................................78
Files Used .............................................78
Business Scenario .................................79
Explanation of Sample ..........................79
Input Tag Group ....................................80
Field Tag Group ....................................81
Rules ..............................................82
Banner Tag Group ..................................82
Output Tag Group .................................82
Presort Tag Group .................................82
Run Time ...........................................83
<table>
<thead>
<tr>
<th>Chapter Advanced Sample 5: Extracting Information and Creating a Flat File (AFPDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong> .............................. 164</td>
</tr>
<tr>
<td><strong>Files Used</strong> ............................ 164</td>
</tr>
<tr>
<td><strong>Business Scenario</strong> ..................... 165</td>
</tr>
<tr>
<td><strong>Explanation of Sample</strong> .............. 166</td>
</tr>
<tr>
<td><strong>Input Tag Group</strong> ...................... 166</td>
</tr>
<tr>
<td><strong>Field Tag Group</strong> ...................... 167</td>
</tr>
<tr>
<td><strong>Rules</strong> ................................. 168</td>
</tr>
<tr>
<td><strong>Output Tag Group</strong> ..................... 169</td>
</tr>
<tr>
<td><strong>Sidefile Tag Group</strong> ................... 169</td>
</tr>
<tr>
<td><strong>AFPIndex Tag Group</strong> ................... 169</td>
</tr>
<tr>
<td><strong>Run Time</strong> .............................. 170</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter Advanced Sample 6: Adding Barcodes to DJDE or Impact Printer Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong> .............................. 174</td>
</tr>
<tr>
<td><strong>Files Used</strong> ............................ 174</td>
</tr>
<tr>
<td><strong>Business Scenario</strong> ................. 175</td>
</tr>
<tr>
<td><strong>Explanation of Sample</strong> .............. 175</td>
</tr>
<tr>
<td><strong>Input Tag Group</strong> ...................... 177</td>
</tr>
<tr>
<td><strong>Field Tag Group</strong> ...................... 177</td>
</tr>
<tr>
<td><strong>Rules</strong> ................................. 178</td>
</tr>
<tr>
<td><strong>Add Tag Group</strong> ....................... 178</td>
</tr>
<tr>
<td><strong>Output Tag Group</strong> ................. 178</td>
</tr>
<tr>
<td><strong>An Alternative Approach: Using the Insertrec Group to Add Barcodes</strong> .... 179</td>
</tr>
<tr>
<td><strong>Run Time</strong> .............................. 181</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter Advanced Sample 7: Postal Processing with Complex Print</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong> .............................. 186</td>
</tr>
<tr>
<td><strong>Files Used</strong> ............................ 186</td>
</tr>
<tr>
<td><strong>Business Scenario</strong> ..................... 187</td>
</tr>
<tr>
<td><strong>Explanation of Sample (First Pass)</strong> ........ 189</td>
</tr>
<tr>
<td><strong>Input Tag Group</strong> ...................... 189</td>
</tr>
<tr>
<td><strong>Field Tag Group</strong> ...................... 190</td>
</tr>
<tr>
<td><strong>Rules</strong> ................................. 190</td>
</tr>
<tr>
<td><strong>Output Tag Group</strong> ..................... 191</td>
</tr>
<tr>
<td><strong>Explanation of Sample (Second Pass)</strong> .......... 192</td>
</tr>
<tr>
<td><strong>Input Tag Group</strong> ...................... 193</td>
</tr>
<tr>
<td><strong>Field Tag Group</strong> ...................... 194</td>
</tr>
</tbody>
</table>
Basic sample 1: Extracting information from a document and creating a flat file

In this section

Overview........................................................................................................ 12
Files Used by Sample.................................................................................. 12
Business Scenario.......................................................................................... 13
Explanation of Sample.................................................................................... 14
Overview

This sample application extracts data from a single AFP line data input and uses the extracted data to construct a flat file to be used by another program. This sample illustrates the following concepts:

- Input tag group
  - <NAME> tag
  - <FILE> tag
  - <TYPE> tag
  - <DOCUMENT> tag
  - <PAGE> tag
- Carriage controls
- Field tag group
  - <FIELD> tag
  - <LOCATION> tag
- Common system variables
- Document breaks
- Page breaks
- Side files
- Output tag group
  - <NAME> tag
  - <FILE> tag
- Sidefile tag group
  - <FIELD> tag
  - <SIDEPART> tag

Files Used by Sample

This sample application uses the following files.

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
<td>D:\apps\basic1\b01inpt.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(b01inpt)</td>
</tr>
<tr>
<td>Rule File</td>
<td>N/A</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\basic1\b01cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(B01CONT)</td>
</tr>
<tr>
<td></td>
<td>B01CONT.ws.CON</td>
</tr>
<tr>
<td>Output Print Stream</td>
<td>D:\apps\basic1\b01outp.lin</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B01OUTP)</td>
</tr>
</tbody>
</table>
Basic sample 1: Extracting information from a document and creating a flat file

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Other Output</th>
<th>D:\apps\basic1\b01side.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B01SIDE)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Run Time</th>
<th>JCL:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D:\apps\basic1\b01jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B01JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic1\b01scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B01SCR)</td>
</tr>
</tbody>
</table>

Business Scenario

This sample application generates an audit report that shows every customer’s invoice amount from this customer invoice file. The audit report is used as input to another program that requires a fixed record format. The layout of the audit report should be:

Table 2: Layout of Audit Report

<table>
<thead>
<tr>
<th>Column</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>Customer number</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Invoice amount</td>
</tr>
</tbody>
</table>

Any data past column 20 in the audit report is ignored.

In the sample, above:

- Customer invoices are always one page in length.
Basic sample 1: Extracting information from a document and creating a flat file

- Extract the customer number, which is up to 8 characters in length, and the invoice amount, which is up to 11 characters long, for example: $999,999.99. Allow for the minimum invoice amount due for every document. Both pieces of data are at fixed positions on the printed pages - the customer number on the 8th print line and the invoice amount on the 9th print line, both at column 60.

Explanation of Sample

The control file reads a print stream, extracts data from it, and writes the information to a side file.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B01CONT.ws.CON contains these modifications.

Input Tag Group

As the control file shows, the Input tag group is identified in the `<NAME>` tag as EXM1. In this application, the `<NAME>` tag value is used only in the processing report that Enrichment generates for every processing run. The input file is found at DD: INPUT1 on the mainframe system, as indicated by the Input group `<FILE>` tag. The application refers to a DD: statement in the JCL, but you could also specify the file’s data set name in the `<FILE>` tag instead.

**Note:** On a UNIX system, you would set the `<FILE>` tag either to the explicit file name or use a pseudonym that the shell script would link. On a Windows system, you would set the `<FILE>` tag to the explicit file name.

The input print stream is AFP line data with ANSI carriage controls, so the `<TYPE>` tag in the Input group is set to `AFPL A`, allowing the `charset` parameter to default to `E` (EBCDIC).

Enrichment can locate pages automatically, so no `<PAGE>` tag is required.

Each document in the input is one page in length, as indicated by `<DOCUMENT>1`. A quick examination of the input print stream reveals that the input contains three one-page documents.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.
Basic sample 1: Extracting information from a document and creating a flat file

To define the data to extract you use the Field tag group. The application needs to extract two pieces of data from each document, so two Field tag groups are defined in the Input group.

In the first Field tag group, the `<FIELD>` tag defines a field variable called `%%Customer_Number` that stores the customer account number from each document. This variable is referenced later to include it in the report file. Because we only want to extract the first occurrence of the variable value from each document (that is, we don’t want to replace, delete, or move it), we set the `<FIELD>` tag action parameter to `K` (keep). The `<LOCATION>` tag indicates that the customer number information always begins on row 8 in column 60, and is up to 8 characters in length, as illustrated in the figure below.

In the second Field group, the `<FIELD>` tag defines a field variable called `%%invoice` that will extract the invoice amount from each document. Again, we set the action parameter to `K` because we only want to extract the first occurrence of the variable value from each document. The `<LOCATION>` tag indicates that the invoice amount...
always begins on row 9 in column 60, and is up to 10 characters in length, as illustrated in the following figure.

![Image of the document content]

**Output Tag Group**

Next, we specify the Output tag group, and the Sidefile group within it, to define the audit report to create from data extracted from the input print stream. Much as we did in the Input group, we use the Output group `<NAME>` tag to identify the Output group in the Enrichment report (MEMO). We also used the `<FILE>` tag to specify the output file, DD:OUTPUT1.

**Sidefile Tag Group**

The Sidefile tag group’s `<FILE>` tag identifies the audit report to create (DD:AUDIT). We use `<SIDEPART>` tags to specify the record layout and contents of the side file (the audit report). Each record of the side file will contain the data defined in each of the `<SIDEPART>` tags, in the order in which the tags are used in the Sidefile tag group.

The first `<SIDEPART>` tag specifies that the value of `%%Customer_Number`, which will be a maximum of 8 characters in length, should be the first data item on each record of the audit report. Further, the value should be right-justified (R), and if the value is less than 8 characters in length Enrichment should pad it with blanks to a length of 8 (’ ’).

The second `<SIDEPART>` tag specifies that the value of `%%invoice`, which will be a maximum of 11 characters in length, should be the second data item on each record of the audit report. The value should be right-justified, and if it is less than 11 characters in length, Enrichment should pad it with blanks to a length of 11. Notice that, while the field definition for `%%invoice` specifies a maximum length of 10, the `<SIDEPART>` tag specifies its length as 11 so the audit report will always contain an extra blank separating the invoice amount from the customer number. Although this isn’t necessary (Enrichment doesn’t care if there’s an extra space or not), it will make the audit report easier to read.
Basic sample 1: Extracting information from a document and creating a flat file

The third `<SIDEPART>` tag specifies that the value of the system variable `%%DOCUMENT_NO`, which will be a maximum of 5 characters in length, should be the last data item on each record of the audit report. The `%%DOCUMENT_NO` system variable represents the sequential number of each document in the input print stream. We use it in the audit report so we'll know that each document in the input is represented in the side file. Again, Enrichment will right-justify the value, but this time if it is less than the specified length, we want it padded with zeroes.

Run Time

The following example scripts could be used to run the sample application on mainframe and UNIX.

Mainframe

The following figure shows the JCL we used to run the application.

```plaintext
// JOB
// JOB DD DSN=PDR.STREAMW.LACDCRUN, DISP=SHR
// // DD DSN=SYS3.V4R1M0.ISPLOAD, DISP=SHR
// // DD DSN=SYS3.CLIB22.SEDCLINK, DISP=SHR
// // DD DSN=SYS3.PLI230.SIBMLNK, DISP=SHR
// // SW EXEC PGM=PDRSW000, REGION=0M
// // REPORT DD SYSOUT=*  
// // SYSPRINT DD SYSOUT=* 
// // CONTROL DD DSN=PDR.STREAMW.HANDSON.CONTROL(B01CONT)
// // INPUT DD DSN=PDR.STREAMW.HANDSON.INPUT(B01INPT), DISP=SHR
// // OUTPUT DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B01OUT1), DI SP=SHR
// // AUDIT DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(B01OUT2), DI SP=SHR
// // SYSTERM DD SYSOUT=*  
// // SYSSDUMP DD SYSOUT=* 
// // OUTPUT DD SYSOUT=*  
// // SYSIN DD DUMMY
// // -------------------------------------------------------------
// // COMP01 EXEC PGM=ISRSUPC, PARM=(DELTAL,LICENSEM,'','')
// // NEWDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B01OUT1), DI SP=SHR
// // OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B01OUT1), DI SP=SHR
// // OUTDD DD SYSOUT=* 
// // -------------------------------------------------------------
// // COMP02 EXEC PGM=ISRSUPC, PARM=(DELTAL,LICENSEM,'','')
// // NEWDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(B01OUT2), DI SP=SHR
// // OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(B01OUT2), DI SP=SHR
// // OUTDD DD SYSOUT=* 
// * -------------------------------------------------------------
```
Basic sample 1: Extracting information from a document and creating a flat file

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
rm DD:INPUT1
rm DD:OUTPUT1
rm DD:AUDIT
rm DDCONTROL
ln -s /share/home1/wzanone/SunOS/handson/LINEDATA.LIN DD:INPUT1
ln -s /share/home1/wzanone/SunOS/handson/B01out1 DD:OUTPUT1
ln -s /share/home1/wzanone/SunOS/handson/B01audit DD:AUDIT
ln -s /share/home1/wzanone/SunOS/handson/B01sidef.con DDCONTROL
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=B01CONT.ws.CON
```
Basic Sample 2: Sorting and outsorting documents within a single print stream

In this section

Overview ................................................................. 20
Files Used ............................................................ 20
Business Scenario .................................................... 21
Explanation of Sample .............................................. 21
Overview

This sample illustrates how to sort documents within a single AFP line data print stream by a field and outsort documents to multiple outputs by page count. This sample introduces the following topics.

- Outsorting
- Rule file
- Sortmatch tag group
  - \texttt{<INPUTNAME>} tag
  - \texttt{<SORT>} tag
- Rule tag group
  - Conditions
- Sortmatch tag group
  - Using \texttt{%%TOTALPAGES} to assign output
  - \texttt{<OUTPUT>} command
- Content tag group
  - Multiple Output tag groups

Files Used

This sample application uses the following files.

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Input Print Stream</th>
<th>D:\apps\basic2\b02inpt.afl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule File</td>
<td>Inline</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\basic2\b02cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(B02CONT)</td>
</tr>
<tr>
<td></td>
<td>B02CONT.ws.CON</td>
</tr>
<tr>
<td>Output Print Stream</td>
<td>D:\apps\basic2\b02out1.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic2\b02out2.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic2\b02out3.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B02OUT1)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B02OUT2)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B02OUT3)</td>
</tr>
<tr>
<td>Other Output</td>
<td>N/A</td>
</tr>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic2\b02jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B02JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic2\b02scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B02SCR)</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application takes a customer invoice file and outsor ts three separate outputs based on page count: one-page invoices, two-page invoices, and invoices over two pages. It also sorts the account summaries by customer account number within each grouping.

In the sample, above:

- Page 1 is on the first page of every document.
- The customer number is 10 characters in length and is always on the 15th print line starting in column 60.

Explanation of Sample

The control file reads a print stream that contains multiple documents of varying page count, sorts the documents by account number and outsor ts them to separate outputs by page count.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B02CONT.ws.CON contains these modifications.
Basic Sample 2: Sorting and outsorting documents within a single print stream

Input Tag Group

In the control file, the Input group is identified in the <NAME> tag as Sortout, and the input file is found on the mainframe system at DD:INPUT1, as indicated by the <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A, allowing charset to default to E.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```
<input>
  <name> Sortout </name>
  <file> DD:INPUT1 </file>
  <type> AFPL A </type>
  <document> TOP %%Customer_Number CHANGE </document>
  <field> %%Customer_Number KA </field>
  <location> 15 60 10 </location>
</input>

<sortmatch>
  <inputname> Sortout SORT YES </inputname>
  <sort> %%Customer_Number ASCEND </sort>
</sortmatch>

<rule>
  <content>
    DOCUMENT: if %%TOTAL_PAGES > 2 then MORE3PAGE elseif %%TOTAL_PAGES > 1 then TWOPAGE else ONEPAGE endif
  </content>
</rule>

<output>
  <name> ONEPAGE </name>
  <file> DD:OUTPUT1 </file>
</output>

<output>
  <name> TWOPAGE </name>
  <file> DD:OUTPUT2 </file>
</output>

<output>
  <name> MORE3PAGE </name>
  <file> DD:OUTPUT3 </file>
</output>
```

Field Tag Group

We need to sort the documents within the output by a field, so we define a field variable called %%Customer_Number in the Field tag group to extract the customer account.
number from each document. We only want to extract the $\%\text{Customer\_Number}$ value from every page of each document, so we set the `<FIELD>` tag `action` parameter to `KA` (keep all occurrences of the field in each document). The `<LOCATION>` tag indicates that the customer number always begins on row 15 in column 60, and is up to 10 characters in length.

We also use the $\%\text{Customer\_Number}$ value as a top-of-document indicator. In the Input tag group, we specified `<DOCUMENT>TOP $\%\text{Customer\_Number} \text{ CHANGE}` to indicate that each time the $\%\text{Customer\_Number}$ value changes in the input, a new document begins. If we look at the input we see that, based on the criteria we specified in the `<DOCUMENT>` tag, the input contains three one-page documents, two two-page documents, and one four-page document.

Note: As indicated in “Business Scenario” on page 21, we could have used the string `Page1` followed by a space as the top-of-document indicator. However, we used $\%\text{Customer\_Number}$ instead to save processing time in the application.

Sortmatch Tag Group

The Sortmatch tag group defines how to sort the documents in the input by customer account number. First, the Sortmatch group `<INPUTNAME>` tag specifies the input to sort. The `<INPUTNAME>` tag name parameter value duplicates the name value of the Input group `<NAME>` tag (in this case, `Sortout`). Since we know there’s only one document per customer number in the input, there’s no need to match the documents. Therefore, we set the match parameter to `SORT`. Finally, we want the sorted documents to print in the output, so we set the `<INPUTNAME>` tag `printYN` parameter to `YES`.

The `<SORT>` tag identifies the sort criterion and direction. In the `<SORT>` tag, we specify the field by which to sort ($\%\text{Customer\_Number}$). We want to sort the documents from lowest customer number (147-934783) to highest customer number (831-511116), so we set the `<SORT>` tag `order` parameter to `ASCEND`.

Output Tag Group

We will need to use conditional processing (rules) to outsort the documents by page count. We want to outsort one-page documents to one output, two-page documents to a second output, and all remaining documents (that is, documents with more than two pages) to a third output.

As required, we’ll use a separate Output tag group to define each output. In the first Output group, we set the `<NAME>` tag to `ONEPAGE` and we set the `<FILE>` tag to `DD:OUTPUT1`, the location of the corresponding output file. Similarly, we name the
Basic Sample 2: Sorting and outsorting documents within a single print stream

second Output group TWOPAGE and set up a corresponding file at DD:OUTPUT2 and we name the third Output group MORE3PAGE and set up a corresponding file at DD:OUTPUT3.

Rules

The rules we need to perform the outsort are relatively simple, so rather than setting up a separate rule file, we’ll include the rules in a Content tag group within the Rule tag group.

Since we are using rules to process documents in the input, we define rule processing in the DOCUMENT: section of the inline rule.

We need to know the page count of each document so Enrichment can properly outsort it. an Enrichment system variable, %%TOTAL_PAGES, keeps track of the total number of logical front pages in each document. Thus, we can use the %%TOTAL_PAGES value as the outsort criterion.

To make the application as efficient as possible, we want to build the rules so that processing occurs in an “inverted pyramid” fashion—that is, the majority of the documents are processed first. In the application, we expect that the majority of the documents contain three or more pages. If this is true, a significant number of documents are eliminated before Enrichment processes the second rule, thus improving performance. Enrichment passes documents for which the first rule is not true to the second rule, which outsorts documents of two pages in length. Documents for which the second rule is not true must contain a single page. Our last rule outsorts all the remaining documents accordingly. Our rules, then, look like this:

```
IF %%TOTAL_PAGES > 2 THEN
<OUTPUT> MORE3PAGE
ELSEIF %%TOTAL_PAGES > 1 THEN
<OUTPUT> TWOPAGE
ELSE
<OUTPUT> ONEPAGE
ENDIF
```

We used <OUTPUT> commands in the rules to direct the documents to the proper outputs. Notice that each <OUTPUT> command value duplicates the value of one of the Output group <NAME> tags we set up.

Note: Since documents in the input are sorted before the rules are processed, all outputs will also be sorted.

So, when the application runs, Enrichment sorts all of the documents in ascending order by customer number. Then, as defined in the rules, Enrichment uses each document’s %%TOTAL_PAGES value to outsort the documents. First, Enrichment outsorts all documents that are three or more pages in length to the output we called MORE3PAGE. Then, of the remaining documents (documents that are less than three pages in
Basic Sample 2: Sorting and outsourcing documents within a single print stream

length), Enrichment outsports all two-page documents to the output we called TWOPAGE. Finally, Enrichment places remaining documents in the last output we defined, ONEPAGE.

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```plaintext
//*JOB
//JOB DSN=PDR.STREAM.WLOADCRUN,DISP=SHR
// DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
// DD DSN=SYS3.PLI230.SIBMLINK,DISP=SHR
// SW EXEC PGM=PDRSW000,REGION=0M
// REPORT DD SYSOUT=* 
// SYSPRINT DD SYSOUT=* 
// CONTROL DD DSN=PDR.STREAM.W.HANDSON.CONTROL(B02CONT)
// INPUT1 DD DSN=PDR.STREAM.W.HANDSON.INPUT(B02INPT),DISP=SHR
// OUTPUT1 DD DSN=PDR.STREAM.W.HANDSON.NEWOUT1(B02OUT1),DISP=SHR
// SYSTEM DD DSN=PDR.STREAM.W.HANDSON.NEWOUT2(B02OUT2),DISP=SHR
// SYSOUT DD DSN=PDR.STREAM.W.HANDSON.NEWOUT3(B02OUT3),DISP=SHR
// SYSPRINT DD SYSOUT=* 
// SYSPRINT DD SYSOUT=* 
// SYSEIN DD DUMMY
// *** -------------------------------------------------------------
// COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
// NEWDD DD DSN=PDR.STREAM.W.HANDSON.NEWOUT1(B02OUT1),DISP=SHR
// OLDDD DD DSN=PDR.STREAM.W.HANDSON.GOODOUT1(B02OUT1),DISP=SHR
// OUTDD DD SYSOUT=* 
// *** -------------------------------------------------------------
// COMP02 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
// NEWDD DD DSN=PDR.STREAM.W.HANDSON.NEWOUT2(B02OUT2),DISP=SHR
// OLDDD DD DSN=PDR.STREAM.W.HANDSON.GOODOUT2(B02OUT2),DISP=SHR
// OUTDD DD SYSOUT=* 
// *** -------------------------------------------------------------
// COMP03 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
// NEWDD DD DSN=PDR.STREAM.W.HANDSON.NEWOUT3(B02OUT3),DISP=SHR
// OLDDD DD DSN=PDR.STREAM.W.HANDSON.GOODOUT3(B02OUT3),DISP=SHR
// OUTDD DD SYSOUT=* 
// *** -------------------------------------------------------------
```

EngageOne Enrichment 7.3.0 Sample Applications Guide
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
  sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
  rm DD:INPUT1
  rm DD:OUTPUT1
  rm DD:OUTPUT2
  rm DD:OUTPUT3
  rm DDCONTROL
  ln -s /share/home1/wzanone/SunOS/handson/MULTILINE.AFL DD:INPUT1
  ln -s /share/home1/wzanone/SunOS/handson/B02out1 DD:OUTPUT1
  ln -s /share/home1/wzanone/SunOS/handson/B02out2 DD:OUTPUT2
  ln -s /share/home1/wzanone/SunOS/handson/B02out3 DD:OUTPUT3
  ln -s /share/home1/wzanone/SunOS/handson/SORTOUT.CON DDCONTROL
  sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
  sweaver -c=B02CONT.ws.CON
```
Basic sample 3: Modifying document contents

In this section

Overview ........................................................................................................................................................................ 28
Files Used ..................................................................................................................................................................... 28
Business Problem ....................................................................................................................................................... 29
Explanation of Sample .................................................................................................................................................. 30
Overview

This sample application illustrates how to delete existing OMR marks from a single AFP line data input and change the font in which the company name and document title print. It also shows how to remove a page number from and change the title on the first page of each document. This sample illustrates the following topics.

• TRC or font index column
• Field tag group
  – Replace and Replace All actions
  – <REFERENCE> tag
  – Delete and Delete All actions

Using Field Replacement to Add Information to a Page

You can use field replacement to add information to a page. For example, the following figure shows coding to add the customer’s name to a blank area located on the second line of the first page in column 50.

\[
\text{<FIELD> } \%\text{CustName R} \\
\text{<LOCATION> } 2 \ 50 \ 30 \\
\text{</FIELD>}
\]

The Field group specifies a location that is initially blank, so the value of \%\text{Cust Name} will be blank when it is extracted. If you set the value of \%\text{Cust Name} to the customer’s name, then the blank area on the document will be replaced.

Files Used

This sample application uses the following files.

Table 1: Files Used in This sample

<table>
<thead>
<tr>
<th>File Type</th>
<th>File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
<td>D:\apps\basic3\b03inpt.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(B03INPT)</td>
</tr>
<tr>
<td>Rule File</td>
<td>Inline</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\basic3\b03cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON_CONTROL(B03CONT)</td>
</tr>
<tr>
<td></td>
<td>B03CONT.ws.CON</td>
</tr>
</tbody>
</table>
Table 1: Files Used in This sample

| Output Print Stream | D:\apps\basic3\b03outp.afl  
PDR.STREAMW.HANDSON.GOODOUT1(B03OUTP) |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Output</td>
<td>N/A</td>
</tr>
<tr>
<td>Run-time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic3\b03jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B03JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic3\b03scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B03SCR)</td>
</tr>
</tbody>
</table>

Business Problem

This sample application alters the customer invoices as shown below. It also changes the font used to print the company name and document title.

In the sample, above:

- Delete the OMR marks on the left.
- Remove “Page 1” from the first page of all documents, leaving the page number on all other pages.
- Change the title on the first page of every document to “Summary Page”. The title should remain as “Account Summary” on all other pages.
Basic sample 3: Modifying document contents

Explanation of Sample

The control file reads a print stream, and deletes or replaces field information based on criteria specified in rules.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B03CONT.ws.CON contains these modifications.

Input Tag Group

As the control file shows, the Input tag group is identified in the `<NAME>` tag as `Modify`, and the input file is found on the mainframe system at DD:INPUT1, as indicated by the `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPL A`, allowing `charset` to default to `E`.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```xml
<input>
  <name> Modify </name>  <!-- Identifiable name. -->
  <file> DD:INPUT1 </file>  <!-- Input file. -->
  <type> AFPL A </type>  <!-- File is AFP line data. -->
  <document> TOP  %%Page1Only EXIST </document>  <!-- First page of a document. -->
  <field> %%font_change RA </field>  <!-- Replace all occurrences. -->
    <reference> ' ' '2' 2 </reference>  <!-- Reference of 2 found in column 2. -->
    <location> 0 0 1 </location>  <!-- No horiz. or vertical movement. -->
  </field>
  <field> %account_summary R </field>  <!-- Change only first occurrence. -->
    <reference> ' ' 'Account Summary' 34 </reference>  <!-- String starts in column 34. -->
    <location> 0 -14 15 </location>  <!-- Set the field equal to the reference. -->
  </field>
  <field> %OMR_marks DA </field>  <!-- Delete all old OMR marks. -->
    <reference> ' ' '... 3 </reference>  <!-- OMR mark starts in column 3. -->
    <location> 0 -2 3 </location>  <!-- Pick up the 3 bytes of reference. -->
  </field>
  <field> %Page1Only DA </field>  <!-- Delete every occurrence of 'Page 1'. -->
    <reference> ' ' 'Page 1' 67 </reference>  <!-- Reference starts in column 67. -->
    <location> 0 -6 7 </location>  <!-- Set to the reference string. -->
</input>

<rule>
  <content>
    DOCUMENT:  <!-- This is the document section of the rule. -->
    %%font_change = '5'  <!-- Reset the font replacement field. -->
    %account_summary = "Summary Page"  <!-- Reset replacement field. -->
  </content>
</rule>
```
Basic sample 3: Modifying document contents

</rule>
</output>

<name> ENHANCED </name> <! Identifiable name. >
</output>

**Note:** If you want to use a separate rule file and call it with the Rule group <FILE> tag (instead of using the Content tag group with inline rules), replace the Content tag group with <FILE>DD: rulename.</FILE>

**Field Tag Group**

For the application, we need to extract four data items from documents in the input:

- The font index of records on which the company name and document title print
- The first occurrence in each document of the document title *Account Summary*
- All OMR marks
- The string Page 1 followed by a space

To this end, we defined four Field tag groups.

We used the first Field group to define a variable (**font_change**) to replace all occurrences of font index 2 with font index 5. The value of the field (that is, the data we want to replace) is the Table Reference Character (TRC) that identifies the font used to print data in a particular record. We don’t need to extract the actual data for which the font is to change. Since we’ll replace all occurrences of the field data with new information, we set the <FIELD> tag action parameter to RA.

The TRC is often found in column two of the input print stream. Since the location of records that contain a TRC of 2 may vary from document to document, we used the <REFERENCE> and <LOCATION> tags to define the field. The <REFERENCE> tag instructs Enrichment to set a reference point for the field on every record in each document that has a blank carriage control character in column 1 and contains the character 2 in column 2. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins in the same column, and is one character in length. Thus, the value of **font_change** is 2, the same string identified by the <REFERENCE> tag as the reference point.

The second Field group identifies the heading *Account Summary* in the input. We want to replace the heading with *Summary Page*, but only on the first page of each document. The <FIELD> tag defines a variable called **account_summary**. We want to replace the first occurrence of the variable value in each document with new information, so we set the <FIELD> tag action parameter to R. The <REFERENCE> tag instructs Enrichment to set a reference point for the field on all records in each document that have a blank carriage control character and contain the string *Account Summary* in column 34. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins 14 columns before the
reference point, and is 15 characters in length. Thus, the value of \%\%account\_summary is the phrase Account Summary.

Our third Field tag group defines the locations of OMR marks to delete. The <FIELD> tag defines a variable called \%\%OMR\_marks. Since we want to delete every occurrence of the field information, we set the <FIELD> tag action parameter to DA. The <REFERENCE> tag instructs Enrichment to set a reference point for the field on all records in each document that have a blank carriage control and contain the string --- (three hyphens) starting in column 3. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins 2 columns before the reference point, and is 3 characters in length. Thus, the value of \%\%OMR\_marks is ---.

The final Field tag group serves a dual purpose in the application. It defines the location of the top-of-document indicator for the print stream, and it defines occurrences of the string 'Page 1' to delete. The <FIELD> tag defines a variable called \%\%Page1Only. We want to extract the field information and delete all occurrences of it, so we set the <FIELD> tag action parameter to DA (use and delete all occurrences of the field information in each document). The <REFERENCE> tag instructs Enrichment to set a reference point for the field on all records that have a blank carriage control and contain the string 'Page 1' in column 67. We added an extra space after Page 1 so Enrichment doesn’t inadvertently use a similar string (such as Page 11) as the reference point. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins 6 columns before the reference point, and is 7 characters in length. Thus, the value of \%\%Page1Only is always Page 1 followed by a space.

The Input group <DOCUMENT> tag uses the value of \%\%Page1Only to set top-of-document. Whenever Enrichment encounters the \%\%Page1Only value in the print stream (that is, whenever the value of \%\%Page1Only exists), a new document begins.

**Rules**

We set two of the field definitions (\%\%font\_change and \%\%account\_summary) with replace actions, so we used rules to tell Enrichment what to replace the values with. Because we didn’t have many rules to specify, we included them inline in the control file by specifying the Content tag group within the Rule group. We then placed the rules between the <CONTENT> and </CONTENT> tags.

We can’t put the rules in the START: section because Enrichment resets the variable values each time a document is processed.

Since we’re setting field variable values for each document, we define processing in the DOCUMENT: section of the rule.

For \%\%font\_change, we specified the following rule to replace font 2 with font 5:
Basic sample 3: Modifying document contents

Similarly, to replace the string Account Summary with *Summary Page*, we specify:

```
%%account_summary = '*Summary Page*
```

The new values of these fields automatically replace the old values because the field definition specifies it.

Output Tag Group

Finally, we specified an Output tag group to tell Enrichment where to put the enhanced print stream. We set the `<NAME>` tag to ENHANCED and we set the `<FILE>` tag to the location of the corresponding output file, DD: OUTPUT1.

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```jcl
//*JOB CARD
//*JOB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
//* DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//* DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
//* DD DSN=SYS3.PLI230.SIBMLINK,DISP=SHR
//*SW EXEC PGM=PDRSW000,REGION=OM
//* REPORT DD SYSOUT=* 
//* SYSPRTNT DD SYSOUT=* 
//* CONTROL DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.CONTROL(B03CONT) 
//* INPUT DD DSN=PDR.STREAMW.HANDSON.INPUT(B03INPT),DISP=SHR 
//* OUTPUT DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B03OUT1),DISP=SHR 
//* SYSTEM DD SYSOUT=* 
//* SYSDUMP DD SYSOUT=* 
//* OUTDD DD SYSOUT=* 
//* SYSOOUT DD SYSOUT=* 
//* SYSEIN DD DUMMY 
//* -------------------------------------------------------------
//* COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','') 
//* NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B03OUT1),DISP=SHR 
//* OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B03OUT1),DISP=SHR 
//* OUTDD DD SYSOUT=* 
//* -------------------------------------------------------------
```
Basic sample 3: Modifying document contents

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
swaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
rm DD:INPUT1
rm DD:OUTPUT1
rm DDCONTROL
ln -s /share/home1/wzanone/SunOS/handson/TRC.AFL DD:INPUT1
ln -s /share/home1/wzanone/SunOS/handson/B03out DD:OUTPUT1
ln -s /share/home1/wzanone/SunOS/handson/MODIFY.CON DDCONTROL
swaver -c=DDCONTROL
```

**Note:** If you use a <FILE> tag in the Rule group to specify a rule file instead of using an inline rule in the Content group, add the following line to your shell script:

```
ln rule.file DD:RULENAME
```

Add the line before Enrichment is executed in the shell script.

Windows

Run the application by issuing the following command:

```
swaver -c=B03CONT.ws.CON
```
Basic sample 3: Modifying document contents
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

In this section

Overview........................................................................................................38
Files Used....................................................................................................38
Business Scenario........................................................................................39
Explanation of Sample..............................................................................40
Overview

This sample application illustrates how to cleanse and standardize addresses in customer invoices and add an Intelligent Mail Barcode above the address block so that the IMB and the address are clearly readable through the envelope window. Documents with invalid addresses are written to a separate output. This sample illustrates the following topics.

- Postal cleansing
- System variables
  - %%LPCRC
  - %%ZIPCODE
- Barcodes
  - Drawn barcodes
  - Font-based barcodes
- CASS tag group
  - <CASS> and </CASS> tag
  - <CASSTYPE> tag
  - <DOUBLESORT> tag
  - <LPCRPLACE> tag
- System variables
  - <CASS>
  - </CASS>
  - <CASSTYPE>
  - </CASSTYPE>
  - <DOUBLESORT>
  - </DOUBLESORT>
  - <LPCRPLACE>
  - </LPCRPLACE>
- Add tag group
  - <ADD> and </ADD> tags
  - <ADDTYPE> tag
  - <ADDPART> tag
  - <POSITION> tag
  - <BARS> tag
  - <HEIGHT> tag
  - <ONPAGE> tag
  - <TRACKER> tag
- United States Intelligent Mail barcodes
- Input tag group
  - <CLEANSE> tag
  - <ADDRESSBLOCK> tag
- Address tag group
  - <ADDRESS> and </ADDRESS> tags
  - <LINE> tag
- Barcodes
  - <LPCRPLACE> tag
  - <ADD> and </ADD> tags
  - <ADDTYPE> tag
  - <ADDPART> tag
  - <POSITION> tag
  - <BARS> tag
  - <HEIGHT> tag
  - <ONPAGE> tag
  - <TRACKER> tag
- Using the Add tag group within and outside the Output group
- Adding objects to DJDE and impact print streams

Files Used

This sample application uses the following files.

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Input Print Stream</th>
<th>D:\apps\basic4\b04inpt.afl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(B04INPT)</td>
</tr>
<tr>
<td>Rule File</td>
<td>Inline</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\basic4\b04cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(B04CONT)</td>
</tr>
<tr>
<td></td>
<td>B04CONT.ws.CON</td>
</tr>
</tbody>
</table>
Basic sample 4: Cleansing addresses, adding IMBs and outsoritng bad addresses

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Output Print Stream</th>
<th>D:\apps\basic4\b04out1.afm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D:\apps\basic4\b04out2.afm</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B04OUT1)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B04OUT2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Outputs</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic4\b04jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B04JCL)</td>
</tr>
</tbody>
</table>

Business Scenario

This sample application standardizes the addresses on customer invoices. It also adds a delivery point barcode above the address block.

In the sample, above:

- The address block is in the same location for all documents so it will be properly positioned in the envelope address window. There needs to be enough space above the address for the barcode to be visible in the window.
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

- Write documents that contain invalid addresses to a separate output so that there is 100% accuracy in output 1. The longest address line is 35 bytes.

Explanation of Sample

Our control file (shown below) reads a print stream that contains multiple documents, passes address information to a cleansing program, and adds the Intelligent Mail barcode above the address on each document. Documents with invalid addresses are written to a separate output file.

Note: We could achieve the same outcome if we defined four Field tag groups (one for each address line) and an Address tag group instead of using the <ADDRESSBLOCK> tag. However, using <ADDRESSBLOCK> is much easier and can significantly reduce the size of your control file.

This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the <FILE> tag for your OS.B04CONT.ws.CON contains these modifications.

Input Tag Group

As the control file shows, the Input tag group is identified in the <NAME> tag as CLEANSE, and the input file is found on the mainframe system at DD:INPUT1, as indicated by the Input group <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A, allowing charset to default to E. Because we will use address cleansing software to standardize addresses in the input, we set the <CLEANSE> tag to YES.

Because the input is AFP line data and the addresses to be cleansed are in a consistent location on the first page of every document, we used an <ADDRESSBLOCK> tag to define the location of the address lines. Our address block begins on line 9 in column 9 and is 35 characters in length. The address consists of 4 lines, and we want to replace the first occurrence of the address in each document.

