## What is U-SQL?

U-SQL is the new big data query language of the Azure Data Lake Analytics service.

It evolved out of Microsoft's internal Big Data language called SCOPE and combines a familiar SQL-like declarative language with the extensibility and programmability provided by C# types and the C# expression language and big data processing concepts such as “schema on reads”, custom processors and reducers. It also provides the ability to query and combine data from a variety of data sources, including Azure Data Lake Storage, Azure Blob Storage, and Azure SQL DB, Azure SQL Data Warehouse, and SQL Server instances running in Azure VMs.

It is however not ANSI SQL. For starters, its keywords such as SELECT have to be in UPPERCASE. And its expression language inside SELECT clauses, WHERE predicates etc is C#. This for example means, that the comparison operations inside a predicate follow C# syntax (e.g., a == "foo"), and that the language uses C# null semantics which is 2-valued and not 3-valued as in ANSI SQL. To help SQL users to get familiar with U-SQL, a section is providing the mapping of some common SQL expressions and how to express them in U-SQL.

**U-SQL Built in Functions**

# **Built-in Functions (U-SQL)**

While U-SQL is using C# as its expression language and thus has a large set of the Common Language Runtime (CLR) libraries and all of C#’s operators at the user’s disposal, it also adds a couple of common Built-in U-SQL Functions.

|  |  |
| --- | --- |
| **Function** | **Description** |
| **Aggregate Functions** | An aggregator will compute a single result value over a group of values and will have an identity value for the case that the group is empty. |
| **Analytic Functions** | Analytic functions compute an aggregate value based on a group of rows. However, unlike aggregate functions, they can return multiple rows for each group. |
| **Metadata Functions** | Returns information about the database and database objects. |
| **Ranking Functions** | Returns a ranking value for each row in a partition. |
| **Built-in U-SQL UDOs** | Provides extraction from and outputting to some common data formats. |

# **Aggregate Functions (U-SQL)**

U-SQL provides both built-in aggregation functions and the ability for the user to define user-defined aggregators.

An aggregator will compute a single result value over a group of values and will have an identity value for the case that the group is empty.

In U-SQL, aggregators can only be used in the following syntactic contexts:

* Inside a SELECT’s SELECT clause:
  + It will calculate the aggregated value for each group specified by a GROUP BY clause. If the rowset is empty and thus all groups are empty, the SELECT expression will return an empty rowset.
  + If no GROUP BY has been specified, then every column in the SELECT clause has to be an expression that needs to contain at least one aggregator. The aggregations will be computed across the whole rowset of the SELECT expression. If the rowset is empty, then the aggregators return their identity value, which is the value that is would not change the aggregation result if it was added. I.e., 0 for COUNT() and null for all others.
  + As part of a U-SQL windowing expression where it will calculate the aggregated value inside each window partition. Some built-in aggregators are not supported in windowing expressions (see each aggregator section for more details).
  + Aggregations can appear in expressions
  + Aggregations always need to be aliased in the SELECT clause.
* Inside a GROUP BY’s HAVING clause to provide a filter condition on the group based on the aggregation’s value.

# **ANY\_VALUE (U-SQL)**

The ANY\_VALUE aggregator arbitrarily picks one value from the group including potentially a null value. While the operation picks an arbitrary value, it does so based on the efficiency of execution and not based on some random sampling.

# **ARRAY\_AGG (U-SQL)**

The ARRAY\_AGG aggregator creates a new SQL.ARRAY value per group that will contain the values of group as its items. ARRAY\_AGG is not preserving order of values inside a group. If an array needs to be ordered, a LINQ OrderBy can be used. ARRAY\_AGG and EXPLODE are conceptually inverse operations.

# **AVG (U-SQL)**

The AVG aggregator computes the average of a set of numeric values by dividing the sum of those values by the count of nonnull values. If the sum exceeds the maximum value for the data type of the return value an error will be returned.

