

What Is New in CSI-Data-Miner 6.0A and B?

Data-Miner 6.0A

"Auto" and "Task" mode scripts

Data-Miner will now run in either "Auto" mode or "Task" mode. Auto is the way that previous versions of Data-Miner have operated with reading and writing of the files being done automatically at the start and end of the script. This mode can still be used.

Task mode lets you do all of your own I/O so that you can decide when and how to read and write your records. You are also responsible for looping back to the start of the script to retrieve each record.

A task running in Auto mode has a "Main input file" and, possibly, a "Main output file." These are the first input and output file defined in the script and they will be automatically read and written at the start and end of the script in the same way as happens now. An Auto mode script can also do I/O to other files and extra I/Os to the main files and create reports.

Multiple input and output files

A Data-Miner script can now include any number of input and output files. Files can be read and written sequentially or by key (including "skip-sequential" processing). Files are defined using the usual INPUT= and OUTPUT= commands.

File I/O result

The result of an I/O operation to a file is returned after each I/O operation for the file as a 3 character alphabetic return code. The result is put into a field called "filename:IO-RESULT." These return codes are the same as those returned by Data-Miner's VSAM access modules VNATB and VNAT with the exception that successful I/O completion is shown by a result of "OK" rather than "000." Some of the more commonly encountered file results include EOF (end of file), NFD (keyed record not found), DPK (duplicate key when trying to add a keyed record). Many of the same result codes are returned for table operations (see "FIND command"). There is also a general IO-RESULT value that is set to the result of the most recent I/O operation performed by any file.

INPUT= and OUTPUT= commands

The number of reads or writes done to a file can now be limited with the MAX= parameter. For example MAX=50 says that only 50 records should be read from, or written to, this file. When an input file exceeds its MAX count, Data-Miner treats it as if end of file was reached, including setting the file's result to EOF. When an output file exceeds its MAX count, the output record is not written but all other processing continues as if it had been.

When running in AUTO mode, passing the MAX value on the main input file will terminate the script normally.

Report writer

The report writer allows you to define a report layout and then to print a line or group of lines to the report using the PRINT command. A report definition allows you to lay out a report with up to 255 different lines per page, a line length of 250 characters and up to 32,000 bytes of information on each page. The report can be printed in any order you like with headings and sub-totals being inserted when necessary. You can create summary reports consisting solely of counts and totals. Any number of reports can be created in a single execution of Data-Miner and with a single pass of the input file. Data can be combined from several files and / or work fields to create a report.

Mailing labels

Data-Miner can print several labels across the page. This facility can be used to print any kind of multi-column output that is required such as side-by-side packing slips. Anything that can be printed in a report can be printed on a label.

Table look up

Data-Miner supports two types of table look-up. The first is where the table is loaded into memory from a file at the start of the script and the second is where the table's contents are provided as part of the script. The total size of all tables is limited to a little less than 2Gb and each row of a table can be up to 8Kb long (it can be loaded from a file with any size of record). Within those limits, tables can have any number of rows and columns.

Defining a table

A table is defined with the TABLE command with its fields being defined in the same way as a file layout. The TABLE command is in the form:

```
TABLE      name [FILE filename] [SIZE nnnnn]
```

If a filename is specified then it must be defined with an INPUT= command before the TABLE command. Fields in the table are defined the same way as fields in a file. A table can have as many fields as you like.

The optional SIZE parameter tells Data-Miner roughly how many rows you expect the table to have. If the SIZE you give is not large enough to contain the entire table, Data-Miner acquires enough storage for a further "SIZE" rows and continues to do so until it has got enough memory. The default value for SIZE is 1000 rows.

Fields in a table are defined in the same way as fields in a file. The "Start position" value on the field definition tells Data-Miner the start position of the field in the file or card image from which the table is being loaded. Fields are squeezed together in the table to remove any unused space.

