# Importing Text-based data: Workbench

red-gate.com/simple-talk/sql/t-sql-programming/importing-text-based-data-workbench

October 23, 2007

Robyn and Phil return with some fresh ideas about how to import text files into SQL Server, without resorting to DTS or SSIS scripting. They go on to show how much can be done in TSQL

## Contents

## Introduction

It is hard to estimate the enormous number of unnecessary and unmaintainable SSIS and DTS files that are written merely to import data from text into SQL Server. For performance, and for the sanity of the DBA, it is usually better to allow SQL Server to import text and to pummel it into normalised relational tables, rather than rely on procedural techniques.

There are many ways to read text into SQL Server including, amongst others, `BCP, BULK INSERT, OPENROWSET, OPENDATASOURCE, OPENQUERY`, or by setting up a linked server.

Normally, for reading in a table from an external source such as a text file, one would use an OpenRowSet, which can be referenced in the `FROM` clause of a query as though it were a table name. This is a topic that would take too long to tackle in this workbench, though we'll show you an example of its use for reading in a CSV file. Perhaps one day we'll do an OpenRowSet Workbench!...

## Fast import with the Quirky Update technique

So, you think you're good at importing text-based data into SQL Server? A friend of ours made that mistake too, recently, when he tried to get a highly paid consultancy job in London. The interviewer guided him to an installation of SQL Server and asked him to import a text file. It had a million rows in it which were rather poorly formatted. As our friend stared at the data, his confident laugh turned to a gurgle of panic, as he suddenly realised that he wasn't looking at simple columnar data, or delimited stuff, but something else, and something that looked tricky. Our friend realised too late that it was a 'curved ball' and floundered embarassingly. Let's simulate a few of the million rows just so you can see the problem.

```
frizbees	59787	654 c
cricket bats	807453	9245 c
stumps	80675	1348 s
tennis rackets	74009	34	t
woggle 74009	34	t
Running shoes 4570	132	c
football shorts and shirt (small, medium or large) 5928 132 c
```

There are, of course, several different approaches to turning this sort of mess into a table. we can `BCP` or `BULK INPUT` it into an imput table, in order to pummel it into shape. Actually, where record-lengths are short, one can do it even more simply this way.

```
CREATE TABLE #Textimport ( line VARCHAR(8000) )

INSERT  INTO #textImport
        ( line )
        EXECUTE MASTER..xp_cmdShell 'Type MyFile.TXT'

/*
But for this exercise... we'll just create a sample '
*/
DROP TABLE #import
CREATE TABLE #import
  (
    line VARCHAR(8000),
    firstone INT,
    secondone INT,
    thirdone INT
  )
INSERT  INTO #import  ( line )
        SELECT  'frizbees     59787   654 c'
UNION ALL
        SELECT  'cricket bats     807453   9245 c'
UNION ALL
        SELECT  'stumps    80675   1348 s'
UNION ALL
        SELECT  'tennis rackets    74009   34  t'
UNION ALL
        SELECT  'woggle 74009   825  t'
UNION ALL
        SELECT  'Running shoes 4570   132  c'
UNION ALL
        SELECT
      'football shorts and shirt (small, medium or large) 5928 132 c'
```

And so the answer to the interview question was perfectly simple. With a million rows, one daren't hang about, so here is a solution that does the trick quickly without a cursor in sight. Can you spot a neater method? Neither Phil nor I can.

```
DECLARE
  @first INT,
  @second INT,
  @third INT

UPDATE  #import
SET     @first = firstone = PATINDEX('%[0-9][0-9]%', line),
        @second = secondone = @first + PATINDEX('%[^0-9][0-9]%',
                        SUBSTRING(line, @first + 1, 2000)) + 1,
        thirdone = @second + PATINDEX('%[^0-9][a-z]%',
                        SUBSTRING(line, @second + 1, 2000)) + 1

SELECT  [product] = CONVERT(VARCHAR(50), RTRIM(
                        SUBSTRING(line, 1, firstone - 1))),
        [sales] = CONVERT(INT, RTRIM(
                        SUBSTRING(line, firstone,secondone - firstone))),
        [Salesman_id] = CONVERT(INT, RTRIM(
                        SUBSTRING(line, secondone, thirdone - secondone))),
        [type] = CONVERT(CHAR(1), RTRIM(
                        SUBSTRING(line, thirdone, 2000)))
FROM    #import
```