### Return Type

The return type is determined by the type of the evaluated result of the expression as follows:

|  |  |  |
| --- | --- | --- |
| **Expression type** | **Return type** | **Comment** |
| Integral types (short, int, uint, etc) implicitly castable to long and long? | long? | To be aligned with SQL semantics. The resulting value will be truncated to its integral part. |
| ulong, ulong? | double? | Returns double to be able to handle large values. |
| decimal, decimal? | decimal? |  |
| float, float?, double, double? | double? |  |

# **COUNT (U-SQL)**

The COUNT aggregator returns the number of items in a group. If COUNT(\*) is specified the count will include null values, if an expression or column reference is provided, then null values will not be counted

# **MAX (U-SQL)**

The MAX aggregator choses the largest value in the group or null if the expression returns only nulls in the group. The values have to be comparable. For string types, it uses a culture-invariant UTF-8 byte ordering.

# **MIN (U-SQL)**

The MIN aggregator choses the smallest value in the group or null if the expression returns only nulls in the group. The values have to be comparable. For string types, it uses a culture-invariant UTF-8 byte ordering.

# **SUM (U-SQL)**

The SUM aggregator returns the sum of all the values in the group. The input values have to be of a numeric type or an error is raised. Null values are ignored.

# **Analytic Functions (U-SQL)**

U-SQL supports a set of SQL-based analytic functions that compute an aggregate value based on a group of rows. However, unlike [aggregate functions](https://docs.microsoft.com/en-us/u-sql/functions/aggregate-functions), they can return multiple rows for each group. You can use analytic functions to compute moving averages, running totals, percentages or top-N results within a group.

In U-SQL, analytics functions can only be used in the following syntactic contexts:

* As a window function in a [windowing expression](https://docs.microsoft.com/en-us/u-sql/statements-and-expressions/select/over-expression) with the [OVER](https://docs.microsoft.com/en-us/u-sql/statements-and-expressions/select/over-expression) operator where it will calculate the value for each window partition.

Some of the analytics functions are type-polymorphic, meaning that they can operate on many different input types and return potentially different types based on their input type.

Analytics functions cannot be nested.

# **CUME\_DIST (U-SQL)**

The CUME\_DIST analytic function calculates the cumulative distribution of a value in a group of values. That is, CUME\_DIST computes the relative position of a specified value in a group of values. For a row r, assuming ascending ordering, the CUME\_DIST of r is the number of rows with values lower than or equal to the value of r, divided by the number of rows evaluated in the group. Thus, the range of values returned by CUME\_DIST is greater than 0 and less than or equal to 1. Tie values always evaluate to the same cumulative distribution value. NULL values are included by default and are treated as the highest possible values

CUME\_DIST is similar to the PERCENT\_RANK function.

# **FIRST\_VALUE (U-SQL)**

The FIRST\_VALUE analytic function returns the first value in an ordered set of values provided by the windowing expression.

# **LAST\_VALUE (U-SQL)**

The LAST\_VALUE analytic function returns the last value in an ordered set of values provided by the windowing expression.

# **PERCENT\_RANK (U-SQL)**

The PERCENT\_RANK analytic function calculates the relative rank of a row within a group of rows specified by the windowing expression.

The range of values returned by PERCENT\_RANK is greater than or equal to 0 and less than or equal to 1. The first row in any set has a PERCENT\_RANK of 0. NULL values are included by default and are treated as the highest possible values.

Use PERCENT\_RANK to evaluate the relative standing of a value within the window.

PERCENT\_RANK is similar to the CUME\_DIST function.

PERCENT\_RANK can only be used in the context of the OVER expression.

# **Metadata Functions (U-SQL)**

The following scalar function(s) return information about the database and database objects:

|  |  |
| --- | --- |
| **Function** | **Description** |
| [FILE](https://docs.microsoft.com/en-us/u-sql/functions/metadata/file-functions) | Returns certain file properties. |
| [PARTITION](https://docs.microsoft.com/en-us/u-sql/functions/metadata/partition) | Tests for the presence of a specified table partition. |

These intrinsic function(s) are currently always evaluated at compile time (and thus considered constant-foldable).