Filling a table

Entries in a table can be filled in two ways. If the table is defined as being associated with a file then Data-Miner reads the whole file into the table before doing any processing. If there is no file associated with the table then you provide the data for the table as part of your script. The data follows the table's field definitions, preceded by the command "DATA" and followed by the

command “ENDTABLE.” There is no need to tell Data-Miner how many rows there are in the table. Tables that are loaded with data as part of the script will generally contain character or zoned numeric data as this is all that can be entered with a keyboard. However, instream data can contain any characters other than X'FE' and X'FF'.

Data does not have to be in any particular order - Data-Miner treats the first field you define as the key field for the table but that doesn't need to be in any particular order either.

Examples

1 Load a table called BRANCHES with information from a file called BRFILE

```
TABLE          BRANCHES  FILE BRFILE
BRANCH-NO      1         5    C
CITY           9         20   C
RATE          40         6    P    2
```

2 Load a table called DEPTS with information contained in the script

```
TABLE          DEPTS
DEPT-CODE      1         4    C
DEPT-NAME      6         15   C
DEPT-MANAGER   18        30   C
DATA
ACCT ACCOUNTING      Joe Bloggs
SALE SALES           Penny Wise
MAIN MAINTENANCE     Fred Boreham
ENDTABLE
```

Finding data in a table

Finding data in a table is very similar to reading a record by key from a keyed file. You can retrieve a row in a table by finding a match for any of the fields in the table, using the FIND command. The MATCHING keyword is optional. The tablename:IO-RESULT field is set to “OK” if a matching entry is found or to “NFD” if no matching entry is found.

For example:

```
FIND DEPTS DEPT-CODE='SALE' or
FIND BRANCHES MATCHING CITY=INFILE:TOWN
```

Fieldnames

Fieldnames can include hyphens to make Data-Miner more “COBOL programmer friendly.” All internal field names and commands used by Data-Miner and that are not accessible to the user now start with “@#_” so that it is unlikely that users will accidentally use conflicting field names. You are allowed to start your own field names with @#_ but any resulting errors are your problem!

Field names are shared between files if the files share the same field definitions. (This will happen, for example if you enter an INPUT= command immediately followed by an OUTPUT= command. The two files will have the same fields defined in the same positions but they will not share the actual contents of those fields). To refer to a field in a specific file, prefix the field

name with the file name and a colon (e.g. INFILE:BALANCE). Otherwise a reference to a field name such as BALANCE is taken to mean the first field defined with that name.

Statement labels

In addition to the @LBL command, you can now label a statement as a name with a period on the end. So “@LBL HERE” and “**HERE.**” are equivalent.

Error messages

There are over 30 new or changed error messages.

Obsolete commands

The **BGNFIELDS** and **ENDFIELDS** commands are no longer needed and will be ignored if they are entered. They are still reserved words.

The **PTEND** command is no longer needed and PTEND is no longer a reserved word.

The **TRACE** internal command has been renamed and TRACE is no longer a reserved word.

The **DEFINE** keyword used to define a work field is now optional. Data-Miner treats a field definition that has no starting position as a work field.

The **SET** keyword is now optional on commands such as SET TOTAL = 0

The “**FROM filename**” part of a SELECT command is now ignored and can be omitted

Field definitions

Commas in field definitions are optional so that the following all mean the same

AMOUNT 10,5,P

AMOUNT 10 5 P and

AMOUNT 10 5,P

Fields, both in records and work fields, can be redefined. For example

DATE 20 8 Z

MONTH DATE 2 Z

DAY DATE+2 2 Z

YEAR DAY+2 4 C

A field start position of * can be used to mean “the next byte in the record” and ‘ ‘ ‘ (ditto) can be used to mean “starts in the same place as the previous field. * and ‘ ‘ can both be adjusted with + or – values to move to a different place in the record.

Opening files

All **referenced** files are opened at the start of a script, provided the script is valid. A “referenced file” is one for which the script contains a READ, WRITE or DELETE command, (whether that command actually gets executed or not) or that is included in a COPY, EXTRACT or DELETE run as one of the main files. This means that unused files need not be removed from a script, making it easier to set up “skeleton” general purpose scripts. It also means that, in the event of a WRITE not being issued to an output file, an empty file is created rather than leaving an obsolete version of the file in place.

