SQL#

Expanding the capabilities of T-SQL

Version **4.2.100 / 4.2.101**
November 13th, 2018
(doc. rev. 20181113)

Sql Quantum Lift

SQLsharp.com

Table of Contents

General Information

Copyrights

JsonFx:
The MIT License

Introduction

Welcome to SQL# (SQLsharp). SQL# is a .NET / CLR library that resides in a SQL Server 2005 (or newer) database and provides a suite of User-Defined Functions, Stored Procedures, User-Defined Aggregates, and User-Defined Types. This set of tools is designed to make the lives of countless SQL Server professionals easier by providing the broad range of commands available in most other languages outside of SQL (we will not speak of JCL or IBM's horrendous Net.Data). The User-Defined Functions and Stored Procedures are all prefixed with a "category" name much like the class names that you would find in C#, Java / J#, C++, VB.Net, etc. The names of the commands are in mixed case but SQL Server is not typically case-sensitive so it will not matter if you use capitals or not (unless the Database you install SQL# into has a binary or case-sensitive default Collation). Most of the functions will work in SAFE mode (this refers to security setting of the SQL# Assemblies within Microsoft SQL Server). If you want to use the functions that access anything outside of SQL Server (e.g. the Internet, the file system, the OS, etc), then you will have to change the security setting of the SQL# Assembly that has the function(s) that you wish to use. The security setting will need to be *at least* EXTERNAL_ACCESS or maybe even UNRESTRICTED (very few functions require this security level). See details on SQLsharp_SetSecurity for more information on security levels.

Requirements

1) Microsoft SQL Server 2005 (with SP3) or newer
2) CLR Integration must be enabled (this will be done by the setup script if not done already)

Contact Information

Product Website: https://SQLsharp.com/

Company Website: https://SqlQuantumLift.com/

Suggestions for improvements, Feedback: https://SQLsharp.com/contact/

Problems to report / Questions: check the website at https://SQLsharp.com/faq/ or use the contact form at: https://SQLsharp.com/contact/

Installation and Updating

Installation / Setup

If you do not already have "CLR Integration" enabled (it is disabled by default when creating a new SQL Server Instance), then it will be enabled automagically by the setup SQL script.

1) If you do not have the install SQL script for SQL#, or if you want the most recent version, then obtain the current installation SQL script from the SQL# website at: https://SQLsharp.com/
2) Save the SQL script in case you need it later or need to install on more than one machine.
3) Open the SQL script (named SQLsharp_Setup.sql) in SQL Server Management Studio.
4) Be sure to edit the USE statement just under the header comment block by replacing the `{replace_with_DB_name}` with the name of the target Database.
5) Add two dashes / hyphens just to the right of the "];" after the DB name to comment out the comment.
6) If you do not want to install one of the optional Assemblies – SQL#_2, SQL#.DB, SQL#.DotNetZip (Full Version only), SQL#.FileSystem (Full Version only), SQL#.JsonFx, SQL#.Network, SQL#.OS, SQL#.SgmlReader, SQL#.Twitterizer, SQL#.TypesAndAggregates, SQL#.TypesAndAggregatesPlus, and SQL#.XML – just update the variables just below the USE statement to equal 0, each of which follows this form:
   `SET @InstallSQL#{AssemblyName} = 1;`
7) The script will create an Asymmetric Key and a Key-based Login (both in [master]). The Login provides the ability to set any of the SQL# Assemblies to a security level other than SAFE without needing to set the Database to TRUSTWORTHY ON. It will also be the owner of the SQL# Assemblies which will create an App Domain separate from any other potential SQLCLR code that would be owned by "dbo". Whether or not any of the SQL# Assemblies can be set to either External Access or Unrestricted Access depends on what permission this SQL# Login is granted, and that is determined by the setting of the @MaxAllowedAccessLevel variable (just below the USE statement). Please see the comments immediately following the "SET @MaxAllowedAccessLevel =" line for a detailed description of each value.
8) Click on !Execute OR press F5 OR press Control-E (so many choices!) to run the script, and it will display status information in the "Messages" tab.
9) All SQL# Functions, Stored Procedures, and User-Defined Types reside in the SQL# Schema, so you might need to GRANT permissions to any Users or Roles that will be using them via:
   `EXEC SQL#.SQLsharp_GrantPermissions N'UserName_A, Role_B';`
   Note: this is either a single name or a comma-separated list.
10) Enjoy!

Updating

Internally (not available in Free version)

This method will download the most current SQL# installer from the SQLsharp.com website (no information from your computer is transmitted to the site, just the version number being upgraded), and save it in the location that you specify.

1) Make sure that SQLsharp is allowed access to Internet resources.
   a. You can see the current setting by running:
      `EXEC SQL#.SQLsharp_SetSecurity 0;`
   b. If the value of "permission_set" is 1, then run:
      `EXEC SQL#.SQLsharp_SetSecurity 2;`

2) Execute the download Stored Procedure:
   ```
   EXEC SQL#.SQLsharp_Download N'{LicenseKey}', N'{Download\Path[\File.ext]}';
   ```
3) You will see status information displayed in the Messages tab.
4) If you previously had a "permission_set" of 1 and would like to change it back to that, then run:
   ```
   EXEC SQL#.SQLsharp_SetSecurity 1;
   ```

Externally

This method is not preferred as it is not as easy (or cool!) as the internal method using the SQLsharp_Download Stored Procedure. But if you must, then it does happen to work:

1) Obtain the current installation SQL script from the SQL# website at:
   a. Free version: https://SQLsharp.com/free/
   b. Full version: https://SQLsharp.com/full/
2) Save the SQL script in case you need it later or need to install on more than one machine.
3) You do not need to uninstall the previous version as that will be done by the install script
4) Open the SQL script in SQL Server Management Studio.
5) Follow the general installation / setup instructions, noted above, starting at Step 4.
6) Assemblies that have been set to security levels higher than SAFE will be reset to the current security level (unless it is the main SQL# Assembly, in which case it will always be reset to SAFE as of SQL# Version 4.0).

Function, Stored Procedure, Type Reference

These functions are prefixed with a category name to give a logical separation as well as to provide a name-space (for example, there are two Split functions, one in String and one in RegEx). Functions and Stored Procedures are in the main SQL# Assembly unless specified otherwise in each section.

Regular Expressions (RegEx)

For ALL RegEx functions, the @StartAt input parameter specifies the absolute location in the string @ExpressionToValidate to start at. The first position in a string in SQL Server is 1 and that holds true with all of these SQL# functions; they do NOT start at position 0 (zero). The syntax for a Regular Expression can be found at Microsoft's MSDN site (https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference) as well as right here:

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and "\(" matches "(".
^	Matches the position at the beginning of the input string. If the **RegExp** object's **Multiline** property is set, ^ also matches the position following '\n' or '\r'.
$	Matches the position at the end of the input string. If the **RegExp** object's **Multiline** property is set, $ also matches the position preceding '\n' or '\r'.
*	Matches the preceding character or subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
+	Matches the preceding character or subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
?	Matches the preceding character or subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
{*n*}	*N* is a nonnegative integer. Matches exactly *n* times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
{*n*,}	*N* is a nonnegative integer. Matches at least *n* times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "foooood". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
{*n*,*m*}	*M* and *n* are nonnegative integers, where *n* <= *m*. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.
?	When this character immediately follows any of the other quantifiers (*, +, ?, {*n*}, {*n*,}, {*n*,*m*}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", 'o+?' matches a single "o", while 'o+' matches all 'o's.
.	Matches any single character except "\n". To match any character including the '\n', use a

	pattern such as '[\s\S]'.
(*pattern*)	A subexpression that matches *pattern* and captures the match. The captured match can be retrieved from the resulting Matches collection using the **$0…$9** properties. To match parentheses characters ( ), use '\(' or '\)'.
(?:*pattern*)	A subexpression that matches *pattern* but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (|). For example, 'industr(?:y|ies) is a more economical expression than 'industry|industries'.
(?=*pattern*)	A subexpression that performs a positive lookahead search, which matches the string at any point where a string matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?=95|98|NT|2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
(?!*pattern*)	A subexpression that performs a negative lookahead search, which matches the search string at any point where a string not matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95|98|NT|2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
x|*y*	Matches either *x* or *y*. For example, 'z|food' matches "z" or "food". '(z|f)ood' matches "zood" or "food".
[*xyz*]	A character set. Matches any one of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
[^*xyz*]	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
[*a-z*]	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
[^*a-z*]	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
\b	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
\B	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".
\c*x*	Matches the control character indicated by *x*. For example, \cM matches a Control-M or carriage return character. The value of *x* must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character. Equivalent to \x0c and \cL.
\n	Matches a newline character. Equivalent to \x0a and \cJ.
\r	Matches a carriage return character. Equivalent to \x0d and \cM.

\s	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
\S	Matches any non-white space character. Equivalent to [^ \f\n\r\t\v].
\t	Matches a tab character. Equivalent to \x09 and \cI.
\v	Matches a vertical tab character. Equivalent to \x0b and \cK.
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'.
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'.
\x*n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, '\x41' matches "A". '\x041' is equivalent to '\x04' & "1". Allows ASCII codes to be used in regular expressions.
\num	Matches *num*, where *num* is a positive integer. A reference back to captured matches. For example, '(.)\1' matches two consecutive identical characters.
\n	Identifies either an octal escape value or a backreference. If \n is preceded by at least *n* captured subexpressions, *n* is a backreference. Otherwise, *n* is an octal escape value if *n* is an octal digit (0-7).
\nm	Identifies either an octal escape value or a backreference. If \nm is preceded by at least *nm* captured subexpressions, *nm* is a backreference. If \nm is preceded by at least *n* captures, *n* is a backreference followed by literal *m*. If neither of the preceding conditions exists, \nm matches octal escape value *nm* when *n* and *m* are octal digits (0-7).
\nml	Matches octal escape value *nml* when *n* is an octal digit (0-3) and *m* and *l* are octal digits (0-7).
\u*n*	Matches *n*, where *n* is a Unicode character expressed as four hexadecimal digits. For example, \u00A9 matches the copyright symbol (©).

Additional syntax found at:

- Regular Expression Language - Quick Reference (https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference)
- http://www.regular-expressions.info/reference.html
- http://www.regular-expressions.info/refadv.html
- Be sure to review the .NET-specific syntax noted here:
  - http://www.regular-expressions.info/refext.html
  - http://www.regular-expressions.info/refflavors.html
  - http://www.regular-expressions.info/refreplace.html

RegEx Options

Also, for all RegEx functions the RegExOptionsList parameter is a pipe-separated list of options that can be specified in any combination to control how the Regular Expression pattern matching is performed. You can pass in NULL or empty string ‘’ for “none”. Values are NOT case-sensitive. The valid values are:

Compiled	Specifies that the regular expression is compiled to an assembly. This yields faster execution but increases startup time. Please note that the compiled definition stays in memory until the App Domain is unloaded.
CultureInvariant	Specifies that cultural differences in language are ignored. Ordinarily, the regular expression engine performs string comparisons based on the conventions of the current culture. If the **CultureInvariant** option is specified, it uses the conventions of the invariant culture.

ECMAScript	Enables ECMAScript-compliant behavior for the expression. This value can be used only in conjunction with the **IgnoreCase** and **Multiline** values. The use of this value with any other values results in an exception.
ExplicitCapture	Specifies that the only valid captures are explicitly named or numbered groups of the form (?<name>…). This allows unnamed parentheses to act as noncapturing groups without the syntactic clumsiness of the expression (?:…).
IgnoreCase	Specifies case-insensitive matching.
IgnorePatternWhitespace	Eliminates unescaped white space from the pattern and enables comments marked with #. However, the **IgnorePatternWhitespace** value does not affect or eliminate white space in character classes
Multiline	Multiline mode. Changes the meaning of ^ and $ so they match at the beginning and end, respectively, of any line, and not just the beginning and end of the entire string.
RightToLeft	Specifies that the search will be from right to left instead of from left to right.
Singleline	Specifies single-line mode. Changes the meaning of the dot (.) so it matches every character (instead of every character except \n).

Examples:
'IgnoreCase' or 'ignorecase|CultureInvariant'

RegEx_CaptureGroup

RegEx_CaptureGroup(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), CaptureGroupNumber INT, NotFoundReplacement NVARCHAR(MAX), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- CaptureGroupNumber >= 0
- CaptureGroupNumber of 0 returns the whole capture
- NotFoundReplacement can be empty string ‘’, any value, or NULL
- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Length of -1 = Search until the end of the ExpressionToValidate
- If input data will *never* be over 4000 characters, please use RegEx_CaptureGroup4k as that function offers better performance.

EXAMPLES:
```
SELECT SQL#.RegEx_CaptureGroup('there were 123 web errors + 5 ftp errors',
'(\d+) (web|ftp) errors', 2, NULL, 1, -1, '')
-- web
SELECT SQL#.RegEx_CaptureGroup('there were 123 web errors + 5 ftp errors',
'(\d+) (web|ftp) errors', 2, NULL, 1, 8, '')
-- NULL
SELECT SQL#.RegEx_CaptureGroup('there were 123 web errors + 5 ftp errors',
'(\d+) (web|ftp) errors', 2, NULL, 15, -1, '')
-- ftp
SELECT SQL#.RegEx_CaptureGroup('errors', '(\d+) (web|ftp) errors', 2, 'Not
Found', 1, -1, '')
-- Not Found
```

RegEx_CaptureGroup4k

RegEx_CaptureGroup4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), CaptureGroupNumber INT, NotFoundReplacement NVARCHAR(4000), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to RegEx_CaptureGroup except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_CaptureGroup when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_CaptureGroupCapture

RegEx_CaptureGroupCapture(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), CaptureGroupNumber INT, MatchNumber INT, CaptureNumber INT, NotFoundReplacement NVARCHAR(MAX), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- {will add details later}

RegEx_CaptureGroupCapture4k

RegEx_CaptureGroupCapture4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), CaptureGroupNumber INT, MatchNumber INT, CaptureNumber INT, NotFoundReplacement NVARCHAR(4000), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to RegEx_CaptureGroupCapture except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_CaptureGroupCapture when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_CaptureGroupCaptures (Not available in Free version)

RegEx_CaptureGroupCaptures(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, GroupNum INT, CaptureNum INT, Value NVARCHAR(MAX), StartPos INT, EndPos INT, Length INT)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Please also see explanation after the example code below for additional details

EXAMPLES:
```
DECLARE @Test NVARCHAR(500),
        @Expression NVARCHAR(500);
SET @Test = N'A123R2Z4B89:blue,red,green;Y999Q3:C,M,Y,K';
SET @Expression = N'(?:([A-M]\d+)|([N-Z]\d+))+:(?:(\w+),?)+(?:;|$)';

SELECT * FROM SQL#.RegEx_CaptureGroupCaptures(@Test, @Expression, 1, NULL);
SELECT * FROM SQL#.RegEx_CaptureGroups(@Test, @Expression, 1, NULL);
/*
MatchNum    GroupNum    CaptureNum    Value    StartPos    EndPos    Length
1           1           1             A123      1           4        4
1           1           2             B89       9          11        3
1           2           1             R2        5           6        2
1           2           2             Z4        7           8        2
1           3           1             blue     13          16        4
1           3           2             red      18          20        3
1           3           3             green    22          26        5
2           2           1             Y999     28          31        4
2           2           2             Q3       32          33        2
2           3           1             C        35          35        1
2           3           2             M        37          37        1
2           3           3             Y        39          39        1
2           3           4             K        41          41        1


MatchNum    GroupNum                  Value    StartPos    EndPos    Length
1           1                         B89       9          11        3
1           2                         Z4        7           8        2
1           3                         green    22          26        5
2           1                                   1           0        0
2           2                         Q3       32          33        2
2           3                         K        41          41        1
*/
```

For the example above:

- "(?: …)" is a non-capturing group. This is why the ";" and "," do not get captured, but capturing groups within non-capturing groups can still capture as expected.
- "([A-M]\d+)|([N-Z]\d+)" is two groups:
  - Group 1 is single character A-M followed by 1 or more decimal digits
  - Group 2 is single character N-Z followed by 1 or more decimal digits
  - "|" is typical "or", so capture group 1 XOR group 2, not both
- "+" means "one or more", so "(?: ...)+" means get 1 or more of whatever is in that grouping
- "\w" is a word character, so "\w+" means "one or more word characters"
- "(?:(\w+),?)+" captures "one or more groupings of one or more word characters (being Group 3) followed by 0 or 1 commas". The comma, if present, is not captured as it is inside of a non-capturing but outside of the capturing group
- "(?:;|$)" is a non-capturing group stating that the match ends with either a ";" or "end-of-input"
- The output shows that:
  - CaptureGroupCaptures works for repeating (captured) groups within one or more matches. In contrast, CaptureGroups can handle one or more matches, but doesn't handle repeating groups within a match; it can only show the value for the last instance of a group within a match.
  - CaptureGroupCaptures does not return rows for groups that did not match anything (this is why there is no row for GroupNum 1 in MatchNum 2). In contract, CaptureGroups always returns a row for every defined capturing group, whether or not it captured anything (this is why there is a row for GroupNum 1 in MatchNum 2, but "EndPos" is "0").