# FILE Functions (U-SQL)

# **FILE.CREATED**

Returns the creation timestamp of the file at the specified location at compile time as a DateTime value

# **FILE.EXISTS**

Checks if a file in the specified location exists at compile time.

# **FILE.LENGTH**

Returns the logical size in bytes of the file at the specified location at compile time

# **FILE.MODIFIED**

Returns the last modified timestamp of the file at the specified location at compile time as a DateTime value

# **PARTITION**

The PARTITION intrinsic object returns true if the specified partition for the given table exists and the user has access to said table. Otherwise, false is returned. The function will be evaluated at compile-time (and is thus constant-foldable).

# **Ranking Functions (U-SQL)**

U-SQL provides some built-in SQL-based ranking functions that return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows. Ranking functions are nondeterministic.

In U-SQL, ranking functions can only be used in the following syntactic contexts:

* As a window function in a windowing expression with the OVER clause where it will calculate the value for each window partition.

Ranking functions cannot be nested.

U-SQL provides the following built-in ranking functions (follow the links for more information):

# **DENSE\_RANK (U-SQL)**

The DENSE\_RANK ranking function returns the rank of rows within the partition of a window, without any gaps in the ranking. The rank of a row is one plus the number of distinct ranks that come before the row in question.

DENSE\_RANK can only be used in the context of a windowing expression.

# **NTILE**

The NTILE ranking function returns the number of the group to which the row belongs from among the groups that the windowing function has distributed the rows using an ordered partition. The groups are numbered, starting at one.

If the number of rows in a partition is not divisible by the provided integer, it will cause groups of two sizes that differ by one member. Larger groups come before smaller groups in the order specified by the OVER clause. For example, if the total number of rows is 53 and the number of groups is five, the first three groups will have 11 rows and the two remaining groups will have 10 rows each. If on the other hand the total number of rows is divisible by the number of groups, the rows will be evenly distributed among the groups. For example, if the total number of rows is 50, and there are five groups, each bucket will contain 10 rows.

# **RANK (U-SQL)**

The RANK ranking function returns the rank of each row within the window. The rank of a row is one plus the number of ranks that come before the row in question.

If two or more rows tie for a rank, each tied rows receives the same rank and the next highest row will receive the rank as if the previous rows were not tied. For example, if the three top salespeople have the same SalesYTD value, they are all ranked one. The salesperson with the next highest SalesYTD is ranked number four, because there are three rows that are ranked higher. Therefore, the RANK function does not always return consecutive integers.

The sort order that is used for the whole query determines the order in which the rows appear in a result set.

RANK can only be used in the context of a windowing expression.

# **ROW\_NUMBER (U-SQL)**

The ROW\_NUMBER ranking function returns the sequential number of a row within a window, starting at 1 for the first row in each window.

There is no guarantee that the rows returned by a query using ROW\_NUMBER will be deterministically ordered exactly the same with each execution unless all of the following conditions are true.

1. Values of the partitioned column are unique.
2. Values of the [ORDER BY](https://docs.microsoft.com/en-us/u-sql/statements-and-expressions/order-by-and-offset-fetch-clause) columns are unique.
3. Combinations of values of the partition column and [ORDER BY](https://docs.microsoft.com/en-us/u-sql/statements-and-expressions/order-by-and-offset-fetch-clause) columns are unique.

ROW\_NUMBER can only be used in the context of a [windowing expression](https://docs.microsoft.com/en-us/u-sql/statements-and-expressions/select/over-expression).

# Built-in U-SQL UDOs [User Defined Extractors]

U-SQL provides some common UDOs already built-in. currently the built-in UDOs are providing extraction from and outputting to some common data formats. More UDOs may be added over time.

# U-SQL Built-in Extractors

U-SQL provides a built-in extractor class called Extractors that provides the following three built-in extractors to generate a rowset from the input file or files:

* Extractors.Text() : Provides extraction from delimited text files of different encodings.
* Extractors.Csv() : Provides extraction from comma-separated value (CSV) files of different encodings.
* Extractors.Tsv() : Provides extraction from tab-separated value (TSV) files of different encodings.