Arithmetic commands

You can use an arithmetic command such as `PRICE = COST + FREIGHT * TAX` to do calculations. The expression is evaluated from left to right and parentheses are ignored. Operators can be `+`, `-`, `*` and `/`.

Random access for VSAM files

Data-Miner has always had the “FIRSTKEY” command to say where sequential reading is to start. You can now process VSAM files in any order using the “POINT” command. POINT lets you point at the first record in the file, the last record in the file, the previous record, the next record, a record with a specific key or a record with a key greater than or equal to the one you specify. Following the next READ command, the VSAM file is set to read the next record unless you issue another POINT command to point to a different place. POINT does not read a record from a file; it merely tells Data-Miner which record you want to read next time you issue a READ command. You may change your mind about which record you want to read next. For example

```
POINT FILEA KEY '123'  
POINT FILEA KEY '789'  
READ FILEA
```

will read the record whose key is '789'

POINT does not set the IO-RESULT.

New system variables and constants

A number of new system variables and constants have been introduced including:

EOF - a constant value of “EOF”

HIGH-VALUES - 16 bytes of X'FF'

LOW-VALUES or NULLS - 256 bytes of X'00'

SPACES or BLANKS - 256 bytes of spaces

NUMERIC - for use in IF commands such as “IF A IS NUMERIC”

Reading and writing files

There are a number of ways to read or write a file. First, you must tell Data-Miner about the file with an INPUT= or OUTPUT= command. If you are running a COPY, DUMP, PRINT, EXTRACT or DELETE script, you do not need to tell Data-Miner to read or write the main files as it will read a record at the start of the script, write (or print or dump or delete) the record at the end of the script and then go back to read another record, carrying on until there are no more records to be read.

The “Main files” for these types of script are the first input and output file defined in your script. So a script that contains:

```
COPY
  INPUT=VSAM  FILENAME=FILE1
  INPUT=DISK  FILENAME=FILE2
  OUTPUT=DISK FILENAME=FILE3
etc.
```

will copy FILE1 to FILE3 as they are the first input and output files defined, respectively. Your script can do whatever additional reading it likes from FILE2 but Data-Miner will not read any records from it automatically.

READ command

The READ command reads the next record from a sequential file, from a keyed file that is being read sequentially or by key from a keyed file.

The record is read into the record area that is defined following the definition of the file. The command to read the next record from FILE1 is:

```
READ FILE1
```

Reading randomly

To read a keyed file randomly using a key, you have two options. You can either POINT at the record you want to read and then read it or you can read directly by putting the key in the read command. For example:

```
POINT FILE1 "999000"
READ FILE1
```

points FILE1 at the record with a key of 999000 and then reads that record (the POINT command does not read a record, it merely points the file to the right place). If the key that you point at is shorter than the actual key of the file then the next READ will retrieve the first record whose key starts with the partial key you have given. For example, suppose FILE1 has a 6 byte key:

```
POINT FILE1 "123"
READ FILE1
```

returns the first record whose key starts with “123.”

The other way to read a file randomly is to put the key on the READ command as in:

```
READ FILE1 KEY "456789"
```

Errors during reading - return code

A number of things, other than getting a record, can happen as a result of trying to read a record. Most of these error conditions are quite normal, indicating that you have reached end of file, tried to read a record that doesn't exist etc. Following every read or write to a file, Data-Miner places a result code in a field called **filename:IO-RESULT**. That value remains there until you next read or write that file. (Note that POINT does not change the result field). The result field is 3 characters long and the most common values you will find in there are:

- OK - the read or write was successful
- EOF - there are no more records in the input file. The last read issued by your script did not get a record from the file; it only got the EOF indication
- DPK - duplicate key - you are trying to write a record to a keyed file but a record of that key already exists
- NRF - no record found - you have issued a read for a record by key but no record with that key exists.