RegEx_CaptureGroupCaptures4k (Not available in Free version)

RegEx_CaptureGroupCaptures4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, GroupNum INT, CaptureNum INT, Value NVARCHAR(4000), StartPos INT, EndPos INT, Length INT)

NOTES:

- Functionally equivalent to RegEx_CaptureGroupCaptures except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_CaptureGroupCaptures when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_CaptureGroups (Not available in Free version)

RegEx_CaptureGroups(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, GroupNum INT, Value NVARCHAR(MAX), StartPos INT, EndPos INT, Length INT)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate

EXAMPLES:

```
SELECT * FROM SQL#.RegEx_CaptureGroups('phone: 800-555-1212', '((\d{3})-(\d{3})-
(\d{4}))', 1, 'IgnoreCase')
/*
MatchNum    GroupNum    Value             StartPos    EndPos      Length
1           1           800-555-1212      8           19          12
1           2           800               8           10          3
1           3           555               12          14          3
1           4           1212              16          19          4
*/

SELECT * FROM SQL#.RegEx_CaptureGroups('phone: 800-555-1212 and 800-555-5555',
'((\d{3})-(\d{3})-(\d{4}))', 1, NULL)
/*
MatchNum    GroupNum    Value             StartPos    EndPos      Length
1           1           800-555-1212      8           19          12
1           2           800               8           10          3
1           3           555               12          14          3
1           4           1212              16          19          4
2           1           800-555-5555      25          36          12
2           2           800               25          27          3
2           3           555               29          31          3
2           4           5555              33          36          4
*/
```

RegEx_CaptureGroups4k (Not available in Free version)

RegEx_CaptureGroups4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, GroupNum INT, Value NVARCHAR(4000), StartPos INT, EndPos INT, Length INT)

NOTES:

- Functionally equivalent to RegEx_CaptureGroups except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_CaptureGroups when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_Escape

RegEx_Escape(ExpressionToEscape NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Escapes a minimal set of metacharacters (\, *, +, ?, |, {, [, (,), ^, $,., #, and white space) by replacing them with their escape codes.

NOTES:

- See also: RegEx_Unescape

EXAMPLES:

```
SELECT SQL#.RegEx_Escape('test(*.)')
-- test\(\*\.\)
```

RegEx_Escape4k

RegEx_Escape4k(ExpressionToEscape NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Escapes a minimal set of metacharacters (\, *, +, ?, |, {, [, (,), ^, $,., #, and white space) by replacing them with their escape codes.

NOTES:

- Functionally equivalent to RegEx_Escape except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_Escape when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_GetCacheSize

RegEx_GetCacheSize()

RETURNS: INT

Returns the current number of Regular Expression patterns that *can* be cached by the App Domain.

NOTES:

- Default cache size in .NET is 15
- See also: RegEx_SetCacheSize

RegEx_Index

RegEx_Index(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: INT

Returns the location of the first character of the first match that is found starting at StartAt.

NOTES:

- StartAt:
  - Must be >= 1
  - cannot be greater than the length of ExpressionToValidate
- Length:
  - -1 = search entire ExpressionToValidate
  - 0 always returns 0
  - cannot be greater than the length of ExpressionToValidate

EXAMPLES:

```
SELECT SQL#.RegEx_Index('thisAA is a isA fisAA bob', 'is[A]{2}', 1, 0, '')
-- 3
SELECT SQL#.RegEx_Index('thisAA is a isA fisAA bob', 'is[A]{2}', 4, 0, '')
-- 18
SELECT SQL#.RegEx_Index('thisAA is a isA fisAA bob', 'is[A]{2}', 4, 2, '')
-- 0
```

RegEx_Index4k

RegEx_Index4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: INT

Returns the location of the first character of the first match that is found starting at StartAt.

NOTES:

- Functionally equivalent to RegEx_Index except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_Index when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_IsMatch

RegEx_IsMatch(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: BIT

NOTES:

- StartAt
  - Must be >= 1
  - Cannot be greater than the length of ExpressionToValidate
- If input data will *never* be over 4000 characters, please use RegEx_IsMatch4k as that function offers better performance.

EXAMPLES:

```
SELECT SQL#.RegEx_IsMatch('zo', 'zo{2}', 1, '')
-- 0
SELECT SQL#.RegEx_IsMatch('zoo', 'zo{2}', 1, '')
-- 1
SELECT SQL#.RegEx_IsMatch('Zoo', 'zo{2}', 1, '')
-- 0
SELECT SQL#.RegEx_IsMatch('Zoo', 'zo{2}', 1, 'IgnoreCase')
-- 1
```

RegEx_IsMatch4k

RegEx_IsMatch4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: BIT

NOTES:

- Functionally equivalent to RegEx_IsMatch except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_IsMatch when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_Match

RegEx_Match(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, Value NVARCHAR(MAX), StartPos INT, EndPos INT, Length INT)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Returns the FIRST match; will only return 1 row

EXAMPLES:

```
SELECT * FROM SQL#.RegEx_Match('This is a test that shows matching', 'th.{2}',
1, '')
--MatchNum  Value StartPos    EndPos      Length
--1         that  16          19          4
SELECT * FROM SQL#.RegEx_Match('This is a test that shows matching', 'th.{2}',
1, 'IgnoreCase')
--MatchNum  Value StartPos    EndPos      Length
--1         This  1           4           4
```

RegEx_Matches

RegEx_Matches(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, Value NVARCHAR(MAX), StartPos INT, EndPos INT, Length INT)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Same as RegEx_Match but returns all matches

EXAMPLES:
```
SELECT * FROM SQL#.RegEx_Matches('This is a test that shows matching', 'th.{2}',
1, 'IgnoreCase')
--MatchNum  Value StartPos    EndPos      Length
--1         This  1           4           4
--2         that  16          19          4
```

RegEx_MatchLength

RegEx_MatchLength(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

Returns the first match that is found starting at StartAt but only looking as far as Length characters rather than until the end of the ExpressionToValidate.

NOTES:

- @StartAt
  - Must be >= 1
  - Cannot be greater than the length of ExpressionToValidate
- @Length
  - Must be >= 0
  - Cannot be greater than the length of ExpressionToValidate
  - 0 will return an empty string ''

EXAMPLES:
```
SELECT SQL#.RegEx_MatchLength('This is a test that shows matching', 'th.{2}', 1,
20, '')
-- that
SELECT SQL#.RegEx_MatchLength('This is a test that shows matching', 'th.{2}', 1,
10, '')
-- (empty string)
SELECT SQL#.RegEx_MatchLength('This is a test that shows matching', 'th.{2}',
10, 10, '')
-- that
```

RegEx_MatchLength4k

RegEx_MatchLength4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, Length INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Returns the first match that is found starting at StartAt but only looking as far as Length characters rather than until the end of the ExpressionToValidate.

NOTES:

- Functionally equivalent to RegEx_MatchLength except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_MatchLength when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_MatchSimple

RegEx_MatchSimple(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Returns the first match that is found starting at StartAt but unlike RegEx_MatchLength it will search until the end of the ExpressionToValidate
- If input data will *never* be over 4000 characters, please use RegEx_MatchSimple4k as that function offers better performance.

EXAMPLES:

```
SELECT SQL#.RegEx_MatchSimple('This is a test that shows matching', 'th.{2}', 1,
'IgnoreCase')
-- This
SELECT SQL#.RegEx_MatchSimple('This is a test that shows matching', 'th.{2}',
10, '')
-- that
```

RegEx_MatchSimple4k

RegEx_MatchSimple4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to RegEx_MatchSimple except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_MatchSimple when input data will *never* be over 4000 characters as this function offers better performance.

RegEx_Replace

RegEx_Replace(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), Replacement NVARCHAR(4000), Count INT, StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Count cannot be less than -1
- Count of -1 = unlimited replacements
- Allows for capturing groups using parenthesis (look at the third example) and those groups can be used in the replacement expression using $ notation
- If no matches are found, ExpressionToValidate is returned without changes
- If input / output data will *never* be over 4000 characters, please use RegEx_Replace4k as that function offers better performance.

EXAMPLES:

```
SELECT SQL#.RegEx_Replace('This is a test that shows matching', 'th.{2}', 'bob',
2, 1, '')
-- This is a test bob shows matching
SELECT SQL#.RegEx_Replace('This is a test that is edifying', 'is', 'IS', 2, 1,
'')
-- ThIS IS a test that is edifying
SELECT SQL#.RegEx_Replace('This is a test that shows matching',
'(a)\s+(t.{2}t)\s+(that)', 'NOT $3 s$1me ol'' $2 which sometimes', 2, 1, '')
-- This is NOT that same ol' test which sometimes shows matching
SELECT SQL#.RegEx_Replace(N'abacab', N'aa', N'#', -1, 1, NULL)
-- abacab
```

RegEx_Replace4k

RegEx_Replace4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), Replacement NVARCHAR(4000), Count INT, StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:
- Functionally equivalent to RegEx_Replace except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_Replace when input / output data will *never* be over 4000 characters as this function offers better performance.

RegEx_ReplaceIfMatched

RegEx_ReplaceIfMatched (ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), Replacement NVARCHAR(4000), NotFoundReplacement NVARCHAR(MAX), Count INT, StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:
- NotFoundReplacement can be empty string ‘’, any value, or NULL
- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Count cannot be less than -1
- Count of -1 = unlimited replacements
- Allows for capturing groups using parenthesis (look at the third example) and those groups can be used in the replacement expression using $ notation
- If no matches are found, NotFoundReplacement is returned
- If input / output data will *never* be over 4000 characters, please use RegEx_ReplaceIfMatched4k as that function offers better performance.

EXAMPLES:

```
SELECT SQL#.RegEx_ReplaceIfMatched(N'abacab', N'a', N'#', N'$$', -1, 1, NULL)
-- #b#c#b
SELECT SQL#.RegEx_ReplaceIfMatched(N'abacab', N'a', N'#', N'$$', 2, 1, NULL)
-- #b#cab
SELECT SQL#.RegEx_ReplaceIfMatched(N'abacab', N'aa', N'#', N'$$', -1, 1, NULL)
-- $$
```

RegEx_ReplaceIfMatched4k

RegEx_ReplaceIfMatched4k(ExpressionToValidate NVARCHAR(4000), RegularExpression NVARCHAR(4000), Replacement NVARCHAR(4000), NotFoundReplacement NVARCHAR(4000), Count INT, StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to RegEx_ReplaceIfMatched except no NVARCHAR(MAX) parameters
- Use this function in place of RegEx_ReplaceIfMatched when input / output data will *never* be over 4000 characters as this function offers better performance.

RegEx_SetCacheSize (Not available in Free version)

RegEx_SetCacheSize(NewCacheSize INT)

RETURNS: INT

Sets the number of Regular Expression patterns that *can* be cached by the App Domain.

NOTES:

- Expressions (i.e. RegEx patterns) take time and CPU resources to parse. Consider increasing the number of expressions that can be cached if the number of expressions *used repeatedly* is more than 10 (10 instead of 15 since some SQL# functions use regular expressions as well, and might use some of those cache slots).
- Returns the previous number of Regular Expression patterns that *could* be cached.
- Default cache size in .NET is 15
- See also: RegEx_GetCacheSize

RegEx_Split

RegEx_Split(ExpressionToValidate NVARCHAR(MAX), RegularExpression NVARCHAR(MAX), Count INT, StartAt INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (MatchNum INT, Value NVARCHAR(MAX), StartPos INT, EndPos INT, Length INT)

NOTES:

- StartAt >= 1
- StartAt cannot be greater than the length of ExpressionToValidate
- Count cannot be less than -1
- Count of -1 (or 0) = unlimited “parts”
- Works like String_Split but uses a Regular Expression to break a string into multiple records rather than a static character or string delimiter
- If the “part” is empty (Length = 0), StartPos and EndPos both equal -1

EXAMPLES:

```
SELECT * FROM SQL#.RegEx_Split('This is a test iff i ever saw one',
'i[fs]{1,2}', 0, 1, '')
/*
MatchNum    Value                StartPos    EndPos      Length
```

```
1            Th                 1            2            2
2                               5            5            1
3             a test            8            15           8
4             i ever saw one    19           33           15
*/
```

RegEx_Unescape

RegEx_Unescape(ExpressionToUnescape NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Unescapes any escaped characters in the input string.
.
NOTES:

- See also: RegEx_Escape

EXAMPLES:

```
SELECT SQL#.RegEx_Unescape('test\(\*\.\)')
-- test(*.)
```

Strings

For String functions, the @StartIndex input parameter specifies the absolute location in the string @StringValue to start at. The first position in a string in SQL Server is 1 and that holds true with all of these SQL# functions; they do NOT start at position 0 (zero). This also holds true of the IndexOf and LastIndexOf functions that return the position of a string within another string.

The following chart lists the options (*not* case-sensitive) to be used in any function that accepts a parameter named @StringCompareOptions. Multiple options, separated by a comma "," or pipe symbol / vertical bar "|", can be used in each case. This chart is taken from the MSDN page for CompareOptions Enumeration.

Option	**Description**
IgnoreCase	Indicates that the string comparison must ignore case.
IgnoreKanaType	Indicates that the string comparison must ignore the Kana type. Kana type refers to Japanese hiragana and katakana characters, which represent phonetic sounds in the Japanese language. Hiragana is used for native Japanese expressions and words, while katakana is used for words borrowed from other languages, such as "computer" or "Internet". A phonetic sound can be expressed in both hiragana and katakana. If this value is selected, the hiragana character for one sound is considered equal to the katakana character for the same sound.
IgnoreNonSpace	Indicates that the string comparison must ignore nonspacing combining characters, such as diacritics. The Unicode Standard defines combining characters as characters that are combined with base characters to produce a new character. Nonspacing combining characters do not occupy a spacing position by themselves when rendered.
IgnoreSymbols	Indicates that the string comparison must ignore symbols, such as white-space characters, punctuation, currency symbols, the percent sign, mathematical symbols, the ampersand, and so on.
IgnoreWidth	Indicates that the string comparison must ignore the character width. For example, Japanese katakana characters can be written as full-width or half-width. If this value is selected, the katakana characters written as full-width are considered equal to the same characters written as half-width.
Ordinal	Indicates that the string comparison must use successive Unicode UTF-16 encoded values of the string (code unit by code unit comparison), leading to a fast comparison but one that is culture-insensitive. A string starting with a code unit $XXXX_{16}$ comes before a string starting with $YYYY_{16}$, if $XXXX_{16}$ is less than $YYYY_{16}$. ***This value cannot be combined with other CompareOptions values and must be used alone***.
OrdinalIgnoreCase	String comparison must ignore case, then perform an ordinal comparison. This technique is equivalent to converting the string to uppercase using the invariant culture and then performing an ordinal comparison on the result. ***This value cannot be combined with other CompareOptions values and must be used alone***.
StringSort	Indicates that the string comparison must use the string sort algorithm. In a string sort, the hyphen and the apostrophe, as well as other nonalphanumeric symbols, come before alphanumeric characters.

String_CompareSplitValues (Not available in Free version)

String_CompareSplitValues(StringValueA NVARCHAR(MAX), StringValueB NVARCHAR(MAX), RegExDelimiter NVARCHAR(4000), @RegExOptionsList NVARCHAR(4000), @CaseSensitive BIT, @NullHandling NVARCHAR(10))

RETURNS: BIT

Compares two delimited lists of after the lists have been split into individual items. The lists must have the same number of elements once split. Meaning, an item from one list cannot match more than one item, even if the same text, in the other list.