```xml
<field> %Address1 R
<location> 9 9 35
</field>
<field> %Address2 R
<location> 10 9 35
</field>
<field> %Address3 R
<location> 11 9 35
</field>
<field> %Address4 R
<location> 12 9 35
</field>
```
Basic sample 4: Cleansing addresses, adding IMBs and outsourcing bad addresses

Field Tag Group

We defined one Field tag group for the application to define top-of-document. The Field group identifies a variable called %AcctNum. Because some customer invoices may contain several pages, we must find a unique criteria by which to identify the first page of each invoice (that is, the top of each document). In the input print stream, the account number appears at a fixed location on every page. The value of the field is the customer’s account number on each page of the invoices. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <LOCATION> tag indicates that the field information begins on record 13 in column 60 and is 8 characters in length.

The Input group <DOCUMENT> tag uses the value of %AcctNum to set top-of-document. When the %AcctNum value changes in the input (that is, when the account number changes), a new document begins.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

CASS Tag Group

We included a CASS tag group in the control file because we want to cleanse the addresses in the input. We specified Finalist as the address cleansing program by setting the <CASS TYPE> tag to LPC. Note: You must have Finalist to perform address cleansing.

Since processing time is always at a premium, we set the <DOUBLESORT> tag to YES. <DOUBLESORT>YES tells Enrichment to sort documents in ascending order by ZIP Code before sending addresses for cleansing. At the same time, Enrichment sets the value of the %%ZIP CODE system variable for each document. Sorting the documents in this manner allows Finalist to look up ZIP Codes in its database more efficiently.
We want to add a Intelligent Mail barcode to each document, so Enrichment should place only documents whose addresses returned a %%LPCRC value of 0 (indicating that the address was cleansed with a standardized ZIP + 4® and carrier route) in the good output. We want to route all other documents to a separate output. To do this, we had to:

- Specify the processing threshold in the CASS group <LPCREPLACE> tag
- Construct rules for conditional processing of documents based upon their %%LPCRC value.

We specified <LPCREPLACE>1 5 9 6 6 4 4 4 2 to define the processing thresholds for address cleansing. Each of these numbers represents a specific processing threshold for Finalist. If an address exceeds any one of these thresholds, Enrichment won't replace it. We are most concerned with the first number, however, which represents the threshold for the general return code from Finalist when an address is cleansed (%%LPCRC). If Finalist's return code for a specific address exceeds 1, that address is not replaced. In other words, Enrichment will replace any address cleansed by Finalist with a return code of 1 or 0.

**Rules**

An inline rule in the control file decides, based on the %%LPCRC value, which output ("good" or "bad") the documents will go to after processing. Since we are using rules to process documents in the input, we define rule processing in the DOCUMENT: section of the rule. We are able to outsort based on the Finalist return code because DOCUMENT: rules are processed after address cleansing takes place. If the %%LPCRC value for a document is 0 (<1 in the rule), meaning it was cleansed with a standardized ZIP + 4® and carrier route, we want it to go to the "good" output (GoodOut). Documents with any other %%LPCRC value will go to the "bad" output (BadOut) for further processing later. The following figure shows the logic we used in the rule.

**Note:** Since %%POSTNET is only created for documents with an %%LPCRC value of 0, we only want to add the Intelligent Mail barcode to documents placed in the "good" output.

```plaintext
DOCUMENT: <! The rule section for document processing.>
if %%LPCRC < 1 then <! Separate the documents based on the return code from the CASS program. >
<output> GoodOut <! Return code from the CASS program. >
else
<output> BadOut
endif
%%BARCODEID = "00"
%%SERVICETYPE = "300"
%%MAILERID = "123456"
%%SEQUENCENUMBER = JUSTIFY(%%DOCUMENT_NO, R, 9, '0')
%%ZIP5 = SUBSTR(%%ZIPCODE, 1, 5, '0')
/* Construct contents of an IMB */
```
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

```plaintext
%%IMBIN25 = %%BARCODEID | %%SERVICETYPE | %%MAILERID | %%SEQUENCENUMBER | %%ZIP5
/* Call IMB to generate bars */
%%IMBBARS = IMB(%%IMBIN25)
```

Output Tag Group

The `<OUTPUT>` commands in the rule point to the two Output tag groups we set up. In the first Output group, we set the `<NAME>` tag to `GoodOut`, which resides on the mainframe system at DD:OUTPUT1 as indicated by the `<FILE>` tag. In the second Output group, we set `<NAME>` to `BadOut`, which resides at DD:OUTPUT2.

Add Tag Group

Finally, we set up an Add tag group to build and position the Intelligent Mail barcode on the documents. We include the Add group within the Output group that defines GoodOut because we know only these documents will have a valid Intelligent Mail barcode (because the %%LPCRC value for such documents is 0). This way, we only add barcodes to the GoodOut documents.

Since we're adding an Intelligent Mail barcode to the documents in GoodOut, we set the `<ADDTYPE>` tag to `IMB`. Enrichment gets the information that goes into the Intelligent Mail barcode from the result of the IMB function. Thus, we need only one `<ADDPART>` tag. In the control file, the `<ADDPART>` tag tells Enrichment to add the value of %%IMBBARS, which is 65 characters in length, to each document. Since each document in GoodOut will have an Intelligent Mail barcode, we do not need to specify values for the `<ADDPART>` tag justify or pad parameters.

We used the `<BARS>` and `<HEIGHT>` tags to specify the size of the Intelligent Mail barcode Enrichment would add. We print on an all points addressable (APA) AFP printer, so we can use AFP line draw commands in the print stream. The combination of `<ADDTYPE>IMB` and the `<POSITION>` tag adds an AFP record to each page on which an address is located. For the Intelligent Mail barcode to appear correctly in the window of the envelopes, we must specify `<POSITION>1.9 2.7 CM` (that is, the barcode will print 1.9 centimeters from the left edge of the paper and 2.7 centimeters from the top edge of the paper).

Finally, since the address appears only on the first page of each document in GoodOut, we set the `<ONPAGE>` tag for the Intelligent Mail barcode to `FIRST`, indicating that Enrichment will add the barcode only to the first page of each GoodOut document.
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application. In the JCL, we must add the Finalist load library to the JOBLIB. We also have to make sure that the second output file (OUTPUT2), for documents with incorrect addresses, is allocated in the JCL. Finalist may require additional DD statements, as indicated in the JCL.

```jcl
//JOBCARD
//JOBLIB DD DSN=PDR.STREAMW.LOADCRUN, DISP=SHR
//   DD DSN=SYS3.CLIB22.SEDCLINK
//   DD DSN=SYS3.PLI230.SIBMLINK
//   DD DSN=DIST.FNS670.PLOD, DISP=SHR
//   DD DSN=SYS3.V4R1M0.ISPLOAD, DISP=SHR
//*************************************************
//SW   EXEC PGM=PDRSW000, REGION=0M
//REPORT DD SYSOUT=* 
//INPUT1 DD DSN=PDR.STREAMW.HANDSON.INPUT(B04INPT), DISP=SHR
//OUTPUT1 DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B04OUT1), DISP=SHR
//OUTPUT2 DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(B04OUT2), DISP=SHR
//CONTROL DD DISP=SHR, DSN=PDR.STREAMW.HANDSON.CONTROL(B04CONT)
//***
//*** Finalist DATA SETS
//***
//CBDATA DD DSN=DIST.FNM310.PVSM.DATAFILE, DISP=SHR
//CBCTYST DD DSN=DIST.FNM310.PVSM.CITYFILE, DISP=SHR
//SYSTEM DD SYSOUT=* 
//SYSUDUMP DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSPRT1 DD SYSOUT=* 
//SYSPRT2 DD SYSOUT=* 
//SYSPRT3 DD SYSOUT=* 
//SYSPRT4 DD SYSOUT=* 
//SYSPRT5 DD SYSOUT=* 
//SYSPRT6 DD SYSOUT=* 
//SYSPRT7 DD SYSOUT=* 
//SYSPRT8 DD SYSOUT=* 
//CBSMPLE DD SYSOUT=* 
//CBSYSIN DD *
// PDR Enrichment TEST
//EXCPIN DD DUMMY 
//SYSPRT9 DD SYSOUT=* 
//* ------------------------------- -------------------------------
//COMP01 EXEC PGM=ISR5UPC, PARM=('DELTA L, LIN C', '')
//NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B04OUT1), DISP=SHR
//OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B04OUT1), DISP=SHR
//OUTDD DD SYSOUT=* 
//* ------------------------------- -------------------------------
//COMP02 EXEC PGM=ISR5UPC, PARM=('DELTA L, LIN C', '')
```
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

    sweaver -c=<controlfile>

To run the control file as-is on UNIX, use this script:

    rm DDCONTROL
    rm DD:INPUT1
    rm DD:OUTPUT1
    rm DD:OUTPUT2
    rm B04out1
    rm B04out2
    ln -s /share/home1/SunOS/handson/b04cont.con DDCONTROL
    ln -s /share/home1/SunOS/handson/b04inp.t afl DD:INPUT1
    ln -s /share/home1/SunOS/handson/B04out1 DD:OUTPUT1
    ln -s /share/home1/SunOS/handson/B04out2 DD:OUTPUT2
    sweaver -c/DDCONTROL

Windows

Run the application by issuing the following command:

    sweaver -c=B04CONT.ws.CON
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses
Basic sample 4: Cleansing addresses, adding IMBs and outsorting bad addresses
Basic sample 5: Adding OMR Marks and coding 3of9 Barcodes

In this section

Overview ................................................................................................. 52
Files Used .............................................................................................. 52
Business Scenario .................................................................................. 53
Explanation of Sample .......................................................................... 55
Basic sample 5: Adding OMR Marks and coding 3of9 Barcodes

Overview

This sample illustrates how to separate customer invoices into outputs by page count, adding OMR marks to one-page invoices and Code 3of9 barcodes to multipage invoices. This sample introduces the following topics.

- OMR marks
- Code 3of9 barcodes
- Rule file
  - Initialization
  - Function calls
  - CHECKSUM function
  - JUSTIFY function
  - SUBSTR function
  - Concatenation
- Using different Add tag groups within different Output groups
- Add tag group
  - <BARS> tag
  - <HEIGHT> tag
  - <ORIENT> tag

Note: Enrichment can add many other types of barcodes in addition to Code 3of9 and OMR marks. Refer to the <ADDTYPE> tag discussion in the Enrichment Language Reference for more information.

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\basic5\b05inpt.afl</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.INPUT(B05INPT)</td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td>Inline</td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td>D:\apps\basic5\b05cont.con</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.CONTROL(B05CONT)</td>
</tr>
<tr>
<td>B05CONT.ws.CON</td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\basic5\b05out1.afm</td>
</tr>
<tr>
<td>D:\apps\basic5\b05out2.afm</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B05OUT1)</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B05OUT2)</td>
</tr>
</tbody>
</table>
Basic sample 5: Adding OMR Marks and coding 3of9 Barcodes

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Other Output</th>
<th>JCL:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D:\apps\basic5\b05jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B05JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX Shell Script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic5\b05scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B05SCR)</td>
</tr>
</tbody>
</table>

**Business Scenario**

This sample application separates customer invoices into two outputs: one-page invoices and multi-page invoices. It adds OMR marks to the one-page invoices so they can be tracked through the inserter. Since the multi-page invoices will be going through a more advanced inserter, the application adds 3of9 barcodes to the multi-page invoices.
In the sample, above:

- For multi-page invoices, use the account number (numbers only), document number and check digit to create a 3of9 barcode.
- For one page invoices, add three OMR marks and cycle the added OMR marks from 1 to 7, then back to 1.
**Explanation of Sample**

The control file we created for the application (shown below) reads an input print stream that contains several multipage documents. Enrichment adds OMR marks to one-page documents and writes them to an output. Enrichment adds Code 3of9 barcodes to multipage documents and writes them to a separate output.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B05CONT.ws.CON contains these modifications.

**Input Tag Group**

As the control file shows, the Input tag group is identified in the `<NAME>` tag as `INPUT`, and the input file is found on the mainframe system at `DD:INPUT1`, as indicated by the Input group `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPL A`.

**Field Tag Group**

We defined one field, which Enrichment will use to set top-of-document and as data for the Code 3of9 barcodes.

Because some of the customer invoices may contain several pages, we must find a unique criteria by which to identify the first page of each invoice (that is, the top of each document). In the print stream, the account number appears relative to a string on the first page of each document. Therefore, the `<FIELD>` tag identifies a variable called `%%AcctNum`. The value of the field is the customer’s account number on each page of the invoices. We only want to extract the field information, so we set the `<FIELD>` tag action parameter to `KA`.

As the `<REFERENCE>` tag shows, Enrichment will set a reference point for the field on all records that have a blank carriage control and contain the string `Account Number:` beginning in column 44. The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins 2 columns after the reference point, and is 8 characters in length. Whenever the value of `%%AcctNum` changes in the input (that is, whenever the account number changes), a new document begins.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.
Basic sample 5: Adding OMR Marks and coding 3of9 Barcodes

<content>
START:
%%OMRCount = 0
%%Counter = 0
<! If a one page document, set OMR marks and route to that output.>
DOCUMENT:
if %%TOTAL_PAGES = 1 then
  %%OMRCount = %%OMRCount + 1   <! setup OMR mark and  >
  if %%OMRCount > 7 then  <! write to first output >
    %%OMRCount = 1
  endif
<output> Output1
<! Document must have more than one page. So set the 3of9 barcode>  <! and route to the appropriate output. >
else
  %%Counter = %%Counter + 1
  %%Doc = JUSTIFY(%%Counter,R,5,0)
  %%Barcode = SUBSTR(%%AcctNum,1,3) | SUBSTR(%%AcctNum,5,4) | %%Doc
  %%ChkDigit = CHECKSUM(%%Barcode)
<output> Output2
endif
</content>

<add>
<addtype> OMR
<addpart> 1 1     <! Mark used for Benchmarking  >
<addpart> %%OMRCount 3     <! Variable set in rule for 3 bytes.>
<position> .5 6 in     <! Placement of OMR marks.  >
</add>

<add>
<addtype> 3OF9
<addpart> *     <! Framing character>
<addpart> %%Barcode 12 L |     <! 12-bytes, left justified.  >
<addpart> %%ChkDigit 1 L |     <! 1-byte check digit.  >
<addpart> *     <! Framing character>
<position> .5 10.5 0 0 IN     <! Placement of 3of9 barcode.  >
<height> 40 PELS     <! Height of bars.  >
<orient> 4     <! Rotate the barcode 270 degrees.  >
Rules

START: Section

We use a START: section in the inline rule to set the initial value of %%OMRCount (which is total pages sent to Output 1) to 0. We also set %Counter (which keeps a count of the documents sent to Output 2) to 0.

DOCUMENT: Section

The DOCUMENT: section rules check the system variable %TOTAL_PAGES to see how many pages the invoice contains. If the %TOTAL_PAGES value for a document is 1, Enrichment sets up the OMR mark and sends the document to Output 1. The %OMRCount variable cycles the OMR marks between 1 and 7. We use an Add tag group in the Output group defining Output 1 to add the %OMRCount value (the OMR marks) to one-page documents.

If the %TOTAL_PAGES value for a document is anything other than 1, we use the value of the %AcctNum field variable along with the value of the %Counter variable and a checksum digit to construct a Code 3of9 barcode to add to each multi-page document before sending it to Output 2.

First, we increment the value of %Counter by one and set that as the new value of %Counter. We want to use the value of %Counter in the Code 3of9 barcodes, but since we have many documents we know the value could potentially consist of from one to five digits. Our Code 3of9 barcodes must always contain the same number of characters, so we used the JUSTIFY function to right-justify the %Counter value, padding it with zeroes to a length of 5 and returned it as the value of a new variable called %Doc. In this way, we ensure the document number information always has a constant length in the barcode. If the document number is 1, the %Doc value will be 00001, if the document number is 351, the %Doc value will be 00351, and so on.

The %AcctNum field variable value includes the dash in the middle of the account number. We don’t want the dash in the Code 3of9 barcode, so we used the SUBSTR function to parse the %AcctNum value into two pieces and concatenated them together (along with %Doc) into a new variable called %Barcode.

Then, we used the CHECKSUM function to calculate a checksum from the %Barcode value. We set the checksum as the value of a new variable called %ChkDigit.
Output Tag Group

The `<OUTPUT>` commands in the rule point to the two Output tag groups we set up. In the first Output group, we set the `<NAME>` tag to Output1, which resides on the mainframe system at DD:OUTPUT1 as indicated by the `<FILE>` tag. In the second Output group, we set `<NAME>` to Output2, which resides at DD:OUTPUT2. All one-page documents are routed to Output1 in the DOCUMENT: rules. All multipage documents are routed to Output2.

Add Tag Group

We set up an Add tag group in each Output group to position the OMR marks or Code 3of9 barcodes on the documents. We put the Add groups inside the respective Output tag groups so Enrichment will add the OMR marks or barcodes only to documents with the appropriate page counts. If we place the Add groups outside the Output groups, Enrichment will add OMR marks and Code 3of9 barcodes to all of the documents in both outputs.

Note: Xerox printers use a line drawing and shading font called FORMSX to draw lines of varying sizes. Enrichment uses this font to draw barcodes and OMR marks in Metacode data.

Since we want to apply OMR marks to the documents in Output1, we set the Add group `<ADDTYPE>` tag to OMR for that output. Enrichment gets the information that goes into the OMR marks from the `%OMRCount` variable and uses AFP line draw commands to print the OMR marks on the output. Most inserters require a “benchmark” OMR mark that is always added. To add this benchmark, we specify an `<ADDPART>` tag that adds an OMR mark 1 that is 1 character in length to every document. We then specify the `<ADDPART>` tag that instructs Enrichment to add the value of `%OMRCount`, which is up to 3 characters in length, to each document. The `<POSITION>` tag tells Enrichment to begin printing OMR marks 5 inches from the left edge and 6 inches from the top of the paper.

To add Code 3of9 barcodes to the documents in Output2, we set the `<ADDTYPE>` tag to 3OF9. We set four `<ADDPART>` tags:

- We set the first `<ADDPART>` tag to the constant value *. This is the beginning framing character for the barcode.
- We set the second `<ADDPART>` tag to instruct Enrichment to add the left-justified value of `%Barcode`, which is up to 12 characters in length, padding the value with blanks if necessary.
- We set the third `<ADDPART>` tag to instruct Enrichment to add the left-justified value of `%ChkDigit`, which is 1 character in length.
- We set the last `<ADDPART>` tag to the constant value *. This is the ending framing character for the barcode.
The `<BARS>` and `<HEIGHT>` tags specify the size of the Code 3of9 barcode Enrichment will add. Our `<BARS>` tag specifies that the narrow bars and spaces in the Code 3of9 Barcode must be 6 pels in width and the wide bars and spaces must be 18 pels in width. Similarly, the `<HEIGHT>` tag indicates that the barcode must be 40 pels tall.

Finally, since we want the Code 3of9 barcode to print so its top edge is parallel to the left edge of the paper, we specify `<ORIENT>` 4. If we allow the `<ORIENT>` tag to default to 1, the barcode will print with its top edge running parallel to the top edge of the paper. We also set the `<POSITION>` tag so that the top left corner of the barcode is 5 inches from the left edge of the paper and 10.5 inches from the top edge of the paper. Our Code 3of9 barcode will print without HRI, because we set the third and fourth `<POSITION>` tag parameters to 0.

Run Time

The following information describes how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following figure shows the JCL we used to run the application. In the JCL, we identify the input data set and the two output data sets.

```plaintext
// *jobcard
// JOBLIB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
//   DD DISP=SHR, DSN=SYS3.CLI B22.SEDCLI N K
//   DD DSN=SYS3.PLI230.SIBMLINK
//   DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// *************************************************
// SW   EXEC PGM=PDRSW000,REGION=0M
// REPORT DD SYSOUT=*   
// SYSOUT DD SYSOUT=*   
// INPUT1 DD DSN=PDR.STREAMW.HANDSON.INPUT(B05INPT),DISP=SHR
// OUTPUT1 DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B05OUT1),DISP=SHR
// OUTPUT2 DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(B05OUT2),DISP=SHR
// CONTROL DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// *************************************************
// COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,','','')
// NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(B05OUT1),DISP=SHR
// OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B05OUT1),DISP=SHR
// OUTDD DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// *************************************************
// COMP02 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,','','')
// NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(B05OUT2),DISP=SHR
// OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(B05OUT2),DISP=SHR
// OUTDD DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// *************************************************
```
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```bash
rm DD:INPUT1
rm DD:OUTPUT1
rm DD:OUTPUT2
rm DDCONTROL
ln -s /share/home1/wzanone/SunOS/handson/OMR2.CON DDCONTROL
ln -s /share/home1/wzanone/SunOS/handson/OMR3OF9.AFL DD:INPUT1
ln -s /share/home1/wzanone/SunOS/handson/omrout1 DD:OUTPUT1
ln -s /share/home1/wzanone/SunOS/handson/omrout2 DD:OUTPUT2
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=B05CONT.ws.CON
```
Basic sample 6: Postal presorting

In this section

Overview ............................................................................................................................................... 62
Files Used ........................................................................................................................................ 62
Business Scenario ............................................................................................................................... 63
Explanation of Sample ......................................................................................................................... 63
Overview

This sample illustrates how to use MailStream Plus to perform postal presort on documents. The sample routes rejected documents to a separate reject file. This sample introduces the following topics.

- Presorting automated mail
- Rejected mail
- The `PRESORTED:` section of the rule file
- System variables
  - `%%%DOCINDEX`
  - `%%%TRAY_NO`
- System Sort
- Presort tag group for system sort instead of presort program
- On mainframe systems, setting up JCL for multiple system sorts
- Presort tag group
  - `<PRETYPE>` tag
  - `<FILE>` tag
  - `<STEP>` tag
  - `<OUTFILE1>` tag
  - `<INDEXCOL>` tag
  - `<REJECTFILE>` tag

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\basic6\b06inpt.afl</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.INPUT(B06INPT)</td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td>Inline</td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td>D:\apps\basic6\b06cont.con</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.CONTROL(B06CONT)</td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\basic6\b06out1.afl</td>
</tr>
<tr>
<td>D:\apps\basic6\b06rej.afl</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B06OUT1)</td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.NEWOUT2(B06OUT2)</td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
</tr>
<tr>
<td>JCL:</td>
</tr>
<tr>
<td>D:\apps\basic6\b06jcl.jcl</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.JCL(B06JCL)</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application presorts invoices to achieve a postal discount. It uses MailStream Plus to perform the postal presort. It also creates a report which has the total number of customers and pages in each tray. If there is any mail that doesn’t get presorted, it gets placed in a separate reject file.

In the sample, above:

- Use the POSTNET information as the presort criterion.

Explanation of Sample

The sample control file (shown below) reads a print stream, calls MailStream Plus for postal presort, creates a tray report, and writes presorted documents to an output file. The control file also writes documents that could not be presorted to a reject file.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS.
Input Tag Group

As the control file shows, the Input tag group is identified in the <NAME> tag as INPUT, and the input file resides on the mainframe system at DD: INPUT, as indicated by the Input group <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

Note: If the POSTNET™ information had not been available, we would have had to use postal cleansing to generate a POSTNET™ or the <ADDRESSBLOCK> tag to find the ZIP Code.

```xml
<input>
  <name> INPUT     <! Identifiable name.  >
  <file> DD:INPUT      <! Input file name.    >
  <type> AFPL A   <! AFP line data w/ANSI controls.>
  <doc> T %AcctNum CHANGE   <! Top of document when field changes. >
  <field> %AcctNum KA   <! Find all occurrences >
    <loc> 13 60 8   <! Line 13, column 60, for 8 bytes. >
  </field>
  <field> %SortBar K     <! The delivery point barcode field >
    <loc> 12 10 12    <! Print line 12, column 10, 12 bytes. >
  </field>
</input>

<rule>
  <content>
    <! Initialize the variables used in the rule.  >
    START:
    %Oldtray = '
    %Traydocs = 0
    %Traypages = 0
    <! Everytime there is a new tray, make an entry in the report >
    <! file that has the accumulated total documents pages. >
    PRESORTED:
    if %TRAY_NO <> '' THEN  <! don't increment for rejects >
      if %Oldtray <> %TRAY_NO AND %Oldtray <> '' then
        %Report = %Traydocs | ',' | %Traypages | ',' | %Oldtray
        WRITE('DD:TRAYRPT', %Report, FB, 80, 8000)
      endif
      %Traydocs = %Traydocs + 1
      %Traypages = %Traypages + %TOTAL_PAGES
    endif
    <! Make the last entry in the report for the last tray. >
    FINISH:
    %Report = %Traydocs | ',' | %Traypages | ',' | %Oldtray
    WRITE('DD:TRAYRPT', %Report, FB, 80, 8000)
  </content>
</rule>

<output>   <! The presorted output file. >
  <name> GoodOut  <! Identifiable name. >
```

EngageOne Enrichment 7.3.0 Sample Applications Guide
Field Tag Group

We defined two Field tag groups in the application. The first field sets top-of-document. The second field defines the information to extract from each document for use by MailStream Plus in presorting the documents.

Some invoices may contain several pages, so the first <FIELD> tag identifies a variable called %%AcctNum whose value is the account number, which appears at a fixed location on every page of the input. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <LOCATION> tag indicates that the field information begins on print line 13 in column 60 and is 8 characters in length.

The Input group <DOCUMENT> tag uses the value of %%AcctNum to set top-of-document. Whenever the value of %%AcctNum changes in the input, a new document begins.

The second <FIELD> tag defines a variable called %%SortBar, whose value is the delivery point barcode information from the address. Enrichment is to extract the first occurrence of the field information in each document without changing it, so we set the <FIELD> tag action parameter to K. The <LOCATION> tag indicates that the value of %%SortBar begins 12 lines from the top of the page in column 10. The %%SortBar value is 12 characters in length.

Rules

START: Section

We initialize three variables in the START: section of the rules. The first variable, %%Oldtray, which we initially set to a blank, represents the tray number of the previous tray. We initialize the second and third variables, %%Traydocs and %%Traypages, with values of 0. %%Traydocs is the number of documents in each tray. %%Traypages is the number of pages in each tray.
**PRESORTED: Section**

In the `PRESORTED:` section, we check to see if the tray number is blank. If it is, we know the document is a reject (that is, Enrichment will write it to the reject file) and that it's not in any tray, so we don't increment the report incorrectly. If the presort program rejected the document, Enrichment sets presort system variables for that document to blank (`''`). Then, we check to see if the tray number has changed. If it has, then we create a variable called `%Report` that contains the tray information. It is written to a file called 'DD:TRAYRPT' using the WRITE function. The tray totals are then updated.

**Note:** The `PRESORTED:` computations could not have been performed in the `DOCUMENT:` section of the rules because no tray information was available then.

**FINISH: Section**

The information for the last tray is written in the `FINISH:` section of the rules.

**Output Tag Group**

We set the Output group `<NAME>` tag to `GoodOut`, which resides on the mainframe system at `DD:OUTPUT1` as indicated by the `<FILE>` tag. Enrichment will place the presorted documents in this output file.

**Presort Tag Group**

If we want to perform a postal presort, we have to explicitly tell Enrichment which program to use. There are reasons we might not want to use a presort program to presort (for example, international users may need to specify a their own presort program), but for this sample we want a USPS®-certified postal presort. Thus, we specify `<PRETYPE>LPC`, indicating the use of MailStream Plus.

Enrichment uses the values of fields and system variables identified in `<SORTPART>` tags to build a presort index. The presort index contains one record for each document in the input print stream. The Presort group `<FILE>` tag indicates that the presort index resides on the mainframe system at `DD:MSSTIN`. We defined two `<SORTPART>` tags for the application:

- Our first `<SORTPART>` tag specifies a system variable, `%DOCINDEX`, whose value is an eight-digit index number for each document in the input print stream. As the `<SORTPART>` tag indicates, Enrichment will left-justify the `%DOCINDEX` value in the presort index and pad it with blanks if it is less than 8 characters in length.
Our second `<SORTPART>` tag specifies the field variable `%%SortBar`, which extracts the POSTNET™ value from the documents for use as the presort criterion. The `%%SortBar` value is up to 12 characters in length. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 12 characters in length.

Our MailStream Plus parameter settings will further define the column positions of data in the presort index. Specifically, we will specify that the ZIP Code begins in column 9, the sector and segment (the “+4” portion of the ZIP Code) begins in column 14, and the delivery point begins in column 18 of the presort index.

After Enrichment creates the presort index and calls MailStream Plus, MailStream Plus compares the `%%SortBar` value with its production database and sorts the records in the presort index accordingly, writing the records in their new order (along with other data) to the MailStream Plus Inkjet file. The `<OUTFILE1>` tag indicates that the MailStream Plus Inkjet file resides on the mainframe system at `DD:MSWKIJ` and its records are 400 characters in length. The Inkjet file also contains one record per mail piece (that is, one record per document).

Enrichment scans the Inkjet file for the `%%DOCINDEX` value. If you use default settings, Enrichment expects the `%%DOCINDEX` value to begin in column 16 of the Inkjet file. As Enrichment locates each `%%DOCINDEX` value in the Inkjet file, it finds the corresponding document in the input print stream and writes it to the output file (`OUTPUT1`). Thus, the output file contains the documents in proper presort order. Enrichment writes input documents not listed in the Inkjet file to the reject file (`DD:REJECT`).

The `<REJECTFILE>` tag indicates that the file to use for rejected documents resides on the mainframe system at `DD:REJECT`. Enrichment places documents for which MailStream Plus cannot find ZIP Codes in its database in the reject file. Enrichment will place a document from the application in the reject file if:

- The `%%SortBar` value is blank; for example, if the application responsible for applying the POSTNET™ barcode to the document failed to do so
- The `%%SortBar` value contains an invalid ZIP Code according to the MailStream Plus database.

If MailStream Plus successfully presorts all of the documents in the input (that is, if each document is represented by a record in the presort index file), Enrichment creates an empty reject file.