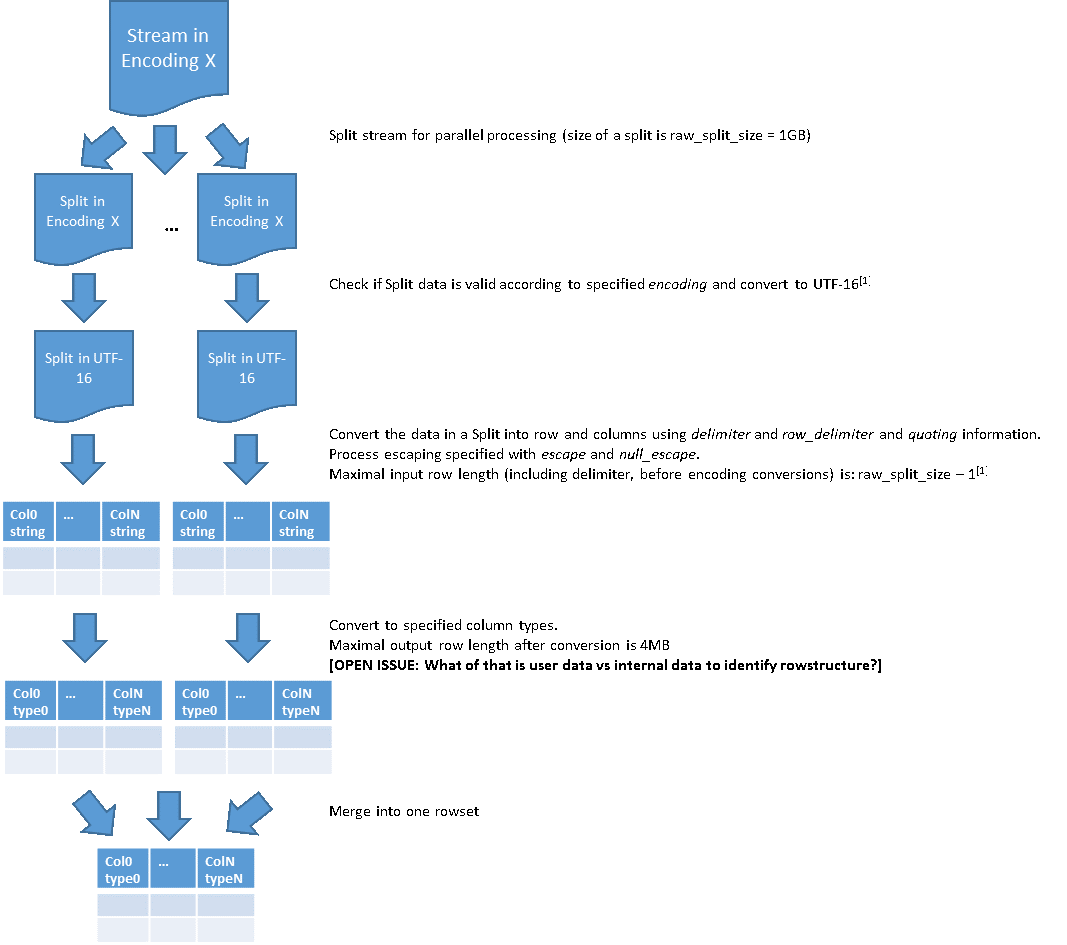
Technically speaking these are factory methods that generate an instance of the IExtractor class and they can be used in the USING clause of the EXTRACT expression. Since they create the extractor object, one does not need to call them with new.

The Csv() and Tsv() extractors are special versions of the generic Text() extractor where the delimiter has been fixed to comma and tab respectively.

If the EXTRACT expression specifies a file set pattern, then the extractor parameters will be applied to all the selected files equally. If different files require different parameter values, then different EXTRACT expressions need to be used

### **Built-in Extractor Processing Model**

The built-in extractors transforms a byte stream in parallel into a rowset that can be further processed with U-SQL statements. The following figure provides a logical view of the processing model (that in turn is based on the general UDO Extractor processing model).