Which gives:

```
product                                          sales   S_id   type
------------------------------------------------ ------- ------- ----

frizbees                                         59787    654    c
cricket bats                                      807453   9245   c
stumps                                            80675    1348   s
tennis rackets                                      74009    34     t
woggle                                            74009    825    t
Running shoes                                          4570     132    c
football shorts and shirt (small, medium or large) 5928     132    f
```

Of course, this needs a bit of explanation. What we are doing is to use the ‘Quirky Update’ syntax in Sybase and SQL Server to allow us to update some special columns in the import table that tell us the column positions of the various pieces of data for each row, as they will be different in every row.

The first column is terminated by the number (number of sales), so we need to use `PATINDEX` to tell us where this is. Then we have to look for the next number. The trouble with `PATINDEX` is that one cannot specify the start (or end) position of the search, so you have to use `SUBSTRING` for that. Finally we need to find that pesky character at the end.

Now we have the column positions we can then parse it all neatly with a select statement.

You’ll see that it would work even with spurious characters in the way such as [ ], and so on.

Sometimes, one gets strange delimiters in data. Here is an example of how one might input a file from a monitoring system.

```
[stop-cock opened] <<<<(Matt)>>>>>   [12/3/2007 12:09:00]
[stop-cock closed] <<<(Tony)>>>>   [12/3/2007 12:10:00]
#not authorised [stop-cock opened] <(Timothy)>   [12/3/2007 13:21:00]
[stop-cock closed] <<(Dave)>>>   [12/3/2007 13:30:00]
[stop-cock opened] <<<<(Matt)>>>>>   [12/3/2007 15:18:00]
#post-sign-off [stop-cock closed] <<<(Matt)>>>>   [12/3/2007 15:20:00]
```

```
CREATE TABLE #importDelimited
  (
    line VARCHAR(8000),
    firstone INT,
    secondone INT,
    thirdone INT,
    fourthone INT,
    fifthone INT,
    Sixthone INT
  )
INSERT  INTO #importDelimited  ( line )
        SELECT  ' [stop-cock opened] <<<<(Matt)>>>>>   [12/3/2007 12:09:00] '
UNION ALL
        SELECT  ' [stop-cock closed] <<<(Tony)>>>>   [12/3/2007 12:10:00] '
UNION ALL
        SELECT  '#not authorised [stop-cock opened] <(Timothy)>   [12/3/2007
13:21:00] '
UNION ALL
        SELECT  ' [stop-cock closed] <<(Dave)>>>   [12/3/2007 13:30:00] '
UNION ALL
        SELECT  ' [stop-cock opened] <<<<(Matt)>>>>>   [12/3/2007 15:18:00] '
UNION ALL
        SELECT  '#post-sign-off [stop-cock closed] <<<(Matt)>>>>   [12/3/2007
15:20:00] '
/* OK, here is a bit of luck! The delimitors show us where the fields are. They may
be inconsistent but that doesn't worry us. Heaven only knows what was going
through the mind of the programmer who came up with this data format.*/
DECLARE
  @first INT,
  @second INT,
  @third INT,
  @Fourth INT,
  @Fifth INT

UPDATE  #importDelimited
SET     @first = firstone = CHARINDEX('[', line),
        @second = secondone = CHARINDEX(']',line,@first+1),
        @third = thirdone = CHARINDEX('(',line,@second+1),
        @fourth = fourthone = CHARINDEX(')',line,@third+1),
        @fifth = fifthone = CHARINDEX('[',line,@fourth+1),
       Sixthone = CHARINDEX(']',line,@fifth+1)

SELECT
   CONVERT(VARCHAR(20),SUBSTRING(line,firstone+1,secondone-firstone-1)),
   CONVERT(VARCHAR(10),SUBSTRING(line,thirdone+1,fourthone-thirdone-1)),
   CONVERT(DATETIME,SUBSTRING(line,fifthone+1,sixthone-fifthone-1),103)