WRITE command

The WRITE command is used to write a record to a file. If the file is being written sequentially then WRITE writes the next record to the file. If the file is being written by key then the record you write is either inserted into the file or added on the end of the file, depending on the key value. There is no need to tell Data-Miner what the key of the record is as Data-Miner finds it in the record that it is writing. If a record with that key already exists then the file's result field is set to "DPK" and the record is not written.

Examples of READ, WRITE and POINT

Example 1 - copy a file and write every record with a negative balance to another file

```
COPY
INPUT=VSAM      FILENAME=INFILE
OUTPUT=VSAM     FILENAME=OUTFILE
OUTPUT=DISK FILENAME=NEGFILE  VARBLK BLKSIZE 16000
WHOLE-RECORD   1 1000 C
BALANCE 98 10 P
IF  BALANCE < 0
    MOVE  INFILE:WHOLE-RECORD TO NEGFILE:WHOLE-RECORD
    WRITE NEGFILE
ENDIF
```

Note that there is no need to read or write INFILE or OUTFILE - as they are the first input and output files defined, Data-Miner uses them as the input and output files for the copy operation. Because no separate record layout has been defined for INFILE, OUTFILE and NEGFILE, all three files share the same field definitions.

Example 2 - read a file sequentially, get some information from a keyed file using data from the input record and write a record containing some fields from both the main input file and the keyed file

```
INPUT=DISK FILENAME=INFILE LRECL=800 BLKSIZE=800 FIXED
ACCT-NO    1     8     C
BALANCE    98    10    P
OUTPUT=VSAM FILENAME=NEWFILE
NEWACCT    1     8     C
ADDRESS    9     300   C
BALANCE    400   12    P
INPUT=VSAM FILENAME=CUSTFL
ADDRESS    200   300   C
READNEXT.
READ INFILE
IF  INFILE:IO-RESULT = "EOF"
    GOTO FINISHED
ENDIF
IF  INFILE:BALANCE < 0
    READ CUSTFL KEY ACCT-NO
    IF  CUSTFL:IO-RESULT = 'NRF'
        GOTO NOCUST
    ENDIF
    MOVE ACCT-NO TO NEWACCT
    MOVE INFILE:BALANCE TO NEWFILE:BALANCE
    MOVE CUSTFL:ADDRESS TO NEWFILE:ADDRESS
    WRITE      NEWFILE
NOCUST.
ENDIF
GOTO READNEXT
FINISHED.
```

Here we are doing all of the reading and writing ourselves and checking the I/O result after each read to see if we have reached end of file on the main input file or if we have tried to read a non-existent record from the keyed file, CUSTFL.

Because we have used the same field names for the various files, we have needed to prefix the field names in the MOVE commands with the name of the file we want the field to come from or go to.

Opening and closing files

There is no need to open or close the files used in a Data-Miner script. Data-Miner will open all of the files used by a script (that is, referred to in a READ or WRITE command or defined as one of the main files for a COPY, EXTRACT, DUMP, DELETE or PRINT script). When the script finishes, Data-Miner closes all the files that it opened.

Creating a report

There are two ways to create a report with Data-Miner. The first (using PRINT and SELECT) is most useful for a quick one-off tabular report while the report writer (using PRINT and REPORT) gives you more control over the layout and contents of your report.

Creating a tabular report

Suppose that you want a report showing the outstanding balance of all the accounts in your BILLING file. This is most easily done using a PRINT script. For example:

```
PRINT
INPUT=VSAM FILENAME=BILLING
ACCT-NO    1    8    C
NAME 9     32    C
ADDRESS   100  55    C
BALANCE   240  10    P    2
SELECT (ACCT-NO NAME ADDRESS BALANCE)
```

That's all there is to it. Data-Miner reads the BILLING file sequentially and prints a tabular report with each line containing the required fields. The report looks something like this:

| ACCT-NO | NAME | ADDRESS | BALANCE |
|----------|--------------|----------------|-----------|
| 12345678 | BLOGGS INC | 47 MAIN ST | 426.65 |
| 33333333 | JONES LLC | 96 ANYWHERE ST | 12,217.32 |
| 44444444 | ALBRECHT INC | 62 DURER AVE | 0.00 |