NOTES:

- RegExDelimiter
  - What to split both lists on
- RegExOptionsList
  - Please see list of options here
  - Option “IgnoreCase” affects only the split, not the comparison
- CaseSensitive
  - How to compare the lists after being split
  - Does not affect the case-sensitivity of the split
- NullHandling
  - When to return a NULL if either or both StringValue parameters are NULL
  - Options are NOT case-sensitive
  - Options are:
    - Empty string ‘’ = never; return FALSE if either or both are NULL
    - Either = return NULL if either or both are NULL
    - Both = return NULL only if both are NULL or FALSE is only one is NULL

EXAMPLES:

```
SELECT SQL#.String_CompareSplitValues('a B', 'b a', ' ', '', 0, '')
-- 1
SELECT SQL#.String_CompareSplitValues('a B', 'b     a', '\s+', '', 0, '')
-- 1
SELECT SQL#.String_CompareSplitValues('a B', 'b     a', '\s+', '', 1, '')
-- 0
SELECT SQL#.String_CompareSplitValues(NULL, 'b     a', '\s+', '', 0, 'either')
-- NULL
SELECT SQL#.String_CompareSplitValues('a B', NULL, '\s+', '', 0, 'both')
-- 0
SELECT SQL#.String_CompareSplitValues(NULL, NULL, '\s+', '', 0, 'both')
-- NULL
SELECT SQL#.String_CompareSplitValues(NULL, NULL, '\s+', '', 0, '')
-- 0
```

String_Contains

String_Contains(StringValue NVARCHAR(MAX), SearchValue NVARCHAR(MAX))

RETURNS: BIT

String_Contains is a Case-Sensitive replacement for the following LIKE pattern that allows for the @SearchValue to be NVARCHAR(MAX) whereas the value passed into LIKE is constrained to being either VARCHAR(8000) or NVARCHAR(4000):

```
WHERE StringValue LIKE '%' + @SeachValue + '%'
```

If case-sensitivity is the only concern and you are working with either VARCHAR(8000) or NVARCHAR(4000), then consider using the COLLATE clause at is will likely be more efficient:

```
SELECT 1 WHERE 'ABC' LIKE '%a%' COLLATE Latin1_General_100_CS_AS
```

The COLLATE clause also allows for using other sensitivities and/or locales. For example:

```
SELECT 1 WHERE 'ABC' LIKE '%a%' COLLATE Latin1_General_100_CS_AS
         OR 'a' = 'A' COLLATE Hebrew_100_CI_AS
```

FYI: the _CS_ means CaseSensitive whereas the _CI_ means CaseInsensitive. Specifying COLLATE works only for the predicate (i.e. WHERE condition) it is a part of.

EXAMPLES:

```
SELECT SQL#.String_Contains('ABC', 'ab');
-- 0
SELECT SQL#.String_Contains('Abacab', 'ab');
-- 1
```

String_Count

String_Count(StringValue NVARCHAR(MAX), SearchValue NVARCHAR(MAX), StartAt INT, ComparisonType INT, CountOverlap BIT)

RETURNS: INT

String_Count returns the number of times SearchValue is found in StringValue.

NOTES:

- StartAt must be >= 1 and <= the length of StringValue
- ComparisonType:
    - 1 (case-sensitive)
    - 2 (case-INsensitive)
- If CountOverlap is set to 0 / False then the search will resume at the end of the previous match.
- If CountOverlap is set to 1 / True then the search will continue from the next character after the start of the previous match.

EXAMPLES:

```
SELECT SQL#.String_Count('aaAAaaa', 'aa', 1, 2, 0); --insensitive, no ovrlap
-- 3
SELECT SQL#.String_Count('aaAAaaa', 'aa', 1, 2, 1); -- insensitive, overlap
-- 6
SELECT SQL#.String_Count('aaAAaaa', 'aa', 1, 1, 1); -- sensitive, overlap
-- 3
SELECT SQL#.String_Count('aaAAaaa', 'aa', 1, 1, 0); -- sensitive, no ovrlap
-- 2
SELECT SQL#.String_Count('aaAAaaa', 'aa', 3, 1, 1); -- sensitive, overlap
-- 2
```

String_Cut

String_Cut(StringValue NVARCHAR(MAX), Delimiter NVARCHAR(4000), Fields NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

String_Cut returns a string of the fields specified in Fields from StringValue. String_Cut emulates the UNIX "cut" command when using the –f flag.

NOTES:
- Delimiter cannot be NULL
- IF StringValue IS NULL returns NULL
- Empty StringValue returns empty string
- IF Delimiter is empty string returns StringValue unchanged
- Fields = [BeginField-] | [-EndField] | [BeginField- EndField] | [FieldNum] [,]

EXAMPLES:
```
SELECT SQL#.String_Cut('one two three four five', ' ', '1,3')
-- one three
SELECT SQL#.String_Cut('one two three four five', ' ', '-2')
-- one two
SELECT SQL#.String_Cut('one two three four five', ' ', '3-')
-- three four five
SELECT SQL#.String_Cut('one two three four five', ' ', '1,2,4-')
-- one two four five
SELECT SQL#.String_Cut('one two three four five', ' ', '-2, 5')
-- one two five
```

String_DamerauLevenshteinDistance (Not available in Free version)

String_DamerauLevenshteinDistance(String1 NVARCHAR(4000), String2 NVARCHAR(4000), MaxDistance INT)

RETURNS: INT

Calculate the number of changes (inserts, updates, deletes, *and* transpositions of two adjacent characters) it takes to get from String1 to String2.

NOTES:
- NULL input returns NULL.
- Calculation will exit if @MaxDistance is reached, rather than continuing to waste CPU and time.
- If @MaxDistance < 0 or > the longer of @String1 and @String2, then no real "max" distance.
- Comparison assumes same LCID / Locale and sensitity settings as Database's default Collation.
- For more information, see: http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance
- Also see: String_LevenshteinDistance

EXAMPLES:
```
SELECT SQL#.String_DamerauLevenshteinDistance(N'Whale', N'Banana', -1);
-- 5
SELECT SQL#.String_DamerauLevenshteinDistance(N'Whale', N'Banana', 2);
-- 3
SELECT SQL#.String_DamerauLevenshteinDistance(N'Whale', N'Whlae', -1);
-- 1
```

String_DamerauLevenshteinDistancePlus (Not available in Free version)

String_DamerauLevenshteinDistancePlus(String1 NVARCHAR(4000), String2 NVARCHAR(4000), MaxDistance INT, StringCompareOptions NVARCHAR(100), LCID INT)

RETURNS: INT

Calculate the number of changes (inserts, updates, deletes, *and* transpositions of two adjacent characters) it takes to get from String1 to String2.

NOTES:

- NULL input returns NULL.
- Calculation will exit if @MaxDistance is reached, rather than continuing to waste CPU and time.
- If @MaxDistance < 0 or > the longer of @String1 and @String2, then no real "max" distance.
- Unlike the non-"Plus" version, this version allows for customizing how a character is determined to be equal to another. Sensitivity to Case, Accents, etc can be turned on and off, similar to being able to specify a collation dynamically, if that were possible.
- Use this version if you need the comparison to use a "custom" Collation, one that does not match your databases default Collation.
- @StringCompareOptions
  - Comma "," or Pipe "|" separated list of options
  - NULL or empty string '' = None = everything sensitive (but *not* binary / ordinal)
  - 'Database_Default' (not case sensitive; cannot be combined with other values) uses sensitivity settings associated with default Collation of Database where SQL# is installed.
  - Full list of options is shown at the beginning of the Strings section of this manual.
- @LCID options:
  - -1 = Invariant culture
  - 0 = LCID that is associated with default Collation of Database where SQL# is installed.
  - Valid values found at: https://msdn.microsoft.com/en-us/library/cc233982.aspx (Language ID)
- For more information, see: http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance
- See also: String_LevenshteinDistancePlus

EXAMPLES:

```
SELECT SQL#.String_DamerauLevenshteinDistancePlus(N'WhAle', -1, N'Whlae', '',
1033);
-- 2
SELECT SQL#.String_DamerauLevenshteinDistancePlus(N'WhAle', -1, - N'Whlae',
'iGnorecAsE', 1033);
-- 1
SELECT SQL#.String_DamerauLevenshteinDistancePlus(N'WhAle', -1, - N'Whläe',
'iGnorecAsE', 1033);
-- 2
SELECT SQL#.String_DamerauLevenshteinDistancePlus(N'WhAle', -1, - N'Whläe',
'iGnorecAsE,IgnoreNonSpace', 1033);
-- 1
```

String_EndsWith

String_EndsWith(StringValue NVARCHAR(MAX), SearchValue NVARCHAR(4000), ComparisonType INT)

RETURNS: BIT

String_EndsWith is an optionally Case-Sensitive replacement for:

```
WHERE StringValue LIKE '%' + @SeachValue
```

NOTES:

- ComparisonType:
  - 1 (case-sensitive)
  - 2 (case-INsensitive)
- See discussion of COLLATE clause in String_Contains

EXAMPLES:
```
SELECT SQL#.String_EndsWith('Frankly, Mr. Shankly', 'shankly', 1)
-- 0
SELECT SQL#.String_EndsWith('Frankly, Mr. Shankly', 'shankly', 2)
-- 1
```

String_Equals

String_Equals(StringValueA NVARCHAR(MAX), StringValueB NVARCHAR(MAX))

RETURNS: BIT

NOTES:
- See discussion of COLLATE clause in String_Contains

String_Equals is a Case-Sensitive replacement for:
```
WHERE StringValue = @SeachValue
```

EXAMPLES:
```
SELECT SQL#.String_Equals('Plainsong', 'plainsong')
-- 0
SELECT SQL#.String_Equals('Plainsong', 'Plainsong')
-- 1
```

String_FixedWidthIndex (Not available in Free version)

String_FixedWidthIndex(StringToSearch NVARCHAR(MAX), StringToFind NVARCHAR(4000), Width INT, StartAt INT, CaseSensitive BIT, ReturnValue NVARCHAR(50))

RETURNS: INT

Finds the first occurrence of a string within another string which is parsed into fixed-width elements

NOTES:
- Returns 0 if not found
- ReturnValue:
  - NOT case-sensitive
  - Options:
    - **PositionFromBeginning**: the position, starting at the beginning of the StringToSearch regardless of StartAt
    - **PositionFromStartAt**: the position relative to the StartAt value
    - **ElementNumber**: which element within the set that was parsed into fixed-width elements

EXAMPLES:
```
SELECT SQL#.String_FixedWidthIndex('1234567890', '34', 0, 1, 0, 'ElementNumber')
-- 2
SELECT SQL#.String_FixedWidthIndex('1234567890', '45', 0, 1, 0, 'ElementNumber')
-- 0
SELECT SQL#.String_FixedWidthIndex('zzz1234567890', '34', 0, 4, 0,
'ElementNumber') -- 2
SELECT SQL#.String_FixedWidthIndex('zzz1234567890', '34', 0, 4, 0,
'PositionFromBeginning') -- 6
SELECT SQL#.String_FixedWidthIndex('zzz1234567890', '34', 0, 4, 0,
'PositionFromStartAt') -- 3
```

```
SELECT SQL#.String_FixedWidthIndex('zzz1234567890', '34', 4, 1, 0,
'PositionFromStartAt') -- 6
SELECT SQL#.String_FixedWidthIndex('zzz1234567890', '34', 4, 4, 0,
'ElementNumber') -- 1
SELECT SQL#.String_FixedWidthIndex('aabbccdd', 'CC', 0, 1, 0, 'ElementNumber') -
- 3
SELECT SQL#.String_FixedWidthIndex('aabbccdd', 'CC', 0, 1, 1, 'ElementNumber') -
- 0
```

String_FixedWidthSplit (Not available in Free version)

String_FixedWidthSplit(StringToSplit NVARCHAR(MAX), Width INT, StartAt INT)

RETURNS: TABLE (ElementNumber INT, ElementValue NVARCHAR(4000))

Splits a string into chunks of the specified number of characters each

NOTES:

- Final ElementValue might be less than Width characters if there are not Width characters left in the StringToSplit

EXAMPLES:

```
SELECT * FROM SQL#.String_FixedWidthSplit('1234567890', 2, 1)
/*
ElementNumber     ElementValue
1                       12
2                       34
3                       56
4                       78
5                       90
*/

SELECT * FROM SQL#.String_FixedWidthSplit('1234567890', 4, 2)
/*
ElementNumber     ElementValue
1                       2345
2                       6789
3                       0
*/
```

String_IndexOf

String_IndexOf(StringValue NVARCHAR(4000), SearchValue NVARCHAR(4000), StartIndex INT, ComparisonType INT)

RETURNS: INT

String_IndexOf returns the position of the *first* occurrence of SearchValue in StringValue starting at StartIndex. If no occurrence of SearchValue is found within that range, String_IndexOf returns 0 (zero).

NOTES:

- StartIndex >= 1
- StartIndex <= LEN(StringValue)
- ComparisonType:
    - 1 (case-sensitive)

- 2 (case-INsensitive)

EXAMPLES:
```
SELECT SQL#.String_IndexOf('Sound of Music', 's', 1, 1)
-- 12
SELECT SQL#.String_IndexOf('Sound of Music', 's', 1, 2)
-- 1
SELECT SQL#.String_IndexOf('Sound of Music', 'o', 1, 1)
-- 2
SELECT SQL#.String_IndexOf('Sound of Music', 'o', 5, 1)
-- 7
```

String_InitCap

String_InitCap(StringValue NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

String_InitCap capitalizes the first letter of each word as separated by spaces.

EXAMPLE:
```
SELECT SQL#.String_InitCap('the boy with the thorn in his side')
-- The Boy With The Thorn In His Side
```

String_IsNumeric

String_IsNumeric(StringValue NVARCHAR(MAX), NumberTypeMask INT)

RETURNS: BIT

Determines if the StringValue is a number as defined by NumberTypeMask. Works much like the T-SQL built-in ISNUMERIC() but can handle more than 8000 bytes, can handle more numeric formats, and can distinguish between numeric formats.

NOTES:

- NumberTypeMask:
  - A bit-mask value used to specify one or more (or all) numeric formats
  - Number Format Values:
    - 1 = No thousands separator, Period as radix point, optional Scientific Notation. Example: [+/-]12345[.67][e/E+/-10]
    - 2 = No thousands separator, Comma as radix point, optional Scientific Notation. Example: [+/-]12345[,67][e/E+/-10]
    - 4 = Comma as thousands separator, Period as radix point, optional Currency symbol. Example: [[+/-$]or[$+/-]][12,]345[.67]
    - 8 = Period as thousands separator, Comma as radix point, optional Currency symbol. Example: [[+/-$]or[$+/-]][12.]345[,67]
    - 16 = Space as thousands separator, Period as radix point, optional Currency symbol. Example: [[+/-$]or[$+/-]][12]345[.67]
    - 32 = Space as thousands separator, Comma as radix point, optional Currency symbol. Example: [[+/-$]or[$+/-]][12]345[,67]
    - 63 = ALL formats

Symbol	Currency	Hexadecimal value
$	Dollar sign	0024
¢	Cent sign	00A2
£	Pound sign	00A3
¤	Currency sign	00A4
¥	Yen sign	00A5
৲	Bengali Rupee mark	09F2
৳	Bengali Rupee sign	09F3
฿	Thai currency symbol Baht	0E3F
៛	Khmer currency symbol Riel	17DB
₠	Euro-Currency sign	20A0
₡	Colon sign	20A1
₢	Cruzeiro sign	20A2
₣	French Franc sign	20A3
₤	Lira sign	20A4
₥	Mill sign	20A5
₦	Naira sign	20A6
₧	Peseta sign	20A7
₨	Rupee sign	20A8
₩	Won sign	20A9
₪	New Sheqel sign	20AA
₫	Dong sign	20AB
€	Euro sign	20AC
₭	Kip sign	20AD
₮	Tugrik sign	20AE
₯	Drachma sign	20AF
₰	German Penny sign	20B0
₱	Peso sign	20B1
﷼	Rial sign	FDFC
﹩	Small Dollar sign	FE69
＄	Fullwidth Dollar sign	FF04
￠	Fullwidth Cent sign	FFE0
￡	Fullwidth Pound sign	FFE1
￥	Fullwidth Yen sign	FFE5
￦	Fullwidth Won sign	FFE6

- Above chart shows all valid Currency symbols (as taken from Microsoft SQL Server Books Online)
- Additional info: http://en.wikipedia.org/wiki/Decimal_separator

EXAMPLE:
```
SELECT SQL#.String_IsNumeric('$123,121.55', 4)
-- 1
SELECT SQL#.String_IsNumeric('$123,121.55', 8)
-- 0
```

String_Join

String_Join(SQL NVARCHAR(MAX), Separator NVARCHAR(4000), JoinOption INT)

RETURNS: NVARCHAR(MAX)

String_Join will create a single string from the rows of a single string column. If you want to combine numbers, you must convert them to a string datatype in the SQL. The SQL can be any SELECT statement as long as it returns a single column of a text datatype. The SELECT statement can even be from a Temp Table (but not a Table Variable).