```

```
FROM #importDelimited
/*
-------------------- ---------- ----------------------
stop-cock opened    Matt      2007-03-12 12:09:00.000
stop-cock closed    Tony      2007-03-12 12:10:00.000
stop-cock opened    Timothy   2007-03-12 13:21:00.000
stop-cock closed    Dave      2007-03-12 13:30:00.000
stop-cock opened    Matt      2007-03-12 15:18:00.000
stop-cock closed    Matt      2007-03-12 15:20:00.000

(6 row(s) affected)
```

## CSV Importing- Comma-delimited and Comedy-Limited.

CSV, if done properly, is actually a very good way of representing a table as an ASCII file, even though its use has now been overtaken by XML. CSV is different from a simple comma-delimited format. The simple use of commas as field separators is often called ‘Comedy Limited’, because it is so incredibly useless and limiting.

The real CSV allows commas or linebreaks in fields: well anything actually. It is described in The Comma Separated Value (CSV) File Format, or CSV Files

BCP is not a good way of reading CSV files; Unless you use a Format file, it will only do ‘comedy-limited’ files. A much better method is to use ADODB provider MSDASQL, which does it properly.

```
SELECT *
FROM
    OPENROWSET('MSDASQL',--provider name (ODBC)
      'Driver={Microsoft Text Driver (*.txt; *.csv)};
        DEFAULTDIR=C:\;Extensions=CSV;',--data source
      'SELECT * FROM sample.csv')
```

This assumes that the first row is the header, so you may need to add a first row.

The ODBC TEXT driver will not output a table as a CSV file, unfortunately. The reason for this is mysterious. It would have been very useful.

Sometimes, for a special purpose where a simple method like this won’t do, you have to develop a TSQL way. Sometimes, for example, you will find that records are separated by ‘[]’ markers, or that comment or header lines are inserted with a prepended ‘#’. Sometimes quotes are ‘escaped’ by a ‘\’ character.

The first stage is to read the entire file into a SQL Server variable. Reading text into a VARCHAR(MAX) is very easy in SQL Server 2005. (For other ways in SQL Server 7 and 2000, see Reading and Writing Files in SQL Server using T-SQL

```
DECLARE @CSVfile VARCHAR(MAX)
SELECT  @CSVfile = BulkColumn
FROM    OPENROWSET(BULK 'C:\sample.csv', SINGLE_BLOB) AS x
SELECT @CSVfile
```

For this test, we’ll put the CSV file in a VARCHAR(MAX) variable.

```
SET NOCOUNT ON
DECLARE @CSVFile VARCHAR(MAX)

SELECT  @CSVFile = '
Tony Davis,,,,
Rev D. Composition,02948 864938,10TH 7TH,"The Vicarage,
Blakes End,
Shropshire",
Phil Factor,04634 845976,FD4 5TY,"The Lighthouse,
Adstoft,
Norfolk",Phil@notanemail.com
Polly Morphick,04593 584763,,"""The Hollies"",
Clumford High Street,
Chedborough,
Hants DF6 4JR",Polly@NotAnEmail.com
Sir Relvar Predicate CB,01549 69785,FG10 6TH,"The Grange,
Southend Magna,
Essex.",'
```