And so on

Being selective

You can create a report of records that meet certain criteria very easily using the "ONLY" and "SKIP" commands. "ONLY" means "I want to see only these records while "SKIP" means "skip these records." So, in the example above, if we wanted only those records where the balance is greater than \$1,000.00, we could add a command before the SELECT saying either:

```
ONLY BALANCE > 1000 or
```

```
SKIP BALANCE <= 1000
```

Laying out a report

Besides using the SELECT command and letting Data-Miner lay out your report out for you, you can lay out your own report exactly how you want it. There are a number of steps involved although you can usually skip most of them and use Data-Miner's defaults. Three commands help you lay out and print a report:

The field definition

Data-Miner will format a field for printing based on how you define the field in your record layout. For example, if you say it is a number with 2 places of decimals, Data-Miner will, by default, insert commas between groups of 3 digits and print the number with 2 places of

decimals. The field name is used as the heading.

You can override these defaults by specifying a field layout, called a mask, and / or a field heading in the field definition. Data-Miner provides 5 built-in masks - to use one, you add its name to the field definition - for example:

```
BALANCE 60 8 P 4 M5
```

tells Data-Miner that BALANCE is an 8 byte packed field starting at byte 60, with 4 places of decimals and is to be printed using mask M5. Because mask M5 has only 2 places of decimals, the last two places of decimals are not printed.

You can define your own mask, too. To define your own mask for printing a field, you add something like "MASK '123,456.78 DOLLARS'" to the end of the field definition instead of using an "M" value.

The characters you put in your mask serve a special purpose in showing how the field should appear:

A number in the mask will be replaced with a number from the field

A "Z" in the mask is replaced with a number from the field unless the number in the field is a leading zero in which case the Z is replaced with a space. For example, if the field contains '000123456' and the mask is "ZZZZ99999", it will be displayed as " 123456"

If the first character of your mask is a currency symbol, it will be "floated" down the field so that it is printed immediately in front of the number.

If the first character of your mask is an asterisk, the number in the field will be padded on the left with asterisks.

A leading or trailing minus sign will be added to the field if the number is negative.

You can also define a column heading for the field to appear in any report where the field is used. The column heading can be from 1 to 3 lines with each line specified as a literal in quotes. For example

```
BALANCE 60 8 P 4 M5
```

will put a heading of "BALANCE" at the top of the column in which the BALANCE field is printed whereas

```
BALANCE 60 8 P 4 M5 HEADING ('Current' 'account' 'balance')
```

will print three heading lines for the field.

A report can be sorted, printed "2-up" (or any number up - used mostly for address labels), totaled, broken by control value and a number of other options as set out with the REPORT command

The REPORT command

Use the REPORT command to define the layout of a report and select which fields are to appear in the report. You use the PRINT command to tell Data-Miner actually to print a "subpage" (usually a line) of the report. A report definition can contain one or more lines and these are referred to as a "subpage." The PRINT command prints the entire subpage in one go.

Examples

A column report of overdue accounts

```
* Define the input file
INPUT=DISK FILENAME=ACCOUNT etc
* Field definitions for the input file
```

```
ACCT-NO    1      8      C      HEADING ('ACCOUNT' 'NUMBER')
BALANCE    64     9      P      2
BAL-60     90     9      P      2      HEADING ('60 DAYS' 'OVERDUE')
BAL-120    135    9      P      2      HEADING ('>120
                                DAYS' 'OVERDUE')

*      Define the printer
OUTPUT=PRINTER FILENAME=SYSPRT      etc
*      Now set up automatic reading of the input file
AUTO INPUT ACCOUNT
*      Print the report
PRINT      OVERDUE
*      Define the report layout - let Data-Miner do most of
      the work
REPORT      OVERDUE
TITLE      "OVERDUE ACCOUNTS REPORT"
LINE ACCT-NO BALANCE      BAL-60 BAL-120
```

That's the entire job. The print line is laid out by Data-Miner so that the fields fit in, a page heading of "OVERDUE ACCOUNTS REPORT" is printed on every page and the columns in the report get headings of "ACCOUNT NUMBER", "BALANCE", "60 DAYS OVERDUE" AND ">120 DAYS OVERDUE").