NOTES:

- Separator can be more than 1 character
- CombineOption:
    - 1 = for Keep Empty Entries
    - 2 = Remove Empty Entries
- Depending on your situation, you might be able to get away with this:
  ```
  DECLARE @JoinedString VARCHAR(MAX)
  SET @JoinedString = ''
  SELECT      @JoinedString = @JoinedString + alias.Column + ','
  FROM        SchemaName.TableName alias
  ```

EXAMPLE:
```
SELECT so.name INTO #temp_name FROM sys.objects so
SELECT SQL#.String_Join('SELECT name FROM #temp_name', ',', 1)
DROP TABLE #temp_name
GO
-- sysrowsetcolumns,sysrowsets,sysallocunits,sysfiles1,...
```

String_LastIndexOf

String_LastIndexOf(StringValue NVARCHAR(MAX), SearchValue NVARCHAR(4000), StartIndex INT, ComparisonType INT)

RETURNS: INT

String_LastIndexOf returns the position of the *last* occurrence of SearchValue in StringValue starting at StartIndex and proceeding backwards, towards the start of the string. If no occurrence of SearchValue is found within that range, String_LastIndexOf returns 0 (zero).

NOTES:

- StartIndex
    - >= 1
    - <= LEN(StringValue)
    - Where search begins at and proceeds backwards
- ComparisonType:
    - 1 (case-sensitive)
    - 2 (case-INsensitive)

EXAMPLES:
```
SELECT SQL#.String_LastIndexOf('Temptation', 'T', LEN('Temptation'), 1)
-- 1
SELECT SQL#.String_LastIndexOf('Temptation', 'T', LEN('Temptation'), 2)
-- 7
SELECT SQL#.String_LastIndexOf('Temptation', 't', LEN('Temptation'), 1)
-- 7
SELECT SQL#.String_LastIndexOf('Temptation', 't', 5, 1)
-- 5
```

String_LevenshteinDistance (Not available in Free version)

String_LevenshteinDistance(String1 NVARCHAR(4000), String2 NVARCHAR(4000), MaxDistance INT)

RETURNS: INT

Calculate the number of changes (inserts, updates, and deletes) it takes to get from String1 to String2.

NOTES:
- NULL input returns NULL.
- Calculation will exit if @MaxDistance is reached, rather than continuing to waste CPU and time.
- If @MaxDistance < 0 or > the longer of String1 and String2, then no real "max" distance.
- Comparison assumes same LCID / Locale and sensitity settings as Database's default Collation.
- For more information, see: http://en.wikipedia.org/wiki/Levenshtein_distance
- Also see: String_DamerauLevenshteinDistance

EXAMPLES:
```
SELECT SQL#.String_LevenshteinDistance(N'Whale', N'Banana', -1);
-- 5
SELECT SQL#.String_LevenshteinDistance(N'Whale', N'Banana', 3);
-- 4
SELECT SQL#.String_LevenshteinDistance(N'Whale', N'Whlae', -1);
-- 2
```

String_LevenshteinDistancePlus (Not available in Free version)

String_LevenshteinDistancePlus(String1 NVARCHAR(4000), String2 NVARCHAR(4000), MaxDistance INT, StringCompareOptions NVARCHAR(100), LCID INT)

RETURNS: INT

Calculate the number of changes (inserts, updates, and deletes) it takes to get from String1 to String2.

NOTES:
- NULL input returns NULL.
- Calculation will exit if @MaxDistance is reached, rather than continuing to waste CPU and time.
- If @MaxDistance < 0 or > the longer of @String1 and @String2, then no real "max" distance.
- Unlike the non-"Plus" version, this version allows for customizing how a character is determined to be equal to another. Sensitivity to Case, Accents, etc can be turned on and off, similar to being able to specify a collation dynamically, if that were possible.
- Use this version if you need the comparison to use a "custom" Collation, one that does not match your databases default Collation.
- @StringCompareOptions

    - Comma "," or Pipe "|" separated list of options
    - NULL or empty string '' = None = everything sensitive (but *not* binary / ordinal)
    - 'Database_Default' (not case sensitive; cannot be combined with other values) uses sensitivity settings associated with default Collation of Database where SQL# is installed.
    - Full list of options is shown at the beginning of the Strings section of this manual.
- @LCID options:
    - -1 = Invariant culture
    - 0 = LCID that is associated with default Collation of Database where SQL# is installed.
    - Valid values found at: https://msdn.microsoft.com/en-us/library/cc233982.aspx (Language ID)
- For more information, see: http://en.wikipedia.org/wiki/Levenshtein_distance
- See also: String_DamerauLevenshteinDistancePlus

EXAMPLES:

```
SELECT SQL#.String_LevenshteinDistancePlus(N'ABC' + NCHAR(304), N'ABci', -1,
N'none', 0x0409);
-- 2
SELECT SQL#.String_LevenshteinDistancePlus(N'ABC' + NCHAR(304), N'ABci', -1,
N'IgnoreCase', 0x0409);
-- 1
SELECT SQL#.String_LevenshteinDistancePlus(N'ABC' + NCHAR(304), N'ABci', -1,
N'IgnoreCase|IgnoreNonSpace', 0x0409);
-- 0
SELECT SQL#.String_LevenshteinDistancePlus(N'ABC' + NCHAR(304), N'ABci', -1,
N'IgnoreCase', 0x041f);
-- 0
SELECT SQL#.String_LevenshteinDistancePlus(N'ABC' + NCHAR(304), N'ABci', -1,
N'database_default', 0);
-- 1  (database default Collation = Latin1_General_100_CI_AS_SC)
```

String_Newline

String_Newline(EOLType NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

String_Newline returns the newline character(s) for the particular OS / End-Of-Line Type that is specified.

NOTES:

- EOLType can be:
    - CRLF, WIN, WINDOWS, DOS, or OS/2
    - LF or UNIX
    - CR or MAC
    - FF
    - NEL, 390, OS/390, or EBCDIC
    - HTML
    - XHTML
- EOLType is not case-sensitive

EXAMPLES:

```
PRINT 'This is a test of ' +
      SQL#.String_Newline('win') +
      'how newlines' +
      SQL#.String_Newline('html') +
      'work in SQL strings.'
```

```
This is a test of
how newlines<br>work in SQL strings.
```

String_NthIndexOf

String_NthIndexOf(StringValue NVARCHAR(MAX), Search NVARCHAR(MAX), StartAt INT, NthOccurance INT, ComparisonType INT, CountOverlap BIT)

RETURNS: INT

String_NthIndexOf finds the location of the specified occurance of Search in StringValue. It returns 0 if not found.

NOTES:

- If StringValue is NULL, 0 is returned
- Search cannot be NULL or empty string
- StartAt >= 1
- StartAt <= LEN(StringValue)
- NthOccurance >= 1
- ComparisonType = 1 (case-sensitive) or 2 (case-INsensitive)

EXAMPLES:

```
SELECT SQL#.String_NthIndexOf('aaa  AAaa', 'aa', 1, 3, 1, 1)
-- StartAt 1, 3rd Occurance, case Sensitive, do count OverLaps
-- 8
SELECT SQL#.String_NthIndexOf('aaa  AAaa', 'aa', 1, 3, 2, 1)
-- StartAt 1, 3rd Occurance, case INsensitive, do count OverLaps
-- 6
SELECT SQL#.String_NthIndexOf('aaa  AAaa', 'aa', 1, 3, 2, 0)
-- StartAt 1, 3rd Occurance, case INsensitive, do NOT count OverLaps
-- 8
SELECT SQL#.String_NthIndexOf('aaa  AAaa', 'aa', 2, 3, 2, 1)
-- StartAt 2, 3rd Occurance, case INsensitive, do count OverLaps
-- 7
SELECT SQL#.String_NthIndexOf('aaa  AAaa', 'aa', 1, 3, 1, 0)
-- StartAt 1, 3rd Occurance, case INsensitive, do NOT count OverLaps
-- 0
```

String_PadBoth (Not available in Free version)

String_PadBoth(StringValue NVARCHAR(MAX), StringWidth INT, LeftPadCharacter NCHAR(1), RightPadCharacter NCHAR(1), FavorLeftOrRight NCHAR(1))

RETURNS: NVARCHAR(MAX)

Combines both PadLeft and PadRight, essentially centering StringValue based on StringWidth. It allows for choosing different characters for each side.

NOTES:

- StringWidth >= LEN(StringValue)
- LeftPadCharacter and RightPadCharacter can only be a single character; any characters after the first one will be ignored.
- FavorLeftOrRight:
    - NOT case-sensitive
    - 'L' or 'R'

  - ‘L’ will place string just left of center when sides are uneven
  - ‘R’ will place string just right of center when sides are uneven
- For @StringValue values that will never be over 4000 bytes, please use String_PadBoth4k

EXAMPLES:
```
SELECT SQL#.String_PadBoth(N'Test', 8, N'$', N'%', N'R');
-- $$Test%%
SELECT SQL#.String_PadBoth(N'Test', 9, N'$', N'%', N'L');
-- $$Test%%%
SELECT SQL#.String_PadBoth(N'Test', 9, N'$', N'%', N'r');
-- $$$Test%%
```

String_PadBoth4k (Not available in Free version)

String_PadBoth4k(StringValue NVARCHAR(4000), StringWidth INT, LeftPadCharacter NCHAR(1), RightPadCharacter NCHAR(1), FavorLeftOrRight NCHAR(1))

RETURNS: NVARCHAR(4000)

NOTES:
- Functionally equivalent to String_PadBoth except no NVARCHAR(MAX) parameters
- Use this function in place of String_PadBoth when input / output data will *never* be over 4000 characters as this function offers better performance.

String_PadLeft

String_PadLeft(StringValue NVARCHAR(MAX), StringWidth INT, PadCharacter NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

String_PadLeft returns a string of StringWidth characters with StringValue right-justified and padded on the left with PadCharacter.

NOTES:
- StringWidth >= LEN(StringValue)
- PadCharacter can only be a single character; any characters after the first one will be ignored.

EXAMPLE:
```
SELECT SQL#.String_PadLeft('Kathy'' Song', 30, '*')
-- ******************Kathy' Song
```

String_PadRight

String_PadRight(StringValue NVARCHAR(MAX), StringWidth INT, PadCharacter NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

String_PadRight returns a string of StringWidth characters with StringValue left-justified and padded on the right with PadCharacter.

NOTES:
- StringWidth >= LEN(StringValue)
- PadCharacter can only be a single character; any characters after the first one will be ignored.

EXAMPLE:
```
SELECT SQL#.String_PadRight('Colorwheel', 30, '-')
-- Colorwheel********************
```

String_RemoveDiacritics (Not available in Free version)

String_RemoveDiacritics(@StringValue NVARCHAR(MAX), @UseCompatibilityForm BIT)

RETURNS: NVARCHAR(MAX)

Removes accents and other diacritical marks from letters.

NOTES:
- String is normalized such that letter characters have any accents, tildes, macrons, diaeresis, cedilla, etc removed
- @UseCompatiblityForm
  - If set to 1 / true, characters are broken down into multiple basic characters if possible.For example: the single character " ¼ " broken into the three characters of " 1⁄4 "
- For @StringValue values that will never be over 4000 bytes, please use String_RemoveDiacritics4k

EXAMPLES:
```
SELECT SQL#.String_RemoveDiacritics(N'sdfsd', 0); -- sdfsd (no change)

SELECT SQL#.String_RemoveDiacritics(N'â', 0); -- a
SELECT SQL#.String_RemoveDiacritics(N'â', 1); -- a (same change as above)

SELECT SQL#.String_RemoveDiacritics(N'¼', 0); -- ¼ (no change)
SELECT SQL#.String_RemoveDiacritics(N'¼', 1); -- 1⁄4 (broken into 3 characters)
```

String_RemoveDiacritics4k (Not available in Free version)

String_RemoveDiacritics4k(@StringValue NVARCHAR(MAX), @UseCompatibilityForm BIT)

RETURNS: NVARCHAR(MAX)

Removes accents and other diacritical marks from letters.

NOTES:
- Functionally equivalent to String_RemoveDiacritics except no NVARCHAR(MAX) parameters
- Use this function in place of String_RemoveDiacritics when input / output data will *never* be over 4000 characters as this function offers better performance.

String_Replace

String_Replace(Expression NVARCHAR(MAX), Find NVARCHAR(MAX), Replacement NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Works the same as the builtin T-SQL REPLACE function except:
- It accepts NVARCHAR(MAX) for all parameters
- The comparison is always case-sensitive

EXAMPLES:
```
DECLARE @String NVARCHAR(MAX),
            @Find NVARCHAR(MAX),
            @Replacement NVARCHAR(MAX),
            @Result NVARCHAR(MAX)
SET @String = REPLICATE(CONVERT(NVARCHAR(MAX), 'a'), 12000)
SET @String = @String + REPLICATE(CONVERT(NVARCHAR(MAX), 'b'), 12000)
SET @String = @String + REPLICATE(CONVERT(NVARCHAR(MAX), 'a'), 12000)
SET @String = @String + REPLICATE(CONVERT(NVARCHAR(MAX), 'c'), 12000)
SET @String = @String + REPLICATE(CONVERT(NVARCHAR(MAX), 'a'), 12000)

SET @Find = REPLICATE(CONVERT(NVARCHAR(MAX), 'a'), 11000)
SET @Replacement = REPLICATE(CONVERT(NVARCHAR(MAX), 'z'), 20000)

SELECT @Result = SQL#.String_Replace(@String, @Find, @Replacement)
SELECT      LEN(@String) AS [String], LEN(@Find) AS [Find],
            LEN(@Replacement) AS [Replacement], LEN(@Result) AS [Result]

SET @Find = REPLICATE(CONVERT(NVARCHAR(MAX), 'c'), 9000)
SET @Replacement = REPLICATE(CONVERT(NVARCHAR(MAX), 'y'), 1000)
SELECT @Result = SQL#.String_Replace(@String, @Find, @Replacement)
SELECT      LEN(@String) AS [String], LEN(@Find) AS [Find],
            LEN(@Replacement) AS [Replacement], LEN(@Result) AS [Result]
```

String_Split

String_Split(StringValue NVARCHAR(MAX), Separator NVARCHAR(4000), SplitOption INT)

RETURNS: TABLE (SplitNum INT, SplitVal NVARCHAR(MAX))

String_Split takes a delimited string (based on the Separator) and returns a table of its elements after removing the Separator.

NOTES:

- Separator can be more than 1 character
- SplitOption:
  - 1 = for Keep Empty Entries
  - 2 = Remove Empty Entries
- For StringValues that will never be over 4000 characters, use String_Split4k, which is faster.

EXAMPLES:

```
SELECT * FROM SQL#.String_Split('12,1,45,646,8978,90,4,3,6,15', ',', 1)
/*
1      12
2      1
3      45
4      646
5      8978
6      90
7      4
8      3
9      6
10     15
*/
SELECT * FROM SQL#.String_Split('Bob<br><br>Sally<br><br>', '<br>', 1)
/*
1      Bob
2
```

```
3      Sally
4
5
*/
SELECT * FROM SQL#.String_Split('Bob<br><br>Sally<br><br>', '<br>', 2)
/*
1      Bob
2      Sally
*/
```

String_Split4k

String_Split4k(StringValue NVARCHAR(4000), Separator NVARCHAR(4000), SplitOption INT)

RETURNS: TABLE (SplitNum INT, SplitVal NVARCHAR(4000))

Identical to String_Split except for the datatype for StringValue (input) and SplitVal (output). This function is much faster than String_Split for input strings of no more than 4000 characters. Only use String_Split with strings longer than 4000 characters, or if you cannot guarantee that they will not be over 4000 characters. Please see String_Split for Notes and Examples.