The `<STEP>` tag calls MailStream Plus.
Run Time

The following information describes how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application. We made some specific additions to the JCL because we used MailStream Plus for presort:

```plaintext
//
// *************************************************************************************
// * INSERT A JOBCARD ABOVE THIS COMMENT BOX *
// *************************************************************************************

* Member Name: CALLMSP
* Last Revised: 12/15/2004
* Description: This job:
*   Executes a Streamweaver job which calls MSP.
*   - STEP1OF3 - Executes IDCAMS to remove allocated files from a prior run.
*   - STEP2OF3 - Executes IDCAMS to remove MSXX00WK VSAM file from a prior run.
*   - STEP3OF3 - Executes Enrichment using the control file yourhlq.STREAMW.CONTROL(MSPSAMPL) copied from the installation media
*   Expected Return Codes:
*    - STEP1OF3: 0
*    - STEP2OF3: 0
*    - STEP3OF3: 0

* Variables: The following lower-case variables must be changed in order to run this job:
* ** NOTE **
* Please make sure to set "CAPS ON" prior to editing.
* VARIABLE: REPLACE WITH:
* yourhlq | your high-level qualifier
* SWhlq | Location of Enrichment
* MSPhlq | Location of MSP
* volume | VSAM DASD pack volser.
```
Basic sample 6: Postal presorting

/  
/ *  
/ *  
/ * STEP1OF2 EXEC PGM=IDCAMS  
/ * SYSPRINT DD SYSOUT=*  
/ * SYSPRINT DD SYSOUT=*  
/ *      
DELETE yourhlq.MSP.MSSTN PURGE  
DELETE yourhlq.MSP.MSSTOR PURGE  
DELETE yourhlq.MSP.MSWKSS PURGE  
DELETE yourhlq.MSP.MSWKCM PURGE  
/ * Start of delete when RUNMSP column 10 is Y   */  
/ * When removing these Deletes also change the   */  
/ * DISP of these files to DISP=(OLD) in STEP2OF2 */  
DELETE yourhlq.MSP.MSWKMS PURGE  
DELETE yourhlq.MSP.MSWKCF PURGE  
/ * Start of delete when RUNMSP column 12 is Y   */  
/ * When removing these Deletes also change the   */  
/ * DISP of these files to DISP=(OLD) in STEP2OF2 */  
DELETE yourhlq.MSP.MSWMRNC PURGE  
DELETE yourhlq.MSP.MSNOUN PURGE  
/ * Start of delete when RUNMSP column 14 is Y   */  
/ * When removing these Deletes also change the   */  
/ * DISP of these files to DISP=(OLD) in STEP2OF2 */  
DELETE yourhlq.MSP.MSRDBAG PURGE  
DELETE yourhlq.MSP.MSRDTRA PURGE  
DELETE yourhlq.MSP.MSRDPLT PURGE  
DELETE yourhlq.MSP.DAT.MSMDCMR PURGE  
DELETE yourhlq.MSP.DAT.MSMDCS PURGE  
DELETE yourhlq.MSP.DAT.MSMDI CR PURGE  
DELETE yourhlq.MSP.DAT.MSMDMAC PURGE  
DELETE yourhlq.MSP.DAT.MSMDMPA PURGE  
DELETE yourhlq.MSP.DAT.MSMDMPU PURGE  
DELETE yourhlq.MSP.DAT.MSMDPDR PURGE  
DELETE yourhlq.MSP.DAT.MSMDPLR PURGE  
DELETE yourhlq.MSP.DAT.MSMDPQT PURGE  
DELETE yourhlq.MSP.DAT.MSMDSEG PURGE
DELETE yourhlq.MSP.DAT.MSMDSNR PURGE
DELETE yourhlq.MSP.DAT.MSMDSPR PURGE
DELETE yourhlq.MSP.DAT.MSMDSWR PURGE
DELETE yourhlq.MSP.DAT.MSRDCF PURGE

SET MAXCC = 0

/*
//*
//*STEP2OF3 EXEC PGM=IDCAMS
//*/SYSPRINT DD SYSOUT=* /SYSIN DD *
DELETE (yourhlq.MSP.DAT.MSXX00WK) CLUSTER PURGE
SET MAXCC = 0
DEFINE CLUSTER (NAME(yourhlq.MSP.DAT.MSXX00WK) -
  KEYS (14,0) -
  SHR(3 3) -
  REUSE -
  CYL(5 5) -
  VOL(volume) -
  RECORDSIZE(150 3014) -
  CI SZ(23704))
/*
//*
//***************************************************************************
//*  *
//* This step executes the Steamweaver program: PDRSW000 *
//*  *
//***************************************************************************
//STEP3OF3 EXEC PGM=PDRSW000,REGION=35M
//STEPLIB DD DSN=SWhlq.LOAD,DISP=SHR
//   DD DSN=MSPhlq.LOADLIB,DISP=SHR
//SYSPRINT DD SYSOUT=* //SYSTERM DD SYSOUT=* //SYSDUMP DD SYSOUT=* //SYSDUMP DD SYSOUT=* //REPORT DD SYSOUT=* //TRACECTL DD DUMMY //CONTROL DD DSN=yourhlq.STREAMW.CONTROL(MSPSAMPLE),DISP=SHR //INPUT1 DD DSN=yourhlq.STREAMW.INPUT1(MSPSAMPLE),DISP=SHR //OUTPUT1 DD DSN=yourhlq.STREAMW.OUTPUT1,
  //  DI SP=(NEW, CATLG, CATLG), UNI T=SYSDA,
  //  SPACE=(CYL, (2, 2), RLSE),
  //  DCB=(DSORG=PS, RECFM=VB, LRECL=8204)
//REJECTS DD DSN=yourhlq.STREAMW.REJECTS,
  //  DI SP=(NEW, CATLG, CATLG), UNI T=SYSDA,
  //  SPACE=(CYL, (2, 2), RLSE),
  //  DCB=(DSORG=PS, RECFM=VB, LRECL=8204)
//MSSTIN DD DSN=yourhlq.MSP.MSSTIN,
  //  UNI T=SYSDA, DI SP=(NEW, CATLG),
  //  SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
  //  DSORG=PS, RECFM=FB, LRECL=20
/*
***************************************************************************
//*  POSTAL REFERENCE FILES *
***************************************************************************
Basic sample 6: Postal presorting

// MSRFDC DD DSN=MSPhlq.USPSRFDC, DISP=SHR
// MSRFDI DD DSN=MSPhlq.USPSRFDI, DISP=SHR
// MSRFMP DD DSN=MSPhlq.USPSRFMP, DISP=SHR
// MSRFPS DD DSN=MSPhlq.USPSRFPS, DISP=SHR
// MSRFSQ DD DSN=MSPhlq.USPSRFSQ, DISP=SHR
// MSRFZD DD DSN=MSPhlq.USPSRFZD, DISP=SHR
// MSRFZM DD DSN=MSPhlq.USPSRFZM, DISP=SHR

//*****************************************************************
//*     MAILSTREAM PARAMETER FILE  *
//*****************************************************************
// MSPRICP DD DSN=yourhlq.MSP.PARMFILE, DISP=SHR

//*****************************************************************
//*     MAILSTREAM LICENSE FILE  *
//*****************************************************************
// GILICEN DD DSN=MSPhlq.GILICEN, DISP=SHR

//*****************************************************************
//*    Mailstream Input Test File  *
//*****************************************************************

//*****************************************************************
//*   Generated Files  *
//*****************************************************************
// MSNAON DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSNAON,
// DSORG=PS, RECFM=FB, LRECL=450
// MSWKIJ4 DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSWKIJ4,
// DSORG=PS, RECFM=FB, LRECL=400
// MSWKIJ6 DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSWKIJ6,
// DSORG=PS, RECFM=FB, LRECL=600
// MSSTON DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSSTON,
// DSORG=PS, RECFM=FB, LRECL=292
// MSSTOR DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSSTOR,
// DSORG=PS, RECFM=FB, LRECL=20
// MSWKCF DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=yourhlq.MSP.MSWKCF,
// DSORG=PS, RECFM=FB, LRECL=105
// MSWKCM DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=yourhlq.MSP.MSWKCM,
// DSORG=PS, RECFM=FB, LRECL=170
// MSWKNRC DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=yourhlq.MSP.MSWKNRC,
// DSORG=PS, RECFM=FB, LRECL=288
// MSWKMS DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=yourhlq.MSP.MSWKMS,
// DSORG=PS, RECFM=FB, LRECL=400
Basic sample 6: Postal presorting

// MSWKSS DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=your hlq. MSP. MSWKSS,
// DSNR=PS, RECFM=FB, LRECL=120
// MSRDBAG DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDBAG,
// RECFM=VB
// MSRDPLT DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDPLT,
// RECFM=FB, LRECL=625
// MSRDTRA DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDTRA,
// RECFM=VB
// TRAYRPT DD UNIT=SYSDA, DISP=(NEW, CATLG, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. TRAYRPT,
// DSNR=PS, RECFM=FB, LRECL=80, BLKSIZE=8000
/********************************************************************
// Report Files
********************************************************************/
// MSRPFAC DD SYSOUT=*, RECFM=FBA
// MSRPPLT DD SYSOUT=*, RECFM=FBA
// MSRPRT DD SYSOUT=*, RECFM=FBA, LRECL=133
// MSRPSTM DD SYSOUT=*, RECFM=FBA
// MSRPSTG DD SYSOUT=*, RECFM=FBA
// MSRPXLF DD SYSOUT=*, RECFM=FBA, LRECL=133
/********************************************************************
// Alternate Print files
********************************************************************/
// AUDITTRL DD SYSOUT=*, RECFM=FBA
// CONFIRM DD SYSOUT=*, RECFM=FBA
// CRRTWALK DD SYSOUT=*, RECFM=FBA
// DETAIL DD SYSOUT=*, RECFM=FBA
// MAILDAT DD SYSOUT=*, RECFM=FBA
// MANIFEST DD SYSOUT=*, RECFM=FBA
// PALSUMM DD SYSOUT=*, RECFM=FBA
// RUNSTATS DD SYSOUT=*, RECFM=FBA
// SUMMARY DD SYSOUT=*, RECFM=FBA
// USPSQUAL DD SYSOUT=*, RECFM=FBA
// USPSSTMT DD SYSOUT=*, RECFM=FBA
// ZONESUMM DD SYSOUT=*, RECFM=FBA
// ZSOETAIL DD SYSOUT=*, RECFM=FBA
/********************************************************************
// Mail.dat Files
********************************************************************/
// MSMDCLR DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your hlq. DAT. MSMDCLR
// MSMDCPT DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your hlq. DAT. MSMDCPT
// MSMDCQT DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
Basic sample 6: Postal presorting

```plaintext
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDCQT
// MSMDCSM DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDCSM
// MSMHDR DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMHDR
// MSDICR DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSDICR
// MSMDCRM DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDCRM
// MSMDCRA DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDCRA
// MSMDMPA DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDCPR
// MSMDPR DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDSNR
// MSMDSR DD UNI T=SYSDA, DI SP=( NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=your.hlq.MSP.DAT.MSMDSR
```

EngageOne Enrichment 7.3.0 Sample Applications Guide 73
Basic sample 6: Postal presorting

// MSRDCF DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
//   SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
//   RECFM=VB,
//   DSN=your hlq. MSP. DAT. MSRDCF
//**
//Temporary files
//**
// MSWKDT DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKDT
// MSWKES DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKES
// MSWKNS DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKNS
// MSWKQS DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKQS
// MSWKZS DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKZS
// MSWKS1 DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKS1
// MSWKS2 DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKS2
// MSWKS3 DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKS3
// MSWKZ1 DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKZ1
// MSWKZ2 DD UNIT=SYSDA, DISP=(NEW, PASS),
//   SPACE=(CYL, (20, 20), RLSE),
//   DSN=your hlq. MSP. MSWKZ2
// MSWKWK DD DSN=your hlq. MSP. DAT. MSXX00WK, DISP=SHR
// SORTWK01 DD UNIT=SYSDA, SPACE=(CYL, (90, 90),)
// SORTWK02 DD UNIT=SYSDA, SPACE=(CYL, (90, 90),)
// SORTWK03 DD UNIT=SYSDA, SPACE=(CYL, (90, 90),)
// SORTWK04 DD UNIT=SYSDA, SPACE=(CYL, (90, 90),)
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

sweaver -c=<controlfile>

To run the control file as-is on UNIX, use this script:

```
r m DD:INPUT1
r m DD:OUTPUT1
r m DD:TRAYRPT
r m DD:REJECT
r m B06out1
r m B06trayrpt
r m B06reject
r m DDCONTROL
l n - s /share/home1/SunOS/handson/b06cont.con DDCONTROL
l n - s /share/home1/SunOS/handson/b06inpt.afl DD:INPUT1
l n - s /share/home1/SunOS/handson/B06out1 DD:OUTPUT1
l n - s /share/home1/SunOS/handson/B06trayrpt DD:TRAYRPT
l n - s /share/home1/SunOS/handson/B06reject DD:REJECT
sweaver -c=DDCONTROL
```

Windows

To run the sample application on a Windows system you must modify the <FILE> tags in the control file to specify file paths using Windows conventions.

For example, you could change this:

```
<FILE> DD:INPUT1
```

To this:

```
<FILE> C:\samples\input1.lin
```

Note: For information on the appropriate file extensions to use when specifying print streams, see the Enrichment Language Reference.

Then, run the application by issuing the following:

```
sweaver -c=<controlfile>
```
Basic sample 7: Adding banner pages and manipulating output files

In this section

Overview........................................................................................................... 78
Files Used......................................................................................................... 78
Business Scenario........................................................................................... 79
Explanation of Sample.................................................................................... 79
Overview

This sample illustrates how to read a print stream and break it into separate outputs by department. It dynamically allocates the file for each department and adds a banner page before each file break. This sample introduces these topics.

- Adding banner pages to print streams
- Convenience breaks and thresholding
- Allocating output files
  - Static
  - Dynamic
- Banner tag group
  - <NAME> tag
  - <TYPE> tag
  - <SUBSTITUTE> tag
  - <ALLOW> tag
- Output tag group
  - <FILEMAX> tag
- <DYNAFILE> tag
  - <ALLOCATE> tag
- Rules
  - <BANNER> command
  - <FILEBREAK> command

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Basic sample 7: Adding banner pages and manipulating output files

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Other Output</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic7\b07jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B07JCL)</td>
</tr>
</tbody>
</table>

Business Scenario

This sample application separates documents by department number. A new output file is created for each department, named by department number. The documents are sorted by department number, then page count, then account number. The application adds a banner page before each document with the department number on it.

In the sample, above:

- Department number is top left.
- Account number is middle right.

Explanation of Sample

The control file we created for the application (shown below) reads an input print stream that contains several documents. Enrichment sorts the documents by department number, page count, and account number, separates the documents into
dynamically allocated output files by department, and adds a banner page before each set.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B07CONT.ws.CON contains these modifications.

### Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `INPUT`, and the input file is found on the mainframe system at `DD:INPUT`, as indicated by the Input group `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPL A`.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

**Note:** The **DEPTCover** banner contains a variable named `%Dept`.

```
<input>
  <name> INPUT </name>  <!-- Identifiable name. -->
  <file> DD:INPUT </file>  <!-- Input file name. -->
  <type> AFPL A </type>  <!-- AFP Line data with ASCII carriage controls. -->
  <doc> T %AcctNum CHANGE </doc>  <!-- New document when Account number changes. -->
  <field> %AcctNum KA </field>  <!-- Find account number. Must reference because address is not always on same line -->
  <loc> 0 2 8 </loc>
</input>

<rule>
  <content>
    PRESORTED:  <!-- Write a banner page each time the department changes. Set the name of output file to include the sequential number. -->
    if CHANGED(%Dept) then
      <banner> DEPTCover BEFORE
        %DSN = "'D966DZB.HANDSON.NEW" | DATE(J) | ".DPT" | %Dept | "."
      </banner>
    endif
  </content>
</rule>
```

```bash
<output> Output1 </output>
```

```bash
</input>
```

```bash
<banner>
  <name> DEPTCover </name>
  <type> I A
```
Basic sample 7: Adding banner pages and manipulating output files

Field Tag Group

We defined two Field tag groups in the application. The first field defines the location of the account number in each document for use by the system sort routine and to set top-of-document. The second field defines the department number information to extract from each document for use by the system sort routine and in the banner pages.

Some invoices may contain several pages, so the first <FIELD> tag identifies a variable called %%AcctNum whose value is the account number, which appears relative to a string on every page of the input. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <REFERENCE> tag tells Enrichment to set a reference point for the field on all records that have a blank carriage control and contain the string Account Number: beginning in column 44. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins 2 after the reference point, and is 8 characters in length.

The Input group <DOCUMENT> tag uses the value of %%AcctNum to set top-of-document. Whenever the value of %%AcctNum changes in the input, a new document begins.

The second <FIELD> tag defines a variable called %%Dept, whose value is the department number information from the document. Enrichment is to extract the first occurrence of the field information in each document without changing it, so we set the <FIELD> tag action parameter to K. The <LOCATION> tag indicates that the value of %%Dept begins on line 3 in column 9 of the first page of each document and is up to 3 characters in length.
Basic sample 7: Adding banner pages and manipulating output files

Rules

The **PRESORTED:** section of the rules runs after Enrichment processes the Presort tag group. In the **PRESORTED:** rules, we instruct Enrichment to do the following if the value of `%%Dept` changes from one document to the next:

- Build a dynamically-allocated data set name by concatenating the Julian date (the return from the `DATE` function) and the value of `%%Dept` with static portions of the data set name
- Add a banner page before the current document
- Break to a new file after the current document.

Banner Tag Group

The `<NAME>` tag identifies the banner page to add to each document as `DEPTCover`. We used the `<NAME>` tag value in the `<BANNER>` command in the **PRESORTED:** rule. Instead of using the `<FILE>` tag to define an external file containing the banner information, we used the Content tag group to define the banner inline. We want to print the `%%Dept` value on the banner for each document, so we set the `<SUBSTITUTE>` tag to `YES`. The banner information is impact data with ANSI carriage controls, so we specified `<TYPE>IA`.

Output Tag Group

We set the Output group `<NAME>` tag to `Output1`, but because we want to dynamically allocate any number of output files, we use the `<DYNAFILE>` tag instead of `<FILE>`. We set `<DYNAFILE>` to `%%DSN`, the name of the variable we defined in the **PRESORTED:** section of the rules. Enrichment will create each output with a unique name based on the `%%DSN` definition in the rules.

Since Enrichment will create the output files on the fly, we used the `<ALLOCATE>` tag to specify each output’s parameters. We want to create variable block (VB) outputs at `SYSDA` with 1 track of primary and secondary storage. The outputs will have a maximum record length of 8204 bytes and each block will be 27998 bytes in size.

**Note:** The `<ALLOCATE>` tag is only needed on z/OS.

Presort Tag Group

We use the Presort tag group to system sort the documents in the input. Since we won’t use a presort program, we set the `<PRETYPE>` tag to `NONE`. 
The Presort group `<FILE>` tag indicates that the presort index resides on the mainframe system at DD: INPUTA. We defined four `<SORTPART>` tags for the application:

- Our first `<SORTPART>` tag specifies a system variable, `%%DOCINDEX`, whose value is an eight-digit index number for each document in the input print stream. As the `<SORTPART>` tag indicates, Enrichment will left-justify the `%%DOCINDEX` value in the index and pad it with blanks if it is less than 8 characters in length.
- Our second `<SORTPART>` tag specifies the field variable `%%Dept`, which extracts the department number from the documents. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 3 characters in length.
- Our third `<SORTPART>` tag specifies the system variable `%%TOTAL_PAGES`. Enrichment right-justifies the total page count and will pad the value with zeroes if it is less than 3 characters in length.
- Our final `<SORTPART>` tag specifies the field variable `%%AcctNum`, which extracts the account number from the documents. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 8 characters in length.

The `<OUTFILE1>` tag indicates that the sorted Index file resides on the mainframe system at DD: OUTA and has a record length of 22. The sorted Index file contains one record per document.

Enrichment scans the sorted Index file for the `%%DOCINDEX` value. If you use default settings, Enrichment expects the `%%DOCINDEX` value to begin in column 16 of the Ink Jet file. However, we used the `<INDEXCOL>` tag to specify that the `%%DOCINDEX` value begins in column 1 of the Index file. As Enrichment locates each `%%DOCINDEX` value in the Index file, it finds the corresponding document in the input print stream and writes it to the dynamically allocated output file (`%%DSN`).

We set one `<STEP>` tag to `SORT`, which is the system sort routine. The parameters specify the sort column starts in column 9 for 3 bytes. This is where `%%Dept` is stored in the file to be sorted (INPUTA). `%%TOTAL_PAGES` starts in column 12 for 3 bytes, `%%AcctNum` starts in column 15 for 8 bytes. As indicated by the `<OUTFILE1>` tag, the sorted file is in OUTA. `%%DOCINDEX` is in column 1 of OUTA.

**Note:** On Windows, we will instead use a batch file on the `<STEP>` tag.

**Run Time**

The following example scripts could be used to run the sample application on mainframe and UNIX.
Mainframe

The following shows the JCL we used to run the application. In the DELETE and CREATE steps, we allocated all the final output files and the intermediate files used during processing. These include:

- The temporary files used during the SORT steps: INPUTA, OUTA, and INPUTB
- OUTB

```
//jobcard
//OBLIT DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
// DD DISP=SHR,DSN=SYS3.CL1B22.SEDCLINK
// DD DISP=SHR,DSN=SYS3.PL1230.SIBMLNK
// DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//*********************************************************
//INIT SET JD=97181   *Set this to the julian date format YYDDD
//*********************************************************
//DELETE EXEC PGM=PDRFILES
//INPUTA DD DSN=PDR.STREAMW.HANDSON.B07.INPUTA,
//   DISP=(MOD,DELETE,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//OUTA DD DSN=PDR.STREAMW.HANDSON.B07.OUTA,
//   DISP=(MOD,DELETE,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//OUTD1 DD DSN=PDR.STREAMW.HANDSON.NEW&JD..DPT100,
//   DISP=(MOD,DELETE,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//OUTD2 DD DSN=PDR.STREAMW.HANDSON.NEW&JD..DPT200,
//   DISP=(MOD,DELETE,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//OUTD3 DD DSN=PDR.STREAMW.HANDSON.NEW&JD..DPT300,
//   DISP=(MOD,DELETE,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//*********************************************************
//CREATE EXEC PGM=PDRFILES
//INPUTA DD DSN=PDR.STREAMW.HANDSON.B07.INPUTA,
//   DISP=(NEW,CATLG,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE),
//   DCB=(DSORG=PS,RECFM=FB,LRECL=22,BLKSIZE=2200)
//OUTA DD DSN=PDR.STREAMW.HANDSON.B07.OUTA,
//   DISP=(NEW,CATLG,DELETE),
//   UNIT=SYSDA,SPACE=(TRK,(1,1),RLSE)
//*********************************************************
//SW EXEC PGM=PDRSW000,REGION=0M
//REPORT DD SYSOUT=* 
//SYSOUT DD SYSOUT=* 
//INPUT DD DSN=PDR.STREAMW.HANDSON.INPUT(B07INPT),DISP=SHR
//INPUTA DD DSN=PDR.STREAMW.HANDSON.B07.INPUTA,DISP=SHR
//OUTA DD DSN=PDR.STREAMW.HANDSON.B07.OUTA,DISP=SHR
//CONTROL DD DSN=PDR.STREAMW.HANDSON.CONTROL(B07CONT)
//*********************************************************
//COMP01 EXEC PGM=SRSUPC,PARM=('DELTA L',L I N C M P,,')
//NEWDD DD DSN=PDR.STREAMW.HANDSON.NEW&D..DPT100,DISP=SHR
//OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B07OUT1),DISP=SHR
//OUTDD DD SYSOUT=* 
//*********************************************************
//COMP02 EXEC PGM=SRSUPC,PARM=('DELTA L',L I N C M P,,')
```
Basic sample 7: Adding banner pages and manipulating output files

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
rm DDCONTROL
rm DD:LUFILE
rm DD:INPUT
rm DD:LUERROR
rm DD:OUTPUT1
rm DD:OUTPUT2
rm DD:INPUTA
rm DD:INPUTB
rm DD:OUTA
rm DD:OUTB
ln -s /share/home1/wzanone/SunOS/handson/LSWJSORT.CON DDCONTROL
ln -s /share/home1/wzanone/SunOS/handson/LUFILE.TXT DD:LUFILE
ln -s /share/home1/wzanone/SunOS/handson/CLEANSE.AFL DD:INPUT
ln -s /share/home1/wzanone/SunOS/handson/luerror DD:LUERROR
ln -s /share/home1/wzanone/SunOS/handson/output1 DD:OUTPUT1
ln -s /share/home1/wzanone/SunOS/handson/output2 DD:OUTPUT2
ln -s /share/home1/wzanone/SunOS/handson/inputa DD:INPUTA
ln -s /share/home1/wzanone/SunOS/handson/inputb DD:INPUTB
ln -s /share/home1/wzanone/SunOS/handson/outa DD:OUTA
ln -s /share/home1/wzanone/SunOS/handson/outb DD:OUTB
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=BO7CONT.ws.CON
```
Basic sample 7: Adding banner pages and manipulating output files
Basic sample 8: Rule I/O

In this section

Overview................................................................. 88
Files Used............................................................. 88
Business Scenario.................................................. 89
Explanation of Sample.......................................... 90
Overview

This sample separates documents for mailing by department and adds appropriate department numbers to each document. It uses the account number from the document as a key for locating the department number in a sequential file. It dynamically allocates the file for each department and adds a banner page before each file break. It writes account numbers not found in the sequential file to a separate sequential file.

This sample introduces the following topics related to the Rule file:

- **LOOKUP** and **UPDATE** functions to read and write “keyed” sequential data files
- **LOOKUPV** and **WRITEV** functions to read and write KSDS VSAM files (mainframe only)
- **READ** and **WRITE** functions to read and write sequential data files on a record basis.

Enrichment provides several functions to read and write data files from the rules. These include:

- **LOOKUP** and **UPDATE** to read and write “keyed” sequential data files
- **LOOKUPV** and **WRITEV** to read and write KSDS VSAM files (on mainframe systems only)
- **READ** and **WRITE** to read and write sequential data files on a record basis.

**Note:** To read and write data from database files or files other than sequential data files or KSDS VSAM files, you must use a user-written function. User-written functions are discussed in Advanced sample 9: Using a user-written function to reverse a string on page 215.

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Control File</th>
<th>D:\apps\basic8\b08cont.con</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(B08CONT)</td>
</tr>
<tr>
<td></td>
<td>B08CONT.ws.CON</td>
</tr>
<tr>
<td>Output Print Stream</td>
<td>D:\apps\basic8\b08out1.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic8\b08out2.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic8\b08out3.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic8\b08out4.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B08OUT1)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B08OUT2)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B08OUT3)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B08OUT4)</td>
</tr>
<tr>
<td>Other Output</td>
<td>D:\apps\basic8\b08luer.txt</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(B08LUER)</td>
</tr>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\basic8\b08jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(B08JCL)</td>
</tr>
</tbody>
</table>

Business Scenario

This sample application separates the documents by department number and sorts them by account number. It dynamically allocates a separate output for each department. It also adds the department number to the documents. could be used to solve the business problem illustrated below. Finally, the application adds a banner page with the department number on it and prints the banner page before each document.
In the sample, above:

- Write the department number in the area circled.
- Use the account number as the key to find the department number in the customer file. The customer file does not contain dashes.

**Explanation of Sample**

The control file we created for the application (shown below) reads a print stream and uses information from a sequential data file to add the appropriate department number to each document. Account numbers not found in the sequential data file are written to a report file. Documents are then written to separate output files based on the department number, and banner pages are added to each document.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. B08CONT.ws.CON contains these modifications.
Input Tag Group

As the control file shows, the Input group is identified in the <NAME> tag as INPUT, and the input file resides on the mainframe system at DD:INPUT, as indicated by the Input group <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPLA.

Field Tag Group

We defined two Field tag groups in the application. The first field defines the information used to determine department number (that is, the account number) and sets top-of-document. The second field defines the area of each document to be replaced with the department number.

Some invoices may contain several pages, so the first <FIELD> tag identifies a variable called %AccountNum whose value is the account number, which appears at a fixed location on every page of the input. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <LOCATION> tag indicates that the field information begins on print line 13 in column 60 and is 8 characters in length.

The Input group <DOCUMENT> tag uses the value of %AccountNum to set top-of-document. Whenever the value of %AccountNum changes in the input, a new document begins.

The %AccountNum value is also used in a number of other places in the control file, as follows:

- In the rules, the value is parsed using the SUBSTR function and concatenated to build the key for the LOOKUP function
- In the rules, the value is used to build records in the file to which Enrichment will write account numbers not found in the input data file are written (LUERROR)
- In the Presort tag group, the value is used to build records of the presort index

**Note:** We used a field replace process to add the %Dept field. You could also add it with an Add tag group.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

**Note:** Since %Dept is a sort key, its value must be determined in the DOCUMENT: section of the rules. Otherwise, its value would not exist for the sort.
Basic sample 8: Rule I/O

```
<! Input file name. >
<type> DD:INPUT   <! AFP Line data with ASCII carriage controls.>
<doc> T %AcctNum CHANGE  <! New document when Account number changes. >
<field> %AcctNum KA
   <loc> 13 60 8  <! Get 8-byte account number. >
<field> %Dept  R  <! Replace the first occurrence on line 5, column 67 for 3 bytes. >
<field> %DocNo  R  <! Replace the first occurrence on line 5, column 67 for 3 bytes. >
<input>
<rule>
<content>
DOCUMENT:
<! Build key for file lookup. >
<! The account number on the file does not include the dash as it does on the document. >
%%Key = SUBSTR(%%AcctNum,1,3) | SUBSTR(%%AcctNum,5,4)
<! File lookup for department number. >
<! If the account number is not in the file a record is written to a sequential file. >
%%Record = LOOKUP('DD:LUFILE',%%Key,1,7,Y)
if %%RC = 0 then
   %%Dept = JUSTIFY(SUBSTR(%%Record,8,3),L,3,' ')
else
   %%Dept = ' ';
%%DocNo = JUSTIFY(%%DOCUMENT_NO,R,5,0)
%%ErrorRecord = %%AcctNum | %%DocNo
%%err = WRITE('DD:LUERROR',%%ErrorRecord,VB,8204)
endif
PRESORTED:
<! Write a banner page each time the department changes. Set the name of output file to include the department number. >
if CHANGED(%%Dept) then
   <banner> DEPTCover BEFORE
   %%DSN = ""D966D2B.HANSON.NEWOUT1.B08D" | %%Dept | ""
   <filebreak>
endif
<output> Output1
</content>
</rule>
<banner>
</banner>
<output>
<name> Output1   <! Identifiable name. >
<dynafile> %%DSN
<allocate> SYSDA 1 1 TRKS VB 8204 27998
<filemax> M
<presort>
<pretype> NONE  <! Not LPC or Group1, but other. >
<file> DD:INPUTA       <! Indexed sort key file for processing. >
```

EngageOne Enrichment 7.3.0 Sample Applications Guide 92
Basic sample 8: Rule I/O

The second <FIELD> tag defines a variable called %Dept, whose value is the blank area on the first page of each document that Enrichment is to replace with the department number. We want to replace the first occurrence of the field in each document, so we set the <FIELD> tag action parameter to R. The <LOCATION> tag indicates that the value of %Dept begins 5 lines from the top edge of the page in column 67. The %Dept value is 3 characters in length.

Note: This illustrates a field replace method for adding information to a document. We could have used an Add tag group instead.

We used field replace to add the department number to the documents because the technique doesn’t actually add records to the print stream. Field replace is a good technique for AFP line data since adding records could adversely affect printing. For example, if we used the Add tag group to add the department number to the documents, we might add an additional record, which could alter AFP page resources.

Rules

DOCUMENT: Section

In the DOCUMENT: section of the rules, we do the following:

• Build the key for the LOOKUP function
• Perform file LOOKUP

The account number contains a dash that does not appear in the input data file. Therefore, we have to remove the dash before we attempt to use the account number as the key for looking up a document's department number.

In the rules, we used the SUBSTR function to extract two substrings from the %AcctNum value. The dash in the account number is in position 4, so the first SUBSTR extracts 3 characters, beginning in position 1. Our second SUBSTR extracts 4 characters, beginning in position 5. We concatenated the two SUBSTR functions into a new variable, %Key.
Then we used a \texttt{LOOKUP} function to search the input data file at DD:LUFILE (which we know has already been sorted by account number) for the value of \texttt{%%%Key}, beginning in column 1 for 7 characters. We set the return from the \texttt{LOOKUP} function as the value of \texttt{%%Record}. The following shows the contents of the file at DD:LUFILE:

\begin{verbatim}
1479347JA2
3015555BA1
3452113AD2
\end{verbatim}

If the \texttt{LOOKUP} function finds the \texttt{%%%Key} value in the data file (\texttt{%%%RC = 0}), we use the \texttt{SUBSTR} function to extract 3 characters from the \texttt{%%%Record} value, beginning in column 8. Using the \texttt{JUSTIFY} function, we left-justify the return from the \texttt{SUBSTR} function and set the return as the value of \texttt{%%%Dept}. The \texttt{%%%Dept} value will:

- Replace the blank space extracted from the input documents
- Complete the file names of dynamically allocated files
- Print on the banner pages we’ll add to each document.

However, if the \texttt{LOOKUP} function doesn’t find the \texttt{%%%Key} value in the input data file, Enrichment:

- Sets \texttt{%%%Dept} to 3 blanks
- Sets a new variable, \texttt{%%%DocNo} to the right-justified value of the system variable \texttt{%%%DOCUMENT\_NO}
- Writes a record to the output data file at DD:LUERROR.

\textbf{Note:} We justified \texttt{%%%DOCUMENT\_NO} so other programs that require a specific format can read DD:LUERROR.

We want to write two data items to the error record: the account number of the document and the document number of the document. We concatenated the values of \texttt{%%%AcctNum} and \texttt{%%%DocNo} into a new variable, \texttt{%%%ErrorRecord}. We used the \texttt{WRITE} function to ensure that Enrichment opens DD:LUERROR with a specific record format (variable blocked—VB) and record length (8204). The variable \texttt{%%%err} is simply a placeholder for the \texttt{WRITE} function. Most functions return a value to the variable on the left of the equal sign. The \texttt{WRITE} function doesn’t. We used the placeholder for consistency.

\textbf{PRESORTED: Section}

The \textbf{PRESORTED:} section of the rules runs after Enrichment processes the Presort tag group. In the \textbf{PRESORTED:} rules, we instruct Enrichment to do the following if the value of \texttt{%%%Dept} changes:

- Add a banner page before the current document
- Build a dynamically-allocated data set name by concatenating \texttt{%%%Dept} with static portions of the data set name
• Break to a new file after the current document.

Banner Tag Group

The banner page to add to each document is identified by the <NAME> tag as DEPTCover. We used the <NAME> tag value in the <BANNER> command in the PRESORTED: rules. Rather than using the <FILE> tag to define an external file containing the banner page information, we used the Content tag group to define the banner page inline. Since we want to print the %%Dept value on the banner page for each document, we set the <SUBSTITUTE> tag to YES. The banner information is impact data with ANSI carriage controls, so we specified <TYPE>I A.

Output Tag Group

We set the Output group <NAME> tag to Output1, but because we want to dynamically allocate any number of output files, we use the <DYNAFILE> tag instead of the <FILE> tag. We set the <DYNAFILE> tag to %%DSN, the name of the variable we defined in the PRESORTED: rules. Each output created by Enrichment will have a unique name based on the %%DSN definition in the rules.

Since Enrichment will create the output files on the fly, we used the <ALLOCATE> tag to specify each output’s parameters. We want to create variable block (VB) outputs at SYSDA with 1 track of primary and secondary storage. The outputs are to have a maximum record length of 8204 bytes and each block will be 27998 bytes in size.

Note: The <ALLOCATE> tag is only needed on z/OS.

Presort Tag Group

We use the Presort tag group to perform a system sort of the documents in the input. Since we won’t use a presort program, we don’t specify a <PRETYPE> tag.

The Presort group <FILE> tag indicates that the presort index resides on the mainframe system at DD:INPUTA. We defined three <SORTPART> tags for the application:

• Our first <SORTPART> tag specifies the system variable %%DOCINDEX. As the <SORTPART> tag indicates, Enrichment will left-justify the %%DOCINDEX value in the presort index and pad it with blanks if it is less than 9 characters in length. The actual %%DOCINDEX value is only eight characters in length, but we added an extra character for readability.
Basic sample 8: Rule I/O

- Our second `<SORTPART>` tag specifies the field variable `%%Dept`, which extracts the department number from the documents. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 3 characters in length.

- Our third `<SORTPART>` tag specifies the field variable `%%AcctNum`, which extracts the account number from the documents. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 8 characters in length.

The `<OUTFILE1>` tag indicates that the sorted Index file resides on the mainframe system at `DD:OUTA` and has a record length of 20. The sorted Index file contains one record per document.

Enrichment scans the sorted Index file for the `%%DOCUMENT INDEX` value. We set the `<INDEXCOL>` tag to 1, indicating that the `%%DOCUMENT INDEX` value begins in column 1 of the Index file. As Enrichment locates each `%%DOCUMENT INDEX` value in the Index file, it finds the corresponding document in the input print stream and writes it to the dynamically allocated output file (`%%DSN`).

We set one `<STEP>` tag to `SORT`, the system sort routine. The parameters specify the sort column starts in column 10 for 3 bytes. This is where `%%Dept` is stored in the file to be sorted (`INPUTA`). `%%AcctNum` starts in column 13 for 8 bytes. As indicated by the `<OUTFILE1>` tag, the sorted file is in `OUTA`. `%%DOCUMENT INDEX` is in column 1 of `OUTA`.

**Note:** Refer to your system documentation for details about system sort programs and parameters.

**Note:** On Windows, we will instead a batch file on the `<STEP>` tag.

Run Time

The following information describes how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following code shows the JCL we used to run the application. In the DELETE and CREATE steps, we allocate all the final output files and the intermediate files used during processing. These include the temporary files used during the SORT steps: `INPUTA`, `OUTA`, and `INPUTB`, `OUTB`.