The "AUTO INPUT ACCOUNT" command tells Data-Miner to add all the logic to read the records from ACCOUNT one at a time, process each one, print the report and keep looping round to read another record until there no more records to be read at which point the script stops.

Sales report, sorted by department

```
*      Define the input file
INPUT=DISK FILENAME=SALES etc
*      Field definitions for the input file:
CUST-NO    1      8      C      HEADING ('CUSTOMER' 'NUMBER')
DEPT       64     12     C      HEADING ('SALES' 'DEPARTMENT')
VALUE      90     9      P      2      HEADING ('INVOICE' 'AMOUNT')
DISCOUNT  135    9      P      2

*      Define the printer
OUTPUT=PRINTER FILENAME=SYSPRT      etc
*      A work field for a calculation
DEFINE DISC-VAL 9      P      2      HEADING('DISCOUNTED' 'VALUE')
*      Now set up automatic reading of the input file
AUTO INPUT SALES
DISC-VAL = VALUE
SUBTRACT DISCOUNT FROM DISC-VAL
*      Print the report
PRINT      SALES-REPORT
*      Define the report layout - let Data-Miner do most of
      the work
REPORT      SALES-REPORT
```

```
TITLE      ("DEPARTMENTAL SALES REPORT" "WITH DISCOUNTS" )
LINE DEPT CUST-NO VALUE DISCOUNT DISC-VAL
ORDER DEPT
```

AUTO command

You can write your own loop control in Data-Miner to read through a sequential file and stop when you reach end of file or you can use the AUTO command to do the job for you. The format of the AUTO command is:

```
AUTO INPUT filename
```

If there is no file to be automatically handled but you want to include an AUTO command anyway, you can write:

```
AUTO INPUT NONE
```

This may be of use in a future release of Data-Miner but serves no purpose in version 6.0.

AUTO INPUT MYFILE is the equivalent of:

```
STARTIT .
READ MYFILE
IF EOF MYFILE
    GOTO ENDIT
ENDIF
```

```
* The rest of your script goes here
GOTO STARTIT
ENDIT.
```

DUMP and PRINT commands

The DUMP and PRINT commands can now be used in the same script and a print line and a dump of the output record will be produced for each record in the file. The DUMP and PRINT output are mixed together and so this option is really useful only for program testing.

SHOW command

The SHOW command is intended mainly for use by computer people who want to see a hex and character snapshot either of a file, part of a file or individual records and fields. The format of it is:

```
SHOW fieldname e.g. SHOW BALANCE or SHOW MYFILE:DEBT-TOTAL or
SHOW filename e.g. SHOW MYFILE
```

SHOW fieldname displays the whole of the named field as it would be displayed in a program dump with hex on the left and a character version on the right. It shows what is in the field when the SHOW command is executed which is not necessarily what was in the field when the record was read.

SHOW filename displays the entire current record for a file. It can be used for both input and output files to show exactly what the current record looks like when the SHOW is issued. If you have changed some fields in the input record then you will see the modified version. If you have built only part of the output record then that is all you will see.

DISPLAY command

DISPLAY lets you display some information in a formatted way without going to the effort of laying out a report. For example

```
DISPLAY "Total sales = " TOTSALLES
```

Data-Miner 6.0B

DO WHILE command

Lets you set up a loop to execute all the commands in the loop while the condition at the start of the loop is met. For example

```
DO WHILE AMOUNT1 > AMOUNT2  
READ FILEX  
ADD BALANCE TO AMOUNT2  
ENDDO
```

ENDWHILE can be used instead of ENDDO.

Restrictions removed from INPUT and OUTPUT commands

The previous restriction on INPUT and OUTPUT commands having to be on a single line has been removed.