String_SplitIntoGuids (Not available in Free version)

String_SplitIntoGuids(StringValue NVARCHAR(MAX), Separator NVARCHAR(4000), EmptyEntryHandling INT, ErrorEntryHandling INT, ReturnNullRowOnNullInput BIT)

RETURNS: TABLE (SplitNum INT, SplitVal UNIQUEIDENTIFIER)

NOTES:

- Separator can be more than 1 character
- EmptyEntryHandling
  - 1 = Value will be NULL
  - 2 = No row returned for empty entry
  - 3 = throw error
- ErrorEntryHandling
  - 1 = Value will be NULL
  - 2 = No row returned for error entry
  - 3 = throw error
- ReturnNullRowOnNullInput
  - 0 = NULL input will return empty record set (no rows)
  - 1 = NULL input will return 1 row with NULL in both fields
- For StringValues that will never be over 4000 characters, use String_SplitIntoGuids4k, which is faster.
- {will add details later}

String_SplitIntoIntegers (Not available in Free version)

String_SplitIntoIntegers(StringValue NVARCHAR(MAX), Separator NVARCHAR(4000), EmptyEntryHandling INT, ErrorEntryHandling INT, ReturnNullRowOnNullInput BIT)

RETURNS: TABLE (SplitNum INT, SplitVal BIGINT)

NOTES:

- Separator can be more than 1 character
- EmptyEntryHandling

    - 1 = Value will be NULL
    - 2 = No row returned for empty entry
    - 3 = throw error
- ErrorEntryHandling
    - 1 = Value will be NULL
    - 2 = No row returned for error entry
    - 3 = throw error
- ReturnNullRowOnNullInput
    - 0 = NULL input will return empty record set (no rows)
    - 1 = NULL input will return 1 row with NULL in both fields
- For StringValues that will never be over 4000 characters, use String_SplitIntoIntegers4k, which is faster.
- {will add details later}

String_SplitInts

String_SplitInts(StringValue NVARCHAR(MAX), Separator NVARCHAR(4000), SplitOption INT, ReturnNullRowOnNullInput BIT)

RETURNS: TABLE (SplitNum INT, SplitVal BIGINT)

NOTES:

- Separator can be more than 1 character
- SplitOption:
    - 1 = for Keep Empty Entries
    - 2 = Remove Empty Entries
- ReturnNullRowOnNullInput
    - 0 = NULL input will return empty record set (no rows)
    - 1 = NULL input will return 1 row with NULL in both fields
- For StringValues that will never be over 4000 characters, use String_SplitInts4k, which is faster.
- {will add details later}

String_SplitResultIntoFields (Not available in Free version)

String_SplitResultIntoFields @Query NVARCHAR(4000), @RegExDelimiter NVARCHAR(4000) [, @ColumnNames NVARCHAR(4000)] [, @DataTypes NVARCHAR(4000)]

PROC: Result set is the NVARCHAR(MAX) field specified in @Query broken into fields based on @RegExDelimiter

NOTES:

- @Query:
    - must return a string field (CHAR, VARCHAR, NCHAR, NVARCHAR, TEXT, NTEXT) as the first column of a SELECT statement
    - Any additional columns returned by @Query will be ignored
- @RegExDelimiter is a full Regular Expression (See RegEx section)
- @ColumnNames:
    - Optional parameter
    - Comma-separated list of values that will be used to name the columns of the result set
    - Extra spaces around each name will be trimmed
    - If more fields are in the data than specified in ColumnNames then additional fields will be named as FieldN where N is the field number
    - If more fields are specified in ColumnNames than in the first row of the result set then extra Column Names will be ignored

    - If not set or set to NULL then all field names will be FieldN where N is the field number starting with 1
- @DataTypes:
    - Optional parameter
    - Value is NOT case-sensitive
    - Comma-separated list of values that will be used to specify the datatype of the columns of the result set
    - If more fields are in the data than specified in DataTypes then additional fields will be set to NVARCHAR(MAX)
    - If more fields are specified in DataTypes than in the first row of the result set then extra values will be ignored
    - If not set or set to NULL then all field datatypes will be set to NVARCHAR(MAX)
    - Empty value in source data will return empty string for (N)(VAR)CHAR / XML datatypes, 0x00 for (VAR)BINARY, and NULL for number / date datatypes.
    - Currently, the TIME and DATETIMEOFFSET datatypes do not work properly.
- Number of fields returned in result set is based on first row of data
- After first row of data, rows with more fields will have the additional fields ignored (see example)
- After number of fields to return is set, rows with fewer fields will return empty strings for the missing fields (see example)
- See also: File_SplitIntoFields and INET_SplitIntoFields

EXAMPLES:
```
CREATE TABLE #SplitTest (Column1 VARCHAR(MAX), Column2 VARCHAR(MAX))

INSERT INTO #SplitTest (Column1, Column2)
VALUES ('Value1 Value2 Value3 Value4  ', 'bob')
INSERT INTO #SplitTest (Column1, Column2)
VALUES ('NewValue1   NewValue2   NewValue3 Value4  ', 'bob')
INSERT INTO #SplitTest (Column1, Column2)
VALUES ('Another1       Another2   Another3', 'bob')
INSERT INTO #SplitTest (Column1, Column2)
VALUES ('a b c d e f', 'bob')

EXEC SQL#.String_SplitIntoFields 'SELECT * FROM #SplitTest', '[ ]+', 'Name,Title
,    Alias    '

/*
Name        Title       Alias       Field4      Field5
---------   ---------   ---------   ---------   ---------
Value1      Value2      Value3      Value4
NewValue1   NewValue2   NewValue3   Value4
Another1    Another2    Another3
a           b           c           d           e
*/
```

String_SplitKeyValuePairs (Not available in Free version)

String_SplitKeyValuePairs(KeyValuePairs NVARCHAR(MAX), PairSeparator NVARCHAR(4000), KeyValueSeparator NVARCHAR(4000), RemoveEmptyPairs BIT, Trim NVARCHAR(50), Decode NVARCHAR(50), Unquote NVARCHAR(1))

RETURNS: TABLE (KeyID INT, Key NVARCHAR(MAX), Value NVARCHAR(MAX))

Splits a delimited set of Key-Value pairs into a result set. A Common use of this is a HTTP Query String which has Key-Value pairs (key=value) separated by ampersands (&).

NOTES:

- PairSeparator:
    - 1 or more characters that separates each set of Key-Value pairs.
    - Typically this is an ampersand (&)
- KeyValueSeparator:
    - 1 or more characters that separates each Key and Value
    - Typically this is an equal-sign (=)
- RemoveEmptyPairs:
    - If set to 1, removes Pairs where both Key and Value are empty
    - Example of empty KeyValuePair: &=&
- Trim:
    - These are NOT case-sensitive
    - “Key” does a Trim on just the Key
    - “Value” does a Trim on just the Value
    - “Both” does a Trim on both Key and Value
    - NULL or empty string ‘’ else does nothing
- Decode:
    - These are NOT case-sensitive
    - “Key” does a URIDecode on just the Key
    - “Value” does a URIDecode on just the Value
    - “Both” does a URIDecode on both Key and Value
    - NULL or empty string ‘’ does nothing
    - Use “Value” or “Both” when splitting a HTTP Query String
- Unquote:
    - Use this only if the Values are quoted
    - Single character to remove from Value only (has no effect on Key)
    - The character is only removed if it is present on both sides of the Value

EXAMPLES:

```
DECLARE @String NVARCHAR(MAX)
SET @String = 'asd=234&=& asd = 234
&qw234234&asd=234&qw=234234&&qw=234234&asd=234&qw=234234&as%3dd=2%203%3e4&f=asd"
&g="234"'
SELECT * FROM SQL#.String_SplitKeyValuePairs(@String, '&', '=', 0, null, null,
null) -- row 2 is empty
SELECT * FROM SQL#.String_SplitKeyValuePairs(@String, '&', '=', 1, null, null,
null) -- empty row is gone
SELECT * FROM SQL#.String_SplitKeyValuePairs(@String, '&', '=', 1, 'key', null,
null) -- row 2 key is trimmed
SELECT * FROM SQL#.String_SplitKeyValuePairs(@String, '&', '=', 1, 'key',
'value', null) -- row 8 value is decoded
SELECT * FROM SQL#.String_SplitKeyValuePairs(@String, '&', '=', 1, 'key',
'value', '"') -- row 10 value is unquoted
```

String_StartsWith

String_StartsWith(StringValue NVARCHAR(4000), SearchValue NVARCHAR(4000), ComparisonType INT)

RETURNS: BIT

String_StartsWith is an optionally Case-Sensitive replacement for:

```
WHERE StringValue LIKE @SeachValue + '%'
```

NOTES:

- ComparisonType:
    - 1 (case-sensitive)
    - 2 (case-INsensitive)
- See discussion of COLLATE clause in String_Contains

EXAMPLES:
```
SELECT SQL#.String_StartsWith('Hey Nineteen', 'hey', 1)
-- 0
SELECT SQL#.String_StartsWith('Hey Nineteen', 'hey', 2)
-- 1
```

String_Trim
String_Trim(StringValue NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

NOTES:
- String_Trim is a replacement for the silly LTRIM(RTRIM()) combination.
- Unlike the LTRIM() and RTRIM() built-in functions that only remove spaces, this function also removes Tabs [CHAR(9)], Line Feeds [CHAR(10)], and Carriage Returns [CHAR(13)].
- SQL Server 2017 added this functionality via the TRIM() built-in function.

EXAMPLE:
```
SELECT '*' + SQL#.String_Trim('   Deacon Blues   ') + '*'
--  *Deacon Blues*
```

String_Trim4k
String_Trim4k(StringValue NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:
- Functionally equivalent to String_Trim except no NVARCHAR(MAX) parameters
- Use this function in place of String_Trim when input / output data will *never* be over 4000 characters as this function offers better performance.

String_TrimChars (Not available in Free version)
String_TrimChars(StringValue NVARCHAR(MAX), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:
- This works like String_Trim, but instead of removing whitespace characters, it removes all occurrences of the characters passed in via @CharsToTrim.
- SQL Server 2017 added this functionality via the TRIM() built-in function.

EXAMPLE:
```
SELECT SQL#.String_TrimChars('"''aasasa34985as"sa398475as''"', '"''as')
-- 34985as"sa398475
```

String_TrimChars4k (Not available in Free version)

String_TrimChars4k(StringValue NVARCHAR(4000), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to String_TrimChars except no NVARCHAR(MAX) parameters
- Use this function in place of String_TrimChars when input / output data will *never* be over 4000 characters as this function offers better performance.

String_TrimEnd (Not available in Free version)

String_TrimEnd(StringValue NVARCHAR(MAX), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

This works like the built-in RTRIM function, but instead of removing whitespace characters, it removes all occurrences of the characters passed in via @CharsToTrim until it reaches the first character NOT in @CharsToTrim.

EXAMPLE:

```
SELECT SQL#.String_TrimEnd('"''aasasa34985as"sa398475as''"', '"''as')
-- "'aasasa34985as"sa398475
```

String_TrimEnd4k (Not available in Free version)

String_TrimEnd4k(StringValue NVARCHAR(4000), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to String_TrimEnd except no NVARCHAR(MAX) parameters
- Use this function in place of String_TrimEnd when input / output data will *never* be over 4000 characters as this function offers better performance.

String_TrimStart (Not available in Free version)

String_TrimStart(StringValue NVARCHAR(MAX), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

This works like the built-in LTRIM function, but instead of removing whitespace characters, it removes all occurrences of the characters passed in via @CharsToTrim until it reaches the first character NOT in @CharsToTrim.

EXAMPLE:

```
SELECT SQL#.String_TrimStart('"''aasasa34985as"sa398475as''"', '"''as')
-- 34985as"sa398475as'"
```

String_TrimStart4k (Not available in Free version)

String_TrimStart4k(StringValue NVARCHAR(4000), CharsToTrim NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Functionally equivalent to String_TrimStart except no NVARCHAR(MAX) parameters
- Use this function in place of String_TrimStart when input / output data will *never* be over 4000 characters as this function offers better performance.

String_TryParseToInt

String_TryParseToInt(StringValue NVARCHAR(4000), TargetDataType NVARCHAR(15), Culture NVARCHAR(10))

RETURNS: BIGINT

NOTES:

- Works nearly identically to TRY_PARSE which was introduced in SQL Server 2012
- If StringValue contains a number that is valid for the TargetDataType, the converted value will be returned
- If StringValue contains non-numeric characters or is not valid for the TargetDataType, NULL will be returned
- TargetDataType:
    - NOT case-sensitive
    - VALUES: TINYINT, SMALLINT, INT, BIGINT
- Culture is optional; pass in empty string '' to use system default
- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.String_TryParseToInt('123 456', 'int', '');
-- NULL
SELECT SQL#.String_TryParseToInt('123 456', 'int', 'en-us');
-- NULL
SELECT SQL#.String_TryParseToInt('123 456', 'int', 'fr-fr');
-- 123456
SELECT SQL#.String_TryParseToInt('123,456', 'INT', 'en-us');
-- 123456
SELECT SQL#.String_TryParseToInt('123,456', 'int', 'fr-fr');
-- NULL
```

String_Unescape

String_Unescape(EscapedString NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Replaces standard escape sequences with their unescaped values.

NOTES:

- Valid escape sequences:
    - **\0** = (null) (i.e. CHAR(0))
    - **\a** = Alert (i.e. “Bell” CHAR(7))
    - **\b** = Backspace (CHAR(8))
    - **\f** = Form Feed (CHAR(12))
    - **\n** = New Line (i.e. “Line Feed” CHAR(10))
    - **\r** = Carriage Return (CHAR(13))

    - **\t** = Horizontal Tab (i.e. “Tab” CHAR(9))
    - **\v** = Vertical Tab (CHAR(11))
    - **\?** = Question Mark (CHAR(63))
    - \\ = Reverse Solidus (i.e. “Backslash” CHAR(92))
    - **\'** = Apostrophe (i.e. “Single Quote” CHAR(39))
    - **\"** = Quotation Mark (i.e. “Double Quote” CHAR(34))
    - **\0[?][?]** = Octal notation (1 - 3 digits in the form of: **\[0-7]** or **\[0-7][0-7]** or **\[0-3][0-7][0-7]**)
    - **\xH[?][?][?]** = variable-length UCS-2 Code Point (1 - 4 hex digits: 0-9, A-F)
    - **\uHHHH** = 4 hex digit UCS-2 Code Point / UTF-16 Code Unit (same as UTF-16 BE; range is \u0000 - \uFFFF)
    - **\U00HHHHHH** = 8 hex digit Unicode Code Point (first two digits are always 00; same as UTF-32 BE; range is \U00000000 - \U0010FFFF)
- NULL input returns NULL

EXAMPLES:
```
PRINT SQL#.String_Unescape(N'A\t1\nB\t2\n\x65 \x065 \x0065 \u0065\n \76 \076');
/*
A      1
B      2
e e e e
 > >
*/

SELECT CONVERT(VARBINARY(20),
SQL#.String_Unescape(N'\U0001F92F\0\uD83E\uDD2F'));
-- 0x3ED8 2FDD 0000 3ED8 2FDD (UTF-16 Code Units separated to be more readable)
```

String_Unescape4k

String_Unescape(EscapedString NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:
- Functionally equivalent to String_Unescape except no NVARCHAR(MAX) parameters
- Use this function in place of String_Unescape when input / output data will *never* be over 4000 characters as this function offers better performance.

String_WordWrap

String_WordWrap(StringValue NVARCHAR(MAX), LineWidth INT, Separator NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

String_WordWrap returns a string broken into lines of LineWidth characters with Separator between each line.

EXAMPLE:
```
DECLARE @String NVARCHAR(MAX);

SELECT      TOP 1 @String = [definition]
FROM   [master].[sys].[all_sql_modules]
WHERE OBJECT_NAME([object_id], DB_ID(N'master')) = N'sp_who2';

SET @String = REPLACE(@String, NCHAR(13)+NCHAR(10), N'');
PRINT SQL#.String_WordWrap(@String, 60, N'<br>' + NCHAR(13)+NCHAR(10));
```

```
/*
create procedure sys.sp_who2  --- 1995/11/03 10:16   <br>
@loginame     sysname = NULLasset nocount ondeclare   <br>
@retcode         intdeclare    @sidlow         varbinary(85)<br>
  ,@sidhigh        varbinary(85)   ,@sid1          <br>
varbinary(85)   ,@spidlow         int   ,@spidhigh       <br>
intdeclare    @charMaxLenLoginName      varchar(6)  <br>
...
*/
```

Math

As with any of the other SQL# functions, any string options sent into a function are NOT case-sensitive; mixed-case options are shown here for easier reading.

Math_BitwiseLeftShift

Math_BitwiseLeftShift(TheValue BIGINT, BitsToShift INT)

RETURNS: BIGINT

NOTES:

- Shifts TheValue the number of bit positions to the left as specified by BitsToShift
- Returns NULL if either input parameter is NULL
-
```
Bit position:           8    7   6   5   4  3  2  1  0
Decimal Equivalents:  256  128  64  32  16  8  4  2  1
Binary Value:                            1  0  0  1  0
Decimal Value:                          16   +    2     = 18
```

EXAMPLES:

```
SELECT SQL#.Math_BitwiseLeftShift(1, 4); -- 16
SELECT SQL#.Math_BitwiseLeftShift(2, 3); -- 16
SELECT SQL#.Math_BitwiseLeftShift(3, 2); -- 12 (3=slots 0 & 1; 12=slots 2 & 3)
SELECT SQL#.Math_BitwiseLeftShift(18, 2); -- 72 (18=slots 1 & 4; 72=slots 6 & 3)
```

Math_BitwiseRightShift

Math_BitwiseRightShift(@TheValue BIGINT, @BitsToShift INT)

RETURNS: BIGINT

NOTES:

- Shifts @TheValue the number of bit positions to the right as specified by @BitsToShift.
- Returns NULL if either input parameter is NULL.
-
```
Bit position:           8    7   6   5   4  3  2  1  0
Decimal Equivalents:  256  128  64  32  16  8  4  2  1
Binary Value:                1   0   1   1  0  0  0  0
Decimal Value:             128   +  32 + 16              = 176
```

EXAMPLES:

```
SELECT SQL#.Math_BitwiseRightShift(12, 2); -- 3
SELECT SQL#.Math_BitwiseRightShift(16, 3); -- 2
SELECT SQL#.Math_BitwiseRightShift(176, 2); -- 44 (44 = slots 5, 3, & 2)
```

Math_CompoundAmortizationSchedule

Math_CompoundAmortizationSchedule(@LoanAmount FLOAT, @AnnualInterestRate FLOAT, @YearsOfLoan INT, @PaymentsPerYear INT, @LoanStartDate DATETIME, @OptionalExtraPayment FLOAT)

RETURNS: TABLE (PaymentNum INT, PaymentDate DATETIME, BeginningBalance FLOAT, ScheduledPayment FLOAT, ExtraPayment FLOAT, TotalPayment FLOAT, Principal FLOAT, Interest FLOAT, EndingBalance FLOAT, CumulativeInterest FLOAT, TotalInterest FLOAT, TotalPayments INT, PaymentsLeft INT, CumulativePrincipal FLOAT, CumulativeAmountPaid FLOAT, TotalAmountPaid FLOAT)

NOTES:

- **If you don't have a specific need to use FLOAT datatypes for input parameters and result set fields, then use Math_CompoundAmortizationSchedule2 as it won't have rounding errors due to FLOAT.**
- @LoanAmount must be >= 0
- @YearsOfLoan must be >= 1
- @PaymentsPerYear must be >= 1
- @OptionalExtraPayment must be >= 0

EXAMPLE:

```
SELECT * FROM SQL#.Math_CompoundAmortizationSchedule(100000, 5.5, 2, 12,
'1/1/2006', 0)
```

PaymentNum	PaymentDate	Beginning-Balance	Scheduled-Payment	ExtraPayment	Total-Payment	Principal	Interest	Ending-Balance	Cum-ulative-Interest	Total-Interest	TotalPayments	PaymentsLeft	Cumulative Principal	CumulativeAmount-Paid	TotalAmount-Paid
1	2006-02-01	100000	4409.57	0	4409.57	3951.24	458.33	96048.76	458.33	5829.53	24	23	3951.24	4409.57	105829.53
2	2006-03-01	96048.76	4409.57	0	4409.57	3969.35	440.22	92079.41	898.55	5829.53	24	22	7920.59	8819.14	105829.53
3	2006-04-01	92079.41	4409.57	0	4409.57	3987.54	422.03	88091.87	1320.58	5829.53	24	21	11908.13	13228.71	105829.53
4	2006-05-01	88091.87	4409.57	0	4409.57	4005.82	403.75	84086.05	1724.33	5829.53	24	20	15913.95	17638.28	105829.53
5	2006-06-01	84086.05	4409.57	0	4409.57	4024.18	385.39	80061.87	2109.72	5829.53	24	19	19938.13	22047.85	105829.53
6	2006-07-01	80061.87	4409.57	0	4409.57	4042.62	366.95	76019.25	2476.67	5829.53	24	18	23980.75	26457.42	105829.53
7	2006-08-01	76019.25	4409.57	0	4409.57	4061.15	348.42	71958.1	2825.09	5829.53	24	17	28041.9	30866.99	105829.53
8	2006-09-01	71958.1	4409.57	0	4409.57	4079.76	329.81	67878.34	3154.9	5829.53	24	16	32121.66	35276.56	105829.53
9	2006-10-01	67878.34	4409.57	0	4409.57	4098.46	311.11	63779.88	3466.01	5829.53	24	15	36220.12	39686.13	105829.53
10	2006-11-01	63779.88	4409.57	0	4409.57	4117.25	292.32	59662.63	3758.33	5829.53	24	14	40337.37	44095.7	105829.53
11	2006-12-01	59662.63	4409.57	0	4409.57	4136.12	273.45	55526.51	4031.78	5829.53	24	13	44473.49	48505.27	105829.53
12	2007-01-01	55526.51	4409.57	0	4409.57	4155.07	254.5	51371.44	4286.28	5829.53	24	12	48628.56	52914.84	105829.53
13	2007-02-01	51371.44	4409.57	0	4409.57	4174.12	235.45	47197.32	4521.73	5829.53	24	11	52802.68	57324.41	105829.53
14	2007-03-01	47197.32	4409.57	0	4409.57	4193.25	216.32	43004.07	4738.05	5829.53	24	10	56995.93	61733.98	105829.53
15	2007-04-01	43004.07	4409.57	0	4409.57	4212.47	197.1	38791.6	4935.15	5829.53	24	9	61208.4	66143.55	105829.53
16	2007-05-01	38791.6	4409.57	0	4409.57	4231.78	177.79	34559.82	5112.94	5829.53	24	8	65440.18	70553.12	105829.53
17	2007-06-01	34559.82	4409.57	0	4409.57	4251.17	158.4	30308.65	5271.34	5829.53	24	7	69691.35	74962.69	105829.53
18	2007-07-01	30308.65	4409.57	0	4409.57	4270.66	138.91	26037.99	5410.25	5829.53	24	6	73962.01	79372.26	105829.53
19	2007-08-01	26037.99	4409.57	0	4409.57	4290.23	119.34	21747.76	5529.59	5829.53	24	5	78252.24	83781.83	105829.53
20	2007-09-01	21747.76	4409.57	0	4409.57	4309.89	99.68	17437.87	5629.27	5829.53	24	4	82562.13	88191.4	105829.53
21	2007-10-01	17437.87	4409.57	0	4409.57	4329.65	79.92	13108.22	5709.19	5829.53	24	3	86891.78	92600.97	105829.53
22	2007-11-01	13108.22	4409.57	0	4409.57	4349.49	60.08	8758.73	5769.27	5829.53	24	2	91241.27	97010.54	105829.53
23	2007-12-01	8758.73	4409.57	0	4409.57	4369.43	40.14	4389.3	5809.41	5829.53	24	1	95610.7	101420.11	105829.53
24	2008-01-01	4389.3	4409.42	0	4409.42	4389.3	20.12	0	5829.53	5829.53	24	0	100000	105829.53	105829.53

Math_CompoundAmortizationSchedule2

Math_CompoundAmortizationSchedule2(@LoanAmount MONEY, @AnnualInterestRate DECIMAL(8, 5), @YearsOfLoan INT, @PaymentsPerYear INT, @LoanStartDate DATETIME, @OptionalExtraPayment MONEY, @UseStandardRounding BIT)

RETURNS: TABLE (PaymentNum INT, PaymentDate DATETIME, BeginningBalance MONEY, ScheduledPayment MONEY, ExtraPayment MONEY, TotalPayment MONEY, Principal MONEY, Interest MONEY, EndingBalance MONEY, CumulativeInterest MONEY, TotalInterest MONEY, TotalPayments INT, PaymentsLeft INT, CumulativePrincipal MONEY, CumulativeAmountPaid MONEY, TotalAmountPaid MONEY)

NOTES:

- **If you don't have a specific need to use FLOAT datatypes for input parameters and result set fields, then use Math_CompoundAmortizationSchedule2 as it won't have rounding errors due to FLOAT.**
- Same basic calculation as Math_CompoundAmortizationSchedule and same result set, but produces more accurate results due to different datatypes for input parameters and result set fields. Math_CompoundAmortizationSchedule2 uses DECIMAL / MONEY instead of FLOAT (which Math_CompoundAmortizationSchedule uses). DECIMAL / MONEY datatypes are "precise" whereas FLOAT is "imprecise" and more prone to rounding errors. Math_CompoundAmortizationSchedule2 also, by default, uses "Banker's Rounding" which is preffered for financial calculations.
- @LoanAmount must be >= 0
- @YearsOfLoan must be >= 1
- @PaymentsPerYear must be >= 1
- @OptionalExtraPayment must be >= 0
- @UseStandardRounding:
  - 1 / True = use "Away From Zero" rounding. This type of rounding is what most people are familiar with. It always goes up to the next (higher) positive value, or to the next (lower) negative value if the value is negative.
  - 0 / False = use "To Even" rounding. This type of rounding is also known as "Banker's Rounding" and does not always go in the same direction.
  - Please see the following MSDN documentation for more details: MidpointRounding Enumeration
  - If unsure which type of rounding to use, then use "To Even" by setting this parameter to 0 / False.

EXAMPLE:

```
SELECT * FROM SQL#.Math_CompoundAmortizationSchedule2(100000, 5.5, 2, 12,
'1/1/2006', 0, 0)
```

Math_Constant

Math_Constant(ConstantName NVARCHAR(4000))

RETURNS: FLOAT

NOTES:

1. SpeedOfLight
2. Gravity
3. GravitationalAcceleration
4. ElectronMass
5. ProtonMass
6. NeutronMass

7. AtomicMassUnit
8. ElectronCharge
9. Planck
10. Boltzmann
11. MagneticPermeability
12. DielectricPermittivity
13. ClassicalElectronRadius
14. FineStructure
15. BohrRadius
16. Rydberg
17. FluxQuantum
18. BohrMagneton
19. ElectronMagnetMoment
20. NuclearMagneton
21. ProtonMagnetMoment
22. NeutronMagnetMoment
23. ComptonElectronWavelength
24. ComptonProtonWavelength
25. Stefan-Boltzmann
26. Avogadro
27. IdealGasVolume
28. Gas
29. Faraday
30. QuantumHoleResistance

EXAMPLES:
```sql
SELECT SQL#.Math_Constant('SpeedOfLight')
-- 299792458
SELECT SQL#.Math_Constant('GAS')
-- 8.31451
```

Math_Convert

Math_Convert(BaseNumber FLOAT, From NVARCHAR(4000), To NVARCHAR(4000))

RETURNS: FLOAT

You can convert between any of the units of measurement within a group, but not between groups (duh!).

NOTES:
- Returns NULL if any input parameter is NULL
- Distance & Length
  1. Nanometer
  2. Micrometer
  3. Millimeter
  4. Centimeter
  5. Meter
  6. Kilometer
  7. Inch
  8. Foot
  9. Yard
  10. Mile
- Temperature
  1. Kelvin
  2. Celsius

3. Fahrenheit
4. Rankine
5. Reaumur

- Computer Data Size
  1. Bit
  2. Byte
  3. Kilobyte
  4. Megabyte
  5. Gigabyte
  6. Terabyte
  7. Petabyte

EXAMPLES:
```
SELECT SQL#.Math_Convert(1.0, 'yard' ,'mile')
-- 0.000568181818181818
SELECT SQL#.Math_Convert(1.0, 'mile' ,'centimeter')
-- 160934.4
SELECT SQL#.Math_Convert(-40.0, 'fahrenheit' ,'celsius')
-- -40
SELECT SQL#.Math_Convert(0, 'kelvin' ,'celsius')
-- -273.15
```

Math_Cosh

Math_Cosh(BaseNumber FLOAT)

RETURNS: FLOAT

Returns the hyperbolic cosine of the specified angle.

EXAMPLE:
```
SELECT SQL#.Math_Cosh(1.5)
-- 2.35240961524325
```

Math_CubeRoot

Math_CubeRoot(BaseNumber FLOAT)

RETURNS: FLOAT

Returns the cube root of the specified number.

NOTES:

- Same as Math_NthRoot(@Number, 3) but faster
- Same as PostgreSQL function: cbrt

EXAMPLE:
```
SELECT SQL#.Math_CubeRoot(27)
-- 3
SELECT SQL#.Math_CubeRoot(28)
-- 3.03658897187566
```

Math_Factorial

Math_Factorial(BaseNumber INT)

RETURNS: INT

NOTES: BaseNumber BETWEEN 0 and 170

EXAMPLE:
```
SELECT SQL#.Math_Factorial(10)
-- 3628800
```

Math_FormatDecimal

Math_FormatDecimal(TheNumber DECIMAL(38, 18), NumberFormat NVARCHAR(4000), Culture NVARCHAR(10))

RETURNS: NVARCHAR(4000)

Returns a string representing the number in the specified format and optional culture.

NOTES:

- NumberFormat
  - Standard formats: http://msdn.microsoft.com/en-us/library/dwhawy9k(VS.80).aspx
  - Custom formats: http://msdn.microsoft.com/en-us/library/0c899ak8(VS.80).aspx
- Culture
  - Optional, use empty string (‘’) to default to “current culture”
  - Available culture names: http://msdn.microsoft.com/en-US/library/system.globalization.cultureinfo(v=vs.80).aspx
- Essentially the same as the new FORMAT command in SQL Server 2012: http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx

EXAMPLES:
```
SELECT SQL#.Math_FormatDecimal(12345678.0987654, 'C2', 'FR-fr')
-- 12 345 678,10 €
SELECT SQL#.Math_FormatDecimal(12345678.0987654, 'C4', 'ja-jp')
-- ¥12,345,678.0988
SELECT SQL#.Math_FormatDecimal(12345678.0987654, 'C', '')
-- $12,345,678.10
SELECT SQL#.Math_FormatDecimal(12345678.0987654, '### - ### . #|#|# // #', '')
-- 12345 - 678 . 0|9|8 // 8
```

Math_FormatFloat (Not available in Free version)

Math_FormatFloat(TheNumber FLOAT, NumberFormat NVARCHAR(4000), Culture NVARCHAR(10))

RETURNS: NVARCHAR(4000)

Returns a string representing the number in the specified format and optional culture.

NOTES:

- NumberFormat
  - Standard formats: http://msdn.microsoft.com/en-us/library/dwhawy9k(VS.80).aspx
  - Custom formats: http://msdn.microsoft.com/en-us/library/0c899ak8(VS.80).aspx
- Culture

    - Optional, use empty string (‘’) to default to “current culture”
    - Available culture names: http://msdn.microsoft.com/en-US/library/system.globalization.cultureinfo(v=vs.80).aspx
- Essentially the same as the new FORMAT command in SQL Server 2012: http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx

EXAMPLES:
```
SELECT SQL#.Math_FormatFloat(123.123192, 'N', 'ja-jp')
-- 123.12
SELECT SQL#.Math_FormatFloat(123.123192, 'p', 'FR-fr')
-- 12 312,32 %
SELECT SQL#.Math_FormatFloat(123.123192, 'P', '')
-- 12,312.32 %
```

Math_FormatInteger (Not available in Free version)

Math_FormatInteger(TheNumber BIGINT, NumberFormat NVARCHAR(4000), Culture NVARCHAR(10))

RETURNS: NVARCHAR(4000)

Returns a string representing the number in the specified format and optional culture.

NOTES:
- NumberFormat
    - Standard formats: http://msdn.microsoft.com/en-us/library/dwhawy9k(VS.80).aspx
    - Custom formats: http://msdn.microsoft.com/en-us/library/0c899ak8(VS.80).aspx
- Culture
    - Optional, use empty string (‘’) to default to “current culture”
    - Available culture names: http://msdn.microsoft.com/en-US/library/system.globalization.cultureinfo(v=vs.80).aspx
- Essentially the same as the new FORMAT command in SQL Server 2012: http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx

EXAMPLES:
```
SELECT SQL#.Math_FormatInteger(9223372036854775807, 'N', 'FR-fr')
-- 9 223 372 036 854 775 807,00
SELECT SQL#.Math_FormatInteger(123112, 'X', '')
-- 1E0E8
SELECT SQL#.Math_FormatInteger(9223372036854775807, '## - ## / ## h ## k ## l ##
(##)bob', '')
-- 9223372 - 03 / 68 h 54 k 77 l 58 (07)bob
```

Math_IEEERemainder (Not available in Free version)

Math_IEEERemainder(Dividend FLOAT, Divisor FLOAT)

RETURNS: FLOAT

NOTES:
- From the MSDN documentation:

This operation complies with the remainder operation defined in Section 5.1 of ANSI/IEEE Std 754-1985; IEEE Standard for Binary Floating-Point Arithmetic; Institute of Electrical and Electronics Engineers, Inc; 1985.

The IEEERemainder method is not the same as the modulus operator. Although both return the remainder after division, the formulas they use are different. The formula for theIEEERemainder method is:

```
IEEERemainder = dividend - (divisor * Math.Round(dividend / divisor))
```

In contrast, the formula for the modulus operator is:

```
Modulus = (Math.Abs(dividend) - (Math.Abs(divisor) *
          (Math.Floor(Math.Abs(dividend) / Math.Abs(divisor))))) *
          Math.Sign(dividend)
```

EXAMPLES:
```
SELECT SQL#.Math_IEEERemainder(25.4, 4.5), 25.4 % 4.5
--     -1.6                        2.9
SELECT SQL#.Math_IEEERemainder(-16.3, 4.1), -16.3 % 4.1
--     0.0999999999999979          04.0
```

Math_IsPrime

Math_IsPrime(BaseNumber BIGINT)

RETURNS: BIT

EXAMPLE:
```
SELECT SQL#.Math_IsPrime(12318237133333)
-- 1
```

Math_NthRoot (Not available in Free version)

Math_NthRoot(BaseNumber FLOAT, Root FLOAT)

RETURNS: FLOAT

NOTES:

- If using Root value of 3, use Math_CubeRoot instead as it is slightly faster

EXAMPLES:
```
SELECT SQL#.Math_NthRoot(27, 3)
-- 3
SELECT SQL#.Math_NthRoot(27.5, 3.5)
-- 2.57773288800724
```

Math_RandomRange

Math_RandomRange(Seed INT, LowerBound INT, UpperBound INT)

RETURNS: INT

NOTES:

- Seed can be NULL
- Random number generation might not work as you anticipate; please see examples and notes below
- LowerBound must be less than or equal to the @UpperBound

EXAMPLE:

```
SELECT SQL#.Math_RandomRange(NULL, -10, 10), RAND()
SELECT SQL#.Math_RandomRange(NULL, -10, 10), RAND(4)
SELECT SQL#.Math_RandomRange(NULL, -10, 10), RAND(4), RAND()

SELECT *, RAND(ints.IntNum), RAND(),
       SQL#.Math_RandomRange(ints.IntNum, 1, 8),
       SQL#.Math_RandomRange(NULL, 1, 8)
FROM   SQL#.Util_GenerateInts(1, 200, 2) ints

Num    Val    RAND(Num)         RAND()            RR(Num)     RR(NULL)
1      1      0.713591993212924 0.842605911809958 2           2
2      3      0.713610626184182 0.842605911809958 6           2
3      5      0.71362925915544  0.842605911809958 3           2
4      7      0.713647892126698 0.842605911809958 6           2
5      9      0.713666525097956 0.842605911809958 3           1
6      11     0.713685158069215 0.842605911809958 7           1

Num    Val    RAND(Num)         RAND()            RR(Num)     RR(NULL)
1      1      0.713591993212924 0.842605911809958 2           4
2      3      0.713610626184182 0.842605911809958 6           4
3      5      0.71362925915544  0.842605911809958 3           4
4      7      0.713647892126698 0.842605911809958 6           4
5      9      0.713666525097956 0.842605911809958 3           4
6      11     0.713685158069215 0.842605911809958 7           4

SELECT RAND(),
       SQL#.Math_RandomRange(NULL, 1, 8)
FROM   SQL#.Util_GenerateInts(1, 200, 2) ints
```

Randomize functions, in native T-SQL or even in .Net languages do not produce truly random numbers. How they are used plays a large role in how they generate numbers. For example, to run RAND() by itself across several executions will produce a different number each time. However if you pass in a seed, such as calling RAND(4) will produce the same number each time. One nuance that is not obvious is that the number generate by RAND() will only be random if called only once in a batch—yes, in a batch, not just a single query. To see the effect of this, try the top three examples above. Run each one independently several times. Then, run all three at the same time several times. You will notice that once RAND() and RAND(4) are combined in the same batch (such as when highlighting just the top 2 SELECT statements and executing) the RAND() function works differently than when only the first SELECT statement above is ran several times.

The Math_RandomRange function provides something that the native RAND() function cannot: changing values between executions and even sometimes within a single execution in a result set of more than one row. The output shown below the SELECT statements is the first six rows returned from the fourth SELECT statement above (the one with the FROM clause). As you can see, the RAND() function returns the same value across all rows, whether or not a seed value is passed in. Across several executions of this query the same numbers are always produced for; although if you removed the RAND(ints.IntNum) column the RAND() column would produce a different number each time, but it would still be the same across all rows. Now, looking at the two Math_RandomRange columns (RR) we can see that when passing in a see value, the return values are different across each row, but they will also be same the values across multiple executions of the query, just like we see with the RAND(ints.IntNum) function call. What is truly different here is the

Math_RandomRange call when passing in NULL as the seed value. In this case we get two benefits over the T-SQL RAND() function: first is that across several executions of the query the return values will be different, and second is that sometimes the return values per row can differ—something not possible even when using RAND() by itself in a query! When you run the fourth query above, look through all 200 rows returned and you can see that the value changes at least once. The fifth (and final) query above is a simple side-by-side comparison of this so you can see more clearly.

Math_RoundToEvenDecimal

Math_RoundToEvenDecimal(BaseNumber DECIMAL(28,10), DecimalPlaces TINYINT)

RETURNS: DECIMAL(28,10)

Returns the @BaseNumber, rounded to @DecimalPlaces, using the “To Even” method of rounding (a.k.a. “Bankers’ Rounding”)

NOTES:

- The built-in ROUND() function uses the typical “away from zero” method. The “away from zero” method will round a positive number up and a negative number down.
- The “to even” method of rounding (i.e. “Bankers’ Rounding”) is often used in financial calculations. It rounds the exact midpoint (5 to the right of where the number is being rounded to, with no non-zero digits to the right of the 5) either up or down such that the resulting rounded number is even, regardless of @BaseNumber being positive or negative. Meaning:
  - 7.5, rounded to 0 decimal places, rounds up to 8.0
  - 8.5, rounded to 0 decimal places, rounds down to 8.0
  - 8.5000001, rounded to 1 decimal places, rounds up to 9.0
- DECIMAL is more precise (meaning: better for financial calculations) than FLOAT, but it is slightly slower as a parameter / return type. The performance difference is not noticeable for single operations, but can become noticeable when used in set-based operations over a large number of rows. Test, and if necessary, you can try using Math_RoundToEvenFloat.

EXAMPLE:

```
SELECT SQL#.Math_RoundToEvenDecimal(7.5, 0),       -- 8.00000
       SQL#.Math_RoundToEvenDecimal(8.5, 0),       -- 8.00000
       SQL#.Math_RoundToEvenDecimal(8.5000001, 0); -- 9.00000
```

Math_RoundToEvenFloat

Math_RoundToEvenFloat(BaseNumber FLOAT, DecimalPlaces TINYINT)

RETURNS: FLOAT

Returns the @BaseNumber, rounded to @DecimalPlaces, using the “To Even” method of rounding (a.k.a. “Bankers’ Rounding”)

NOTES:

- The built-in ROUND() function uses the typical “away from zero” method. The “away from zero” method will round a positive number up and a negative number down.
- The “to even” method of rounding (i.e. “Bankers’ Rounding”) is often used in financial calculations. It rounds the exact midpoint (5 to the right of where the number is being rounded to, with no non-zero digits to the right of the 5) either up or down such that the resulting rounded number is even, regardless of @BaseNumber being positive or negative. Meaning:
  - 7.5, rounded to 0 decimal places, rounds up to 8.0
  - 8.5, rounded to 0 decimal places, rounds down to 8.0

  - 8.5000001, rounded to 1 decimal places, rounds up to 9.0
- DECIMAL is more precise (meaning: better for financial calculations) than FLOAT, but it is slightly slower as a parameter / return type. The performance difference is not noticeable for single operations, but can become noticeable when used in set-based operations over a large number of rows. If doing financial calculations, try to use Math_RoundToEvenDecimal.

EXAMPLE:
```
SELECT SQL#.Math_RoundToEvenFloat(7.5, 0),        -- 8
       SQL#.Math_RoundToEvenFloat(8.5, 0),        -- 8
       SQL#.Math_RoundToEvenFloat(8.5000001, 0); -- 9
```

Math_Sinh

Math_Sinh(BaseNumber FLOAT)

RETURNS: FLOAT

Returns the hyperbolic sine of the specified angle

EXAMPLE:
```
SELECT SQL#.Math_Sinh(1.5)
-- 2.12927945509482
```

Math_Tanh

Math_Tanh(BaseNumber FLOAT)

RETURNS: FLOAT

Returns the hyperbolic tangent of the specified angle.

EXAMPLE:
```
SELECT SQL#.Math_Tanh(1.5)
-- 0.905148253644866
```

Math_Truncate

Math_Truncate(BaseNumber FLOAT, DecimalPlaces TINYINT)

RETURNS: FLOAT

NOTES:
- This does not round up or down; it merely chops the value off at the specified decimal place
- This is the same as the PostgreSQL function: trunc

EXAMPLE:
```
SELECT SQL#.Math_Truncate(123.4567, 2)
-- 123.45
```

Network

The **INET** functions reside in the SQL#.Network assembly. The following assemblies require the SQL#.Network assembly to be installed in order to use them: SQL#.DB.

If you use any of the functions that access the file system or network, then this assembly will need a security setting of EXTERNAL_ACCESS (2). You can set this by executing the following query:

```
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.Network'
```

If you do not want to have this assembly in your system at all, you can do either of the following:

- Do not install the SQL#.Network assembly by setting the `@InstallSQL#Network` variable (towards the top of the script) to 0 before installing
- Uninstall the assembly by running:
  `EXEC SQL#.SQLsharp_Uninstall N'SQL#.Network'`

Please note:

- When accessing the file system, the Operating System user account that will be used is the one that is currently running (i.e. “log on as”) the main SQL Server process (it might be Local System Account or an account created specifically for SQL Server).
- The default *concurrent* connection limit to any particular URI (i.e. protocol + host name + domain name, *not* including path or query string) is 20. The .NET default limit is 2 which causes a bottleneck when more than 2 requests are made to the same URI and the initial requests have not completed yet.

  The way .NET handles network connections is that when the *concurrent* connection limit (for any particular URI) has been reached, all additional requests are forced to wait until one of the active connections completes. The concurrent connection limit is per App Domain, and all methods within an Assembly (and all Assemblies within the same Database and having the same owner) are in the same App Domain. This means that *all* Sessions executing the same SQLCLR Function (or Stored Procedure, etc) are using the same App Domain and hence the same current connection count and concurrent connection limit.

  If there is a process using any networking function that does actual network access (typically INET_GetWebPages) to the same URI *and* multiple Sessions execute it at the same time, then it is very easy to experience delays when using the default concurrent connection limit of 2. This is why, as of SQL# Version 4.1, the default concurrent connection limit has been raised to 20. If you need to set a limit that is higher (or even lower, to prevent hitting the remote resource too frequently), or are using SQL# Version 4.0 and need to set a limit that is higher than 2, then do one of the following:
  1. SELECT or EXECUTE INET_SetConnectionLimitForURI
  2. Use the “Connection

INET_AddressToNumber

INET_AddressToNumber(IPAddress NVARCHAR(4000))

RETURNS: BIGINT

Converts standard four-part dotted IP Address (IPv4) into a single numerical equivalent. This function mirrors (mostly) INET_ATON in MySQL and ip2long in PHP.

NOTES:

- IF IPAddress IS NULL, is an empty string, or is not a valid IP Address, NULL is returned

- See also: INET_NumberToAddress

EXAMPLES:
```
SELECT SQL#.INET_AddressToNumber('192.168.1.100')
-- 3232235876
SELECT SQL#.INET_AddressToNumber('192.168.1.300')
-- NULL
```

INET_DownloadFile (Not available in Free version)

INET_DownloadFile(URI NVARCHAR(4000), FileName NVARCHAR(4000))

RETURNS: VARBINARY(8000)

Retrieves a file from the specified URI either as a scalar value OR to a file.

NOTES:

- URI:
  - Is the full location of the file, starting with the protocol ("http://", etc.)
  - If NULL or empty string '' a NULL will be returned
- FileName:
  - IF NULL or empty string '' the contents of the remote file will be returned as the scalar value
  - IF a value, it should be the full path to the filename that will be created from the contents of the remote file
  - IF a value, scalar value returned is 0x00
- Downloading directly can also be done via INET_GetWebPages but this is easier if you don't need all of the options available in GetWebPages or if you want to download directly to a file.

EXAMPLES:
```
SELECT SQL#.INET_DownloadFile('http://google.com/images/logos/ps_logo2.png', '')
-- 0x89504E470D0A1A0A0000000D49484452000016C0000007E08020000008F94BCCA...
SELECT SQL#.INET_DownloadFile('http://google.com/images/logos/ps_logo2.png',
'c:\GoogleLogo.png')
-- 0x00
DECLARE @HTML VARCHAR(MAX)
SELECT @HTML = CONVERT(VARCHAR(MAX),
      SQL#.INET_DownloadFile('http://www.google.com', ''))
SELECT @HTML
```

INET_FTPDo (Not available in Free version)

INET_FTPDo(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, RenameTo NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Address must begin with "ftp://"
- Options for FTPCommand:
  - del | delete | DeleteFile
  - mk | mkdir | MakeDirectory
  - rm | rmdir | RemoveDirectory
  - ren | rename
- FTPCommand options are NOT case-sensitive

- RenameTo is only used and required if FTPCommand = ren | rename
- Return value is completion status

EXAMPLES:
```
SELECT SQL#.INET_FTPDo('ftp://sqlsharp.com/favicon.ico', 'xxxx', 'xxxx', 'ren',
0, 'favicon.txt')

SELECT SQL#.INET_FTPDo('ftp://sqlsharp.com/favicon.txt', 'xxxx', 'xxxx', 'del',
0, '')
```

INET_FTPGet (Not available in Free version)

INET_FTPGet(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, ContentOffset BIGINT)

RETURNS: NVARCHAR(MAX)

Gets a non-Binary file from an FTP location to a return value.

NOTES:
- This is for ASCII / Text files only; this command does NOT work with Binary files or VARBINARY data
- Address must begin with “ftp://”
- Options for FTPCommand:
  - get | recv | DownloadFile
  - ls | ListDirectory
  - dir | ListDirectoryDetails
- FTPCommand options are NOT case-sensitive
- ContentOffset is how many bytes from the beginning of the file to skip. This number has to be >= 0 and if > 0 then FTP will use the RESTART command which can resume a previously stopped download.
- SQL Server Integration Services (SSIS) can FTP a file from a server, but only to disk, not to a local variable and hence not directly to a column in a table, nor does SSIS support incremental downloads.

EXAMPLES:
```
SELECT SQL#.INET_FTPGet('ftp://sqlsharp.com/ ', 'xxxx', 'xxxx', 'dir', 0, 0)
SELECT SQL#.INET_FTPGet('ftp://sqlsharp.com/index.html', 'xxxx', 'xxxx', 'get',
0, 0)
SELECT SQL#.INET_FTPGet('ftp://sqlsharp.com/index.html', 'xxxx', 'xxxx', 'get',
0, 100)
```

INET_FTPGetBinary (Not available in Free version)

INET_FTPGetBinary(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, ContentOffset BIGINT)

RETURNS: VARBINARY(MAX)

Gets a Binary file from an FTP location into a return value.

NOTES:
- This is mainly for Binary files or VARBINARY data; if used for ASCII files, it will NOT do the translation of OS specific items such as CRLF <=> LF
- Address must begin with “ftp://”
- Options for @FTPCommand:

    - get | recv | DownloadFile
    - ls | ListDirectory
    - dir | ListDirectoryDetails
- FTPCommand options are NOT case-sensitive
- ContentOffset is how many bytes from the beginning of the file to skip. This number has to be >= 0 and if > 0 then FTP will use the RESTART command which can resume a previously stopped download.
- SQL Server Integration Services (SSIS) can FTP a file from a server, but only to disk, not to a local variable and hence not directly to a column in a table, nor does SSIS support incremental downloads.
- Sometimes an FTP error is thrown (`The remote server returned an error: (503) Bad sequence of commands.`), just try again and it should work.

EXAMPLES:

```
DECLARE @File VARBINARY(MAX)
SELECT @File = SQL#.INET_FTPGetBinary('ftp://www.domain.com/file.zip', 'login',
'passwd', 'get', 0, 0)
SELECT @File = SQL#.INET_FTPGetBinary('ftp://www.domain.com/file.zip', 'login',
'passwd', 'get', 0, 100)
```

INET_FTPGetFile (Not available in Free version)

INET_FTPGetFile(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, BinaryMode BIT, FilePath NVARCHAR(4000), FileHandling TINYINT)

RETURNS: NVARCHAR(4000)

Gets a file from an FTP location directly to disk.

NOTES:

- If BinaryMode is set to False / 0, it will NOT do the translation of OS specific items such as CRLF <=> LF
- Address must begin with “ftp://”
- Options for @FTPCommand:
    - get | recv | DownloadFile
    - ls | ListDirectory
    - dir | ListDirectoryDetails
- FTPCommand options are NOT case-sensitive
- FileHandling values:
    - 0 – do NOT overwrite an existing file – if the file already exists, you will get an FTP error (`The remote server returned an error: (451) Local error in processing.`)
    - 1 – overwrite existing files
    - 2 – Incremental download – if the file already exists, autodetect where to start the download from in the remote file to “resume” the download

EXAMPLES:

```
SELECT SQL#.INET_FTPGetFile('ftp://www.domain.com/file.zip', 'login', 'passwd',
'get', 0, 1, 'C:\file.zip', 0)
```

INET_FTPPut (Not available in Free version)

INET_FTPPut(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, FileData NVARCHAR(MAX))

RETURNS: NVARCHAR(4000)

Sends a non-Binary file from an input parameter to an FTP location.

NOTES:

- This is for ASCII / Text files only; this command does NOT work with Binary files or VARBINARY data
- Address must begin with "ftp://"
- Options for FTPCommand:
    - app | append | AppendFile
    - put | send | UploadFile
- FTPCommand options are NOT case-sensitive
- Return value is completion status
- SQL Server Integration Services (SSIS) can FTP a file to a server, but only from disk, not from a local variable and hence not directly from a column in a table.

EXAMPLES:

```
SELECT SQL#.INET_FTPPut('ftp://sqlsharp.com/test.txt', 'xxxx', 'xxxx', 'put', 0,
'this is a test, duh!')