```mlist
// *JOBCARD
// JOB JOB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
// DD DD DS=SHR,DS=SYS3.CLIB22.SEDCLINK
// DD DS=SHR,DS=SYS3.PLI230.SIBMLINK
// DD DS=SYS3.V4R1M0.ISPLOAD,DISP=SHR
```
Basic sample 8: Rule I/O

//*************************************************
//DELETE  EXEC PGM=PDRFILES
//INPUTA   DD DSN=PDR.STREAMW.HANDSON.B08.INPUTA,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//OUTA     DD DSN=PDR.STREAMW.HANDSON.B08.OUTA,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//OUT1     DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08D,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//OUT2     DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DAD2,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//OUT3     DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DBA1,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//OUT4     DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DJA2,
//   DI SP=|MOD, DELETE, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//*************************************************

//CREATE  EXEC PGM=PDRFILES
//INPUTA   DD DSN=PDR.STREAMW.HANDSON.B08.INPUTA,
//   DI SP=|NEW, CATLG, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//   DCB=|DSORG=PS, RECFM=FB, LRECL=20, BLKSIZE=2000|
//OUTA     DD DSN=PDR.STREAMW.HANDSON.B08.OUTA,
//   DI SP=|NEW, CATLG, DELETE|
//   UNI T=SYSDA, SPACE={TRK,(1,1), RLSE}
//*************************************************

//SW   EXEC PGM=PDRSW000, REGION=0M
//REPORT   DD SYSOUT=* 
//SYSOUT   DD SYSOUT=* 

// -------------------------------------------------------------
//COMP01   EXEC PGM=ISRSUPC, PARM=(DELTAL, LINECMP, '','')
//NEWDD    DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08D, DI SP=SHR
//OLDDD    DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(B08OUT1), DI SP=SHR
//OUTDD    DD SYSOUT=* 

// -------------------------------------------------------------
//COMP02   EXEC PGM=ISRSUPC, PARM=(DELTAL, LINECMP, '','')
//NEWDD    DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DAD2, DI SP=SHR
//OLDDD    DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(B08OUT2), DI SP=SHR
//OUTDD    DD SYSOUT=* 

// -------------------------------------------------------------
//COMP03   EXEC PGM=ISRSUPC, PARM=(DELTAL, LINECMP, '','')
//NEWDD    DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DBA1, DI SP=SHR
//OLDDD    DD DSN=PDR.STREAMW.HANDSON.GOODOUT3(B08OUT3), DI SP=SHR
//OUTDD    DD SYSOUT=* 

// -------------------------------------------------------------
//COMP04   EXEC PGM=ISRSUPC, PARM=(DELTAL, LINECMP, '','')
//NEWDD    DD DSN=PDR.STREAMW.HANDSON.NEWOUT.B08DJA2, DI SP=SHR
//OLDDD    DD DSN=PDR.STREAMW.HANDSON.GOODOUT4(B08OUT4), DI SP=SHR

EngageOne Enrichment 7.3.0  Sample Applications Guide  97
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
rm DDCONTROL
rm DD:LUFILE
rm DD:LUERROR
rm b08luerror
rm DD:INPUT
rm DD:INPUTA
rm b08inpta
rm b08outa
rm DD:OUTA
ln -s /share/home1/SunOS/handson/b08cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/b08inpt.afl DD:INPUT
ln -s /share/home1/SunOS/handson/b08lutb.txt DD:LUFILE
ln -s /share/home1/SunOS/handson/b08luerror DD:LUERROR
ln -s /share/home1/SunOS/handson/b08inpta DD:INPUTA
ln -s /share/home1/SunOS/handson/b08outa DD:OUTA
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=B08CONT.ws.CON
```
Basic sample 9: Cleansing addresses and adding IMB Barcodes

In this section

Overview.............................................................................................................. 100
Files Used........................................................................................................ 100
Business Scenario.......................................................................................... 101
Explanation of Sample.................................................................................... 101
Basic sample 9: Cleansing addresses and adding IMB Barcodes

Overview

This sample illustrates how to cleanse and standardize addresses in customer invoices and add an IMB barcode below the address block so that the IMB and the address are clearly readable through the envelope window. This sample illustrates the following topics:

- Postal cleansing
- IMB barcodes
- Input tag group
- Address tag group
- CASS tag group
- Add tag group
- Using the Add tag group within the Output group

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
</tr>
<tr>
<td>Rule File</td>
</tr>
<tr>
<td>Other Inputs</td>
</tr>
<tr>
<td>Control File</td>
</tr>
<tr>
<td>Output Print Stream</td>
</tr>
<tr>
<td>Other Output</td>
</tr>
</tbody>
</table>
Basic sample 9: Cleansing addresses and adding IMB Barcodes

Business Scenario

This sample application standardizes the addresses on customer invoices. It also adds an IMB barcode below the address block.

Explanation of Sample

Our control file reads a print stream that contains multiple documents, passes address information to a cleansing program and adds the IMB below the address on each document.

Note: This control file is designed for running Enrichment on a Windows system.
Basic sample 9: Cleansing addresses and adding IMB Barcodes

Input Tag Group

As the control file shows, the Input tag group is identified in the `<NAME>` tag as `<INPUT>`, and the input file is found on the window system, as indicated by the Input group `<FILE>` tag. The input is PDF data, so we set the `<TYPE>` tag to `<PDF>` because we will use address cleansing software to standardize addresses in the input. Because the input is PDF data and the addresses to be cleansed are in a consistent position on the first page of every document, we are using `<WINDOW>` tag to capture address block and address consists of 3 lines, and we want to replace the first occurrence of the address in each document.

```
<INPUT>
  <NAME> INPUT1
  <FILE> XPL99_Part1.PDF
  <TYPE> PDF
  <FIELD> %%AccountExec K1 n
    <WINDDOW> 5.00 0.60 7.00 0.80 IN
  </FIELD>
  <FIELD> %%Address R3
    <WINDDOW> 4.00 1.80 6.60 2.40 IN
  </FIELD>
  <document> T %%AccountExec E
    <address>
      <line> %%Address[0]
      <line> %%Address[1]
      <line> %%Address[2]
    </address>
  </INPUT>
```

Field Tag Group

We defined two field tag groups for the application to define top-of-document and address values. The first field group identifies a variable called `%AccountExec`. Because some customer invoices may contain several pages, we must find unique criteria by which to identify the first page of each invoice (that is, the top of each document). In the input print stream, the account executive appears at a fixed location on every page. The value of the field is the customer’s account executive number on each page of the invoices. We only want to extract the field information, so we set the `<FIELD>` tag action parameter to K. The `<WINDDOW>` tag indicates that the field information begins from 5.00 0.60 7.00 0.80 in inches (rectangular). The Input group `<DOCUMENT>` tag uses the value of `%AccountExec` to set top-of-document. When the `%AccountExec` value changes in the input (that is, when the account executive information changes), a new document begins.

The second field group identifies a variable called `%Address` to capture address block information. We want to extract and replace the field information (cleanse from finalist), so we set the `<FIELD>` tag action parameter to R. The `<WINDDOW>` tag indicates that the field information begins from 4.00 1.80 6.60 2.40 in inches (rectangular).
Basic sample 9: Cleansing addresses and adding IMB Barcodes

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```xml
<INPUT>
  <NAME> INPUT1
  <FILE> XPL99_Part.PDF
  <TYPE> PDF
  <FIELD> %%AccountExec K1 n
    <WIN NWOW> 5.00 0.60 7.00 0.80 IN
  </FIELD>
  <FIELD> %%Address R3
    <WIN NWOW> 4.00 1.80 6.60 2.40 IN
  </FIELD>
  <DOCUMENT> T %%AccountExec E
    <CLEANSE> y
    <ADDRESS>
      <LINE> %%Address[0]
      <LINE> %%Address[1]
      <LINE> %%Address[2]
    </ADDRESS>
  </DOCUMENT>
</INPUT>
```

**Cass Tag group**

We included a CASS tag group in the control file because we want to cleanse the addresses in the input. We specified Finalist as the address cleansing program by setting the `<CASSTYPE>` tag to LPC.

**Rules**

We are adding IMB barcode to every document. Most of the values are hardcoded, only `%%DOCUMENT_NO` and `%%ZIPCODE` variable values vary with every document.

**Output Tag Group**

In the `<OUTPUT>` command group, we set the `<NAME>` tag to Output, which resides on the window system as indicated by the `<FILE>` tag. The output file has all the updated addresses in every document.
Add Tag Group

Finally, we set up an Add tag group to build and position the IMB barcode on the documents. We include the Add group within the Output group that defines Output. Since the address appears only on the first page of each document in Output, we set the <ONPAGE> tag for the IMB barcode to FIRST, indicating that Enrichment will add the barcode only to the first page of each Output document.
Basic sample 10: Sorting, splitting and adding QR Barcodes

In this section

Overview ................................................................. 106
Files Used ............................................................. 106
Business Scenario .................................................. 107
Explanation of Sample .......................................... 107
Overview

This sample illustrates how to sort documents within a single PDF data stream by a field, outsort documents to multiple outputs by document count and add an IMB barcode. This sample introduces the following topics:

- Sortmatch tag group
- QR barcodes
- Input tag group
- Rule tag group
- <OUTPUT> command
- Multiple Output group
- Using the Add tag group within the Output group

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
</tbody>
</table>
Basic sample 10: Sorting, splitting and adding QR Barcodes

Business Scenario

This sample application takes a customer invoice file and sorts it on the basis of account number in the increasing order, then outsorsts it into two separate outputs based on document count: odd-document number invoices and even-document number invoices. It also adds the QR barcode within each grouping.

Explanation of Sample

The control file reads a print stream that contains multiple documents of the same page count, sorts the documents by account number, outsorsts them to separate outputs by page count and adds IMB barcodes to each output group.

Note: This control file is designed for running Enrichment on a Windows system.

Input Tag Group

In the control file, the Input group is identified in the <NAME> tag as INPUT, and the input file is found on the windows system at PBSampleData.pdf, as indicated by the <FILE> tag. The input is PDF data, so we set the <TYPE> tag to PDF F.

```xml
<input>
  <name> input</name>
  <type> F</type>
  <file> PBSampleData.pdf</file>
  <field> %%AccountNumber K1</field>
  <window> 7.40 0.90 7.90 1.10 </window>
</input>
```
Basic sample 10: Sorting, splitting and adding QR Barcodes

```
<field>
  <doc> 1 
</input>

<sortmatch>
  <inputname> input
  <sort> %%AccountNumber A Y 
</sortmatch>

<rule>
  <content>
    if %%DOCUMENT_NO # 2 = 0 then
      <OUTPUT> EvenNoOfDocs 
    else
      <OUTPUT> OddNoOfDocs 
    endif
  </content>
</rule>
```

### Field Tag Group

We need to sort the documents within the output by a field, so we define a field variable called `%%AccountNumber` in the Field tag group to extract the customer account number from each document. We want to extract the `%%AccountNumber` value from every page of each document, so we set the `<FIELD>` tag action parameter to `K` (keep the field information). The `<WINDOW>` tag indicates that the customer’s account number starts from `7.40 0.90 7.90 1.10 IN`. If we look at the input, we see that based on the criteria we specified in the `<DOCUMENT>` tag, and since each document is only one page, so the value is set to 1.

### Sortmatch Tag Group

The Sortmatch tag group defines how to sort the documents in the input by customer account number. First, the Sortmatch group `<INPUTNAME>` tag specifies the input to sort. The `<INPUTNAME>` tag parameter value duplicates the name value of the Input group `<NAME>` tag (in this case, input). Since we know there is only one document per customer number in the input, there is no need to match the documents.

The `<SORT>` tag identifies the sort criterion and direction. In the `<SORT>` tag, we specify the field by which to sort (`%%AccountNumber`). We want to sort the documents from lowest account number (1) to highest customer number (12), so we set the `<SORT>` tag order parameter to `ASCEND`. Also these account numbers are numeric only, we have to set the third parameter to `Y`.
Output Tag Group

We will need to use conditional processing (rules) to outsort the documents by document count. We want to outsort even documents to one output and odd documents to the second output. As required, we will use a separate Output tag group to define each output. In the first Output group, we set the <NAME> tag to EvenNoOfDocs and we set the <FILE> tag to EvenDocsOut.pdf, the location of the corresponding output file. Similarly, we name the second Output group OddNoOfDocs and set up a corresponding file at OddDocsOut.pdf.

Rules

The rules needed in order to perform the outsort are relatively simple, so rather than setting up a separate rule file, we will include the rules in a Content tag group within the Rule tag group. Since we are using rules to process documents in the input, we define rule processing in the DOCUMENT: section of the inline rule. We need to know the document count so Enrichment can properly outsort it. an Enrichment system variable, %%DOCUMENT_NO, keeps track of the total number of documents in the input file. Therefore, we can use the %%DOCUMENT_NO value as the outsort criterion and using simple calculation to outsort the document on the basis of whether the document count is even or odd. Our rules, then, look like this:

```
<rule>
  <content>
    if %%DOCUMENT_NO # 2 = 0 then
      <OUTPUT> EvenNoOfDocs
    else
      <OUTPUT> OddNoOfDocs
    endif
  </content>
</rule>
```

We used <OUTPUT> commands in the rules to direct the documents to the proper outputs. Notice that each <OUTPUT> command value duplicates the value of one of the Output group <NAME> tags we set up.

When the application runs, Enrichment sorts all of the documents in ascending order by account number. Then, as defined in the rules, Enrichment uses the %%DOCUMENT_NO value to outsort the documents. Enrichment starts writing output in even or odd outputs based on the document value. In the end, Enrichment places all odd documents into the OddNoOfDocs group and even documents into the EvenNoOfDocs group.
Add Tag Group

Finally, we set up an Add tag group to build and position the QR barcode on each document. We include one Add group within the Output group that defines output. Both output groups have different QR barcodes.
Basic sample 11: Adding a PDF hyperlink

In this section

Overview ........................................................................................................ 112
Files Used .................................................................................................. 112
Business Scenario .................................................................................... 113
Explanation of Sample ............................................................................. 113
Overview

This sample illustrates how to extract an address and construct a hyperlink using that address. This sample includes the following topics:

- Input tag group
- Add tag group
- Using the Add tag group within the Output group

Files Used

This sample application uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
</tr>
<tr>
<td>Rule File</td>
</tr>
<tr>
<td>Other Inputs</td>
</tr>
<tr>
<td>Control File</td>
</tr>
<tr>
<td>Output Print Stream</td>
</tr>
<tr>
<td>Other Output</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application extracts the address from a customer communication and constructs a URL so the customer could click on a link in a PDF file, and locate their house on a mapping website.

Explanation of Sample

Our control file reads a print stream that contains multiple documents, extracts the address information, and constructs a URL to apply to each document.

Note: This control file is designed for running Enrichment on a Windows system.

Input Tag Group

In the control file, the Input group is identified in the <NAME> tag as INPUT, and the input file is found on the Windows system at AddressSample.pdf, as indicated by the <FILE> tag. The input is PDF data, so we set the <TYPE> tag to PDF F.

```xml
<INPUT>
  <NAME> INPUT1
  <FILE> AddressSample.PDF
  <TYPE> PDF
  <FIELD> %%AddressLines K2
          <WINDOW> 1.06 1.55 3.08 1.92 IN
  </FIELD>
  <DOCUMENT> 1
</INPUT>
```
Field Tag Group

We need to extract the address lines, so we define a field variable called %AddressLines in the Field tag group to extract the address lines from each document. The <WINDOW> tag indicates that the customer’s address line starts from 1.06 1.55 3.08 1.92 IN. If we look at the input, we see each document is only one page, so the value for <DOCUMENT> is set to 1.

Rules

The rules needed in order to perform the outsort are relatively simple, so rather than setting up a separate rule file, we will include the rules in a Content tag group within the Rule tag group. Since we are using rules to process documents in the input, we define rule processing in the DOCUMENT: section of the inline rule. These rules, at time of publication, will create a URL that will take the user to a map of the specified address. This could change in the future, or a different mapping website could be used.

Our rules, then, look like this:

```xml
<rule>
  <content>
    %Line1 = TRANSLATE(%AddressLines[0], ' ', '+')
    %Line2 = TRANSLATE(%AddressLines[1], ' ', '+')
    %url = "https://www.google.com/maps/place/" | %Line1 | ',' | %Line2
  </content>
</rule>
```

Add Tag Group

Finally, we set up an Add tag group to build and position the hyperlink on each document. We use the <hyperlink> tag to define what the destination URL will be. We use <underline> to make the text underlined.

```xml
<ADD>
  <addtype> TEXT
  <addpart> "Click here to find your house"
  <orient> 1
  <position> 3.41 1.79 IN
  <hyperlink> %url
  <color> BLUE
  <underline> YES
</ADD>
```
Run Time

The following information describes how to run the sample application on Windows systems

To run the sample application on a Windows system, issue the following:

```
swaevr -c=b11cont.con
```
Basic sample 11: Adding a PDF hyperlink
Advanced sample 1: Adding inserts to documents

In this section

Overview .................................................................................................................. 118
Files Used ............................................................................................................... 118
Business Scenario .............................................................................................. 119
Explanation of Sample ........................................................................................ 120
Overview

This sample adds a banner page when a field value changes. It accumulates the total amount due and writes a total page at the end of the output file. It also inserts a personalized page and a record for each document. This sample introduced the following topics.

- Insertpage tag group
  - <SUBSTITUTE> tag
  - Creating files that will be the last document in an output print stream
- Insertrec tag group
  - <PLACE> tag
  - <SUBSTITUTE> tag
- Using the Field tag group to indicate placement of an insert page or insert record
- Rule file
  - Comparing a variable to its previous value
  - Setting counters for totals
  - <APPEND> command

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Advanced sample 1: Adding inserts to documents

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Other Output</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv1\a01jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A01JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv1\a01scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A01SCR)</td>
</tr>
</tbody>
</table>

Business Scenario

The input is one set of invoices sorted by territory. You need to process them as shown below and produce a single output. You also need to add a gift certificate notification to each document, personalized by account number. The gift amount is based on the customer's amount due. The customer will call to get the amount.

In the sample, above:

- Add the circled message to the top of every document.
- Territory is indicated by the first three characters of the account number. When territory changes, insert a banner page so that the printer operator can break the output there.
- Print total number of invoices and the total amount due on the new last page of the document.
Advanced sample 1: Adding inserts to documents

Explanation of Sample

The control file uses a number of techniques to add inserts to a print stream:

- The Banner tag group is used to add banner pages
- The Insertpage tag group is used to add additional pages
- The Insertrec tag group is used to add an additional record to a page

Note: This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A01CONT.ws.CON contains these modifications.

Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `Input`, and the input file is found on the mainframe system at `DD:INPUT1`, indicated by the `<FILE>` tag. The input is impact data with ANSI carriage controls, so we set the `<TYPE>` tag to `IA`, allowing `charset` to default to E.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.
For the application, we need to extract three data items from documents in the input:
• The account number
• The amount due
• A blank area on the first page of each document to use as a placeholder for an inserted record.

To this end, we defined three Field tag groups.

The first Field tag group serves a dual purpose in the application: it defines the location of the top-of-document indicator for the print stream, and it defines the account number used in the gift letter we’ll insert in every document. The <FIELD> tag defines a variable called %AcctNo. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA (use and keep all occurrences of the field information in each document). Because the amount due occupies the same position on every document, we did not need to specify a reference point to find the field. Therefore, we didn’t use a <REFERENCE> tag. The <LOCATION> tag indicates that the field information occupies line 13 of the page, begins in column 60, and is 8 characters in length.

The Input group <DOCUMENT> tag uses the value of %AcctNo to set top-of-document. Whenever the %AcctNo value changes in the print stream, a new document begins.

The second Field group identifies a variable (%AmtDue) whose value is the customer’s amount due. Since we only want to extract the first occurrence of the information, we set the <FIELD> tag action parameter to K. The <LOCATION> tag indicates that the amount due information always begins on row 14 in column 60, and is up to 10 characters in length.

The final Field tag group defines a field used only as a marker so Enrichment will know where to insert a record. The <FIELD> tag defines a variable called %Stmt_Terr. We want to extract the first occurrence of the variable from each document, so we set the <FIELD> tag action parameter to K. The <LOCATION> tag indicates that the field information always begins on row 1 in column 1, and is 1 character in length.

Banner Tag Group

We used two Banner tag groups to identify the locations of banner pages for the application. <BANNER> commands in the rules define where to place the banner pages in the output.

The first Banner group is identified in the <NAME> tag as Totals_Page. It defines the banner page to place at the end of the output to report the total number of invoices in the output and the total amount due for all of the invoices. We used the <NAME> tag value to identify the banner in the rules. The <FILE> tag specifies that the banner information is on the mainframe system at DD: TOTALS. The banner is impact data with ANSI carriage controls, so we set the <TYPE> tag to IA. The information in the banner
Advanced sample 1: Adding inserts to documents

Page is shown below. The banner contains two variables, \texttt{%%TotalDocs} and \texttt{%%TotalDue}. We set the \texttt{<SUBSTITUTE>} tag to \texttt{Y} so Enrichment would substitute these variables with the appropriate values during processing. The variables are defined in the rules.

\begin{verbatim}
*******************************************************************************
* TOTAL INVOICES = %%Total_Docs
* TOTAL AMOUNT DUE = %%Total_Due
*******************************************************************************
\end{verbatim}

The second Banner group is identified in the \texttt{<NAME>} tag as Territory. It defines a banner that Enrichment will insert at the beginning of each territory's documents in the output. Again, we used the \texttt{<NAME>} tag value to identify the banner in the rules. We used a Content tag group to include the banner in the control file. The banner is impact data with ANSI carriage controls, so we set the \texttt{<TYPE>} tag to \texttt{IA}. The information in the banner page is shown below. Notice that the banner contains one variable, \texttt{%%Territory}, which we defined in the rules. We set the \texttt{<SUBSTITUTE>} tag to \texttt{Y} so Enrichment would substitute this variable with the appropriate values during processing. We set the \texttt{<ALLOW>} tag to \texttt{PAGECOUNT}, which specifies that the banner information is to be included in calculating the output page count.

\begin{verbatim}
<content> <! Instream content for data >
1
Banner Page
Banner Page Territory: %%Territory
Banner Page
</content>
\end{verbatim}

\textbf{Note:} In UNIX, unless the file is completely ASCII and the Input data is also completely ASCII, using the Content tag group to include inline files could cause a compatibility problem with the input print stream and the insert record or insert page.

Insertpage Tag Group

The Insertpage tag group identifies the location of the gift letter we want to insert in each document. The insert page is identified in the \texttt{<INSERTPAGE>} tag as \texttt{%%Award}. The file containing the gift letter is found on the mainframe system at \texttt{DD:AWARD}, as indicated by the \texttt{<FILE>} tag. The letter is impact data with ANSI carriage controls, so we set the \texttt{<TYPE>} tag to \texttt{IA}, allowing \texttt{charset} to default to E. The following shows the information in the insert page. Notice that the insert contains two variables, \texttt{%%AcctNo} and \texttt{%%AmtDue}. We set the \texttt{<SUBSTITUTE>} tag to \texttt{Y} so Enrichment would substitute these variables with the appropriate values during processing. We defined both variables in Field tag groups. We specified no \texttt{<PLACE>} tag in the Insertpage group because we used the \texttt{<APPEND>} command in the rules to place the insert.

\begin{verbatim}
1
\end{verbatim}
Dear Customer,

Your account number, %AcctNo, has been selected to receive a gift certificate of your choice. We appreciate your business.

The gift certificate is based on your current monthly charge amount your gift certificate.

Your current charge amount is: %AmtDue

We hope you can put this gift to good use. If you have any questions, please call us.

Sincerely,

John Smith
Vice President

Insertrec Tag Group

The Insertrec tag group identifies a record we want to insert in each document and specifies where to add it. The insert record is identified in the <INSERTREC> tag as %Stmt_Terr. We used a Content tag group to include the insert record in the control file. The record is impact data with ANSI carriage controls, so we set the <TYPE> tag to IA, allowing charset to default to E. The information in the insert record is shown below. Notice that the insert contains the variable %Territory. We set the <SUBSTITUTE> tag to Y so Enrichment would substitute the variable with the appropriate values during processing. %Territory is defined in the rules. The <PLACE> tag instructs Enrichment to insert the record WITHIN each document. This means Enrichment will insert the record after the record on which it encounters %Stmt_Terr in the document.

<content>

Use this territory code with all correspondence: %Territory

</content>
Rules

START: Section

We used a START: section in the inline rule to set the initial values of %Total_Due and %Total_Docs to 0.

DOCUMENT: Section

We used the DOCUMENT: section rules to:

- Extract the territory number as a substring of the account number
- Decide when to insert the banner Territory
- Increment a counter that sets the %Total_Docs variable
- Convert the value of %AmtDue to a numeric value
- Calculate the value of %Total_Due
- Append the gift letter %Award to each document.

The territory number is a substring of the account number. We used the SUBSTR function to extract the first three characters of %AcctNo and return them as the value of %Territory.

We then used the CHANGED function to compare the value of %Territory in the current document to its value from the previous document. If the value changed, we used the <BANNER> command to insert the banner Territory before the current document.

Then we incremented the document counter by adding 1 to the value of %Total_Docs. Because DOCUMENT: rules are processed once for each document in the input, the value of %Total_Docs will always be the total number of documents processed. Initializing %Total_Docs in the START: section of the rules allowed Enrichment to start at 0 before processing the first document.

To calculate the total amount due for all clients, we needed to remove all extraneous characters from the value of %AmtDue. We used the FINDNUM function to do this, and then added the return from FINDNUM (that is, the value of %Amt) to the value of %Total_Due from the previous document, setting this as the new %Total_Due value. Initializing %Total_Due in the START: section of the rules gave Enrichment a value to add to %Amt.

Finally, we used the <APPEND> command to insert the gift letter (%Award) after each document.
FINISH: Section

We used the FINISH: section of the rules to place the final %Tota l_Due value in the proper monetary format (we used the FORMAT function to do this). We also used the <BANNER> command to add the Tot als_P age banner at the end of the output.

Output Tag Group

We used the Output group <NAME> tag to identify the Output group in the Enrichment report (Output). We also used the <FILE> tag to specify the output file, DD: OUTPUT1.

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

This sample application uses DELETE and CREATE steps for the output file. The insert page and banner that the sample references are defined in the control file.

```
JOBCARD
// JOBLIB DD DSN=PDR.STREAMW.LOADCRUN,Disp=SHR
// DD DSN=SYS3.CLI22.SEDCLINK,Disp=SHR
// DD DSN=SYS3.PLI230.SIBMLNK,Disp=SHR
// DD DSN=SYS3.V4R1M0.ISPLOAD,Disp=SHR
// *------------------------------------------------------------------
// SW   EXEC PGM=PDRSW000,REGION=0M
// INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A01INPT)
// TOTALS DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A01INSR)
// AWARD DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A01INSTP)
// OUTPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A01OUT1)
// REPORT DD SYSOUT=* 
// SYSPRINT DD SYSOUT=* 
// SYSTEM DD SYSOUT=* 
// SYSDUMP DD SYSOUT=* 
// CONTROL DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.CONTROL(A01CONT)
// *------------------------------------------------------------------
// COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINCEM,'','')
// NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A01OUT1),Disp=SHR
// OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A01OUT1),Disp=SHR
// OUTDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A01OUT1),Disp=SHR
// *------------------------------------------------------------------
```
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
rm DDCONTROL
rm DD:INPUT1
rm DD:OUTPUT1
rm DD:AWARD
rm DD:TOTALS
rm A01out1
ln -s /share/home1/SunOS/handson/A01cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/A01inpt.afl DD:INPUT1
ln -s /share/home1/SunOS/handson/A01insr.lin DD:TOTALS
ln -s /share/home1/SunOS/handson/A01instp.afl DD:AWARD
ln -s /share/home1/SunOS/handson/A01out1 DD:OUTPUT1
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=A01CONT.ws.CON
```
Advanced sample 2: Outsorting based on an exception file

In this section

Overview .......................................................... 130
Files Used ..................................................... 130
Business Scenario ......................................... 131
Explanation of Sample ................................. 131
Advanced sample 2: Outsorting based on an exception file

Overview

This sample uses the account number from the input documents as a key to look up an exception byte in a mainframe VSAM file. The value of the exception byte determines how to direct output. This sample introduces the following topics.

- Outsort based on information from LOOKUPV

Note: This sample is intended for mainframe users because it uses the LOOKUPV function. A02CONT.ws.CON uses the LOOKUP function can be used on Windows, Unix, and Linux.

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Rule File</td>
</tr>
<tr>
<td>Other Inputs</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Control File</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Output Print Stream</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Other Output</td>
</tr>
<tr>
<td>Run Time</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Business Scenario

Extract the account number to use as a lookup key to a VSAM file. The VSAM file contains exception characters that will redirect documents to the correct output as follows:

- Write documents with account numbers that are not in the VSAM file to OUTPUT1.
- Write documents with exception E to OUTPUT2.
- Delete documents with exception X.
- Write all other documents to OUTPUT3

In the sample, above:

- Use the constant string "Account number" to locate the account number in the print stream.

Explanation of Sample

The control file we created for the application (shown below) reads a print stream and uses information from a VSAM file to outsort documents to an appropriate output file. Account numbers not found in the VSAM file are not considered for exception processing—they are considered regular accounts. If the account number is in the VSAM file, the exception byte in the file should be either an E or X. We want to write all the E documents to one output, delete all the X documents, and route all documents that don’t have an exception code of E or X to an error file. A02CONT.ws.CON contains these modifications.
Input Tag Group

As the control file shows, the Input group is identified in the <NAME> tag as INPUT, and the input file resides on the mainframe system at DD: INPUT, as indicated by the Input group <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A.

Field Tag Group

We defined two Field tag groups in the application. The first field defines the information used to set top-of-document. The second field defines the account number to use as the LOOKUPV key.

Some invoices may contain several pages, so the first <FIELD> tag identifies a variable called %%DOC_Page1 whose value is the string 'Page 1'. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <REFERENCE> tag instructs Enrichment to set a reference point for the field on all records that have a blank carriage control and contain the string 'Page 1' beginning in column 67. We added an extra space after Page 1 so Enrichment doesn't inadvertently use a similar string (such as Page 11) as the reference point. The <LOCATION> tag indicates that the field information occupies the same record as the reference point, begins 6 columns before the reference point, and is 7 characters in length. Thus, the value of %%DOC_Page1 is always Page 1 followed by a space.

The Input group <DOCUMENT> tag uses the value of %%DOC_Page1 to set top-of-document. If the value of %%DOC_Page1 exists in the input, a new document begins.

The second <FIELD> tag identifies a variable called %%AcctNo whose value is the account number. We only want to extract the first occurrence of the field information from each document, so we set the <FIELD> tag action parameter to K. The <LOCATION> tag indicates that the field information begins on print line 15, column 60 of the document, and is 10 characters in length.

We used the %%AcctNo value in the rules as the key in the LOOKUPV function.