/*here is the XML version by comparison

```
<document>
<row>
  <Col0>Tony Davis</Col0 >
  <Col1></Col1 >
  <Col2></Col2 >
  <Col3></Col3 >
  <Col4></Col4 >
</row>
<row>
  <Col0>Rev D. Composition</Col0 >
  <Col1>02948 864938</Col1 >
  <Col2>10TH 7TH</Col2 >
  <Col3>The Vicarage,
Blakes End,
Shropshire</Col3 >
  <Col4></Col4 >
</row>
<row>
  <Col0>Phil Factor</Col0 >
  <Col1>04634 845976</Col1 >
  <Col2>FD4 5TY</Col2 >
  <Col3>The Lighthouse,
Adstoft,
Norfolk</Col3 >
  <Col4>Phil@notanemail.com</Col4 >
</row>
<row>
  <Col0>Polly Morphick</Col0 >
  <Col1>04593 584763</Col1 >
  <Col2></Col2 >
  <Col3>"The Hollies",
Clumford High Street,
Chedborough,
Hants DF6 4JR</Col3 >
  <Col4>Polly@NotAnEmail.com</Col4 >
</row>
<row>
  <Col0>Sir Relvar Predicate CB</Col0 >
  <Col1>01549 69785</Col1 >
  <Col2>FG10 6TH</Col2 >
  <Col3>The Grange,
Southend Magna,
Essex.</Col3 >
  <Col4></Col4 >
</row>
</document>
```

```
  DECLARE @StartOfRecord INT,
    @RecordNo INT,
    @FieldNo INT,
    @WhatsLeftInText VARCHAR(MAX),
    @DelimiterType VARCHAR(20),
    @EndOfField INT,
    @Delimiter VARCHAR(8),
  @eat INT,
  @jj INT,
  @jjmax INT,
  @Escape INT,
  @MoreToDo INT
DECLARE @OurTable TABLE (Field INT, record INT,Contents VARCHAR(8000))



SELECT  @CSVFile = LTRIM(@CSVfile),
       @StartOfRecord = 1,
       @RecordNo = 1, @FieldNo = 1, @MoreToDo = 1
--iterate for each field
WHILE @MoreToDo = 1
  BEGIN
   --identify the delimiter for this field
    SELECT  @Delimiter = SUBSTRING(LTRIM(@CSVfile), @StartOfRecord, 1),
            @eat = 0
    IF @Delimiter = ','
      SELECT  @DelimiterType = 'Field'
    ELSE
      IF @Delimiter IN ( CHAR(13), CHAR(10) )
--The end of record delimiters are sometimes other characters such as a
semicolon
        SELECT  @DelimiterType = 'RecordEnd'/* Records are separated with
CRLF (ASCII 13 Dec or 0D Hex and ASCII 10 Dec or 0A Hex respectively) for
Windows, LF for Unix, and CR for Mac*/
      ELSE
        IF @Delimiter LIKE '"'
          SELECT  @DelimiterType = 'Complex'
        ELSE
          SELECT  @DelimiterType = 'RecordStart'
    IF @DelimiterType = 'Field'
      BEGIN --this starts with a comma
        SELECT  @eat = 1
        --check to see if it is quotes-delimited
        IF ( SUBSTRING(LTRIM(@CSVfile), @StartOfRecord + @eat, 1) = '"' )
          SELECT  @eat = 2, @DelimiterType = 'Complex'
      END
    --let's work on the remaining text rather than the whole file'
    SELECT  @WhatsLeftInText = STUFF(@CSVFile, 1, @StartOfRecord + @eat
- 1,
                                     '')
    IF @DelimiterType IN ( 'Field', 'RecordStart' )
      BEGIN--and we will get the end of the simple field
        SELECT  @EndOfField = PATINDEX('%[,' + CHAR(13) + CHAR(10) + ']%',
                                       @WhatsLeftInText)
        IF @EndOfField = 0 --of not there then we are at the end of the file
          SELECT  @EndOfField = LEN(@WhatsLeftInText), @MoreToDo = 0
      END
    ELSE
      IF @DelimiterType = 'Complex'  --this is where it gets tricky!
        BEGIN
          SELECT  @jj = 1, @jjMax = LEN(@WhatsLeftInText), @escape = 0
          WHILE @jj <= @jjMax
            BEGIN
       IF ( SUBSTRING(@WhatsLeftInText, @jj, 1) = '"' )
        BEGIN --walk over double 'escaped' quotes
--The double quote char is sometimes replaced with a single quote or
apostrophe
         SELECT  @escape = CASE @escape
                   WHEN 1 THEN 0
                   ELSE 1
                  END
        END
       ELSE
        IF @Escape = 1
         BREAK--then it was a  quote by itself
       SELECT  @jj = @jj + 1
      END
     SELECT  @EndOfField = @jj - 1, @eat = @eat + 1
     IF @jj > @jjMax
      SELECT  @MoreToDo = 0 --reached end of file
    END
  IF @EndofField = 0
   SELECT  @EndOfField = 1--prevent invalid parameter
  IF @DelimiterType = 'RecordEnd' --The last record in a file may or
  -- may not be ended with an end of line character
   SELECT  @RecordNo = @RecordNo + 1, @FieldNo = 1,
       @StartOfRecord = @StartOfRecord + 2
  ELSE
   BEGIN
    INSERT INTO @OurTable (Field,Record,contents)
    SELECT  @FieldNo, @RecordNo,
       --turn paired quotes into single quotes
        CASE WHEN @DelimiterType = 'Complex'
          THEN REPLACE(SUBSTRING(@WhatsLeftInText, 1,
                     @EndOfField - 1), '""', '"')
          ELSE SUBSTRING(@WhatsLeftInText, 1, @EndOfField - 1)
        END
--sometimes, Non-printable characters in a field are escaped with one of
--several  character escape sequences such as \### and \o### (Octal),
-- \x## (Hex), \d### (Decimaal), and \u#### (unicode)
    SELECT  @FieldNo = @FieldNo + 1,
        @StartOfRecord = @StartOfRecord + @eat + @EndOfField - 1
   END
 END

SELECT  [name]=t1.contents,
   [phone]=t2.contents,
   [Postcode]=t3.contents,
   [Address]=t4.contents,
   [Email]=t5.contents
FROM    @ourtable t1
 INNER JOIN @ourtable t2
  ON t1.field = 1 AND t2.field = 2 AND t1.record = t2.record
 INNER JOIN @ourtable t3
  ON   t3.field = 3 AND t1.record = t3.record
 INNER JOIN @ourtable t4
  ON   t4.field = 4 AND t1.record = t4.record
 INNER JOIN @ourtable t5
  ON   t5.field = 5 AND t1.record = t5.record
```