If the maximal input row length or the maximal output row length are being exceeded, errors are raised.

### **Built-in Type Conversions**

The extractors will convert the string values val in the stream to an instance of the specified type in the extractor schema after the processing of encoding and escaped values have occurred.

Per default, an empty field is mapped to a zero-length string if target type is string and null otherwise. Most types follow the standard C# <Type>.Parse(val) behaviour without a specific culture behaviour and being defaulted to the cluster machine’s locale. The table below provides more details for each type.

# Extractor Parameters (U-SQL)

The supported parameters and their defaults are:

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| delimiter | char | ',' (comma) |

This parameter specifies the column separator character that separates columns in the file. The default column separator for is a comma (','). It can be any valid Unicode character including those that are represented with multi-byte encodings in any of the Unicode-Transfer-Formats (i.e. UTF-\*). The delimiter parameter is not available for Extractors.Csv() and Extractors.Tsv().

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default Value** |
| encoding | System.Text.Encoding | Encoding.UTF8 |

Per default, files are assumed to be stored in UTF-8 encoding. However, some files may be stored using a different encoding. The encoding parameter provides the option to specify the file’s actual encoding and also translate non-UTF-8 encoded files.

The supported encodings are:

|  |  |
| --- | --- |
| **Encoding Property** | **Description** |
| Encoding.[ASCII] | Assumes that the file is encoded using the ASCII (7-bit) character set. Since ASCII is an all uppercase keyword, it needs to be quoted to not conflict with U-SQL’s reserved keyword rule. |
| Encoding.BigEndianUnicode | Assumes that the file is encoded using the UTF-16 format that uses the big endian byte order. |
| Encoding.Unicode | Assumes that the file is encoded using the UTF-16 format using the little endian byte order. |
| Encoding.UTF7 | Assumes that the file is encoded using the UTF-7 format. |
| **Encoding.UTF8** | Assumes that the file is encoded using the UTF-8 format. This is the default. |
| Encoding.UTF32 | Assumes that the file is encoded using the UTF-32 format using the little endian byte order. |

If the file contains a code point that is invalid for the specified encoding, a runtime error during extraction will occur regardless of the silent parameter setting that will indicate the offending data value.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| escapeCharacter | char? | *null* |

The escapeCharacter parameter – if not set to null – specifies the character in the file that is used to escape itself and all delimiter values in the file.

If the escape character is followed by a value other than itself or any of the delimiter values, the escape character is dropped when reading the value.

If it set to null, there is no escape character specified.

The escapeCharacter parameter will be applied regarding of whether quoting is enabled or not. It however will not be used to escape the quoting character. The quoting character will get escaped with double-quotes in alignment with the Excel CSV behaviour.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| quoting | bool | true |

The quoting parameter if set to true indicates that the extractor should consider " (double-quote) as a column field quotation which inhibits the interpretation of the column delimiters inside the quoted field. A double-quote inside a quoted field needs to be escaped by doubling it as "" in the file.

If the input data does not use quoting, specifying false for the quoting parameter will be resulting in a faster extraction.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default values** |
| rowDelimiter | string (max length=1) | "\r\n" (carriage return, linefeed) "\r" (carriage return) "\n" (linefeed) |

This parameter specifies the row separator character sequence that separates rows in the file. If the rowDelimiter is set to null, the default values are being used which are carriage return followed by linefeed, or carriage return or linefeed.

If the length of the rowDelimiter is more than one UCS-4 characters, an error is raised\*\*.\*\*

The comparison of the rowDelimiter will be done using a byte-wise comparison after applying the encoding got applied.

Note that the rowDelimiter character inside a quoted string will not be escaped and will be used as a row separator which will lead to incorrect or failing extractions.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default Value** |
| silent | bool | false |

This parameter tells the extractor to ignore and skip rows that have a different number of columns than the requested number of columns. It also replaces corrupt column data with null when the datatype is nullable (for example, it replaces "int?" with null when there is a non-digit character in the column). If the column type is not nullable, an error is raised.