**Note:** You can use Enrichment Visual Engineer's Edit Assistant to get more information on each tag in this control file.
Advanced sample 2: Outsorting based on an exception file

<field> %AcctNo K</field>
<loc> 15 60 10 L</loc>
</input>
</rule>
<content>
DOCUMENT:

<! The following statement attempts to find a record for the current document in the Exception Table, the account number is the key. The "E" specifies an exact match, the "R" signifies that the VSAM file is opened in read only mode.>
%%TableRecord = LOOKUPV('DD:EXCPTAB', %%AcctNo, 50, E, R)

<! The following code checks the return code from the LOOKUPV. If the account number is not in the table (return code = 1) the document is written to the normal output file. If the account number is in the table, the output flag in position 12 is interrogated to determine which output file to write. An "E" signifies an Exception, an "X" signifies that the document is to be deleted from all outputs. If the flag is neither an "E" or an "X" the document is written to the Error output. The substring function is used to extract the output flag from the table record.>
if %%RC = 1 then
  <output> Normal</output>
elseif %%RC = 0 then
  %%OutputFlag = SUBSTR(%%TableRecord, 12, 1, ' ')
  if %%OutputFlag = E then
    <output> Exception</output>
  elseif %%OutputFlag = 'X' then
    <output> NONE</output>
  else
    <output> Error</output>
  endif
else
  <output> Error</output>
endif
</content>
</rule>

<output>
  <name> Normal</name>
  <file> DD:OUTPUT1</file>
</output>
<output>
  <name> Exception</name>
  <file> DD:OUTPUT2</file>
</output>
<output>
  <name> Error</name>
  <file> DD:OUTPUT3</file>
</output>
Rules

In the DOCUMENT: section of the rule, we used the value of %AcctNo (the account number) as a key for the LOOKUPV function to attempt to find a record for the current document in the VSAM file shown below. Enrichment sets the return from the LOOKUPV function as the value of %TableRecord.

301-555522X
637-252332
831-511116E

As the rules below show, if the account number for a document is not found in the VSAM file (that is, if the LOOKUPV function sets the value of system variable %RC to 1), Enrichment writes the document to the Normal output.

```plaintext
if %RC = 1 then
  <output> Normal
else if %RC = 0 then
  %OutputFlag = SUBSTR(%TableRecord, 12, 1, ' ')
  if %OutputFlag = E then
    <output> Exception
  elseif %OutputFlag = 'X' then
    <output> NONE
  else
    <output> Error
  endif
else
  <output> Error
endif
```

If the account number for a document is found in the VSAM file, the LOOKUPV function sets %RC to 0. In this case, we need to determine what exception code is associated with the account number so we’ll know which output to write the document to (if any). We used the SUBSTR function to parse the exception code from column 12 of the %TableRecord value into the variable %OutputFlag. If the %OutputFlag value for a document is E, we used an <OUTPUT> command to write the document to the Exception output. If the %OutputFlag value is X, we used the <OUTPUT> command to discard the document (that is, we don’t write it to an output at all). If the %OutputFlag value is anything other than E or X, or the LOOKUPV function sets %RC to anything other than 1 or 0, we used an <OUTPUT> command to write the document to the Error output. So, if the Exception VSAM file is empty or missing, all documents will be routed to the Error output file.

Output Tag Group

We defined three Output tag groups for this application.
Advanced sample 2: Outsorting based on an exception file

The first Output group defines an output called Normal which resides on the mainframe system at DD: OUTPUT1. Documents whose account number was not found in the VSAM file are written to this output.

The second Output group defines an output called Exception which resides on the mainframe system at DD: OUTPUT2. Documents whose account number was found in the VSAM file with an exception character of E are written to this output.

The last Output group defines an output called Error which resides on the mainframe system at DD: OUTPUT3. Documents whose account number was found in the VSAM file with an exception character other than E or X and documents for which the LookupV function set %%RC to anything other than 0 or 1 are written to this output.

Run Time

The following information shows how to run the sample application on mainframe and Windows systems.

Mainframe

The JCL shown below includes Delete and Create steps for the three Enrichment output files. For this application, we need a VSAM file so we added one IDCAMS step to define the VSAM file and another IDCAMS step to load some data into the VSAM file from a sequential file. In a real application, you would not have these two steps.

In the Enrichment step, we added DD for the VSAM file that points to the name of the cluster we defined in the IDCAMS step.

```plaintext
//*JOBcard
//*------------------------------------------------------------------
//JOBLIB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
// DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
// DD DSN=SYS3.PLI230.SIBMLINK,DISP=SHR
// DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//*------------------------------------------------------------------
//* DELETE AND CREATE VSAM FILE TO USE FOR THE LOOKUPV
//*
//CREVSAM EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=* 
//SYSIN DD *
DELETE (PDR.STREAMW.HANDSON.INDEX) PURGE CLUSTER
DEFINE CLUSTER ( 
 NAME | PDR.STREAMW.HANDSON.INDEX) - 
 SHAREOPTIONS (4) - 
 RECORDSIZE (50 50) - 
 CYLINDERS (1 1) - 
 NOREUSE -

```
Advanced sample 2: Outsourcing based on an exception file

```
VOL(* * *) -
STORCLAS(TDBASE) -
SPEED -
INDEXED -
FREESPACE (15 10) -
   KEYS (10 1) -
   DATA ( 
   NAME (PDR.STREAMW.HANDSON.INDEX.DATA) -
      CISZ(8192) -
   ) -
   INDEX ( 
   NAME (PDR.STREAMW.HANDSON.INDEX.NDX) -
   ) -

*I------------------------------------------------------------------*
*I LOAD DATA FROM FLAT FILE TO VSAM FILE *
*I------------------------------------------------------------------*
LOADVSAM EXEC PGM=IDCAMS
   /I NDD1 DD DSN=PDR.STREAMW.HANDSON.INPUT(A02TABLE),
   /I OUTDD1 DD DSN=PDR.STREAMW.HANDSON.INDEX,DISP=SHR
   /SYSPRINT DD SYSOUT=* 
   /SYSPRINT DD SYSOUT=* 
   /SYSTERM DD SYSOUT=* 
   /SYSUDUMP DD SYSOUT=* 
   /REPRO  DD INFILE(INDD1) -
   /OUTFILE(OUTDD1) -
   /I------------------------------------------------------------------*
   *
   Enrichment PROCESS *
   *
   /SW      EXEC PGM=PDRSW000,REGION=0M 
   /INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A02INPT) 
   /EXCPTAB DD DI SP=SHR,DSN=PDR.STREAMW.HANDSON.INDEX 
   /OUTPUT1 DD DI SP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A02OUT1) 
   /OUTPUT2 DD DI SP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT2(A02OUT2) 
   /OUTPUT3 DD DI SP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT3(A02OUT3) 
   /REPORT DD SYSOUT=* 
   /SYSPRINT DD SYSOUT=* 
   /SYSTERM DD SYSOUT=* 
   /SYSUDUMP DD SYSOUT=* 
   /COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','') 
   /NEWDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A02OUT1),DI SP=SHR 
   /OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(A02OUT2),DI SP=SHR 
   /OUTDD DD SYSOUT=* 
   /COMP02 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','') 
   /NEWDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(A02OUT2),DI SP=SHR 
   /OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT3(A02OUT3),DI SP=SHR 
   /OUTDD DD SYSOUT=* 
   /COMP03 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','') 
   /NEWDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT3(A02OUT3),DI SP=SHR 
   /OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT3(A02OUT3),DI SP=SHR
```
Advanced sample 2: Outsorting based on an exception file

```bash
//OUTDD DD SYSOUT=* 
/* -------------------------------------------------------------

Windows

Run the application by issuing the following command:

```
swaever -c=A02CONT.ws.CON
```

Advanced sample 2: Outsourcing based on an exception file
Advanced sample 3: Consolidating multiple print streams and adding a cover letter

In this section

Overview .................................................................................................................. 140
Files Used ............................................................................................................. 140
Business Scenario ................................................................................................. 141
Explanation of Sample .......................................................................................... 143
Overview

This sample combines multiple print streams and adds a cover letter to each package that contains the address from the first input and tells what was combined. This sample introduces the following topics.

- Sortmatch tag group
  - `<INPUTNAME>` tag
- Input tag group
  - `<COPYGROUP>` tag
  - `<PAGEFORMAT>` tag
- Setting up fields to indicate the absence or presence of an input
- Setting up a cover page for inclusion “always”
- `%INPUT` system variable
- `WORDPOS` function

Files Used

This sample uses the following files.

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Input Print Stream</th>
<th>D:\apps\adv3\a03inp1.afl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D:\apps\adv3\a03inp2.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv3\a03inp3.afl</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv3\a03inp4.lin</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A03INP1)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A03INP2)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A03INP3)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A03INP4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule File</th>
<th>Inline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control File</th>
<th>D:\apps\adv3\A03cont.con</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(A03CONT)</td>
</tr>
<tr>
<td></td>
<td>A03CONT.ws.CON</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Print Stream</th>
<th>D:\apps\adv3\a03out1.afm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D:\apps\adv3\a03out2.afm</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(A03OUT1)</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT2(A03OUT2)</td>
</tr>
</tbody>
</table>
Advanced sample 3: Consolidating multiple print streams and adding a cover letter

**Table 1: Files Used in This Sample**

<table>
<thead>
<tr>
<th>Other Output</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv3\a03jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A03JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv3\a03scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A03SCR)</td>
</tr>
</tbody>
</table>

**Business Scenario**

This sample combines all the pieces going to a customer into a single envelope and adds a cover letter.
In the sample, above:

- Merge all three print streams by matching the identification number.
- Use the address from the first input on the cover letter.
- Some customers will not receive all pieces, so list package contents on the coder letter (circled bottom).
**Explanation of Sample**

We want to include several different pieces of customer correspondence from a number of inputs in the same envelope to the customer. Our control file (shown below) reads three print streams that contain multiple documents of varying page count, sorts and matches the documents by customer identification number, and adds a cover letter that includes the customer’s address to each document package. Since every customer will not get every piece of correspondence, we should list the contents of the package in the cover letter. If a document package doesn't include the first input (a provider letter from which we'll extract the customer’s address), Enrichment outsorts it to a separate output.

We defined four Input tag groups for this application, one for each of the three inputs to sort and consolidate, and one for the cover letter.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A03CONT.ws.CON contains these modifications.

**Input 1**

**Input Tag Group**

In the control file, the first Input group defines the provider letter from which we'll extract the customer’s address for the cover letter. The input is identified in the `<NAME>` tag as `PL`, and the input file is found on the mainframe system at `DD:INPUT1`, as indicated by the `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPLA`. We used the `<COPYGROUP>` tag to specify that Enrichment is to use copy group `F2LDATA0` to process the input. We also specified in the `<PAGEFORMAT>` tag that Enrichment will use page format `P2LINE35`.

**Field Tag Group**

We need to extract four pieces of data from `PL`, so we defined four Field tag groups in the Input group.

**Note:** Fields used as sort criteria must be identically defined in each input to be sorted.

The first Field group serves a dual purpose in the application: it is used by the Input group `<DOCUMENT>` tag to specify top-of-document, and it is used by the Sortmatch
group as the criterion for sorting and matching the inputs. The `<FIELD>` tag defines a variable called `%%ID_number`. We only want to extract the field information, so we set the `<FIELD>` tag `action` parameter to `KA`. The `<REFERENCE>` tag instructs Enrichment to set a reference point for the field on all records in each document that have a blank carriage control character and contain the string `Number:` beginning in column 17. The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins 3 columns after the reference point, and is 10 characters in length.

In the Input tag group, we specified `<DOCUMENT>T %%ID_number C` to indicate that each time the `%%ID_number` value changes in the input, a new document begins.

The last three Field groups define lines of the address from page 1 of each document:

- The first of the three `<FIELD>` tags defines a variable called `%%PL_Addr1`. We only want to extract the first occurrence of the field information, so we set the `<FIELD>` tag `action` parameter to `K`. The address line always occupies the same position for every document in the input, so we used the `<LOCATION>` tag to specify its location on record 10, column 2 for 40 characters.

- The second `<FIELD>` tag defines a variable called `%%PL_Addr2`. We set the `<FIELD>` tag `action` parameter to `K` to extract the first occurrence of the field information. The `<LOCATION>` tag specifies the field’s location on record 11, column 2 for 40 characters.

- The last `<FIELD>` tag defines a variable called `%%PL_Addr3`. We set the `<FIELD>` tag `action` parameter to `K` to extract the first occurrence of the field information. The `<LOCATION>` tag specifies the field’s location on record 12, column 2 for 40 characters.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

Note: We could have used fields to check to see if the final document contained a piece of any of the inputs, but it wouldn’t have been as efficient or secure.
Advanced sample 3: Consolidating multiple print streams and adding a cover letter

```plaintext
<loc> 11 2 40</loc>  <! the 2nd position for 40 chars. >
<field> %PL_Addr3 K</field>
<loc> 12 2 40</loc>
</input>

<input>
<name> PR</name>
<file> DD:INPUT2</file>
<type> AFPL A</type>
<copygroup> F2LDATA0</copygroup>
<pageformat> P2LINE35</pageformat>
<doc> T %D_number C</doc>
<field> %D_number KA</field>
<location> 9 26 10</location>
</input>

<input>
<name> BenS</name>
<file> DD:INPUT3</file>
<type> AFPL A</type>
<copygroup> F2LDATA0</copygroup>
<pageformat> P2LINE35</pageformat>
<doc> T %D_number C</doc>
<field> %D_number KA</field>
<location> 5 26 10</location>
</input>

<input>
<name> CoverLetter</name>
<file> DD:INPUT4</file>
<type> AFPL A</type>
<copygroup> F2COVLT0</copygroup>
<pageformat> P2COVER5</pageformat>
<doc>*</doc>
<substitute> YES</substitute>
</input>

<sortmatch>
<inputname> CoverLetter</inputname>
<inputname> PL</inputname>
<inputname> PR</inputname>
<inputname> BenS</inputname>
<sort> %D_number A</sort>
</sortmatch>

<rule>
<content>
DOCUMENT:
if WORDPOS('PL', %INPUT) then
  %PL = 'Provider Letter'
%ADDR1 = %PL_Addr1
%ADDR2 = %PL_Addr2
%ADDR3 = %PL_Addr3
<output> Output</output>
</content>
</rule>
```

EngageOne Enrichment 7.3.0 Sample Applications Guide 145
Advanced sample 3: Consolidating multiple print streams and adding a cover letter

Input 2

Input Tag Group

The second Input group defines the plan book revisions documents. The input is identified in the <NAME> tag as PR, and the input file is found on the mainframe system at DD: INPUT2, as indicated by the <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A. The <COPYGROUP> tag specifies that Enrichment is to use copy group F2LDA0 to process the input and the <PAGEFORMAT> tag specifies that Enrichment will use page format P2LINE35.

Field Tag Group

We need to extract one piece of data from PR, so we defined one Field tag group in the Input group. The Field group is used by the Input group <DOCUMENT> tag to specify top-of-document, and it is used by the Sortmatch group as the criterion for sorting and matching the inputs. The <FIELD> tag defines a variable called %%ID_number. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <LOCATION> tag indicates that the field information begins on row 9, column 26, and is 10 characters in length.
In the Input tag group, we specified <DOCUMENT>T %ID_number C to indicate that each time the %ID_number value changes in the input, a new document begins.

Input 3

Input Tag Group

The third Input group defines the benefits summary documents. The input is identified in the <NAME> tag as BenS, and the input file is found on the mainframe system at DD: INPUT3, as indicated by the <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A. The <COPYGROUP> tag specifies that Enrichment is to use copy group F2LDATA0 to process the input and the <PAGEFORMAT> tag specifies that Enrichment will use page format P2LINE35.

Field Tag Group

We need to extract one piece of data from BenS, so we defined one Field tag group in the Input group. The Field group is used by the Input group <DOCUMENT> tag to specify top-of-document, and it is used by the Sortmatch group as the criterion for sorting and matching the inputs. The <FIELD> tag defines a variable called %ID_number. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA. The <LOCATION> tag indicates that the field information begins in row 5, column 26 of the documents, and is 10 characters in length.

In the Input tag group, we specified <DOCUMENT>T %ID_number C to indicate that each time the %ID_number value changes in the input, a new document begins.

Input 4

The Final Input group is identified in the <NAME> tag as CoverLetter, and the input file is found on the mainframe system at DD: INPUT4, as indicated by the <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the <TYPE> tag to AFPL A. The <COPYGROUP> tag specifies that Enrichment is to use copy group F2COVLT0 with the input and the <PAGEFORMAT> tag specifies that Enrichment will use page format P2COVER5.

We specified <DOCUMENT>* to indicate that the input contains only one document.

The <SUBSTITUTE> tag setting (YES) indicates that Enrichment will substitute inline variable names in the input with values derived from the control file or rules during processing.
Since we wanted to include Cover Letter with every document package generated by the application, we didn’t need to define the sort and match criterion field (%%%ID_number) for it.

Sortmatch Tag Group

The Sortmatch tag group defines how to sort and match the documents in the inputs by customer ID number. First, the Sortmatch group <INPUTNAME> tags specify the inputs to sort. We included the <INPUTNAME> tags in the control file in the order in which we want the inputs matched. Thus, each document package will be printed with the cover letter first, followed by the provider letter, the plan book revisions, and the benefits summary. We specified four <INPUTNAME> tags:

- <INPUTNAME>CoverLetter AY instructs Enrichment to include the cover letter in every output document package.
- <INPUTNAME>PL M Y instructs Enrichment to sort and match the provider letter with the plan book revisions and benefits summaries and to include it in the output.
- <INPUTNAME>PR M Y instructs Enrichment to sort and match the plan book revisions with the provider letters and benefits summaries and to include it in the output.
- <INPUTNAME>Ben S M Y instructs Enrichment to sort and match the benefits summary with the provider letters and plan book revisions and to include it in the output.

The <SORT> tag identifies the sort criterion and direction. In the <SORT> tag, we specify the field by which to sort and match (%%%ID_number). We want to sort the documents from lowest customer ID number to highest customer ID number, so we set the <SORT> tag order parameter to A (ASCEND).

Rules

The first thing we do in the rules is define the rule code that will confirm the presence of the provider letter in the document package, set the %%PL variable, and print the customer’s address on the cover letter. As the code below shows, if PL is contained in the value of %%%INPUT, Enrichment sets %%PL to Provider Letter; sets address variables %%ADDR1, %%ADDR2, and %%ADDR3 on the cover letter to the values of the address fields extracted from PL; and routes the document to Output. If PL isn’t in %%%INPUT, Enrichment sets %%ADDR1, %%ADDR2, and %%ADDR3 to an ADDRESS UNKNOWN message; sets %%PL to blank; and outsorsts the document to NoAddress.

```plaintext
if WORDPOS('PL', %%%INPUT) then  <! This block of code checks >  
  %%PL = 'Provider Letter'  <! for the existence of the >  
  %%ADDR1 = %PL_Addr1  <! Provider Letter to list >  
  %%ADDR2 = %PL.Addr2  <! in the cover letter. >
```
Advanced sample 3: Consolidating multiple print streams and adding a cover letter

In the next rule, we defined code to confirm the presence of the plan book revisions in the document package and set the %%PBR variable on the cover letter. As the code below shows, if PR is found in %%NPUT, Enrichment sets %%PBR on the cover letter to Plan Book Revisions. If PR isn’t in %%NPUT, Enrichment sets %%PBR to blank.

```
if WORDPOS('PR',%%NPUT) then
  %%PBR = 'Plan Book Revisions'
else
  %%PBR = ' '
endif
```

Then we defined the code to confirm the presence of the benefits summary in the document package and set the %%BS variable on the cover letter. As the code in Figure below shows, if BenS is found in %%NPUT, Enrichment sets %%BS on the cover letter to Benefits Summary. If BenS isn’t in %%NPUT, Enrichment sets %%BS to blank.

```
if WORDPOS('BenS',%%NPUT) then
  %%BS = 'Benefits Summary'
else
  %%BS = ' '
endif
```

Output Tag Group

We set up two Output tag groups to direct the output from the application. The first Output group <NAME> tag defines Output, which resides at DD:OUTPUT1 on the mainframe system. Output will contain all of the document packages that included a provider letter, as instructed in the rules. The second Output group <NAME> tag defines NoAddress, which resides at DD:OUTPUT2 on the mainframe system. NoAddress will contain any document packages that did not include a provider letter (and so did not include a customer address on the cover letter).

So, when the application runs, Enrichment sorts the documents in three of the inputs in ascending order by customer ID number, matches documents with the same ID number into document packages, adds a cover letter to each package, and sets the appropriate variables before printing.
Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```
//*JOBCARD
//*OBLIB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
//* DD DSN=SYS3.CLI822.SEDCLINK,DISP=SHR
//* DD DSN=SYS3.PLL230.SIBMLINK,DISP=SHR
//* DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//*------------------------------------------------------------------
*/
// SW EXEC PGM=PDRSW000,REGION=0M
// INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A03INP1)
// INPUT2 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A03INP2)
// INPUT3 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A03INP3)
// INPUT4 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A03INP4)
// OUTPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A03OUT1)
// OUTPUT2 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT2(A03OUT2)
// REPORT DD SYSOUT=*  
// SYSPRINT DD SYSOUT=*  
// SYSTERM DD SYSOUT=*  
// SYSUDUMP DD SYSOUT=*  
// CONTROL DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.CONTROL(A03CONT)
//*------------------------------------------------------------------
// COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
// NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A03OUT1),DI SP=SHR
// OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A03OUT1),DI SP=SHR
// OUTDD DD SYSOUT=*  
// COMP02 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
// NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(A03OUT2),DI SP=SHR
// OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(A03OUT2),DI SP=SHR
// OUTDD DD SYSOUT=*  
// *------------------------------------------------------------------
```

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

`sweaver -c=<controlfile>`
Advancing sample 3: Consolidating multiple print streams and adding a cover letter

To run the control file as-is on UNIX, use this script:

```bash
rm DDCONTROL
rm DD:INPUT1
rm DD:INPUT2
rm DD:INPUT3
rm DD:INPUT4
rm DD:OUTPUT1
rm DD:OUTPUT2
rm A03out1
rm A03out2
ln -s /share/home1/SunOS/handson/A03cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/A03inpt1.afl DD:INPUT1
ln -s /share/home1/SunOS/handson/A03inpt2.afl DD:INPUT2
ln -s /share/home1/SunOS/handson/A03inpt3.afl DD:INPUT3
ln -s /share/home1/SunOS/handson/A03inpt4.lin DD:INPUT4
ln -s /share/home1/SunOS/handson/A03out1 DD:OUTPUT1
ln -s /share/home1/SunOS/handson/A03out2 DD:OUTPUT2
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=A03CONT.ws.CON
```
Advanced sample 3: Consolidating multiple print streams and adding a cover letter
Advanced sample 4: Extracting information and creating a flat file (Metacode)

In this section

Overview ........................................................................................................... 154
Files Used ........................................................................................................... 154
Business Scenario ............................................................................................. 155
Explanation of Sample ....................................................................................... 155
Overview

This sample extracts data from a single Metacode input and uses the extracted data to construct a flat file to be used by another program. It changes selected data and the font in which it prints in output. This sample introduces the following topics:

- Metacode print streams
- Positional Metacode commands
- Font Metacode commands
- Fields
- Using print records
- Extracting until end criteria
- \texttt{<REFERENCE>} tag
- Input tag group

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Advanced sample 4: Extracting information and creating a flat file (Metacode)

Business Scenario

This sample application generates a report that shows every customer’s name and address from this asset summary file. It uses the audit report as input to another program that requires a fixed record format.

FOR MONTE ERDING JUNR

You are Account Executive

DAVID L. MAN

Federated Finance Co

100 E. Main St.

Lexington, KY 40507

(606) 227-3400

DEBBY JONES

CFO, FEDERATED CUP LTD. INC.

MORRISTOWN, TN 37829-2475

Asset Summary

<table>
<thead>
<tr>
<th>Percent of Assets</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Money Market Funds</td>
<td>$1.87</td>
</tr>
<tr>
<td>Bonds/Options</td>
<td>$24.50</td>
</tr>
<tr>
<td>Municipal Bonds</td>
<td>$37.00</td>
</tr>
<tr>
<td>Corporate Fixed Income</td>
<td>$1.00</td>
</tr>
<tr>
<td>Mutue Funds</td>
<td>$9,253.12</td>
</tr>
<tr>
<td>Other Investment Income</td>
<td>$0.00</td>
</tr>
<tr>
<td>Certificates of Deposit</td>
<td>$0.00</td>
</tr>
<tr>
<td>Annuities/Insurance</td>
<td>$0.00</td>
</tr>
<tr>
<td>Other</td>
<td>$0.00</td>
</tr>
<tr>
<td>Total Asset Value</td>
<td>$32,927.67</td>
</tr>
<tr>
<td>Cash</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Income Summary

<table>
<thead>
<tr>
<th></th>
<th>This Month</th>
<th>Year to Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Money Market Funds</td>
<td>$1.87</td>
<td>$80.11</td>
</tr>
<tr>
<td>Bonds/Options</td>
<td>204.00</td>
<td>1,594.26</td>
</tr>
<tr>
<td>Municipal Bonds</td>
<td>220.56</td>
<td>820.56</td>
</tr>
<tr>
<td>Corporate Fixed Income</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Government Securities</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Mutual Funds</td>
<td>110.86</td>
<td>408.61</td>
</tr>
<tr>
<td>Other Investment Income</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Certificates of Deposit</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Other</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Total Income</td>
<td>$412.76</td>
<td>$2,569.24</td>
</tr>
<tr>
<td>Taxable Income</td>
<td>$274.86</td>
<td>$1,489.74</td>
</tr>
</tbody>
</table>

In the sample, above:

Left Replace the phone number with a new one, using a font that stands out better.

Right Extract the customer's name and address, up to 30 characters.

Explanation of Sample

Our control file (shown below) reads a print stream, extracts information from it, and writes the information to a side file. It also changes the value of a data object and the font in which it prints.

Note: This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the <FILE> tag for your OS. A04CONT.ws.CON contains these modifications.
Input Tag Group

As the control file shows, the Input group is identified in the <NAME> tag as Statements. The input file is found at DD:INPUT1 on a mainframe system, as indicated by the Input group <FILE> tag. The input print stream is Xerox Metacode with machine carriage controls, so we set the <TYPE> tag in the Input group to METACODE MACHINE.

If this application will run on a UNIX system, we must know the characteristics of record length indicators and specify the <RECORD> tag (as shown below). Different composition systems that produce Metacode use different style length indicators. Find out what style is used for your composition system. The <RECORD> tag we defined specifies that the input contains ASCII data with a 2-byte length indicator that is coded least significant byte first. The length indicator value does not include the 2 bytes of the length indicator itself.

If you use the <RECORD> tag on a mainframe system, Enrichment issues the following warning message and continues processing:

PDR1371W <RECORD> tag only valid on UNIX systems.

The input contains some DJDE records that are used to set up the printer. These records are at the top of the input but are not part of the first document. We specified <HEADER>5 YES to instruct Enrichment to ignore the five header records and to copy those records to the beginning of the output.

Each document in the input is two pages in length, so we specified <DOCUMENT>2. This is much more efficient processing for Enrichment than using a Field to denote the top of a document as in other samples. Of course this technique will cause major pagination problems if there happens to be a document that is not 2 pages in length. Don't specify a number of pages as a document identifier unless you are sure each document will have that exact number of pages. Otherwise, use a Field to identify the first or last page of each document.

Field Tag Group

We need to extract three pieces of data from each document, and we defined six Field tag groups in the Input group (one for each of four lines of address information, one for the telephone number, and one for the font in which the telephone number prints).

We analyzed the Metacode print stream looking for the Metacode controls that perform absolute dot and scanline moves. The control for the absolute dot move is x'04' and the absolute scanline move code is x'06'. Each control is followed by a 2 byte location. Each of the address lines has a constant y position x'8D01'. The x position increases for each address line. Once we find this reference string we must be able to consistently move from that reference string to the start of the address line. In addition
to these absolute move controls, you may also have occasion to use the relative move controls, which are x'05' for x and x'07' for y.

**Note:** You can use the Enrichment Visual Engineer to view and get more information on the Metacode commands in a print stream. For information, see the Visual Engineer User Guide.

The first four Field tag groups identify the lines of the address that appears on each document. In the first Field group, the `<FIELD>` tag defines a variable called `%%Addr1` that we’ll use to store the first line of address information from each document. We will reference this variable later to include it in the side file. Because we only want to extract the first occurrence of the variable value from each document, we set the `<FIELD>` tag action parameter to `K` (keep). Since the location of the address information may change from document to document, we used the `<REFERENCE>` tag to identify a reference point for the field information. The `<REFERENCE>` tag for `%%Addr1` instructs Enrichment to set a reference point for the field on every record in each document that has a carriage control character of x'01' in column 1 and contains the string x'8D0104A50B' beginning in column 3. The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins 43 characters after the reference point, and is 30 characters in length. The other three address fields define the positions of subsequent address lines in the same manner.

In the fifth Field group, the `<FIELD>` tag defines a variable called `%%PhoneNo` that will extract the telephone number from each document. We set the action parameter to `R` because we only want to replace the first occurrence of the variable value from each document. The `<REFERENCE>` tag instructs Enrichment to set a reference point for the field on every record in each document that has a carriage control of x'01' in column 1 and contains the string x'8D0104450B' beginning in column 3. The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins 3 characters after the reference point, and is 14 characters in length.

To find the font specification for the phone number, we use the x, y location for the phone number along with the Metacode control for setting the font, which is x'00'. The byte immediately following x'00' is the font number that we will change. Therefore, in the last Field group, the `<FIELD>` tag defines a variable called `%%PhoneFont` that will extract the font index that controls the font in which the telephone number prints. Again, we set the action parameter to `R` because we only want to replace the first occurrence of the variable value from each document. Since the location of records that contain the font index may vary from document to document, we used the `<REFERENCE>` and `<LOCATION>` tags to define the field. The `<REFERENCE>` tag instructs Enrichment to set a reference point for the field on every record in each document that has a carriage control of x'01' in column 1 and contains the string x'8D0104450B00' beginning in column 3. The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins one character after the reference point, and is one character in length.

**Note:** You may want to include the `<ONPAGE>` tag in each Field group to ensure that the data is only found and replaced on the first page.
Note: Remove the <RECORD> tag from the control file if you are not running Enrichment on a UNIX system.

```xml
<name> Statements </name>
<file> DD: INPUT1 </file>
<type> METACODE MACHINE </type>
<record> ASCII 2 LEAST EXCLUSIVE <! used for UNIX systems only. >
            <! Remove this tag for non-UNIX runs. >
<header> 5 YES <! There are 5 header lines of DJDE to keep. >
<doc> 2 <! every document is exactly 2 pages. >
<field> %Addr1 K <! Pick the 4 address lines >
            <! based on dot and scan lines >
            <! in metacode. The scan line >
            <! is always the same but the >
            <! dot address (x'04') changes. >
<ref> x'01' x'8D0104A50B' 3 </ref>
<loc> 0 43 30 </loc>
</field>
<field> %Addr2 K <! Field replace to change >
<ref> x'01' x'8D01048D0B' 3 </ref>
<loc> 0 43 30 </loc>
</field>
<field> %Addr3 K </field>
<field> %Addr4 K </field>
<field> %PhoneNo   R <! Field replace to change >
<ref> x'01' x'8D0104450B' 3 </ref>
<loc> 0 5 14 </loc>
</field>
<field> %PhoneFont R </field>
<rule> DOCUMENT: %PhoneFont = x'01' <! Change font for phone number. >
        %PhoneNo = '(606) 555-1234' <! Change value of phone number. >
</rule>
<output>
<name> Output </name>
<file> DD: OUTPUT1 </file>
<sidefile> DD: SIDE1 </sidefile>
<sidepart> %Addr1 30 L ' ' <! Fields in side file, each is >
        <! 30 characters, left justified >
<sidepart> %Addr2 30 L ' ' <! and, if shorter than 30 >
        <! characters, padded with blanks >
<sidepart> %Addr3 30 L ' ' <! on the right. >
<sidepart> %Addr4 30 L ' '
</sidefile>
</output>
</xml>

Rules

The rules for this sample are very simple. Since we’re setting field variable values for each document, we define processing in the DOCUMENT: section of the rule. To change the font in which the telephone number prints, we simply specify the new value of %PhoneFont (x‘01‘). Likewise, to change the telephone number, we specified the new value of %PhoneNo.

Output Tag Group

Next, we specify the Output tag group, and the Sidefile group within it, to define the side file to create from data extracted from the input print stream. We used the Output group <NAME> tag to identify the Output group in the Enrichment report (Output). We also used the <FILE> tag to specify the output file, DD:OUTPUT1.

Sidefile Tag Group

The Sidefile group <FILE> tag identifies the file to create (DD:SIDE1). We used <SIDE> tags to specify the record layout and contents of the side file. Each record of the side file will contain the data defined in each of the <SIDE> tags, in the order in which the tags are used in the Sidefile tag group. Thus, each record of the side file will contain the four lines of address information defined by the field variables %Addr1, %Addr2, %Addr3, and %Addr4.

Each <SIDE> tag specifies that the value of one of the address line fields, which will be a maximum of 30 characters in length, should be left-justified (L) and padded with blanks (' ') if necessary. As the figure below shows, Enrichment places each line of address information on the record side by side in the order in which the <SIDE> tags are defined.

<table>
<thead>
<tr>
<th>DEBRA A. JONES</th>
<th>2309 DUCK POND CIR APT 1</th>
<th>MORRISVILLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC 27560-9679</td>
<td>33431·0231</td>
<td></td>
</tr>
<tr>
<td>AMANDA K. CULPEPPER</td>
<td>2205 WOODBURN HALL RD</td>
<td>LEXINGTON</td>
</tr>
<tr>
<td>FL 33431·0231</td>
<td>2205 WOODBURN HALL RD</td>
<td>LEXINGTON</td>
</tr>
<tr>
<td>DOUGLAS R. SMITH</td>
<td>40515-1266</td>
<td></td>
</tr>
<tr>
<td>KY 40515-1266</td>
<td></td>
<td></td>
</tr>
<tr>
<td>THOMAS E. GREEN</td>
<td>749 SAN PABLO AVE</td>
<td></td>
</tr>
<tr>
<td>CA 94086·3441</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This will write the data in the same format as the Metacode file (usually ASCII). If you are processing on the mainframe, you can convert the data to EBCDIC using the A2E function in the rules. Refer to the Enrichment Language Reference for a complete discussion of the A2E function.
Use the output document number in the Enrichment Report to verify that the side file contains one record for each document found in the input. A portion of the Enrichment Report from this sample is shown below.

---

**Enrichment Processing Summary**

Copyright (c) 1993-2000, Pitney Bowes Inc.

**Input:** Insurance_Letter

**Storage Method:** Memory (One-At-A-Time Processing)

**Records:** 33

**Documents:** 3

**Total Logical Pages:** 3

**Output:** MovedAddressLetter

**File:** DD:OUTPUT1 (1 of 1)

**Records:** 36

**Documents:** 3

**Total Pages:** 3

---

### Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

#### Mainframe

The following shows the JCL we used to run the application.

