Delimited File Reader

Delimited File Reader is an SSIS source component capable of parsing delimited flat files and loading them into SSIS data flows. The main difference between this parser and the one used by the Flat File source is the ability to parse rows first and fill default values for missing column fields. The data from delimited files will be parsed using two delimiters (column and row delimiter), and there will also be an option of using text qualifiers to allow using delimiters inside data fields. Text qualifiers can be escaped in data fields by repeating the text qualifiers.

# Component Properties

Here is the list of exposed component custom properties:

|  |  |  |
| --- | --- | --- |
| IsUnicode | bool | Determines whether the file is using Unicode or ANSI characters. |
| CodePage | int | Defines the code page the file is encoded with. This property is ignored if IsUnicode is set to true. |
| TextQualifier | string | Defines text qualifier. A single qualifier is used for parsing the entire delimited file. |
| HeaderRowDelimiter | string | Defines a delimiter used to separate header lines. |
| HeaderRowsToSkip | string | Defines the number of header rows to skip before starting to parse data rows. |
| ColumnNameInFirstRow | bool | Instructs the parser that the first data row contains column names. |
| ColumnDelimiter | String | Defines a global column delimiter. Individual columns can override this delimiter if multiple column delimiters are needed. |
| RowDelimiter | String | Used for defining row delimiters. It will implicitly be used as a delimiter of the last column. |
| DataRowsToSkip | int | Number of data rows to skip before starting to add them to the data flow buffers. |
| TreatEmptyStringsAsNull | bool | Whether to convert parsed, empty strings into nulls. |

All of the component properties are expression-able.

# Regular Component Output

The component will have a single data output for passing parsed data downstream. The column metadata to be parsed will be defined by output column objects. Data types of destination output columns will define necessary conversions between parsed string values and buffer types. The component will support the following data types for its output columns:

* DT\_STR
* DT\_WSTR
* DT\_I2
* DT\_I4
* DT\_R4
* DT\_R8
* DT\_CY
* DT\_DATE
* DT\_BOOL
* DT\_DECIMAL
* DT\_I1
* DT\_UI1
* DT\_UI2
* DT\_UI4
* DT\_I8
* DT\_UI8
* DT\_FILETIME
* DT\_GUID
* DT\_STR
* DT\_WSTR
* DT\_NUMERIC
* DT\_DBDATE
* DT\_DBTIME
* DT\_DBTIMESTAMP
* DT\_DBTIME2
* DT\_DBTIMESTAMPOFFSET
* DT\_TEXT
* DT\_NTEXT
* DT\_DBTIMESTAMP2

The following data types will not be supported:

* DT\_EMPTY
* DT\_NULL
* DT\_BYTES
* DT\_IMAGE
* DT\_BYREF\_...

# Error Output

The component will contain an error output to allow redirecting error rows to a separate path. The error output will contain fixed metadata defined for the following columns:

* ErrorCode
* ErrorColumn
* ErrorMessage
* ColumnData
* RowData

The first two columns are standard error output columns, and they will not be used by this component.

The ErrorMessage column will contain a detailed description of the error.

The ColumnData column will contain the parsed string from a field causing the error. If no column data is available, this column will be empty.

The RowData column will contain the original data of the row causing the error.

The component will be able to detect the following error conditions:

1. Parsing buffer overflow. There will be a limited number of characters that can be parsed into a single cell.
2. Number of columns overflow. The number of columns that can be part of a single row will be limited.
3. Data conversion and truncation errors.
4. Too many fields in a row. The number of parsed fields cannot be greater than the number of columns defined in the main output.

The first two errors would always stop the package execution. It will be possible to re-route rows with errors 3 and 4 to the error output.

# Connection

The component will use the connection manager of type FILE to manage its reference to a file.

# Parsing Logic

Rows will be parsed using two delimiters (column and row delimiters). The row ends when row delimiter is matched. If fewer fields are found in a row than the number of output columns in the component, the remaining columns will be filled with null values. If more fields are found then the number of output columns, the Row Overflow error will be raised. This error can be handled by redirecting such rows to the error output, stopping the execution, or ignoring the error. The handling of row errors is defined by the TruncationRowDisposition property of the main output. The fields can be qualified as well, and the qualifiers are defined in the TextQualifier component property.

Here are some rules used for parsing qualified fields:

* Fields are treated as qualified only if they start with a qualifier. If the field does not start with a qualifier all remaining qualifiers in the field will be treated as regular text.
* Qualifiers can be part of a qualified field if they are escaped.
* Qualifiers are escaped inside the field if they are listed twice.
* Qualified text ends when the first un-escaped qualifier is found in the field text.
* When text after the first matching qualifier is found, the end of the field (next delimiter) is treated as not qualified and eventual qualifiers in that space will be ignored.

There are fixed limitations on the size of parsed fields and number of columns in a row. Currently, the fields can have a maximum of 64K characters, and rows can have a maximum of 1024 columns. These values could eventually be changed or parameterized based on users’ feedback.

# Design-time experience

There are two ways to configure this component before it can be used to import data:

* Manually, by using the advanced component editor
* Automatically, by getting the component properties and column metadata from an existing flat file connection manager.