## Unrotating a CSV Pivot-table on import

We’ll end up with one of Phil’s real life routines that is used to get daily exchange rate information for a multi-currency ecommerce site. This gets a text file which is in Comedy-limited format (comma-separated) which is gotten from the Bank of Canada’s internet site. There are several comment lines starting with a `#` character and the first non-comment line contains the headings.

```
Date
(//),10/01/2007,10/02/2007,10/03/2007,10/04/2007,10/05/2007,10/08/2007,10/09/2007
Closing Can/US Exchange Rate,0.9914,0.9976,0.9984,0.9974,0.9818,N/A,N/A
U.S. Dollar (Noon),0.9931,1.0004,0.9961,0.9983,0.9812,NA,0.9846
Argentina Peso (Floating Rate),0.3114,0.3145,0.3131,0.3123,0.3072,NA,0.3083
Australian Dollar,0.8868,0.8848,0.8846,0.8867,0.8828,NA,0.8836
..etc...
```

And we want to ‘unpivot’ it into back into a table in the format .....

```
Date                    currency                       rate
----------------------- ------------------------------ --------
2007-10-01 00:00:00.000 Closing Can/US Exchange Rate   0.991400
2007-10-01 00:00:00.000 U.S. Dollar (Noon)             0.993100
2007-10-01 00:00:00.000 Argentina Peso (Floating Rate) 0.311400
2007-10-01 00:00:00.000 Australian Dollar              0.886800
2007-10-01 00:00:00.000 Bahamian Dollar                0.993100
2007-10-01 00:00:00.000 Brazilian Real                 0.546100
2007-10-01 00:00:00.000 Chilean Peso                   0.001949
2007-10-01 00:00:00.000 Chinese Renminbi               0.132300
```

You’ll see that it is simple to start an archive of daily currency fluctuations with something like this:

To start with we will need to install `CURL` on the server. `CURL` is extraordinarily useful as a way of getting text into SQL Server from awkward places such as secure FTP sites, or simply from internet sites. Then we will need a couple of utility functions which as provided below. You’ll see how easy it is to ‘unpivot’ a pivot table back into a data table!