Rows are checked for the correct number of columns first. For example, if there is corrupt data in a column with the wrong number of rows, the extractor skips the row. There is no conversion. If the row has the correct number of columns, however, the extractor doesn't skip it. The conversion occurs (assuming that the column's type is nullable).

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default Value** |
| skipFirstNRows | Int | 0 |

The parameter specifies the number of rows to skip. The rows being skipped do not need to conform to the column schema in either type or count of columns. Note, you can only skip rows that are in the first segment of a file; otherwise, an error will be raised.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default Value** |
| charFormat | string | "uint16" |

The supported values are:

|  |  |
| --- | --- |
| **Value** | **Description** |
| uint16 or null | This is the default value. Serializes the char value as an integral number (taking all other serialization options into account) and parses the input as the integral character code number to the corresponding character or errors if the input is not an integral character code (or mapable null if extracting it as char?). |
| string | Serializes the char value in its Unicode string representation (taking all other serialization options including encoding into account) and parses the input as the character codepoint using the specified encoding. |

# Extractors.Text()

The Text() extractor supports a variety of text file formats that all follow a row/column format. It provides a set of delimiters to identify the row and column boundaries and several other parameters to parse the text file and produces a rowset based on the EXTRACT expression’s schema.

e.g.

@searchlog =

EXTRACT UserId int,

Start DateTime,

Region string,

Query string,

Duration int?,

Urls string,

ClickedUrls string

FROM "/Samples/Data/SearchLog.tsv"

USING Extractors.Text(delimiter: '\t', skipFirstNRows: 1);

OUTPUT @searchlog

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorText\_SearchLog.csv"

USING Outputters.Csv();

@Drivers =

EXTRACT driver\_id int,

name string,

street string,

city string,

region string,

zipcode string,

country string,

phone\_numbers string // Map

FROM "/Samples/Data/AmbulanceData/Drivers.txt"

USING Extractors.Text(delimiter: '\t', encoding:Encoding.Unicode);

OUTPUT @Drivers

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorText\_Drivers.csv"

USING Outputters.Csv();

// You need to quote ASCII with [] to make sure it is not read as a reserved U-SQL keyword

@Trips =

EXTRACT date DateTime,

driver\_id int,

vehicle\_id int,

trips string // Array

FROM "/Samples/Data/AmbulanceData/DriverShiftTrips.csv"

USING Extractors.Text(encoding: Encoding.[ASCII]);

OUTPUT @Trips

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorText\_DriverShiftTrips.csv"

USING Outputters.Csv();

# **Extractors.Csv()**

The Csv() extractor disallows the delimiter parameter and defaults the field delimiter to ',' (comma). All other parameters are the same. See Extractor Parameters (U-SQL) for all supported parameters and their defaults values.

// Excel CSV (ANSI with ASCII only)

// You need to quote ASCII with [] to make sure it is not read as a reserved U-SQL keyword

@Trips =

EXTRACT date DateTime,

driver\_id int,

vehicle\_id int,

trips string // Array

FROM "/Samples/Data/AmbulanceData/DriverShiftTrips.csv"

USING Extractors.Csv(encoding: Encoding.[ASCII]);

OUTPUT @Trips

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorCsv\_DriverShiftTrips.csv"

USING Outputters.Csv();

# **Extractors.Tsv()**

The Tsv() extractor disallows the delimiter parameter and defaults the field delimiter to '\t' (tab). All other parameters are the same. See Extractor Parameters (U-SQL) for supported parameters and their defaults values.