```bash
// *JOBCARD
// **OBLI B DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
// ** DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
// ** DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
// ** DD DSN=SYS3.PLI230.SIBMLINK,DISP=SHR
// **********************
// **SW **EXEC PGM=PDRSW000,REGION=0M
// ** INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A04INPT)
// ** OUTPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A04OUTP)
// ** SIDE1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT2(A04SIDE)
// ** REPORT DD SYSOUT=* 
// ** SYSPRINT DD SYSOUT=* 
// ** SYSTEM DD SYSOUT=* 
// ** SYSDUMP DD SYSOUT=* 
// ** CONTROL DD SYSOUT=*
// **********************
// ** COMP01 **EXEC PGM=ISRSUPC,PARM=(DELTAL,LI NECMP,' ','')
// ** NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A04OUTP),DISP=SHR
// ** OLDDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A04OUTP),DISP=SHR
// ** OUTDD DD SYSOUT=* 
// **********************
// ** COMP01 **EXEC PGM=ISRSUPC,PARM=(DELTAL,LI NECMP,' ','')
```
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
   sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```
   rm DDCONTROL
   rm DD:INPUT1
   rm DD:OUTPUT1
   rm DD:SIDE1
   ln -s /share/home1/SunOS/handson/A04cont.con DDCONTROL
   ln -s /share/home1/SunOS/handson/A04inpt.met DD:INPUT1
   ln -s /share/home1/SunOS/handson/A04out1 DD:OUTPUT1
   ln -s /share/home1/SunOS/handson/A04side DD:SIDE1
   sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
   sweaver -c=A04CONT.ws.CON
```
Advanced sample 4: Extracting information and creating a flat file (Metacode)
Advanced sample 5: Extracting information and creating a flat file (AFPDS)

In this section

Overview........................................................................................................ 164
Files Used......................................................................................... 164
Business Scenario.............................................................................. 165
Explanation of Sample....................................................................... 166
Overview

This sample extracts the recipient name and address from letters and writes them to a side file. It changes the font for the name field and uses the AFPIndex tag group to add Tag Logical Element (TLE) records to the output so customer service representatives can find documents quickly using an AFP viewer. This sample introduces the following topics.

- AFPDS print streams
- AFPIndex tag group
  - `<part>` tag
- Fields in AFP records

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Input Print Stream</th>
<th>D:\apps\adv5\a05inpt.afp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A05INPT)</td>
</tr>
<tr>
<td>Rule File</td>
<td>Inline</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\adv5\a05cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(A05CONT)</td>
</tr>
<tr>
<td></td>
<td>A05CONT.ws.CON</td>
</tr>
<tr>
<td>Output Print Stream</td>
<td>D:\apps\adv5\a05outp.afp</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(A05OUTP)</td>
</tr>
<tr>
<td>Other Output</td>
<td>D:\apps\adv5\a05side.txt</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOODOUT1(A05SIDE)</td>
</tr>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv5\a05jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A05JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv5\a05scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A05SCR)</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application writes the customer name and address from a “health care plan ID” letter to a side file for use by another department. It also adds TLE records so the information can be indexed and searched by an AFP viewer, using the customer name and the account number.

February 17, 1994

To:  
Harry O'Brian  
340 Water Rd.  
Hopkinsville, KY 40117

From:  
Daniel J. Kloiber  
PDR Insurance Co.  
900 Corporate Drive, Suite 200  
Lexington, KY 40503

Subject:  Health Care Plan

Dear M. O'Brian,

At the bottom of this letter you will find your PDR health insurance identification numbers. If you have family coverage, you should have two numbers. Please read and verify these numbers at this time.

No changes to the plan coverage(s) have been made.

Identification Number(s)
1234512345123456
2234522345223456

Please contact Betsy Baird or Dan Kloiber of PDR at (808) 223-6900 with any questions or comments regarding this addendum or additional questions.

Sincerely,

Daniel J. Kloiber  
Executive Vice President

In the sample, above:

- Pick up each name/address line separately. Change the font of the name line to match the other data.
Explanation of Sample

Our control file (shown below) reads a print stream, extracts information from it, and writes the information to a side file. It also changes the font in which a data item prints and adds TLE records to the output.

Note: This control file is designed for running Enrichment on a mainframe system. You would need to modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A05CONT.ws.CON contains these modifications.

Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `Insurance_Letter`. The input file is found at `DD:INPUT1` on a mainframe system, as indicated by the Input group `<FILE>` tag. We set the `<TYPE>` tag in the Input group to `AFPDS`, allowing the `carriage` parameter to default to `ANSI` and the `charset` parameter to default to `E` (EBCDIC).

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.
Advanced sample 5: Extracting information and creating a flat file (AFPDS)

Field Tag Group

We need to extract three pieces of data from each document, so we defined four Field tag groups in the Input group (one for the customer name, one for each of two lines of address information, and one for the font in which the customer name prints).

All field reference strings use the x and y location controls to extract the information. The absolute y control is x'D3' which is followed by two bytes for the y location going vertically down the page. The absolute x control is x'C7' which is followed by 2 bytes for the x location horizontally across the page. Unlike Metacode, the two bytes for x and y location in AFP are coded most significant byte first. Another way AFPDS differs from Metacode is that each AFP instruction is preceded by its inclusive length specifier. For example, x'04D300F0' is the full AFP instruction for an absolute y move. X'04' shows that the complete length of the code is 4, including the length indicator. The second and third bytes of each AFP record indicate the length of the entire record minus the x'5A' carriage control. Metacode has no such length indicators.

Our first Field tag group identifies the customer name from each document. We’ll add the name to the side file and to the AFP Index added to output. The <FIELD> tag identifies a variable called %Name. We don’t want to change the field information, so we set the <FIELD> tag action parameter to KA. The <REFERENCE> tag instructs Enrichment to set a reference point on every AFP record in each document (that is, every record with a x'5A' (!) carriage control character in column 1) that contains the string x'D3026D04C7017C' beginning in column 77. The <LOCATION> tag indicates that the field occupies the same record as the reference point, begins 18 characters
after the reference point, and is a maximum of 40 characters in length. We also use the %%Name value as a top-of-document indicator. In the Input tag group, <DOCUMENT>TOP %%Name CHANGE indicates that each time the %%Name value changes in the input, a new document begins.

**Note:** Remember to set the <FIELD> tag *action* parameter to KA or RA for any field used as a basis for finding documents.

The next two Field tag groups identify the address lines that appear on each document. In the first of these Field groups, the <FIELD> tag defines a variable called %Address1 that will store the first line of address from each document. We'll reference this variable later to include it in the side file. Because we only want to extract the first occurrence of the value from each document, we set the <FIELD> tag *action* parameter to K. Since the location of the address information may change from document to document, we used the <REFERENCE> tag to identify a reference point for the field information. The <REFERENCE> tag instructs Enrichment to set a reference point on every AFP record in each document that contains the string x'D302904C7017C'. The <LOCATION> tag indicates that the field occupies the same record as the reference point, begins 7 characters after the reference point, and is a maximum of 40 characters in length. The other address field similarly defines the position of the second address line.

In the last Field group, the <FIELD> tag defines the variable %%NameFont to extract the font index that controls the font in which the name prints. The AFP font command is x'F1' and each font sequence is three bytes long (length indicator, font code, and font number). For example, x'03F101' indicates that text after the code uses font number 1. Fonts are assigned in the Active Environment Group (AEG) that precedes each AFPDS page. We set the *action* parameter to R because we only want to replace the first occurrence of the variable from each document. The <REFERENCE> tag instructs Enrichment to set a reference point for the field on every PTX record in each document that contains the string x'D3026D04C7017C' beginning in column 77. The <LOCATION> tag indicates that the field occupies the same record as the reference point, begins 11 characters after the reference point, and is one character in length.

**Rules**

We’re changing a field value for each document, so we define processing in the DOCUMENT: section of the rule. We only want to change the font in which the name prints if it currently has a font index of x'02' (that is, a hexadecimal value of 02). If this is the case, we instruct Enrichment to change the font index to x'04'.

Output Tag Group

Next, we specify the Output tag group and the Sidefile group within it. We use the Output group <NAME> tag to identify the Output group in the Enrichment report (Changed Font Letter). We also used the <FILE> tag to specify the output file, DD:OUTPUT1.

Sidefile Tag Group

The Sidefile group <FILE> tag identifies the file to create (DD:SIDE1). We use <SIDEPART> tags to specify the record layout and contents of the side file. Each record of the side file will contain the data defined in each of the <SIDEPART> tags, in the order in which the tags are used in the Sidefile group. Thus, each record of the side file will contain the %%Name, %%Address1, and %%Address2 values.

Each <SIDEPART> tag specifies that the value of one of the fields, which will be a maximum of 40 characters in length, should be left-justified (L) and padded with blanks (‘ ‘) if necessary. As the following shows, Enrichment places each line of address information on the record side by side in the order in which the <SIDEPART> tags are defined.

<table>
<thead>
<tr>
<th>Name</th>
<th>Address1</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Henry O'Brian</td>
<td>346 Waller Rd.</td>
<td>Hopkinsville,</td>
</tr>
<tr>
<td>Ann Camerinski</td>
<td>2343 Alexander Dr.</td>
<td>Paris,</td>
</tr>
<tr>
<td>Lisa C. Martin</td>
<td>401 Jennifer Court</td>
<td>Lexington,</td>
</tr>
</tbody>
</table>

Use the output document number in the Enrichment Report to verify that the side file contains one record for each document found in the input.

AFPIndex Tag Group

The <AFPIXD> tag group is used to insert the appropriate TLE structured fields as well as Begin Named Group (BNG) and End Named Group (ENG) structured fields. The descriptor and variable value specified in the <PART> tags comprises the information someone using an AFP viewer will use to search the print stream.

We specified two <PART> tags in this sample. The first <PART> tag has a descriptor of Customer Name: and a value of %%Name. Since the value of %%Name can be up to 40 characters in length, Enrichment will truncate it at 25 characters for inclusion in the TLE. The second <PART> tag has a descriptor of EnrichmentDocNumber and a value of %%DOCUMENT_NO. %%DOCUMENT_NO is an Enrichment system variable. The value will have a length of 5 characters in the TLE.
Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run this application.

```jcl
//*JOB
//*OBBL DD DSN=PDR.STREAMW.LOADR3UN,DISP=SHR
//* DD DSN=SYS3.V4R1M0.ISPL0AD,DISP=SHR
//* DD DSN=SYS3.CLIB2.EDCLNK,DISP=SHR
//* DD DSN=SYS3.PLI230.SIBMLNK,DISP=SHR
//* --------------------------------------------------------------
//* SW   EXEC PGM=PDRSW000,PARM='//B', REGION=0M
//* INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A05INPT)
//* OUTPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A05OUTP)
//* SIDE1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT2(A05SIDE)
//* REPORT DD SYSOUT=* 
//* SYSPRINT DD SYSOUT=* 
//* SYSTERM DD SYSOUT=* 
//* SYSUDUMP DD SYSOUT=* 
//* CONTROL DD DISP=SHR, DSN=PDR.STREAMW.HANDSON.CONTROL(A05CONT)
//* -------------------------------------------------------------
//* COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//* NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A05OUTP),DISP=SHR
//* OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A05OUTP),DISP=SHR
//* OUTDD DD SYSOUT=* 
//* -------------------------------------------------------------
//* COMP02 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//* NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT2(A05SIDE),DISP=SHR
//* OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT2(A05SIDE),DISP=SHR
//* OUTDD DD SYSOUT=* 
//* -------------------------------------------------------------
```

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:
Advanced sample 5: Extracting information and creating a flat file (AFPDS)

```
rm DDCONTROL
rm DD:INPUT1
rm DD:OUTPUT1
rm DD:SIDE1
ln -s /share/home1/SunOS/handson/A05cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/A05inpt.afp DD:INPUT1
ln -s /share/home1/SunOS/handson/A05out1 DD:OUTPUT1
ln -s /share/home1/SunOS/handson/A05side DD:SIDE1
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=A05CONT.ws.CON
```
Advanced sample 6: Adding Barcodes to DJDE or impact printer data

In this section

Overview ........................................................................................................ 174
Files Used ...................................................................................................... 174
Business Scenario ........................................................................................ 175
Explanation of Sample ............................................................................... 175
Overview

This sample adds a 3of9 barcode that contains the social security number, page number, and total pages in the document. This sample introduces the following topics.

- Add tag group
  - <POSLINE> tag
  - <FONTNUM> tag
- Setting up an insert record file with barcode components

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\adv6\a06inp1.djd</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.INPUT(A06INP1)</td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td>Inline</td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td>D:\apps\adv6\a06con2.con</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.CONTROL(A06CON2)</td>
</tr>
<tr>
<td>A06CON.ws.CON</td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td>D:\apps\adv6\a06out1.djd</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.GOODOUT1(A06OUT1)</td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
</tr>
<tr>
<td>JCL:</td>
</tr>
<tr>
<td>D:\apps\adv6\a06jcl2.jcl</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.JCL(A06JCL2)</td>
</tr>
<tr>
<td>UNIX shell script:</td>
</tr>
<tr>
<td>D:\apps\adv6\a06scr.sh</td>
</tr>
<tr>
<td>PDR.STREAMW.HANDSON.JCL(A06SCR)</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application adds a 3of9 barcode to each document. The barcode should contain the Social Security number, page number, and total number of pages in the document.

Explanation of Sample

The control file we created for the application (shown below) reads an input print stream and adds Code 3of9 barcodes.
Note: This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A06CONT.ws.CON contains these modifications.

Note: You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```xml
<input>
  <name> Input   <!-- Name of input file -->
  <file> DD:INPUT1  <!-- DD name in JCL -->
  <type> D A  <!-- DJDE, ANSI carriage control -->
  <header> 9 Y  <!-- First 9 records are not processed except to copy to the output. -->
    <iden> e'DJDE' 6 11  <!-- Records with EBCDIC "DJDE" in position 7 are included with the page that follows. The DJDE operands start in position 12. -->
    <doc> T %SSNO CHANGE  <!-- Document determined by change in the value of the SSNO field. -->
      <!-- The following field tag group is for the Social Security Number. The field is located on record 9, column 27 for a length of 11. -->
      <field> %SSNO KA
        <loc> 9  27 11 RECORD  <!-- SSNO, 9 chars, left justified, space filled. -->
      </field>
    </doc>
  </header>
</input>

<rule>
  <content>  <!-- Content tag group allows rule statements within the control file. -->
    DOCUMENT:
      <!-- The dashes are removed from the SSNO variable. The remaining pieces are concatenated together. -->
      %DispSSN = SUBSTR(%SSNO,1,3) | SUBSTR(%SSNO,5,2) | SUBSTR(%SSNO,8,4)
  </content>
</rule>

<add>
  <addtype> T  <!-- Add type is TEXT -->
    <fontnum> 6  <!-- Barcode font number -->
      <addpart> %DispSSN 9 L ' '  <!-- SSNO, 9 chars, left justified, space filled. -->
      <addpart> %PAGE_NO 2 R 0  <!-- System variable, 2 chars, right justified, zero filled. -->
      <addpart> %TOTAL_PAGES 2 R 0  <!-- System variable, 2 chars, right justified, zero filled. -->
      <posline> 2 70 TRC V F R  <!-- Specifies position of added text, row 2, column 70, input has carriage control and TRC, info added in vertical direction, in forward direction. The row column number refer to records in the print stream. -->
    </add>
  </add>

<output>
  <name> Output  <!-- Name of output file -->
```

EngageOne Enrichment 7.3.0 Sample Applications Guide 176
Input Tag Group

As the control file shows, the Input group is identified in the <NAME> tag as Input, and the input file is found on the mainframe system at DD:INPUT1, as indicated by the Input group <FILE> tag. The input is DJDE line data with ANSI carriage controls, so we set the <TYPE> tag to DA.

We don’t want to process all of the DJDE records at the top of the input, so we used a <HEADER> tag to instruct Enrichment to ignore the first nine records in the input. We want to copy those records to the output, however, so we set the <HEADER> tag keepYN parameter to Y.

The <IDEN> tag tells Enrichment how to recognize DJDE records in the print stream. We specified <IDEN> e’DJDE’ 6 11, so Enrichment knows that a DJDE record is present when the EBCDIC string DJDE begins in column 7. The DJDE commands begin in column 12.

Field Tag Group

We defined one field, which Enrichment will use to set top-of-document and as data for the Code 3of9 barcode. Therefore, the <FIELD> tag identifies a variable called %%SSNO. The value of the field is the customer’s social security number. We only want to extract the field information, so we set the <FIELD> tag action parameter to KA.

The <LOCATION> tag indicates that the field information begins on record 9 in column 27 and is 11 characters in length. We set the <LOCATION> tag method parameter to RECORD, indicating that Enrichment is to count records in the print stream to locate the field information.

Note: The <LOCATION> tag method parameter specifies whether Enrichment counts records in the print stream (RECORDS) or lines of printed data (LINE) to locate field information. Refer to the Enrichment Language Reference for more information on the <LOCATION> tag.

We set the Input group <DOCUMENT> tag so that whenever the value of %%SSNO changes in the input (that is, whenever the social security number changes) a new document begins.
Advanced sample 6: Adding Barcodes to DJDE or impact printer data

Rules

The social security numbers in the input print stream include dashes, which we need to remove before we can use the numbers in the barcodes. We used the `SUBSTR` function to extract the numeric characters of the social security number and concatenated the results into a new variable, `%%DispSSN`.

Note: Refer to the *Enrichment Language Reference* for a more in-depth discussion of the `SUBSTR` function.

Add Tag Group

We set up an Add tag group outside the Output group to build and position the Code 3of9 barcodes on the documents.

Since the barcodes we’re adding to the DJDE data must be font-based, we set the `<ADDTYPE>` tag to `T`. We set the `<FONTNUM>` tag to `6` to identify the font Enrichment will use to print the Code 3of9 barcode.

Note: Since we’ll create only one output, there is no difference between placing the Add group within or outside the Output group in the control file.

We specified three `<ADDPART>` tags. The first `<ADDPART>` tag instructs Enrichment to add the left-justified value of `%%DispSSN` (the social security number from the `DOCUMENT:` rule process), which is up to 9 characters in length, to the barcode. If the `%%DispSSN` value is short for some reason, Enrichment will pad it to 9 characters with blanks. The second `<ADDPART>` tag instructs Enrichment to add the right-justified value of the system variable `%%PAGE_NO`, which is up to 2 characters in length, padding the value with zeroes if necessary. Likewise, the last `<ADDPART>` tag instructs Enrichment to add the right-justified value of the system variable `%%TOTAL_PAGES`, which is up to 2 characters in length, padding the value with zeroes if necessary.

We also set the `<POSLINE>` tag so that the barcode will overprint the documents, beginning 2 records from the top of the page and 70 columns from the left edge of the page. The barcode will print vertically down the page in the order in which we specified the `<ADDPART>` tags.

Output Tag Group

In the Output group, we set the `<NAME>` tag to `Output`, which resides on the mainframe system at `DD:OUTPUT1` as indicated by the `<FILE>` tag.
An Alternative Approach: Using the Insertrec Group to Add Barcodes

If you can't use the `<POS Li NE>` tag to position a barcode on your DJDE documents with the Add group, you can use the Insertrec tag group to add the barcode. Usually, you must do this when your barcode font is a different size than the fonts on the records before and after the added barcode record. The control file below shows how to get the same results as using an Insertrec.

**Input Tag Group**

The only difference in the Input group is that we set the `<HEADER>` tag `keepYN` parameter to `N`, indicating that the header records are to be ignored during processing and not included in the output.

**Field Tag Group**

In this implementation, we used a second Field tag group in the Input group to define a field to use as a placeholder for the added barcode. The `<FIELD>` tag identifies a variable called `%%Bar Code`. We only want to extract the field information, so we set the `<FIELD>` tag `action` parameter to `KA`. The `<LOCATION>` tag indicates that the field information begins on record 2 in column 1 and is 1 character in length.

```plaintext
<input>   <! Begin input tag group  >
  <name> Input   <! Name of input file  >
  <file> DD:INPUT1     <! DD name in JCL   >
  <type> D A   <! DJDE, ANSI carriage control  >
  <header> 10 N    <! First 10 records are not copied >
                <! to the output. >
  <iden> e'DJDE' 6 11       <! Records with EBCDIC 'DJDE' in >
                <! position 7 are included with >
                <! the page that follows. The DJDE>  
                <! operands start in position 12. >
  <doc> T %SSNO CHANGE        <! Document determined by change in >
                <! the value of the SSNO field. >
  <! The following field tag group is for the Social Security Number. >
  <! The field is located on record 9, column 27 for a length of 11. >
  <field> %SSNO KA
    <doc> 9 27 11 RECORD
  </field>
  <! The following field is a placeholder used for the placement of >
  <! the Code3of9 barcode. We can pick up any field on each page and >
  <! the insertrec will be added immediately below the last record of >
  <! each page. >
  <field> %BarCode KA
    <doc> 2 1 1 R
  </field>
</input>

<insertrec> %%BarCode  <! Begin insertrec tag group, name is >
    <! %Barcode. >
```
Advanced sample 6: Adding Barcodes to DJDE or impact printer data

```
<type> D A   <! DJDE, ANSI carriage control  >
<place> AFTER  <! Placement of inserted record  >
<!    relative to the placeholder    >
<!    field "%%BarCode".    >
<substitute> YES   <! Variable names in the insert will >
<!   be replaced with the values of >
<!  the fields. >
<! Content tag group below contains the format of the barcode >
<! record that is added to the print stream. The first two >
<! positions are the carriage control and font, followed by the 3 >
<! field names that make up the barcode. >
<!  Content tag group below contains the format of the barcode >
<! record that is added to the print stream. The first two >
<! positions are the carriage control and font, followed by the 3 >
<! field names that make up the barcode. >
<content>
96%%DispSSN%%PN%%TP
</content>
</insertrec>
<brule>  <! Begin rule  >
<content>                    <! Content tag group allows rule  >
DOCUMENT:    <!   statements in the control file. >
<! The dashes are removed from the SSNO variable. The remaining >
<! pieces are concatenated together.  >
%%DispSSN = SUBSTR(%%SSNO,1,3) | SUBSTR(%%SSNO,5,2) | 
SUBSTR(%%SSNO,8,4)
<PAGE:    <! These statements convert the system variables %%PAGE_NO and >
<! %TOTAL_PAGES to fixed length, right justified, with leading >
<! zeroes for use in the insertrec.    >
%%PN = JUSTIFY(%%PAGE_NO,R,4,0)
%%TP = JUSTIFY(%%TOTAL_PAGES,R,4,0)
</content>
</rule>
<output>  <! Begin output tag group.  >
<name> Output   <! Name of output file. >
<file> DD:OUTPUT1  <! DD name in JCL. >
<headerfile> DD:HEADER <! Additional ASSIGN statement to add.>
</output>
```

Insertrec Tag Group

Instead of the Add group, we used an Insertrec tag group in this implementation of the application. The Insertrec tag group identifies a record we want to insert in each document and specifies where to add it. The insert record is identified in the `<INSERTREC>` tag as `%%BarCode`. We used a Content tag group to include the insert record in the control file. The record is DJDE data with ANSI carriage controls, so we set the `<TYPE>` tag to `D A`. The following shows the information in the insert record. We set the `<SUBSTITUTE>` tag to `YES` so Enrichment would substitute the variables in the insert record with the appropriate values during processing. The `<PLACE>` tag instructs Enrichment to insert the barcode after the placeholder field `%%BarCode`.

```
<content>
96%%DispSSN%%PN%%TP
</content>
```
Rules

Another difference between this and the previous implementation of this application is in the rules. We still build the value of %%DispSSN in the DOCUMENT: section of the rules, but we've added a PAGE: section to the rules to convert the system variables %%PAGE_NO and %%TOTAL_PAGES to right-justified, fixed-length values with leading zeroes (%%PN and %%TP) for use in the barcode.

Output Tag Group

Finally, the use of an insert record to add the barcode requires us to add an additional ASSIGN command in the DJDE records. Therefore, we've used the <HEADERFILE> tag in the Output group to identify a file (DD: HEADER) that contains the new DJDE records to include in the output. The following shows the contents of the header file.

```plaintext
+++D| DE ASSIGN=(1, 3);
+++D| DE ASSIGN=(2, 11);
+++D| DE ASSIGN=(6, 23);
+++D| DE ASSIGN=(7, 73);
+++D| DE ASSIGN=(8, 80);
+++D| DE BEGIN=(9.0, 0.4);
+++D| DE BEGIN=(.3, 0);
+++D| DE FORMAT=P0635;
+++D| DE MARGIN=(.05, IN);
+++D| DE DUPLEX=NO, FORMS=(A064), END;
```

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```plaintext
// *>JOB CARD
// */OBJLIB DD DSN=PDR.STREAMW.LOADCRUN, DISP=SHR
// DD DSN=SYS3.CLI.B22.SEDCLINK, DISP=SHR
// DD DSN=SYS3.PL1230.SIBMLINK, DISP=SHR
// DD DSN=SYS3.V4R1MO.ISPLOAD, DISP=SHR
// */-----------------------------------------------
// */ SW EXEC PGM=PDRSW000, REGION=OM
// */ INPUT1 DD DISP=SHR, DSN=PDR.STREAMW.HANDSON.INPUT(A06INPT)
// */ OUTPUT1 DD DISP=SHR, DSN=PDR.STREAMW.HANDSON.NEWOUT1(A06OUT1)
// */ REPORT DD SYSOUT=* 
// */ SYSPRINT DD SYSOUT=* 
// */SYSTEM DD SYSOUT=* 
```
UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```shell
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:

```shell
r m DDCONTROL
r m DD:INPUT1
r m DD:OUTPUT1
r m A06out1
ln -s /share/home1/SunOS/handson/A06cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/A06inpt.djd DD:INPUT1
ln -s /share/home1/SunOS/handson/A06out1 DD:OUTPUT1
s weaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```shell
s weaver -c=A06CONT.ws.CON
```
Advanced sample 6: Adding Barcodes to DJDE or impact printer data
Advanced sample 7: Postal processing with complex print

In this section

Overview.................................................................................................................. 186
Files Used.............................................................................................................. 186
Business Scenario.................................................................................................. 187
Explanation of Sample (First Pass)................................................................. 189
Explanation of Sample (Second Pass)............................................................. 192
Overview

In this sample, the input print stream contains customer invoices and notices. Each document can contain invoice pages, notice pages, or both. We need to standardize the addresses on all of the documents, but the position of address information differs between the invoices and notices. After the addresses are cleansed, we need to presort the output.

Because the address information is inconsistently positioned among the document types (rendering cleansing impossible) and because we cannot presort the documents until the addresses are cleansed, we must perform two Enrichment runs.

In the first Enrichment run, we will separate the documents that contain only a notice from those that begin with an invoice page. The notices must be cleansed and presorted separately because the address is in a different location and will not show in the envelopes used for the invoice documents.

In the second Enrichment run, we will:

• Call Finalist to cleanse addresses
• Update original address blocks with cleansed addresses
• Use MailStream Plus to presort the output

This sample introduces the following topics.

• Multiple document types  • Cleansing multiple address locations
• Multiple Enrichment passes

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample (Part 1 of 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
**Business Scenario**

This AFP line-data print stream contains invoices and customer notices. Unfortunately, address blocks are not consistently placed. The sample application standardizes addresses and presorts the documents. This processing involves two Enrichment passes.
In the sample, above,

- In Enrichment pass 1 separate the documents which have only a notice page from those containing at least one invoice page.
- In Enrichment pass 2, do the following for invoices and notice documents:
  - Use Finalist to standardize addresses
  - Update original address blocks
  - Add a POSTNET barcode immediately above the first address record.
  - Use MailStream Plus to presort documents.
Explanation of Sample (First Pass)

Our control file for the first Enrichment run (shown below) determines which type of document is currently being processed (invoice or notice) and outsorts notice pages that match no invoice pages.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS.

### Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `Input`, and the input file is found on the mainframe system at `DD:INPUT1`, as indicated by the `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPLINE ANSI`. We specified a separate `<DOCUMENT>` tag to handle each type of document in the input. The first `<DOCUMENT>` tag specifies that a new document begins each time the value of `DocTop` is `INVOICE`. The second tag specifies that a new document begins each time the value of `DocTop` is `NOTICE`. The field `DocTop` is defined in the first Field group.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```xml
<input>
  <name> Input </name>
  <file> DD:INPUT1 </file>
  <type> AFPLINE ANSI </type>
  <doc> T DocTop = 'INVOICE' </doc>
  <doc> T DocTop = 'NOTICE' </doc>
  <field> DocTop KA </field>
  <loc> 1 3 7 </loc>
</input>

<rule>
  <content>
    DOCUMENT:
    <%%TypeChg = CHANGED(%%Type)>
    if %%Type = 'INVOICE' then
    elseif %%Type = 'NOTICE' and %%TypeChg then
  </content>
</rule>
```
Field Tag Group

We defined two Field tag groups for this Enrichment run.

The first Field group identifies top-of-document. Each document contains an indicator that specifies whether the document is an invoice or notice. The `<FIELD>` tag defines a field called `DocTop` whose value is this indicator. We want to extract every occurrence of the information in each document, so we set the `<FIELD>` tag action parameter to `KA`. The `<LOCATION>` tag indicates that the field information resides on row 1, column 3 of the document and is 7 characters in length.

The second Field group picks up the first occurrence of each type of document. Since some invoices contain multiple pages and pages 2 and greater have no data in column 3, we must set a separate field from that used for top-of-document. This is because the `DocTop` field must have an `action` parameter value of `KA`. If we used the `DocTop` value elsewhere in the control or rule file, it would keep the last value found. In multi-page invoice documents, then, `DocTop` would be blank. So, we defined the field `%%Type`, which is identical to `DocTop` except that it has an `action` parameter value of `K` (extract only the first occurrence of the field in each document). We used `%%Type` in the rule file to outsort the notice-only documents.

Rules

Rather than setting up a separate rule file, we included the rules in a Content tag group within the Rule tag group. Since we are using rules to process documents in the input, we define rule processing in the DOCUMENT: section.

First, we set a new variable, `%%TypeChg`, to the return from running the `CHANGED` function against the value of `%%Type` for the current document. If the `%%Type` value for the previous document was different than the `%%Type` value for the current document,
Enrichment sets $\%TypeChg$ to 1 (true). If the $\%Type$ value for the previous document is the same as the value for the current document, Enrichment sets $\%TypeChg$ to 0 (false).

As the code below shows, if the value of $\%Type$ is INVOICE, we use the <OUTPUT> command to route invoices to a file identified as Invoice_Output. We use the <OUTPUT> command to direct invoices to a file identified as Output.

```plaintext
If $\%Type$ = 'INVOICE' then
  <output> Invoice_Output
```

If the value of $\%Type$ for the current document is not INVOICE, we need to do some additional processing to ensure that only valid documents end up in the invoice output.

We know that all customers should receive an invoice or a notice, but some customers may not receive both. If the customer receives an invoice or an invoice and a notice, we want to route the document to Invoice_Output output. If the customer receives a notice only, we need to outsort the document to the Notice_Only output. To be able to outsort the notice-only documents, we are treating the invoices and notices as separate documents in this Enrichment run. In the second Enrichment run, each document will include all the pages that go to one customer.

This is where the $\%TypeChg$ variable comes into play. As shown below, if the value of $\%Type$ is NOTICE and Enrichment sets $\%TypeChg$ to 1 (true), Enrichment writes the document to Invoice_Output.

```plaintext
elseif $\%Type$ = 'NOTICE' and $\%TypeChg$ then
  <output> Invoice_Output
```

As the code below shows, if the value of $\%Type$ isn't NOTICE (that is, its INVOICE or the type is unknown), Enrichment will direct the document to a file identified as Notice_Only.

```plaintext
else
  <output> Notice_Only
```

### Output Tag Group

We defined two Output groups in the control file for the first Enrichment run. The first Output group <NAME> tag defines Invoice_Output, which resides at DD:OUTPUT1 on the mainframe system. Output will contain all of the documents for which the value of $\%Type$ was INVOICE or for which the value of $\%Type$ was NOTICE and had changed from the previous document. The second Output group <NAME> tag defines Notice_Only, which resides at DD:OUTPUT2 on the mainframe system. Error will contain any documents for which the value of $\%Type$ was NOTICE but hadn’t changed from the previous document or for which the value of $\%Type$ was unknown.
Explanation of Sample (Second Pass)

Our solution for the second Enrichment run only processes Invoice_Output from the first run. This same process would be valid for processing Notice_Only with a few adjustments:

- The address field locations would be different
- The rules are unnecessary
- The Add group <POSITION> tag would place the POSTNET™ barcode at a different location on the notice page.

Using the output from the first Enrichment run as input for the second run, Enrichment uses the control file shown below to do the following:

- Identifies fields for all the address lines in their original positions on the invoices and notices
- Runs Finalist to cleanse the address information on the first page of all documents
- Replaces the original address fields with their cleansed value from the inserted record
- Adds a POSTNET™ barcode (DPBC) immediately above the address on the first page of each document
- Deletes the document type indicator from all documents
- Runs MailStream Plus to perform a postal presort on all of the documents
- Writes presorted documents to an output file and documents that were not presorted to a reject output file.

Note: This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the <FILE> tag for your OS.