(this was originally in one of Phil’s blogs)

```
CREATE PROCEDURE spGetLatestCanadianExchangeRates

--allow the whereabouts of the CSV file to be specified
@WhereFrom VARCHAR(255)
='http://www.bankofcanada.ca/en/markets/csv/exchange_eng.csv'
AS
/*
Note on the exchange rates:
The daily noon exchange rates for major foreign currencies are
published every business day at about 1 p.m. EST. They are
obtained from market or official sources around noon, and show
```

```
the rates for the various currencies in Canadian dollars
converted from US dollars. The rates are nominal quotations -
neither buying nor selling rates - and are intended for
statistical or analytical purposes. Rates available from financial
institutions will differ.
*/
DECLARE @Command VARCHAR(8000)

--the command line sent to xp_cmdshell

SELECT @Command='curl -s -S "'+@wherefrom+'"'

CREATE TABLE #rawCSV (LineNumber INT IDENTITY(1,1),
    LineContents VARCHAR(8000))--for the output

INSERT INTO #rawCSV(LineContents)
    EXECUTE MASTER..xp_cmdshell @Command--get the data
--find the column headings
    --(indicator will vary from file to file)
DECLARE @Headings VARCHAR(8000)
    --the headings for the columns in the CSV file
SELECT @headings= LineContents
    FROM #rawCSV WHERE LineContents LIKE 'date %'

--and then it is one SQL Call thanks to a couple of
                 --utility functions
SELECT [Date]=CONVERT(DATETIME,item,101),
    [currency]=CONVERT(VARCHAR(50),
             dbo.ufsElement(linecontents,1,',')),
    [rate]=CONVERT(numeric(9,6),
             dbo.ufsElement(linecontents,SeqNo,',')
)
FROM
    (SELECT SeqNo,Item FROM dbo.ufsSplit(@Headings,',')
    WHERE item NOT LIKE 'Date%'
    )f--a table of the headings, with their order
CROSS JOIN
   (SELECT LineContents FROM #rawCSV WHERE lineContents NOT LIKE
'#%'
      AND lineContents NOT LIKE 'Date%')g
WHERE ISNUMERIC(dbo.ufsElement(linecontents,SeqNo,','))>0


GO

--and here are the utility functions-------------------------------

CREATE FUNCTION dbo.ufsSplit
(
@StringArray VARCHAR(8000),
@Delimiter VARCHAR(10)
)
RETURNS
@Results TABLE
  (
  SeqNo INT IDENTITY(1, 1),
  Item VARCHAR(8000)
  )
--splits a string into a table using the specified delimitor. Works like 'Split' in most
languages
--delimiters can be multi-character
AS
BEGIN

DECLARE @Next INT
DECLARE @lenStringArray INT
DECLARE @lenDelimiter INT
DECLARE @ii INT

SELECT @ii=1, @lenStringArray=LEN(
@StringArray), @lenDelimiter=LEN(@Delimiter)

WHILE @ii<=@lenStringArray
  BEGIN
  SELECT @next=CHARINDEX(@Delimiter, @StringArray + @Delimiter, @ii)
  INSERT INTO @Results (Item)
  SELECT SUBSTRING(@StringArray, @ii, @Next - @ii)
  SELECT @ii=@Next+@lenDelimiter
  END
RETURN
END

GO
-----------------------------------------------------------------------
CREATE FUNCTION dbo.ufsElement

(
@String VARCHAR(8000),
@which INT,
@Delimiter VARCHAR(10) = ','
)
--splits a string to get at the nth component in the string using the specified
delimiter
--delimiters can be multi-character
RETURNS VARCHAR(8000) AS

BEGIN
DECLARE @ii INT
DECLARE @Substring VARCHAR(8000)

SELECT @ii=1, @Substring=''

WHILE @ii <= @which
  BEGIN

  IF (@String IS NULL OR @Delimiter IS
NULL )
    BEGIN
    SELECT @Substring=''
    BREAK
    END

  IF CHARINDEX(@Delimiter,@String) = 0
    BEGIN
    SELECT @subString = @string
    SELECT @String=''
    END
  ELSE
    BEGIN
    SELECT @subString = SUBSTRING( @String, 1, CHARINDEX( @Delimiter,
@String )-1)
    SELECT @String = SUBSTRING
( @String, CHARINDEX( @Delimiter, @String )+LEN(@delimiter),LEN(@String))
  END
  SELECT @ii=@ii+1
END

RETURN (@subString)
END
```

So, we hope we've given you a few ideas on how to deal with importing text into a database without resorting to a whole lot of scripting. We've only tackled a few examples and steered clear of thorny topics such as BCP, DTS and SSIS. We'd be interested to hear of any sort of text-based format that you feel would be too hard for TSQL to deal with.

## Further Reading