@searchlog =

EXTRACT UserId int,

Start DateTime,

Region string,

Query string,

Duration int?,

Urls string,

ClickedUrls string

FROM "/Samples/Data/SearchLog.tsv"

USING Extractors.Tsv(skipFirstNRows: 1);

OUTPUT @searchlog

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorTsv\_SearchLog.csv"

USING Outputters.Csv();

// TAB Separated Unicode file

@Drivers =

EXTRACT driver\_id int,

name string,

street string,

city string,

region string,

zipcode string,

country string,

phone\_numbers string // Map

FROM "/Samples/Data/AmbulanceData/Drivers.txt"

USING Extractors.Tsv(encoding:Encoding.Unicode);

OUTPUT @Drivers

TO "/Output/ReferenceGuide/BuiltIn/UDOs/extractorTsv\_Drivers.csv"

USING Outputters.Csv();

# **U-SQL Built-in Outputters**

U-SQL provides a built-in outputter class called Outputters that provides the following three built-in outputters to transform a rowset into a file or set of files:

Outputters.Text(): Provides outputting a rowset into a variety of delimited text formats.

Outputters.Csv(): Provides outputting a rowset into a comma-separated value (CSV) file of different encodings.

Outputters.Tsv(): Provides outputting a rowset into a tab-separated value (TSV) file of different encodings.

The Csv() and Tsv() outputters are special versions of the generic Text() outputter where the delimiter has been fixed to comma and tab respectively.

The built-in outputters serialize a rowset by serializing the individual data types, using a set of delimiters to identify the row and column boundaries and encode certain values according to the set parameters.

Technically speaking these are factory methods that generate an instance of the IOutputter class and they can be used in the USING clause of the OUTPUT statement. Since they create the outputter object, one does not need to call them with new.

If the OUTPUT statement specifies a file set pattern, then the outputter parameters will be applied to all the selected files equally. If different files require different parameter values, then different OUTPUT statements need to be used.

# Outputter Parameters (U-SQL)

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| delimiter | char | ',' |

This parameter specifies the column separator character that separates columns in the file. The default column separator is ',' (comma). The delimiter parameter is not available for [Outputters.Csv()](https://docs.microsoft.com/en-us/u-sql/functions/operators/outputters/outputters-csv) and [Outputters.Tsv()](https://docs.microsoft.com/en-us/u-sql/functions/operators/outputters/outputters-tsv).

Note that per default, the built-in outputters are quoting string values. Thus any delimiter inside a value will be protected. If quoting is turned off, then the escaping needs to be turned on to protect the delimiter character inside a value.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| dateTimeFormat | string | "O" |

The dateTimeFormat parameter sets the dateTime format used when writing a datetime typed column with the built-in outputters. The argument value corresponds to the valid C# formats and defaults to "O" to provide round tripping of values.

If an invalid argument value has been specified, an error will be raised.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default Value** |
| encoding | System.Text.Encoding | Encoding.UTF8 |

Per default, files are assumed to be stored in UTF-8 encoding. However, some files may need to be stored using a different encoding. Since C#’s string types are always UTF-16 encoded, we need to have the ability to convert a string type’s UTF-16 encoding to the encoding used to store it. The encoding parameter gives us the ability to specify the output file’s actual encoding.

|  |  |  |
| --- | --- | --- |
| **Parameter name** | **Parameter type** | **Default value** |
| quoting | bool | true |

This parameter specifies the row separator character sequence that separates rows in the file. If the rowDelimiter is set to null, the default value is being used which is a carriage return followed by linefeed.

If the length of the provided rowDelimiter is more than one UCS-4 character, an error is raised\*\*.\*\*

Note that unless the escapeCharacter parameter is set, the rowDelimiter character inside a value is being written without escaping and may lead to incorrect or failing extractions.

# **Outputters.Text()**

The Text() outputter supports a variety of text file formats that all follow a row/column format. It provides a set of delimiters to identify the row and column boundaries and several other parameters to transform the rowset values into the text file.

# **Outputters.Csv()**

The Csv() outputter disallows the delimiter parameter and defaults the field delimiter to ',' (comma). All other parameters are the same.

# **Outputters.Tsv()**

The Tsv() outputter disallows the delimiter parameter and defaults the field delimiter to '\t' (tab). All other parameters are the same. See [Outputter Parameters (U-SQL)](https://docs.microsoft.com/en-us/u-sql/functions/operators/outputters/outputter-parameters) for supported parameters and their defaults values.