## Manual edits

Use the following steps to configure the component manually:

1. Right-click on its shape after it has been added to the design surface.
2. On the Connection Managers page, select an appropriate file connection.
3. On the ComponentProperties page, assign all needed properties (delimiters, qualifier, etc) and then click on the Refresh button. This will generate output columns by parsing the first data row and extracting its number of columns and their eventual names.
4. After output columns have been added, their metadata (data types, length, etc) can be defined on the Input and Output Properties page.

Error and truncation settings for output columns and truncation settings for the main output, can also be set on the Input and Output Properties page. The settings would determine whether supported errors would cause the execution to stop, redirect or ignore failing rows.

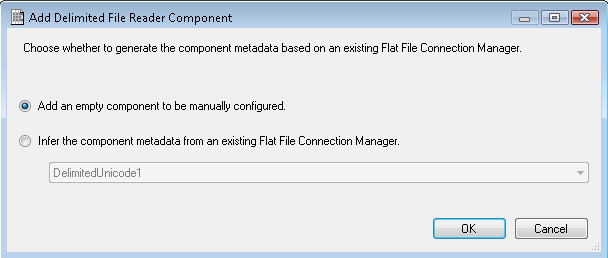
Once output columns have been added and the metadata has been defined, you cannot use the Refresh button to rebuild the columns. To rebuild the columns, remove all the output columns and then click the Refresh button again.

## Using existing Flat File Connection Managers

If your package contains an existing Flat File connection manager or you prefer to define a flat file formatted using the Flat File connection manager, you can modify the connection managers or it seems easier to define a specific flat file format in the Flat File Connection Manager editor to work with the new component (Delimited File Reader).

To modify the Flat File connection manager:

1. Make sure the package contains a flat file connection manager.
2. Add the new component to the design surface. The following dialog appears.



1. Select an appropriate Flat File connection manager by selecting the **Infer the component metadata from an existing Flat File Connection Manager** option. The flat file properties and columns are copied from the connection manager to the new component.

The new component is now ready to use.

In addition to copying the flat file properties and columns, the **Add Delimited File Reader Component** dialog also creates a new file connection and assigns it a file from the original connection manager.

Note: The custom UI is not available yet for this component. Instead of the custom UI a message box would show up and ask users to start the advanced editor. Unfortunately, there was no way to redirect this UI automatically.

# Implementation Details

There are three classes that implement the main parsing logic that this component uses: FieldParser, RowParser and DelimitedFileParser.

## FieldParser Class

The FieldParser class contains the core parsing logic. It parses the data fields using a form of a finite state machine defined for particular fields. Here are the diagrams of currently used parsing state machines:

### Single delimiter without qualifiers



This state machine is used for parsing rows with single columns (without qualifiers) or header rows.

### Single delimiter with qualifiers



This state machine is used for parsing rows with single delimited columns.

### Two delimiters without qualifiers



This state machine is used for parsing unqualified fields in regular rows with column and row delimiters.

### Two delimiters with qualifiers



This parsing state machine is used for parsing regular rows which could have column and row delimiters, and the fields can be qualified.

## RowParser Class

The responsibility of the RowParser objects are to parse all fields in a row. They use the FieldParser object to parse fields one by one until the row delimiter is found. The parsing result will be a RowData structure containing all parsed fields and the original row text (for error reporting).

## DelimitedFileParser Class

DelimitedFileParser is another parsing layer around the row parser. It will contain all relevant parsing properties in addition to delimiters and qualifiers. This class can position the row parser to the rows that will be imported into the data flow buffers by skipping header and initial data rows.

## BufferSink Class

The BufferSink class takes the RowData structure provided by RowParser and sends the data to the data flow buffers. BufferSink also handles eventual error conditions (truncations, conversion errors, etc) by reporting them or rerouting erroneous rows.

## DelimitedFileReaderComponentUI Class

This class is responsible for showing the pop-up dialog when the user drops the component to the design surface and configures the component from a selected flat file connection manager.

## DelimiterStringConverter Class

This is a type converter associated with some properties that allow customers to make specified white characters visible (\t, \r, and \n would become {t}, {CR} and {LF} respectively) and replace empty strings with <none> markers.

## FileReader Class

This class wraps the IO logic and implements services used to stream data to parsers.

## PropertiesManager Class

This is a wrapper class that handles component properties so that the logic could be better unit tested.

## DelimitedFileReaderComponent

This class implements the actual component and its attributes, and associates it with the UI class and the used icon. The component implementation uses the DelimitedFileParser and BufferSink classes for its execution-time logic (PrimeOutput implementation). The rest of the methods are supporting the design-time behavior and define some validation rules.

# Unit Testing

The source code is accompanied with over 100 unit tests. The tests are mainly focused on the parsing logic, but there are additional tests that deal with other servicing classes as well.

To run unit tests, open the provided solution in Visual Studio and activate the [Test/Run->All Tests in Solution] command.

A nice way to browse through unit tests is to group them by the classes they test. Some of the tests could be used as implicit documentation of the logic they test. They can also help in preserving the existing logic if/when you enhance the provided source code.

# Known restrictions

* Custom component UI is not available.
* Currently, the component parser will not be able to handle surrogate characters in Unicode files. This is due to limitations of the IO APIs that are used.
* There are no per-column delimiters like those used in the old flat file parser.