```xml
<input>
  <name> Input </name>
  <file> DD:INPUT1 </file>
  <type> AFPLINE ANSI </type>
  <doc> T %Type = 'INVOICE' </doc>
  <field> %Type DA </field>
  <loc> 1 3 7 </loc>
</input>

<addressblock> 9 9 35 4 L R </addressblock>
<field> %NotAddr1 R </field>
  <ref> '1' 'NOTICE' 3 </ref>
  <loc> 10 22 35 </loc>
</field>

<field> %NotAddr2 R </field>
  <ref> '1' 'NOTICE' 3 </ref>
  <loc> 11 22 35 </loc>
</field>
```

EngageOne Enrichment 7.3.0 Sample Applications Guide
Input Tag Group

As the control file shows, the Input group is identified in the <NAME> tag as Input, and the input file is found on the mainframe system at DD: INPUT1, as indicated by the <FILE> tag. The input is AFP line data with ANSI carriage controls, so we set the
<TYPE> tag to AFPLINE ANSI. We used one <DOCUMENT> tag to specify that a new document begins each time the value of %%Type is INVOICE. Since we are assuming that each document in the input will consist either of an invoice followed by a notice or just an invoice, we don’t need to define top-of-document for notices (we wrote all other policies to Notice_Only in the first Enrichment run). We want to cleanse the addresses in the documents, so we set the <CLEANSE> tag to Y.

We used the <ADDRESSBLOCK> tag to define the invoice address to pass to Finalist. The address block begins on print line 9 in column 9 and has a depth of up to 4 lines. We will replace the first address block in each document.

Note: For the Notice_Only output, the address block would begin on print line 11 in column 22 and have a depth of up to 4 lines. You would replace the first address block in each document.

Field Tag Group

We defined five Field tag groups for this Enrichment run. The fields serve three basic purposes in the application:

- Identify top-of-document
- Delete the document type-of-document indicator
- Identify address information on the notices.

The first Field group identifies the document type indicator, which is also used to set top-of-document. The <FIELD> tag defines a variable called %%Type whose value is the indicator that specifies whether the document is an invoice or notice. We want to extract and delete every occurrence of the information in each document, so we set the <FIELD> tag action parameter to DA. The <LOCATION> tag indicates that the field information resides on record 1, column 3 of the document and is 7 characters in length.

The next four Field groups define the lines of address information on the notice documents in the input. The <FIELD> tags define variables called %%NotAddr1 through %%NotAddr4. We want to replace the first occurrence of each fields information in each document with cleansed address information, so we set the <FIELD> tag action parameters to R. As the <REFERENCE> tags show, Enrichment sets reference points for the fields on all records that have a carriage control 1 and contain the string NOTICE beginning in column 3. The <LOCATION> tags indicate that the fields reside between 10 and 13 records below the reference point, 22 columns after the reference point, and are 35 characters in length.

Note: These fields would be replaced by the <ADDRESSBLOCK> tag if you were processing the Notice_Only documents.
CASS Tag Group

We included a CASS tag group in the control file because we want to cleanse the addresses in the input. We use Finalist for address cleansing, so we set the \texttt{<CASSTYPE>} tag to \texttt{LPC}.

We specified \texttt{<LPCREPLACE>1 5 9 6 6 4 4 4 2} to define the processing thresholds for address cleansing. Each of these numbers represents a specific processing threshold for Finalist. If an address exceeds any one of these thresholds, Enrichment won’t replace it. We are most concerned with the first number, however, which represents the threshold for the general return code from Finalist when an address is cleansed (Enrichment stores this value in the system variable \texttt{%%LPCRC}). If a Finalist return code for a specific address exceeds \texttt{1}, that address is not replaced. In other words, Enrichment will replace only addresses for which Finalist issued a return code of \texttt{1} (five-digit ZIP Code and carrier route assigned) or \texttt{0} (ZIP Code, ZIP + 4®, and carrier route assigned) after cleansing.

Rules

We included the rules in a Content tag group within the Rule tag group. Since we are using rules to process documents in the input, we define rule processing in the \texttt{DOCUMENT:} section of the rule.

In the rules, we assign the cleansed values of \texttt{%%ADDRESSn} variables to replace the original values of the \texttt{%%NotAddrn} fields as appropriate.

\textbf{Note:} No rules are necessary to process the \texttt{Notice Only} documents.

\begin{verbatim}
DOCUMENT:    <! Content tag group allows rule >
             <! file statements in control >
             <! file. >

%%NotAddr1 = %%ADDRESS1
%%NotAddr2 = %%ADDRESS2
%%NotAddr3 = %%ADDRESS3
%%NotAddr4 = %%ADDRESS4
\end{verbatim}

Add Tag Group

We defined an Add tag group to add a drawn POSTNET™ barcode to the documents. To indicate that the barcode is drawn, we set \texttt{<ADDTYPE>} to \texttt{P}. Since we’re only adding the POSTNET™ barcode, we only defined one \texttt{<ADDPART>} tag, \texttt{%%POSTNET}, with a length of \texttt{14}. \texttt{%%POSTNET} is a system variable whose value is generated by Finalist. We add the barcode at \texttt{0.75} inches from the right side of the page and \texttt{2} inches from the top of the page.
Note: When adding a POSTNET™ to the Notice Only output, the <POSITION> tag settings would change to reflect the different address location on the notice pages. This causes the barcode to print immediately above the first address line. <ONPAGE>1 instructs Enrichment to add the barcode only to the first page of each document.

Output Tag Group

We defined one Output group in the control file for the second Enrichment run. The Output group <NAME> tag defines Output, which resides at DD:OUTPUT1 on the mainframe system. Output will contain all of the documents except those in which a bad ZIP Code was identified by MailStream Plus. We want to use MailStream Plus to presort this output, so we included the Presort tag group.

Presort Tag Group

We specified <PRETYPE>L, indicating the use of MailStream Plus. Enrichment uses the values of fields and system variables identified in <SORTPART> tags to build a presort index. The presort index contains one record for each document sent to this <OUTPUT> which in this example is one record for each document in the input print stream since there is only one <OUTPUT> tag group. The Presort group <FILE> tag indicates that the presort index resides on the mainframe system at DD:MSSTIN. The following shows the MSSTIN file created by the second Enrichment run for this sample.

```
7hhtk
7hhv0
7hhwi 06810-4147
7hhym 06810-4147
7hhag 40504-3357
7hhcz 40504-3357
7hhf4 60532-3672
7hhho 60532-3672
```

We defined two <SORTPART> tags for the application:

- Our first <SORTPART> tag specifies a system variable, %%DOCINDEX, whose value is an eight-digit index number for each document in the input print stream. As the <SORTPART> tag indicates, Enrichment will left-justify the %%DOCINDEX value in the presort index and pad it with blanks if it is less than 9 characters in length.
- Our second <SORTPART> tag specifies the system variable %%ZIPCODE, whose value is the ZIP Code information resulting from the address cleansing process. The %%ZIPCODE value is up to 10 characters in length. Enrichment will left-justify the value in the presort index, and will pad the value with blanks if it is less than 10 characters in length.
Our MailStream Plus parameter settings will further define the column positions of data in the presort index.

After Enrichment creates the presort index and calls MailStream Plus, MailStream Plus compares the `%%ZIPCODE` value with its database and sorts the records in the presort index accordingly, writing the records in their new order (along with other data) to the MailStream Plus Inkjet file. The `<OUTFILE1>` tag indicates that the MailStream Plus Inkjet file resides on the mainframe system at DD: MSNAON. The Inkjet file contains one record per mailpiece (that is, one record per document).

Enrichment scans the Inkjet file for the `%%DOCINDEX` value. If you use default settings, Enrichment expects the `%%DOCINDEX` value to begin in column 16 of the Inkjet file. As Enrichment locates each `%%DOCINDEX` value in the Inkjet file, it finds the corresponding document in the input print stream and writes it to the output file (OUTFILE1). Thus, the output file contains the documents in proper presort order. Enrichment writes input documents not listed in the Inkjet file to the reject file (DD: REJECT).

The `<REJECTFILE>` tag indicates that the file to use for rejected documents resides on the mainframe system at DD: REJECT. Enrichment places in the reject file documents for which MailStream Plus cannot find ZIP Codes in its database and that are not returned in the `<OUTFILE1>` file. Documents may be written to the reject file if:

- `%%ZIPCODE` is blank. The application that was responsible for setting the correct ZIP Code failed to generate one for the document.
- `%%ZIPCODE` is invalid according to the MailStream Plus database.

If MailStream Plus successfully presorts all of the documents in the input (that is, if each document is represented by a record in the presort index file), Enrichment creates an empty reject file.

**Note:** Refer to your MailStream Plus documentation for details about MailStream Plus programs and sort parameters.

The `<STEP>` tag calls MailStream Plus.

**Mainframe Run Time**

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

**Mainframe**

The code below shows the JCL for this application. In the `JOBLIB`, we included the load libraries for MailStream Plus and Finalist. The `DELETE` and `CREATE` steps ensure
that all the Enrichment output files and all the temporary files that MailStream Plus uses are deleted and reallocated as new. The MMUSER file is allocated with the exact record length that corresponds to the data that Enrichment will insert.

The output from the first Enrichment step is the input to the second Enrichment step. Because this data set is really temporary, you can delete it once the second Enrichment step is finished.

We need to allocate a data set to hold any documents not included in the MailStream Plus presort output. These documents are rejected because the POSTNET™ was not found in the MailStream Plus database.

```c
// *********************************************************************
//* INSERT A JOBCARD ABOVE THIS COMMENT BOX *
//* Member Name: CALLMSP
//* Last Revised: 12/15/2004
//* Description: This job:
//*   Executes a Streamweaver job which calls MSP.
//*   STEP1OF3 - Executes IDCAMS to remove allocated files from
//*               a prior run.
//*   STEP2OF3 - Executes IDCAMS to remove MSXX00WK VSAM file from
//*               a prior run.
//*   STEP3OF3 - Executes Enrichment using the control file
//*               yourhlq.STREAMW.CONTROL(MSPSAMPL) copied from the
//*               installation media
//* Expected Return Codes:
//*   - STEP1OF3: 0
//*   - STEP2OF3: 0
//*   - STEP3OF3: 0
//* Variables: The following lower-case variables must be changed
//*   in order to run this job:
//* ** NOTE **
//* Please make sure to set "CAPS ON" prior to editing.
+-------------------+------------------------------+
| VARIABLE: | REPLACE WITH: |
+-------------------+------------------------------+
| yourhlq | your high-level qualifier |
| SWhlq | Location of Enrichment |
| MSPhlq | Location of MSP |
| volume | VSAM DASD pack volser. |
```
//**
//*********************************************************************
//**
//**STEP1OF2 EXEC PGM=IDCAMS
//**SYSPRINT DD SYSOUT=*
//**SYSIN DD *

DELETE yourhlq.MSP.MSSTIN PURGE
DELETE yourhlq.MSP.MSWKJ4 PURGE
DELETE yourhlq, MSP, MSWKJ6 PURGE
DELETE yourhlq, STREAMW.REPORT(MSP5AMPL) PURGE
DELETE yourhlq, STREAMW_OUTPUT1 PURGE
DELETE yourhlq, STREAMW.REJECTS PURGE
/* Start of delete when RUNMSP column 8 is Y */
/* When removing these Deletes also change the */
/* DISP of these files to DISP=(OLD) in STEP2OF2 */
DELETE yourhlq, MSP, MSSTON PURGE
DELETE yourhlq, MSP, MSSTOR PURGE
DELETE yourhlq, MSP, MSWKSS PURGE
DELETE yourhlq, MSP, MSWKCM PURGE
/* Start of delete when RUNMSP column 10 is Y */
/* When removing these Deletes also change the */
/* DISP of these files to DISP=(OLD) in STEP2OF2 */
DELETE yourhlq, MSP, MSWKMS PURGE
DELETE yourhlq, MSP, MSWKCF PURGE
/* Start of delete when RUNMSP column 12 is Y */
/* When removing these Deletes also change the */
/* DISP of these files to DISP=(OLD) in STEP2OF2 */
DELETE yourhlq, MSP, MSWKNRC PURGE
DELETE yourhlq, MSP, MSNAON PURGE
/* Start of delete when RUNMSP column 14 is Y */
/* When removing these Deletes also change the */
/* DISP of these files to DISP=(OLD) in STEP2OF2 */
DELETE yourhlq, MSP, MSROBAG PURGE
DELETE yourhlq, MSP, MSRTA PURGE
DELETE yourhlq, MSP, MSRTPLT PURGE
DELETE yourhlq, MSP, DAT, MSMDCLR PURGE
DELETE yourhlq, MSP, DAT, MSMDCPT PURGE
DELETE yourhlq, MSP, DAT, MSMDCQFT PURGE
DELETE yourhlq, MSP, DAT, MSMDCSM PURGE
DELETE yourhlq, MSP, DAT, MSMDHDR PURGE
DELETE yourhlq, MSP, DAT, MSMDICR PURGE
DELETE yourhlq, MSP, DAT, MSMDMCR PURGE
DELETE yourhlq, MSP, DAT, MSMDMDR PURGE
DELETE yourhlq, MSP, DAT, MSMDMPA PURGE
DELETE yourhlq, MSP, DAT, MSMDMPU PURGE
DELETE yourhlq, MSP, DAT, MSMDPDR PURGE
DELETE yourhlq, MSP, DAT, MSMDPLR PURGE
DELETE yourhlq, MSP, DAT, MSMDPQT PURGE
DELETE yourhlq.MSP.DAT.MSMDSEG PURGE
DELETE yourhlq.MSP.DAT.MSMDSNR PURGE
DELETE yourhlq.MSP.DAT.MSMDSPR PURGE
DELETE yourhlq.MSP.DAT.MSMDSNR PURGE
DELETE yourhlq.MSP.DAT.MSMDWSR PURGE
DELETE yourhlq.MSP.DAT.MSRDCF PURGE

SET MAXCC = 0

/*
//
//*****************************************************************************/
//*  *
//* This step executes the Streamweaver program: PDRSW000  *
//*  *
//*****************************************************************************/

DEFINE CLUSTER (NAME(yourhlq.MSP.DAT.MSXX00WK) -
    KEYS (14,0) -
    SHR(3 3) -
    REUSE -
    CYL(5 5) -
    VOL(volume) -
    RECORDSIZE(150 3014) -
    CISZ(23704))

/*
//
*******************************************************************************/

STEP2OF3 EXEC PGM=IDCAMS
    //SYSPRINT DD SYSOUT=* 
    //SYSIN DD * 
    DELETE (yourhlq.MSP.DAT.MSXX00WK) CLUSTER PURGE
    SET MAXCC = 0 

STEP3OF3 EXEC PGM=PDRSW000
    //STEPLIB DD DSN=SWhlq.LOAD,DISP=SHR
    // DD DSN=MSPhlq.LOADLIB,DISP=SHR
    //SYSPRINT DD SYSOUT=* 
    //SYSTERM  DD SYSOUT=* 
    //SYSTERM  DD SYSOUT=* 
    //SYSUDUMP DD SYSOUT=* 
    //SYSOUT   DD SYSOUT=* 
    //REPORT   DD SYSOUT=* 
    //TRACECTL DD DUMMY 
    //CONTROL DD DSN=yourhlq.STREAMW.CONTROL(MSPSAMPL),DISP=SHR 
    //INPUT1 DD DSN=yourhlq.STREAMW.INPUT1(MSPSAMPL),DISP=SHR 
    //OUTPUT1 DD DSN=yourhlq.STREAMW.OUTPUT1, 
        // DI SP=(NEW, CAT LG, CAT LG), UNIT=SYSDA, 
        // SPACE=(CYL, (2,2), RLSE), 
        // DCB=(DSORG=PS, RECFM=VB, LRECL=8204) 
    //REJECT DD DSN=yourhlq.STREAMW.REJECTS, 
        // DI SP=(NEW, CAT LG, CAT LG), UNIT=SYSDA, 
        // SPACE=(CYL, (2,2), RLSE), 
        // DCB=(DSORG=PS, RECFM=VB, LRECL=8204) 
    //MSSTIN DD DSN=yourhlq.MSP.MSSTIN, 
        // UNIT=SYSDA, DI SP=(NEW, CAT LG), 
        // SPACE=(CYL, (50,50), RLSE), VOL=SER=volume, 
        // DSORG=PS, RECFM=FB, LRECL=33 
    //******************************************************************************
Advanced sample 7: Postal processing with complex print

// ********************************************************************
// MSRFDC DD DSN=MSPhlq.USPSRFDC,DISP=SHR
// MSRFDI DD DSN=MSPhlq.USPSRFDI,DISP=SHR
// MSRFMP DD DSN=MSPhlq.USPSRFMP,DISP=SHR
// MSRFPS DD DSN=MSPhlq.USPSRFPS,DISP=SHR
// MSRFSQ DD DSN=MSPhlq.USPSRFSQ,DISP=SHR
// MSRFZD DD DSN=MSPhlq.USPSRFZD,DISP=SHR
// MSRFZM DD DSN=MSPhlq.USPSRFZM,DISP=SHR
// ********************************************************************
/* MAILSTREAM PARAMETER FILE */
// ********************************************************************
// MSPRICP DD DSN=yourhlq.MSP.PARMFILE,DISP=SHR
/* ********************************************************************
// MAILSTREAM LICENSE FILE */
// ********************************************************************
// G1LICEN DD DSN=MSPhlq.G1LICEN,DISP=SHR
// ********************************************************************
/* Mailstream Input Test File */
// ********************************************************************
// Generated Files */
// ********************************************************************
// MSNAON DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSNAON,
// DSORG=PS, RECFM=FB, LRECL=450
// MSWKIJ4 DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSWKIJ4,
// DSORG=PS, RECFM=FB, LRECL=400
// MSWKIJ6 DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSWKIJ6,
// DSORG=PS, RECFM=FB, LRECL=600
// MSSTON DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSSTON,
// DSORG=PS, RECFM=FB, LRECL=292
// MSSTOR DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE), VOL=SER=volume,
// DSN=yourhlq.MSP.MSSTOR,
// DSORG=PS, RECFM=FB, LRECL=192
// MSWKCF DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE),
// DSN=yourhlq.MSP.MSWKCF,
// DSORG=PS, RECFM=FB, LRECL=105
// MSWKCM DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE),
// DSN=yourhlq.MSP.MSWKCM,
// DSORG=PS, RECFM=FB, LRECL=170
// MSWKNC DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE),
// DSN=yourhlq.MSP.MSWKNC,
// DSORG=PS, RECFM=FB, LRECL=288
// MSWKMS DD UNIT=SYSDA,DISP=(,CATLG),
// SPACE=(CYL,(50,50),RLSE),
// DSN=yourhlq.MSP.MSWKMS,
Advanced sample 7: Postal processing with complex print

// DSORG=PS, RECFM=FB, LRECL=400
// MSWKSS DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(CYL, (50, 50), RLSE),
// DSN=your hlq. MSP. MSWKSS,
// DSORG=PS, RECFM=FB, LRECL=120
// MSRDBAG DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDBAG,
// RECFM=FB
// MSRDPLT DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDPLT,
// RECFM=FB, LRECL=625
// MSRDTRA DD UNIT=SYSDA, DISP=(, CATLG),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// DSN=your hlq. MSP. MSRDTRA,
// RECFM=FB
//*****************************************************************
//*     Report Files *
//*****************************************************************
// MSRPFAC DD SYSOUT=*, RECFM=FBA
// MSRPPLT DD SYSOUT=*, RECFM=FBA
// MSRPPT DD SYSOUT=*, RECFM=FBA, LRECL=133
// MSRPSTM DD SYSOUT=*, RECFM=FBA
// MSRPPTAG DD SYSOUT=*, RECFM=FBA
// MSRPXLG DD SYSOUT=*, RECFM=FBA, LRECL=133
//*****************************************************************
//*     Alternate Print files *
//*****************************************************************
// AUDITTRL DD SYSOUT=*, RECFM=FBA
// CONFIRM DD SYSOUT=*, RECFM=FBA
// CRRTWALK DD SYSOUT=*, RECFM=FBA
// DETAIL DD SYSOUT=*, RECFM=FBA
// MAILDAT DD SYSOUT=*, RECFM=FBA
// MANIFEST DD SYSOUT=*, RECFM=FBA
// PALSUMM DD SYSOUT=*, RECFM=FBA
// RUNSTATS DD SYSOUT=*, RECFM=FBA
// SUMMARY DD SYSOUT=*, RECFM=FBA
// USPSQUAL DD SYSOUT=*, RECFM=FBA
// USPSSTMT DD SYSOUT=*, RECFM=FBA
// ZONESUMM DD SYSOUT=*, RECFM=FBA
// ZSDETAI DD SYSOUT=*, RECFM=FBA
//*****************************************************************
//*     Mail.dat Files *
//*****************************************************************
// MSMDCLR DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=FB,
// DSN=your hlq. MSP. DAT. MSMDCLR
// MSMDCPT DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=FB,
// DSN=your hlq. MSP. DAT. MSMDCPT
// MSMDCQT DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
// SPACE=(TRK, (100, 150), RLSE), VOL=SER=volume,
// RECFM=FB,
// DSN=your hlq. MSP. DAT. MSMDCQT
// MSMDCSM DD UNIT=SYSDA, DISP=(NEW, CATLG, DELETE),
Advanced sample 7: Postal processing with complex print

```
// SPACE=(TRK,(100,150),RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDCSM
// MSMDHDR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDHDR
// MSMDICR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDICR
// MSMDMCR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDMCR
// MSMDMIR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDMIR
// MSMDMPA DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDMPA
// MSMDMPU DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDMPU
// MSMDPDR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDPDR
// MSMDPLR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDPLR
// MSMDPQT DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDPQT
// MSMDSEG DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDSEG
// MSMDSNR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDSNR
// MSMDSPR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDSPR
// MSMDWSR DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
// DSN=yourhlq.MSP.DAT.MSMDWSR
// MSRDCF DD UNIT=SYSDA, DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(100,150), RLSE), VOL=SER=volume,
// RECFM=VB,
```
Advanced sample 7: Postal processing with complex print

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps
the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=controlfile
```

To run the control file as-is on UNIX, use this script:

```
rm DDCONTROL
rm DD:INPUT1
rm DD:OUTPUT1
rm A07aout1
rm A07aout2
ln -s /share/home1/SunOS/handson/A07cont1.con DDCONTROL
ln -s /share/home1/SunOS/handson/A07inp.t afl DD:INPUT1
ln -s /share/home1/SunOS/handson/A07aout1 DD:OUTPUT1
ln -s /share/home1/SunOS/handson/A07aout2 DD:OUTPUT2
sweaver -c=DDCONTROL
rm DDCONTROL
rm DD:INPUT1
rm DD:OUTPUT1
rm DD:MSNAON
rm DD:REJECT
rm DD:MSSTIN
rm A07bout2
rm A07bout3
rm A07bout4
ln -s /share/home1/SunOS/handson/A07cont2.con DDCONTROL
ln -s /share/home1/SunOS/handson/A07aout1 DD:INPUT1
ln -s /share/home1/SunOS/handson/A07aout2 DD:OUTPUT1
ln -s /share/home1/SunOS/handson/A07bout3 DD:OUTPUT2
ln -s /share/home1/SunOS/handson/A07bout4 DD:REJECT
sweaver -c=DDCONTROL
```

Windows

To run the sample application on a Windows system you must modify the `<FILE>` tags in the control file to specify file paths using Windows conventions.

For example, you could change this:

```
<FILE> DD:INPUT1
```

To this:

```
<FILE> C:samples\input1.lin
```

**Note:** For information on the appropriate file extensions to use when specifying print streams, see the *Enrichment Language Reference*.

Then, run the application by issuing the following:

```
sweaver -c=controlfile
```
Advanced sample 8: Preparing a print stream for AFP N-Up processing

In this section

Overview........................................................................................................ 208
Files Used................................................................................................... 208
Business Scenario....................................................................................... 209
Explanation of Sample............................................................................... 209
Overview

This sample creates 2-up simplex output (two logical pages placed horizontally on each physical page) from 1-up simplex input. It adds a Code 3of9 barcode containing the document number and page number framed with asterisks to the outside margin of each page. This sample introduces the following topics:

- FORMDEF and N-up
- Mup tag group
  - <MGRID> tag
  - <MPF> tag
  - <MSIZE> tag
- Input tag group
  - <PAGESIZE> tag
  - How to find fields
  - Add tag group
    - <POSINDEX> tag
    - <POSMULTUP> tag

Files Used

This sample uses the following files.

Table 1: Files Used in This Sample

<table>
<thead>
<tr>
<th>Files Used</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
<td>D:\apps\adv8\a08inpt.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.INPUT(A08INPT)</td>
</tr>
<tr>
<td>Rule File</td>
<td>N/A</td>
</tr>
<tr>
<td>Other Inputs</td>
<td>N/A</td>
</tr>
<tr>
<td>Control File</td>
<td>D:\apps\adv8\a08cont.con</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.CONTROL(A08CONT)</td>
</tr>
<tr>
<td></td>
<td>A08CONT.ws.CON</td>
</tr>
<tr>
<td>Output Print Stream</td>
<td>D:\apps\adv8\a08outp.afl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.GOOGOUT1(A08OUTP)</td>
</tr>
<tr>
<td>Other Output</td>
<td>N/A</td>
</tr>
<tr>
<td>Run Time</td>
<td>JCL:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv8\a08jcl.jcl</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.JCL(A08JCL)</td>
</tr>
<tr>
<td></td>
<td>UNIX shell script:</td>
</tr>
<tr>
<td></td>
<td>D:\apps\adv8\a08scr.sh</td>
</tr>
<tr>
<td></td>
<td>PDR.STREAMW.HANDSON.jcl(A08SCR)</td>
</tr>
</tbody>
</table>
Business Scenario

This sample application prints the multi-page documents in the print stream 2-up (two logical pages on every physical page). The FORMDEF will place the pages on the left and right sides of the paper. The application takes into account the fact that the carriage controls in the print stream need to match what is in the FORMDEF.

In the sample, above:

- The inserter equipment requires a 3of9 barcode on each logical page. The barcode for the left page goes in the left margin and the barcode for the right page goes on the right margin. The barcode consists of the document and pages numbers with framing characters (*).

Explanation of Sample

The control file reads an input print stream that contains several multipage documents. Enrichment writes the documents 2-up simplex to the output file and adds Code 3of9 barcodes to the left margin of left-hand pages and the right margin of right-hand pages.

Note: This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A08CONT.ws.CON contains these modifications.
Advanced sample 8: Preparing a print stream for AFP N-Up processing

Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `INPUT`, and the input file is found on the mainframe system at `DD:INPUT1`, as indicated by the Input group `<FILE>` tag. The input is AFP line data with ANSI carriage controls, so we set the `<TYPE>` tag to `AFPL A`.

We used the `<PAGESIZE>` tag to specify the printable area and physical page dimensions for the input. Enrichment will use these dimensions to calculate placement when it produces the 2-up output. The `<PAGESIZE>` tag indicates that the printable area and physical size of each page is 3540 pels wide by 2640 pels long (14.75 inches by 11 inches).

**Note:** If we were adding the barcode to the same relative place on each logical page, we could have specified one `<POSITION>` tag instead of two `<POSMULTUP>` tags.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```
<input>  <! Begin input tag group. >
  <name> Input                        <! File name.  >
  <file> DD:INPUT1  <! DD name in JCL.  >
  <type> AFPL A             <! AFP linedata w/Ansi control >
  <doc> T %%DOC_Page1 EXIST  <! Each "Page 1 " is a doc. >
  <pagesize> 3540 2640 3540 2640 PELS <! Physical page size. >
  <field> %%DOC_Page1 KA    <! This field used to denote a >
    <ref> ' ' 'Page 1 ' 67 <! Each "Page 1 " exists in position 67, a new >
    <loc> 0 -6 7 <! a document exists. >
  </field>
</input>

<output> <! Begin output tag group. >
  <name> Output       <! Name of file.      >
  <file> DD:OUTPUT1   <! DD name in JCL.  >
  <mup>               <! Multiple-up output.  >
    <mpf> M    <! Multiple mapped page format >
    <mgrid> 2 1 R T <! Logical pages are placed 2 >
      <! across and 1 down on the >
      <! physical page. Logical >
      <! pages flow across the >
      <! physical page. There is >
      <! carriage control and TRC.>
    <mcc> 1 1 1 1 <! The base page is row 1 col 1>  
    <mcc> 1 2 1 6 <! and the carriage control >
      <! is 1, the carriage >
      <! control for the right >
      <! side is 6. >
    <msize> 1 1770 2640 PELS <! Size of logical page. >
  </mup>
  <add>               <! Add tag group for barcode. >
    <addtype> T <! Type is text. >
</output>
```
Field Tag Group

Some invoices may contain several pages, so the `<FIELD>` tag identifies a variable called `%%DOC_Page1` whose value is the string 'Page 1'. We only want to extract the field information, so we set the `<FIELD>` tag action parameter to KA. The `<REFERENCE>` tag instructs Enrichment to set a reference point for the field on all records that have a blank carriage control and contain the string 'Page 1' in column 67. We added an extra space after Page 1 so Enrichment doesn't inadvertently use a similar string (such as Page 11) as the reference point.

The `<LOCATION>` tag indicates that the field information occupies the same record as the reference point, begins 6 columns before the reference point, and is 7 characters in length. Thus, the value of `%%DOC_Page1` is always Page 1 followed by a space.

Output Tag Group

We defined one Output group in the control file. In it, we set the `<NAME>` tag to Output, which resides on the mainframe system at DD: OUTPUT as indicated by the `<FILE>` tag. The Output group also contains a Mup tag group that defines the parameter used to print the documents 2-up and an Add tag group that defines the Code 3of9 barcode to be applied to each page.

Mup Tag Group

In the Mup tag group, we specified `<MPF>M` to indicate that Enrichment will use multiple AFP mapped page formats and copy group names to control placement of the 2-up pages. Mapped means that carriage controls are defined and used to position the text on the physical page.

The `<MGRID>` tag specifies that each physical page will contain two logical pages arranged so that logical page 2 follows logical page 1 horizontally on the physical page. The line data contains both carriage controls and TRCs.
We need to be sure the carriage controls in the print stream match how the FORMDEF has defined 2-up placement, so we used two <MCC> tags to change the carriage controls from those in the base page (that is, the left-hand page in the output) to those in the 2-up logical page. The first <MCC> tag indicates that, for the logical page on the left-hand side of the physical page, Enrichment is to leave carriage control 1 alone. The second <MCC> tag indicates that, for the logical page on the right-hand side of the physical page, Enrichment is to change carriage control 1 to carriage control 6.

Finally, we used the <MSIZE> tag to specify the size of one logical page on a 2-up physical page. We set the \texttt{width} and \texttt{maximum} parameters to 1 simply because a value is required. These parameters have no meaning for this application since we specified \texttt{MPF=M}. Enrichment ignores the width and maximum values in this case. We specified that each logical page will be 1770 pels wide by 2640 pels long (7.375 inches by 11 inches).

Add Tag Group

We set up an Add tag group in the Output group to position the Code 3of9 barcodes on the documents. To add font-based Code 3of9 barcodes to the documents in the output, we set the <ADDTYPE> tag to \texttt{T}. We set four <ADDPART> tags as follows to construct the Code 3of9 barcode for each page:

- The first and last <ADDPART> tags specify the asterisks (*) to be used as framing characters for the barcodes.
- The second <ADDPART> tag specifies that Enrichment will include the system variable \texttt{%%DOCUMENT\_NO} in the barcode. The right-justified value will be padded with zeroes if it is less than 5 characters in length.
- The third <ADDPART> tag specifies that Enrichment will include the value of the system variable \texttt{%%PAGE\_NO} in the barcode. The right-justified value will be padded with zeroes if it is less than 5 characters in length.

Since we want the Code 3of9 barcode to print so its top edge is parallel to the right edge of the paper, we specify \texttt{ORI\_ENT=2}.

Since adding a font-based barcode, we must specify the font to use. Since the output will be an AFP print stream, we used the <FONT> tag, setting it to \texttt{5}, which corresponds to a font defined in the FORMDEF.

Finally, we specified two <POS\_MULT\_UP> tags to specify the positions of the barcodes to add. The first <POS\_MULT\_UP> tag specifies the barcode position for the left-hand logical page on each physical page. It instructs Enrichment to place the barcode so its top left corner is 30 pels (.125 inches) to the right of the upper left corner of the physical page and 140 pels (approximately .583 inches) below the upper left corner of the physical page. The second <POS\_MULT\_UP> tag specifies the barcode position for the right-hand logical page on each physical page. It instructs Enrichment to place the barcode so its top left corner is 3420 pels (14.25 inches) to the right of the upper left
corner of the physical page and 140 pels (approximately .583 inches) below the upper left corner of the physical page.

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```jcl
//*JOB CARD
//*OBLIB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
//* DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
//* DD DSN=SYS3.PLI230.SIBMLINK,DISP=SHR
//* DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//*[----------------------------------------------------------------
//* Enrichment PROCESS
//SW EXEC PGM=PDRSW000,REGION=0M
//INPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.INPUT(A08INPT)
//OUTPUT1 DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A08OUT1)
//REPORT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//SYSUDUMP DD SYSOUT=* 
//CONTROL DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.CONTROL(A08CONT) 
//*[----------------------------------------------------------------
//COMP01 EXEC PGM=ISRSUPC,PARM=(DELTAL,LINECMP,'','')
//NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A08OUT1),DISP=SHR
//OLDDD DD DSN=PDR.STREAMW.HANDSON.GOODOUT1(A08OUT1),DISP=SHR
//OUTDD DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.NEWOUT1(A08OUT1),DISP=SHR
//=---------------------------------------------------------------
```

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the `<FILE>` tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:

```
sweaver -c=<controlfile>
```

To run the control file as-is on UNIX, use this script:
Advanced sample 8: Preparing a print stream for AFP N-Up processing

```
rm DDCONTROL
rm DD:INPUT1
rm DD:OUTPUT1
rm A08out1
ln -s /share/home1/SunOS/handson/a08cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/a08inpt.afl DD:INPUT1
ln -s /share/home1/SunOS/handson/A08out1 DD:OUTPUT1
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=a08cont.ws.CON
```
Advanced sample 9: Using a user-written function to reverse a string

In this section

Overview........................................................................................................ 216
Files Used..................................................................................................... 216
Business Scenario.......................................................................................... 216
Explanation of Sample.................................................................................. 217
Overview

Extract a text string from each document and call a COBOL program to reverse the characters in the string. The text string may be several blank-delimited words. The COBOL program should reverse the characters in each word and the words themselves should be reversed. This sample introduces the topic of user-written functions.

Files Used

This sample application uses the following files.

**Note:** This sample application will only run on a mainframe.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Business Scenario

This sample application shows how to use a COBOL program called from Enrichment to reverse the characters in each word of a specified string, and reverse the words. For example, “Dick Wells” would become “slleW kciD”.

EngageOne Enrichment 7.3.0 Sample Applications Guide 216
Explanation of Sample

If we are going to run Enrichment on a mainframe, we must write, compile, and link-edit the COBOL program before we can construct the actual Enrichment application. The load library that contains the finished function’s load module is specified in the J OBLIB statement. The following shows the COBOL code for the user-written function.

This is the JCL for this application:

```cobol
IDENTIFICATION DIVISION. 00010000
*****************************************************************
* File: NORMCOB - COBOL II *
* System: Enrichment *
* Copyright 2007 Group 1 Software, Inc. *
*---------------------------------------------------------------*
* Purpose: Sample COBOL II User Function for 'Normal' type Enrichment UFAPI interface. *
* This example reverses the input string. *
*****************************************************************
PROGRAM-ID. NORMCOB. 00030000
* 00040000
*****************************************************************
00090000
ENVIRONMENT DIVISION. 00110000
* 00130000
*****************************************************************
00090000
DATA DIVISION. 00140000
*----------------------------------------------------------------
00150000
WORKING-STORAGE SECTION. 00160000
01 IDX  PIC 999 COMP.
01 IDX2 PIC 999 COMP.
* 00170000
*****************************************************************
00180000
* LINKAGE SECTION 00190000
*---------------------------------------------------------------*
00210000
* Declarations for Input/Output call areas to pass data between the Enrichment Rule file and this function. 00220000
*****************************************************************
00240000
LINKAGE SECTION. 00250000
* Input Call Area 00260000
01 INPUT-CALL-AREA. 00260002
* --- Required fields:
  05 IN-SIGNATURE PIC X(4). 00270000
  05 CALL-TYPE PIC X(1). 00280000
  05 CALL-FROM PIC X(1). 00290000
  05 FILLER PIC X(2). 00300000
  05 IN-RC PIC S9(9) COMP. 00310000
  05 IN-RV PIC S9(9) COMP. 00320000
  05 FILLER PIC X(20). 00330000
  05 IN-SIZE PIC S9(9) COMP. 00340000
* --- User defined PIC X fields from Rule file arguments:
```
Advanced sample 9: Using a user-written function to reverse a string

```
05 IN-DATA PIC X(40).
*--- Output Call Area -------------------------------
01 OUTPUT-CALL-AREA.
*--- Required fields:
05 OUT-SIGNATURE PIC X(4).
05 OUT-RC PIC S9(9) COMP.
05 OUT-RV PIC S9(9) COMP.
05 FILLER PIC X(24).
05 OUT-SIZE PIC S9(9) COMP.
*--- User defined PIC X fields for combined results:
05 OUT-DATA PIC X(40).
*****************************************************************

PROCEDURE DIVISION USING INPUT-CALL-AREA OUTPUT-CALL-AREA.
FUNCTION-START.
--- Check Input and Output signatures ---------------------
IF IN-SIGNATURE NOT EQUAL 'PDRI' THEN
   MOVE -3 TO OUT-RC
   GO TO FUNCTION-END
END-IF.
IF OUT-SIGNATURE NOT EQUAL 'PDRO' THEN
   MOVE -3 TO OUT-RC
   GO TO FUNCTION-END
END-IF.

FUNCTION-START.
--- Perform User function HERE: ---------------------------
--- Print some stuff
--- Reverse string
INITIALIZE OUT-DATA, IDX2.
PERFORM VARYING IDX FROM IN-SIZE BY –1
   ADD 1 TO IDX2
   MOVE IN-DATA (IDX:1) TO OUT-DATA (IDX2:1)
END-PERFORM.

FUNCTION-START.
--- Store return value, size and set return code & value --
MOVE IN-SIZE TO OUT-SIZE.
MOVE 0 TO OUT-RC.
MOVE 0 TO OUT-RV.
FUNCTION-END.
EXIT.
```

This is the C code for the user written function:

```
/* File : normc.c *
* System : Enrichment *
* Version: 32 *
* Copyright (c) 1993-1998 PDR Advanced Technology. */

05 IN-DATA PIC X(40).
*--- Output Call Area -------------------------------
01 OUTPUT-CALL-AREA.
*--- Required fields:
05 OUT-SIGNATURE PIC X(4).
05 OUT-RC PIC S9(9) COMP.
05 OUT-RV PIC S9(9) COMP.
05 FILLER PIC X(24).
05 OUT-SIZE PIC S9(9) COMP.
*--- User defined PIC X fields for combined results:
05 OUT-DATA PIC X(40).
*****************************************************************

PROCEDURE DIVISION USING INPUT-CALL-AREA OUTPUT-CALL-AREA.
FUNCTION-START.
--- Check Input and Output signatures ---------------------
IF IN-SIGNATURE NOT EQUAL 'PDRI' THEN
   MOVE -3 TO OUT-RC
   GO TO FUNCTION-END
END-IF.
IF OUT-SIGNATURE NOT EQUAL 'PDRO' THEN
   MOVE -3 TO OUT-RC
   GO TO FUNCTION-END
END-IF.

FUNCTION-START.
--- Perform User function HERE: ---------------------------
--- Print some stuff
--- Reverse string
INITIALIZE OUT-DATA, IDX2.
PERFORM VARYING IDX FROM IN-SIZE BY –1
   ADD 1 TO IDX2
   MOVE IN-DATA (IDX:1) TO OUT-DATA (IDX2:1)
END-PERFORM.

FUNCTION-START.
--- Store return value, size and set return code & value --
MOVE IN-SIZE TO OUT-SIZE.
MOVE 0 TO OUT-RC.
MOVE 0 TO OUT-RV.
FUNCTION-END.
EXIT.
```

This is the C code for the user written function:

```
/* File : normc.c *
* System : Enrichment *
* Version: 32 *
* Copyright (c) 1993-1998 PDR Advanced Technology. */

```
/* all rights reserved. Unauthorized use or duplication prohibited. */
/* ------------------------------------------------------------- */
/* Purpose: */
/* ------------------------------------------------------------- */
/* History: */
/* static char SccsID[] = @(#)norm.c 1.1 08/18/97 */
/* 07/09/98 rolson Initial 50 */
/* 08/18/97 tcampbel Userfunction for testcases */
/* ------------------------------------------------------------- */
/*
* norm.c - Example Normal C User Function
* Reverse characters.
*/
/* ------------------------------------------------------------- */
#include <stdio.h>
/* --- UFAPI Input Call Area ---------- */
typedef struct {
  char pSig[4];    /* Signature 'PDRI' */
  char cCallType;  /* Type - Init, Norm, or Term */
  char cCallFrom;  /* Called from - R/P rule/pagerule */
  char pSave1[2];  /* (future) */
  int iInRC;       /* Initial RC */
  int iInRV;       /* Initial RV */
  char pSave2[20]; /* (future) */
  int iInSize;     /* Size of Input data */
  char pInData[2]; /* Input data (blank padded) */
} UFIN, *PUFIN;
/* --- UFAPI Output Call Area --------- */
typedef struct {
  char pSig[4];    /* Signature 'PDRO' */
  int iOutRC;      /* Return RC */
  int iOutRV;      /* Return RV */
  char pSave1[24]; /* (future) */
  int iOutSize;    /* Size of Output data */
  char pOutData[2]; /* Output data (blank padded) */
} UFOUT, *PUFOUT;
/* -- User Function subroutine to be called -- */
long int normc(PUFIN pUFIN, PUFOUT pUFOUT) {
  long int i;
  /* Error exit if signatures not found */
  if (memcmp(pUFIN->pSig, "PDRI", 4)) return(3);
  if (memcmp(pUFOUT->pSig, "PDRO", 4)) return(3);
  pUFOUT->iOutRC = pUFOUT->iOutRV = 0; /* Clear -99s from RC & RV */
  for (i=pUFOUT->iOutSize=0; i<pUFIN->iInSize; i++)
    pUFOUT->pOutData[pUFOUT->iOutSize] = pUFIN->pInData[i];
}
Advanced sample 9: Using a user-written function to reverse a string

```c
return (0);    /* RC=0 -> all ok */
} /* NORMC */
```

The control file we created for the application (shown below) reads an input print stream that contains several one-page documents. Enrichment extracts a field from each document and calls the user-written function NORMCOB to reverse the contents of the string before writing the documents to output.

**Note:** This control file is designed for running Enrichment on a mainframe system. You would need modify this control file to run on UNIX or Windows. For example, you would need to use the appropriate file specification in the `<FILE>` tag for your OS. A09CONT.win.CON contains these modifications.

**Note:** An existing built-in Enrichment function (REVERSE) performs the same function as the user-written function NORMCOB more efficiently. Advanced Sample 9 is just an example of Enrichment calling a user-written function.

### Input Tag Group

As the control file shows, the Input group is identified in the `<NAME>` tag as `InputFile`, and the input file is found on the mainframe system at `DD:INPUT1`, as indicated by the Input group `<FILE>` tag. The input is impact data with ANSI carriage controls, so we set the `<TYPE>` tag to `I A`. We set the `<DOCUMENT>` tag to `1`, indicating that each document in the input is one page in length.

**Note:** You can use Enrichment Visual Engineer’s Edit Assistant to get more information on each tag in this control file.

```xml
<input>
  <name> InputFile </name>
  <file> DD:INPUT1 </file>
  <type> I A </type>
  <document> 1 </document>
  <field> %%Whole_Name R </field>
  <location> 1 2 18 </location>
</input>
```

```xml
<rule>
  <content>
  START:
    UserFunction REV_COB NORMCOB COBOL N 60 60 DOCUMENT:
    %%Whole_Name = REV_COB(%%Whole_Name)
  </content>
</rule>
```
Advanced sample 9: Using a user-written function to reverse a string

Field Tag Group

We defined one field, which Enrichment will use to extract the information to be reversed. The `<FIELD>` tag identifies a variable called `%%Whole_Name`. We want to replace the field information, so we set the `<FIELD>` tag action parameter to `R`. The `<LOCATION>` tag indicates that the field information occupies the first record of each document and begins in column `2` for `18` characters.

Rules

START: Section

In the START: section of the rules, we identify the user-written function to Enrichment using the `USERFUNCTION` command. We specified `REV_COB` as the name (for this application) of the load module `NORMCOB`. We also specified that the user-written function is a normal COBOL function with maximum input and output lengths of `60`.

Note: There is a linking step that is only required when running a COBOL user-written function with SAS/C.

DOCUMENT: Section

In the DOCUMENT: section of the rules, we used `REV_COB` to process the variable `%%Whole_Name`, returning the reversed strings as the new values of `%%Whole_Name`. We didn’t have to use the same variable as input and output, but we saved memory usage by doing so. For example, if the input value of `%%Whole_Name` was `Bob Wobbly`, the `REV_COB` function set the new value of `%%Whole_Name` as `ylbboWbobB`.

The user-written function should also set the system variables `%%RC` and `%%RV` to the appropriate return code and additional return value. `%%RC` and `%%RV` are the same system variables that we saw with built-in functions. The Enrichment Report tells what these variables were set to when we ran the application.
Output Tag Group

We defined one Output group in the control file. In it, we set the <NAME> tag to OutputFile, which resides on the mainframe system at DD:OUTPUT1 as indicated by the <FILE> tag.

Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

Mainframe

The following shows the JCL we used to run the application.

```jcl
//*JOBCCARD
//*OBLLIB DD DSN=PDR.STREAMW.LOADCRUN,DISP=SHR
//* DD DSN=SYS3.CLIB22.SEDCLINK,DISP=SHR
//* DD DSN=SYS3.PLL1230.SIBMLINK,DISP=SHR
//* DD DSN=SYS3.V4R1M0.ISPLOAD,DISP=SHR
//*------------------------------------------------------------------
//* Enrichment PROCESS
// SW EXEC PGM=PDRSW000,REGION=0M
//INPUT1 DD DSN=PDR.STREAMW.HANDSON.INPUT(A09INPT),DISP=SHR
//OUTPUT1 DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A09OUT1),DISP=SHR
//REPORT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSTERM DD SYSOUT=* 
//SYSUDUMP DD SYSOUT=* 
//CONTROL DD DISP=SHR,DSN=PDR.STREAMW.HANDSON.CONTROL(A09CONT) 
//* -------------------------------------------------------------
//COMP01 EXEC PGM=ISRSPFC,PARM=(DELTAL,LINECMP,'','')
//NEWDD DD DSN=PDR.STREAMW.HANDSON.NEWOUT1(A09OUT1),DISP=SHR
//OLDDR DD DSN=PDR.STREAMW.HANDSON.GO002OUT(A09OUT1),DISP=SHR
//OUTDD DD SYSOUT=* 
//* -------------------------------------------------------------
```

UNIX

To run the application on a UNIX system, you may use the shell script shown below. Since the sample control file is written for a mainframe environment, the script maps the mainframe data set specifications (DD:xxxxxx) to UNIX file names. Instead of using this script, you could modify the <FILE> tags in the control file to specify file paths using UNIX conventions. Then, to run the application all you would need to do is issue the command:
Advanced sample 9: Using a user-written function to reverse a string

`sweaver -c=<control file>`

To run the control file as-is on UNIX, use this script:

```sh
r m DDCONTROL
r m DD:INPUT1
r m DD:OUTPUT1
r m A09out1
ln -s /share/home1/SunOS/handson/a09cont.con DDCONTROL
ln -s /share/home1/SunOS/handson/a09inpt.lin DD:INPUT1
ln -s /share/home1/SunOS/handson/A09out1 DD:OUTPUT1
sweaver -c=DDCONTROL
```

Windows

Run the application by issuing the following command:

```
sweaver -c=A09CONT.win.CON
```
Advanced sample 9: Using a user-written function to reverse a string
Advanced sample 10: Accessing data in a relational database

In this section

Overview .................................................................................................................. 226
Files Used .............................................................................................................. 226
Business Scenario ............................................................................................... 227
Explanation of Sample ......................................................................................... 227
Overview

To help explain the ODBC functionality, this sample separates documents into two output categories based on customer accounts contained in a relational database. It uses the account number from the document as a key for locating customers that have signed up for online bill presentment instead of receiving their printed statement in the mail. At the end of the job the application will send collected job processing information to a different database table where the data can be used for future job performance reporting and statistical analysis.

This sample introduces the following concepts to the control and rule file:

- Table Definition using the <RDB>, <SQL>, and <DELIMIT> tags.
- LOOKUP function accessing the results from the SQL query to the database.
- LOOKUP function using a delimiter to enumerate a column for LOOKUP matching.
- SQL function to INSERT records into a relational database.

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Print Stream</strong></td>
</tr>
<tr>
<td><strong>Rule File</strong></td>
</tr>
<tr>
<td><strong>Other Inputs</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Control File</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Output Print Stream</strong></td>
</tr>
<tr>
<td><strong>Other Output</strong></td>
</tr>
<tr>
<td><strong>Run Time</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Business Scenario

This sample application shows how to query a database from within an Enrichment application and use the result set to direct certain accounts to online billing and others to printed billing output files. Additionally, a second database table will be updated with processing statistics at the end of the Enrichment application run. The data collected can later be used for audit and accounting information.

Businesses are faced with providing customer communications for a variety of output channels that reflect real-time activity. The information to determine if a customer receives a printed or electronic statement is typically kept in a relational database that is connected to a web-based enrollment system. Businesses must ensure that a customer receives their communication on their preferred delivery channel. Duplicate statements received can cause confusion and result in extra customer service expense.

The stage is set by defining a typical production print cycle schedule. Print data is extracted nightly at 6:00 PM. As part of each run, Enrichment must query the database to determine if any of the account numbers in the current print data have registered to receive their statements online rather than in print. A table named online_enrollment contains a list of customer accounts that prefer online statements.

The business also requires detailed audit and accounting information for each Enrichment job. At the end of the print job you are required to update a table named job_log with processing statistics.

In the sample, above:

- Extract Account Number and check database to see whether the customer has chosen an online statement.

Explanation of Sample

The control file that was created for the application (shown below) reads a print stream and uses information from a relational database to identify customer billing channel preference: print or electronic. Documents are written to a separate output file based on the information returned from the database. Processing statistics are gathered during the job and sent to a different table in the database.
**Note:** This control file is designed for running Enrichment on the Windows PC platform. You would need to modify this control file to run on UNIX or a mainframe system.

In order to facilitate a connection to the database, the ODBC Data Source Name (DSN) has to be defined to the operating system. A successful test of this connection must be established before the Enrichment application can be written.

The instructions for setting up a USER or SYSTEM ODBC DSN source are different based on the operating systems running on the client machine as well as the database server. Please refer to your OS provider’s documentation on defining ODBC DSN’s for specific information pertaining to your environment.

The dialog box shown below is an example for Windows 7. You may have to download the Remote Server Administration Tools for Windows 7 in order to define a System DSN. Please discuss this step with your Systems and/or Network Administrator if you plan on using a database located on a remote server. Firewall restrictions may prevent you from connecting to a remote database.

For this example you can define a USER DSN ODBC source that will be located using a database on your own system.
Tag or function

For connecting to a relational database you can choose between using tags or a function:

- If the query to the relational database is static, the best implementation is by using a tag.
- If the queried information is based on environmental conditions in the print job (current time/date/job information) use the functions.

Using tags

Tag processing in Enrichment consumes fewer resources than a function. Tags that support the database related in the <TABLE> tag group are:

- <RDB>
- <DELIMIT>
- <SQL>
- <SQLOUTFILE>

The <RDB> tag defines the ODBC Data Source Name to which Enrichment will connect. There are 2 parameters for this tag: User ID and Password. Supply these parameters if your data source requires them. Otherwise, specify NULL values by using sets of double quotation marks "" with no data between them.

The <DELIMIT> tag is used to indicate what character Enrichment will use to delimit the returned column data from the database. This tag requires one parameter which is the actual delimiter character (e.g. ; or #).

The <SQL> tag contains the SELECT statement to query the database with selection criteria you specify. Use this tag whenever the query remains the same every time you run this job. If you expect the query to change each time, you need to use the Enrichment RSQLTABLE function instead.

The <SQLOUTFILE> tag is useful when you are in the initial stages of developing your Enrichment application. This tag writes the results from the <SQL> tag into a plain text file that can be utilized to determine if the application is retrieving the expected data. A single parameter containing the filename is the only requirement for this tag.

Using functions

The only function applicable to this sample is the SQL function in the FINISH: section of the <RULE> which sends the processing of statistics to a database table. For more information on other SQL-based functions that are available, please refer to the Enrichment Language Reference guide.
Advanced sample 10: Accessing data in a relational database

Input Tag Group

As the control file shows, the Input group was identified in the <NAME> tag as INPUT1, and the input file resides in the D:\apps\adv10\ directory with a filename: a10inpt.lin. The input is line data with ANSI (ASCII) carriage controls so set the <TYPE> tag to I AA.

Field Tag Group

Four field tag groups were defined in the application, however just one field (%%AccountNumber) was necessary to perform the database search. The name and address block fields were defined for visual assistance in understanding the application.

The application consisted of simple mailing labels; therefore the <DOC> tag is set to 1, meaning there is 1 page per document. The application used the %%AccountNumber variable to determine if a customer had enrolled to receive online statements.

Table Tag Group

The success of the application depends on the TABLE tag group. In this tag the NAME of the table is defined, the RDB connection information to the database, a static SQL statement and the DELIMIT character used. There is an optional <SQLOUTFILE> tag in this group that can be used for debugging purposes.

In the code sample shown below, the following was performed:

• The <INPUT> tag group contains a field called %%AccountNumber (among other fields).
• A <TABLE> named ACTLKU1 was defined with:
  • An <RDB> tag that identified the data source, ID, and password.
  • A <SQL> tag that contained the SQL query to be sent to the database.
  • A <DELIMIT> tag that specified the delimiter character.
• A LOOKUP function was used to read the extracted database information loaded into the <TABLE> ACTLKU1.
• Logic in the <RULE> to issue the <OUTPUT> command for routing the statements to the appropriate output file.

Note: The samples include both MS Access and MySQL code samples. The following example uses the sample MySQL database.

<! ------------------------------------------------------------------>

EngageOne Enrichment 7.3.0  Sample Applications Guide  230
Advanced sample 10: Accessing data in a relational database

<! Standard Line data Input file >
<!  ------------------------------------------------------------------>

<INPUT> <! Standard input tag group >
  <Name> INPUT1
  <type> 1 AA
  <document> 1
  <file> 'D:\Samples\a10inpt.lin'

  <field> %%CustomerName KA <! Customer Name field >
    <REF> '1' '1' 1
    <location> 1R 2R 30 L
  </FIELD>

  <field> %%Address1 KA <! Address Line 1 >
    <REF> '1' '1' 1
    <location> 2R 2R 30 L
  </field>

  <field> %%Address2 KA <! Address Line 2 >
    <REF> '1' '1' 1
    <location> 3R 2R 30 L
  </field>

  <FIELD>%%AccountNumber <! Account Number >
    <REF> '1' '1' 1
    <location> 3R 40R 6 L
  </field>

</INPUT>

<!  ------------------------------------------------------------------>

<! Table tag group connecting to MySQL database >
<!  ------------------------------------------------------------------>

<table>
  <name>ACTLKU1 <! Table Name for LOOKUP >
  <rdb> SWSAMP <! SYSTEM DSN for database >
  <sql> "SELECT * from online_statement.online_enrollment;";
  <delimit> ; <! delimiter for data ret. >
  <sqloutfile> 'E:\Samples\sqlresults.txt' <! the following can be used to check the results of your SQL query >
</table>

<rule>
  <content>
    start:
    
    %%DocTotal = 0 <! Set Doc & Page counters >
    %%PageTotal = 0
  
</rule>

<!  ------------------------------------------------------------------------->

<! Set Environmental Job Processing variables >
<!  ------------------------------------------------------------------------->

EngageOne Enrichment 7.3.0 Sample Applications Guide 231
\%
\%jobName = "CLET041"     <! Set Job Name for Logs >
\%jobId = "00052981"   <! Set Job ID   >
\%jobStartDate = DATE(I, N)   <! Format Start Date  >
\%jobStartDate = SUBSTR(\%jobStartDate,1,4) | ":" | SUBSTR(\%jobStartDate,5,2) | ":" | SUBSTR(\%jobStartDate,7,2)
\%jobStartTime = TIME(M, Y, B)     <! Job Start Time >

document:
<! ------------------------------------------------------------------ >
<! Perform a lookup against the data from the database >
<! ------------------------------------------------------------------ >

\%ReturnValue = LOOKUP(TABLE:ACTLKU1, \%AccountNumber, 1, ";", N)
\%LKRC = \%RC

if \%LKRC = 0 then
  <! online output file or to >
  <OUTPUT> ONLINE
  <! customer account number >
else
  <OUTPUT> PRINTED
end if

\%DocTotal = \%DocTotal + 1    <! Increment counters >
\%PageTotal = \%PageTotal + 1

finish:
<! ------------------------------------------------------------------ >
<! Collect Job Process infor and update the job log database >
<! ------------------------------------------------------------------ >

\%jobEndDate = DATE(I, N)
\%jobEndDate = SUBSTR(\%jobEndDate,1,4) | ":" | SUBSTR(\%jobEndDate,5,2) | ":" | SUBSTR(\%jobEndDate,7,2)
\%jobEndTime = TIME(M, Y, C)     <! Get Current time >

// Sample INSERT syntax for MySql.
Advanced sample 10: Accessing data in a relational database

// INSERT INTO online_statement.job_log VALUES
(NULL,'CLET030',00052971,
'2012-02-23','08:10:54','2012-02-23','08:10:54',6,6);

%%SQLString = "INSERT INTO online_statement.job_log VALUES (NULL,"
| %%JobName | ", " | %%JobId | ", " | %%JobStartDate |
| " | " | " | " |
| %%JobStartTime | ", " | %%JobEndDate |
| " | " | " | " |
| %%JobEndTime | ", " | %%DocTotal |
| " | " | %%PageTotal | ");"

<!-- Issue the SQL function ->
<!-- and INSERT job log info >

%%SQRet = SQL("SWSAMP","",",", %SQLString)
%%SQLRM = %%RM

// The following WRITE functions are used for debugging and should be
deprecated or commented out
// when the application runs successfully.
 WRITE('E:\Samples\inssql.txt', %SQLString)
 WRITE('E:\Samples\inssql.txt', %SQLRM)

</content>
</rule>

<!-- ----------------------------- >
<!-- The OUTPUT file for ONLINE statements
----------------------------- >

<output>
 <name>ONLINE
 <file>'E:\Samples\a10onl.lin'</file>
</output>

<!-- ----------------------------- >
<!-- The OUTPUT file for PRINTED statements
----------------------------- >

<output>
 <name>PRINTED
 <file>'E:\Samples\a10prt.lin'</file>
</output>

Rules

START: Section

In the START: section of the rules, the following was performed:
Advanced sample 10: Accessing data in a relational database

- %%DocTotal and %%PageTotal variables were set up for audit use at the end of the job.
- A unique JOB ID number was created.
- DATE( ) and TIME( ) functions were used to set the "start time and date" of the job.

The DATE was not formatted in the proper format expected in the database. The SUBSTR( ) function was used to extract the DATE components needed, and separate them with a "-" dash. The proper formatted date looks like this: 2012-04-19.

**DOCUMENT: Section**

In the DOCUMENT: section of the rules, the following was performed:

- The LOOKUP function was used to retrieve information from the <TABLE> tag group.
- The %%RC variable from LOOKUP was checked to determine if data was retrieved.
- Documents were routed to <OUTPUT> ONLINE or PRINTED depending on if the account number was found.
- The %%DocTotal and %%PageTotal counters were incremented.

**FINISH: Section**

In the FINISH: section of the rules, the following was performed:

- The DATE( ) and TIME( ) functions were used to set the "end time and date" of the job. The DATE function could be eliminated if you knew the job was not going to span into a new day, however you still can include it.
- A %%SQLString variable was built to contain all end of job processing information.
- The SQL function was invoked to connect to the table and INSERT the data into the table.

The SQL syntax in this example has been tested using the MySQL database engine. Refer to the documentation for your specific relational database engine for proper syntax.

**Output Tag Group**

There were two <OUTPUT> groups involved in this application: <NAME> ONLINE was used for the output containing statements to be delivered to their recipients. <NAME> PRINTED was used for the statements that were printed. Each OUTPUT tag group specified a <FILE> to which the data was written.
Advanced sample 10: Accessing data in a relational database

It is important to know that Enrichment reads the database only one time and stores the returned information in its <TABLE> in memory. Although this may require more memory footprint for the Enrichment process, this is much more efficient than calling the database repeatedly for each document to determine if the account exists in the online_statement table of the database.

**Note:** To determine the approximate amount of memory Enrichment will consume for the database lookup, you can run a test job without the lookup, and one with a known sample number of online statements. Enrichment reports memory usage in 1k amounts. Your sample should be large enough (over 1k results returned from the database) to facilitate measurement. At that point you can calculate the amount of memory taken for each or n number of result records returned.

### Run Time

The following information shows how to run the sample application on mainframe, UNIX, and Windows systems.

**UNIX/Linux**

Since the sample control file was written for a Windows environment, the control file defines data set specifications for that platform. Modify the <FILE> tags in the control file to specify file paths using UNIX/Linux conventions.

To run the application on a UNIX system, issue the following command:

```
sweaver -c=<controlfile>
```

Enrichment uses the unixODBC library to access relational databases. For more information, please refer to this website: [www.unixodbc.org](http://www.unixodbc.org).

**Windows**

To run the sample application on a Windows, the <FILE> tags need to be modified to reflect the file paths on the desired platform.

Then run the application by issuing the following command at the Windows command prompt:

```
sweaver -c=<controlfile>
```
Advanced sample11: Calling Spectrum

In this section

Overview.......................................................................................... 238
Files Used........................................................................................ 238
Business Scenario........................................................................... 238
Explanation of Sample..................................................................... 239
Overview

This sample illustrates how Enrichment can be used with Spectrum and describes the Spectrum-specific tags used to invoke a Spectrum job.

Before running the control file, you will need to setup a Spectrum job. This job can be created by importing the included data flow, AdvSam11.df. This job assumes you have Validate Address and Geocode US Address configured on your system. You will need to modify the 'Read from File' source and the 'Write to File' sink to match the directory you will be using.

You will need to setup a Java run-time environment, and have the 'java' command in the system path. The current JRE's, as of this writing, are at: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

You will need the 'jobexecutor.jar' jar file associated with your Spectrum installation. Contact Spectrum support to obtain this, if you do not already have it. This file must be in the directory you are running the sample from.

Files Used

This sample uses the following files.

<table>
<thead>
<tr>
<th>Table 1: Files Used in This Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Print Stream</td>
</tr>
<tr>
<td>Rule File</td>
</tr>
<tr>
<td>Other Inputs</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Output Print Stream</td>
</tr>
<tr>
<td>Other Output</td>
</tr>
</tbody>
</table>

Business Scenario

This sample application standardizes the address on customer statements, and provides Location Intelligence information by finding the latitude and longitude for the customer address.
Explanation of Sample

Our control file (shown below) reads a print stream that contains multiple documents, passes address information to Spectrum and generates Location Intelligence data about that address.

This control file is designed for running Enrichment on a Windows or Unix system. It will not run on a mainframe system, as Spectrum is not supported on the mainframe.

Input Tag Group

As the control file shows, the Input tag group is identified in the <NAME> tag as CLEANSE, and the input file is indicated as D:\apps\adv11\adv11in.afp by the Input group <FILE> tag. The input is fully composed AFPDS, so we set the <TYPE> tag to A. Because we will use address cleansing software to standardize addresses in the input, we set the <CLEANSE> tag to YES.

“PAGE 1” appears on the first page of each document, so we will define a field for that value, and a <DOCUMENT> tag using that field.

```
<DOCUMENT> t %Page1 E 
 <FIELD> %Page1 K N
      <REF>'' 'PAGE 1' 1 R
      <LOC>0 . S 6
 </FIELD>
```

Field Tag Group

We need to identify the address lines, so we define a field variable for each address line.

```
<FIELD> %Addr1 K N
     <REF>'' 'PAGE 1' 1 R
     <LOC>3 . 25 16
 </FIELD>
<FIELD> %Addr2 R N
     <REF>'' 'PAGE 1' 1 R
     <LOC>4 . 25 30
 </FIELD>
<FIELD> %Addr3 R N
     <REF>'' 'PAGE 1' 1 R
     <LOC>5 . 25 29
 </FIELD>
```
Cass Tag Group

The `<CASSTYPE>` tag value of “U” indicates we will be invoking UAM by calling a Spectrum job.

Spectrum Tag Group

The Spectrum tag group defines characteristics for how we will call the Spectrum server.

• `<SPECTRUMLOGIN>` requires the server name that Spectrum is installed on and the username and password that will be used.
• `<SPECTRUMJOB>` specifies the Spectrum job that Enrichment will run.

**NOTE:** The specified user must have permission to run the job.

• `<SPECTRUMFILE>` specifies the input file that Spectrum will read from before running the job. This file can be any file, as long as it matches the input source.
• `<SPECTRUMSCHEMA>` specifies the parameters that will be written to the file specified by `<SPECTRUMFILE>`.
• `<SPECTRUMOUTFILE>` specifies the output file that Spectrum will write to after running the job.
• `<SPECTRUMOUTSCHEMA>` specifies the variables that Spectrum will write to the output file. The output schema file must match the output sink for Spectrum.

Rule Tag Group

We write a variety of Spectrum generated variables, such as `%SPECTRUM_USCOUNTYNAME`. The variables are generated based on the contents of `<SPECTRUMOUTSCHEMA>`.

Run Time

The following information shows how to run the sample application on UNIX and Windows systems.

To run the application, issue the command:

```
sweaver -c=adv11.con
```
Index

A

Adding information to a page
   Using field replacement 28
   <ADDRESSBLOCK> tag 40
Addresses
   Cleansing 40
   Defining location of address data 40
Address tag group 40

C

CASS processing
   Specifying thresholds for Finalist 43
CASS tag group 42
   Cass Tag group 103
   <CASSTYPE> tag 42
CHECKSUM function 57
   <CLEANSE> tag 40
Code 39 barcode
   Specifying size 59
   Specifying values for 58
%Counter 57
Counting documents in an output 57
Counting pages in a document 57
Counting pages in an output 57

D

Defining a field 15
   <DELIMIT> tag 229
Document length 14
Document numbers 17
Documents
   Defining top of document 41
   <DOCUMENT> tag 14, 23, 32, 41
%DOCUMENT_NO system variable 17
DOCUMENT: section 32
   <DOUBLESORT> tag 42

E

Extracting data 14
   Defining data to extract 15
   font index 31
   OMR marks 31
   Text 31

F

Field
   Defining location 23

Field location 15
Fields
   Delete every occurrence 32
   <FIELD> tag 15, 23, 32, 41
Field Tag Group 55
Field tag group 15, 22, 31, 41
   <FILE> tag 14, 16, 22, 23, 31, 40
Finalist 42
Font index 31
FORMSX 58

I

Input
   Defining input type 22
   Defining documents 23
   Defining input type 30
Document numbers 17
Documents with varying page counts 21
Specifying print stream type 14
   <INPUTNAME> tag 23
Input tag group 14, 30, 40

J

JUSTIFY function 57

L

<LOCATION> tag 15, 23, 31, 32
LOCATION tag 15
%LPCRC 43
   <LPCREPLACE> tag 43

M

Multiple outputs 21

N

<NAME> tag 14, 16, 22, 23, 30, 40
Number of pages 24

O

ODBC functionality 226
%OMRCount 57
OMR marks 32
   Adding to all documents in an output 58
   Extracting 31
   Positioning 58
Output
   Directing documents to a specific output file 24
   Side file 14
   Sorting documents into separate output files 23
Index

Sorting into multiple outputs 21
Specifying an output file 16
<OUTPUT> command 24, 44
Output tag group 16, 23, 33
Outsoring based on Finalist return code 43

P

Padding values 57
Page count 24
<PAGE> tag 14
Parsing values 57
<POSITION> tag 58

R

<RDB> tag 229
<REFERENCE> tag 31, 32
Replacing data
  Defining replacement values 32
Replacing text 31
Rule file
  Using a separate file as a rule file 31
Rule file sections
  DOCUMENT 32
  START 32

S

Side file 14
  Specifying layout 16
Sidefile tag group 16
<SIDEPART> tag 16
Sorting
  By ZIP Code 42
  Defining data to sort on 22
  Defining sort criteria 23
  Defining sort criteria 23
  Improving performance 24
  Sorting documents into separate output files 23
Sorting documents 21
Sortmatch tag group 23
<SORT> tag 23
<SQLOUTFILE> tag 229
<SQL> tag 229
START: section 32
SUBSTR function 57

T

Text
  Extracting 31
  replacing 31
Top of document 23, 41
Top-of-document 55

EngageOne Enrichment 7.3.0 Sample Applications Guide 242