SELECT so.name, so.object_id INTO #temp_name FROM sys.objects so
SELECT SQL#.INET_FTPPut('ftp://sqlsharp.com/test2.txt', 'xxxx', 'xxxx', 'put',
0, SQL#.String_Join('SELECT name + '','' + CONVERT(NVARCHAR, object_id) FROM
#temp_name', CHAR(13)+CHAR(10), 1))
DROP TABLE #temp_name
```

INET_FTPPutBinary (Not available in Free version)

INET_FTPPutBinary(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, FileData VARBINARY(MAX))

RETURNS: NVARCHAR(4000)

Sends a Binary file from an input parameter to an FTP location.

NOTES:

- This is mainly for Binary files or VARBINARY data; if used for ASCII files, it will NOT do the translation of OS specific items such as CRLF <=> LF
- Address must begin with "ftp://"
- Options for FTPCommand:
    - app | append | AppendFile
    - put | send | UploadFile
- FTPCommand options are NOT case-sensitive
- Return value is completion status
- SQL Server Integration Services (SSIS) can FTP a file to a server, but only from disk, not from a local variable and hence not directly from a column in a table.

EXAMPLES:

```
DECLARE @File VARBINARY(MAX)

SELECT      @File = FileData
FROM        dbo.Documents
WHERE       FileId = @FileId

SELECT SQL#.INET_FTPPutBinary('ftp://www.domain.com/file.zip', 'login',
'passwd', 'put', 0, @File)
```

INET_FTPPutFile (Not available in Free version)

INET_FTPPutBinaryFile(Address NVARCHAR(4000), User NVARCHAR(4000), Password NVARCHAR(4000), FTPCommand NVARCHAR(4000), UseSSL BIT, BinaryMode BIT, FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Sends a file from disk to an FTP location.

NOTES:

- If BinaryMode is set to True / 1, it will NOT do the translation of OS specific items such as CRLF <=> LF
- Address must begin with “ftp://”
- Options for FTPCommand:
    - app | append | AppendFile
    - put | send | UploadFile
- FTPCommand options are NOT case-sensitive
- Return value is completion status

EXAMPLES:
```
SELECT SQL#.INET_FTPPutFile('ftp://www.domain.com/file.zip', 'login', 'passwd',
'put', 0, 1, 'C:\file.zip')
```

INET_GetConnectionLimitForURI (Not available in Free version)

INET_GetConnectionLimitForURI(@URI NVARCHAR(MAX))

RETURNS: INT

Gets the concurrent connection limit for the specified URI.

NOTES:

- URI = protocol + host name + domain name; it does *not* include path or query string.
- Default connection limit for URIs
    - Prior to SQL# Version 4.1: **2** (.NET default)
    - Starting in SQL# Version 4.1: **20**
- You can change the connection limit using:
    - In SQL# Version 4.0: only INET_SetConnectionLimitForURI
    - Starting in SQL# Version 4.1: "ConnectionLimit" pseudo-HTTP header for INET_GetWebPages
- See also: INET_GetCurrentConnectionCountForURI

EXAMPLE:
```
SELECT SQL#.INET_GetConnectionLimitForURI(N'https://host.domain.tld/');
```

INET_GetCurrentConnectionCountForURI (Not available in Free version)

INET_GetCurrentConnectionCountForURI(@URI NVARCHAR(MAX))

RETURNS: INT

Gets the current number of active connections for the specified URI.

NOTES:

- URI = protocol + host name + domain name; it does *not* include path or query string.
- Default connection limit for URIs
    - Prior to SQL# Version 4.1: **2** (.NET default)
    - Starting in SQL# Version 4.1: **20**
- You can change the connection limit using:
    - In SQL# Version 4.0: only INET_SetConnectionLimitForURI
    - Starting in SQL# Version 4.1: "ConnectionLimit" pseudo-HTTP header for INET_GetWebPages
- See also: INET_GetConnectionLimitForURI

EXAMPLE:
```
SELECT SQL#.INET_GetCurrentConnectionCountForURI(N'https://host.domain.tld/');
```

INET_GetHostName (Not available in Free version)

INET_GetHostName(Address NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Returns NULL if NULL passed in.

EXAMPLE:
```
SELECT SQL#.INET_GetHostName('209.73.186.238');
-- f1.www.vip.re3.yahoo.com
```

INET_GetIPAddress (Not available in Free version)

INET_GetIPAddress(HostName NVARCHAR(126))

RETURNS: NVARCHAR(50)

EXAMPLE:
```
SELECT SQL#.INET_GetIPAddress('google.com');
-- 74.125.65.147
```

INET_GetIPAddressList (Not available in Free version)

INET_GetIPAddressList(HostName NVARCHAR(126))

RETURNS: TABLE (IPAddress NVARCHAR(50))

NOTES:

- Returns an empty result set if NULL is passed in for @HostName.

EXAMPLE:
```
SELECT * FROM SQL#.INET_GetIPAddressList('google.com');
--74.125.65.147
--74.125.65.99
--74.125.65.103
--74.125.65.104
```

```
--74.125.65.105
--74.125.65.106
```

INET_GetWebPages (Not available in Free version)

INET_GetWebPages(URI NVARCHAR(4000), SplitLines SMALLINT, TrapErrorInline BIT, MaximumAutomaticRedirections SMALLINT, Timeout INT, MaximumResponseHeadersLength INT, CustomHeaders SQL#.Type_HashTable, Method NVARCHAR(10), PostData NVARCHAR(MAX), ContentDetection NVARCHAR(10))

RETURNS: TABLE (num INT, line_num INT, content_encoding NVARCHAR(50), content_length BIGINT, content_type NVARCHAR(50), Server NVARCHAR(500), Content NVARCHAR(MAX), IsFromCache BIT, LastModified DATETIME, StatusCode INT, StatusDescription NVARCHAR(1000), ResponseUri NVARCHAR(4000), ContentBinary VARBINARY(MAX), ResponseHeaders XML, CharacterSet NVARCHAR(100))

NOTES:

- URI must begin with “http://” or “https://”
- SplitLines:
  - 0 = return 1 row with LineNum = COUNT(lines)
  - 1 = split HTML content per newline
  - -1 = skip content download entirely. The effect of this is getting a very quick response containing the status, headers, content-type, response URI, LastModified date, and sometimes the content-length (this value is not always sent). For example: downloading an 11 MB file, including content (no split) was taking 1200 - 2100 ms. Using the new -1 option to skip downloading the content was returning in 90 - 160 ms.
- TrapErrorInline:
  - 0 (or NULL) = HTTP errors throw an exception that can be caught via TRY / CATCH block
  - 1 = HTTP errors are caught and returned in result set
- MaximumAutomaticRedirections:
  - The number of times the process will automatically redirect after receiving a 300 level response before giving the final response.
  - A value of < 0 will result in the default value of 50 being used.
- Timeout:
  - The number of Milliseconds before the operation times-out.
  - Set to -1 for Unlimited
  - A setting of 0 will always timeout
- MaximumResponseHeadersLength:
  - The maximum size in kilobytes (1024 bytes) of the Response headers.
  - Set to -1 for Unlimited
  - A setting of 0 will make all requests fail
- CustomHeaders:
  - A collection of Name-Value pairs injected into the request headers via the SQL#.Type_HashTable User-Defined Type
  - An alternative to creating, populating, and passing in a Type_HashTable UDT is to pass in a string that could be used to populate a Type_HashTable UDT. The string needs to be in the form of “Header1=Value1&Header2=Value2...”. If either the “&” or “=” characters are needed for the value, use “%26” in place of “&”, and “%3D” in place of “=”.
  - Set to NULL when not using Custom Headers
  - Known headers (do not use “HTTP_” prefix in names: use “REFERER” instead of “HTTP_REFERER”):
    - **Authorization** (please see “Username” and “Password” in SQL#-specific pseudo-headers list; Do not pass this header in: setting both “Username” and “Password” will cause a properly formatted “Authorization” header to be sent)
    - **AutomaticDecompression** (comma or pipe separated options: gzip, deflate, or both)

        - **Connection** (Value needs to be either “Keep-alive” or “Close”)
        - **Content-Type**
        - **Keep-alive** (use by itself; no Value to pass in)
        - **Range** (Value is in the form of “{range-unit} = [{start}] – [{end}]”; typically {range-unit} is “bytes”, and {start} and {end} are 0-based byte offsets from the first byte; possibilities are: “bytes=5-100”, “bytes=-100”, and “bytes=5-”)
        - **Referer**
        - **Transfer-encoding** (setting a value also sets “SendChunked” to “true”)
        - **User-Agent**
    - SQL#-specific pseudo-headers:
        - **ConnectionLimit** (Value is an INT)
        - **Password** (used along with “Username” to send “Authorization” header)
        - **Proxy** (Value is in the form of “{proxy_server}:{proxy_port_number}”; for Fiddler use a value of: `'127.0.0.1:8888'`)
        - **Username** (used along with “Password” to send “Authorization” header)
- Method:
    - NOT case-sensitive
    - Can be: Get, Post, Head, Put, or Delete
    - If using “Post”, two custom headers will be automatically set:
        - ContentType set to “application/x-www-form-urlencoded”
        - ContentLength set to DATALENGTH(PostData)
- PostData:
    - Only needed if Method is set to “Post”
    - Data takes the form of “Var1=Value1&Var2=Value2...”
    - Values for each Variable should be URL encoded using INET_URIEncodeData
- ContentDetection:
    - Values are NOT case-sensitive
    - Text = Always interpret response data as text
    - Binary = Always interpret response data as binary
    - Auto = Interpret response data as text only if ContentType starts with “text\”, else interpret response data as binary
- Either Content XOR ContentBinary fields will be NULL
- Invalid “date_modified” in HTTP header returns 1900-01-01
- See also: INET_DownloadFile, INET_GetConnectionLimitForURI, INET_GetCurrentConnectionCountForURI, and INET_SetConnectionLimitForURI

EXAMPLES:
```
SELECT * FROM SQL#.INET_GetWebPages('http://www.yahoo.com/', 0, 0, 5, -1, -1,
NULL, NULL, NULL, 'Auto')
/*
1      209          -1     text/html; charset=utf-8            <html><head>
<title>Yahoo!</title> <meta http-equiv="...
*/
SELECT * FROM SQL#.INET_GetWebPages('http://www.yahoo.com/', 1, 0, 5, -1, -1,
NULL, NULL, NULL, 'Auto')
/*
1      1            -1     text/html; charset=utf-8            <html><head>
1      2            -1     text/html; charset=utf-8
       <title>Yahoo!</title>
1      3            -1     text/html; charset=utf-8            <meta http-
equiv="Content-Type" content="text/html; charset=UTF-8">
...
*/
```

```
SELECT StatusCode, StatusDescription FROM
SQL#.INET_GetWebPages('http://SQLsharp.com', 0, 1, 0, -1, -1, NULL, NULL, NULL,
'Auto')
-- 301       Moved Permanently
SELECT StatusCode, StatusDescription FROM
SQL#.INET_GetWebPages('http://SQLsharp.com', 0, 1, 2, -1, -1, NULL, NULL, NULL,
'Auto')
-- 200       OK

DECLARE @CustomHeaders SQL#.Type_HashTable -- name/value pairs
SET @CustomHeaders = @CustomHeaders.AddItem('USER_AGENT', 'SQL#')
SET @CustomHeaders = @CustomHeaders.AddItem('REFERER', 'my.site.com?var=val')
SELECT * FROM SQL#.INET_GetWebPages('http://www.SQLsharp.com', 0, 1, 5, -1, -1,
@CustomHeaders, 'GET', '', 'Text')
```

INET_HTMLDecode

INET_HTMLDecode(EncodedHTML NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Decodes a string, translating HTML-encoded characters into their decoded values.

NOTES:

- Handles all 2125 HTML 5 named character entity references
- Handles **&#{decimal_code_point};** and **&#x{hex_code_point};** numeric character references
- Please visit the following pages for details:
  - https://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references
  - https://www.w3.org/TR/html5/syntax.html#named-character-references

EXAMPLES:

```
PRINT SQL#.INET_HTMLDecode(N'&#128584;&#9;test<br>
&#x1F648;<br>
<br />
&Ofr;<BR/>
&lopf; -- &Lopf;');
/*
🙈	test<br>
🙈<br>
<br />
𝔒<BR/>
𝕝 -- 𝕃
*/
```

INET_HTMLEncode

INET_HTMLEncode(DecodedHTML NVARCHAR(MAX), WhiteSpaceHandling NVARCHAR(4000), ContinuousEncoding BIT)

RETURNS: NVARCHAR(MAX)

Encodes a string, translating characters either reserved for HTML mark-up or special characters into their HTML-specific values.

NOTES:

- *Does not handle all 2125 HTML 5 named character references (currently only 102 are handled). Full set will be handled starting in SQL# version 5.0*.
- WhiteSpaceHandling:
  - This option only controls how white-space (spaces and/or returns, but not tabs) are translated; all other applicable characters will always be translated
  - Value cannot be NULL
  - Value is NOT case-sensitive
  - Valid values are:
    - 'None' does not encode spaces or returns
    - 'Spaces' encodes only spaces
    - 'Returns' encodes only returns (\r\n, \r, and \n)
    - 'Both' encodes both spaces and returns
- ContinuousEncoding set to True / 1 will encode values already encoded (i.e. starting with an ampersand (&)). If set to False / 0, values already encoded will not have their ampersand turned into &amp;

EXAMPLES:
```
DECLARE @HTML NVARCHAR(MAX)

SET @HTML = 'This is a <b>test</b>.
Encoded Character: &amp;lt;'

SELECT SQL#.INET_HTMLEncode(@HTML, 'spaces', 0)
/*
This&nbsp;is&nbsp;a&nbsp;&lt;b&gt;test&lt;/b&gt;.
Encoded&nbsp;Character:&nbsp;&amp;lt;
*/
SELECT SQL#.INET_HTMLEncode(@HTML, 'returns', 1)
/*
This is a &lt;b&gt;test&lt;/b&gt;.<br />Encoded Character: &amp;amp;lt;
*/
```

INET_IsValidIPAddress

INET_IsValidIPAddress(IPAddress NVARCHAR(4000))

RETURNS: BIT

Validates whether supplied IPAddress represents a valid IPv4 IP Address is proper four-part dot-notation with each octect being between 0 and 255.

EXAMPLES:
```
SELECT SQL#.INET_IsValidIPAddress('192.168.1.100')
-- 1
SELECT SQL#.INET_IsValidIPAddress('192.168.1.300')
-- 0
```

INET_NumberToAddress

INET_NumberToAddress(IPNumber BIGINT)

RETURNS: NVARCHAR(4000)

Converts a numerical IP Address equivalent to a standard four-part dotted IP Address (IPv4). This function mirrors (mostly) INET_NTOA in MySQL and long2ip in PHP.

NOTES:

- IF @IPNumber IS NULL, < 0, or > 4294967295, NULL is returned
- See also: INET_AddressToNumber

EXAMPLES:
```
SELECT SQL#.INET_NumberToAddress(3232235876)
-- 192.168.1.100
SELECT SQL#.INET_NumberToAddress(4294967296)
-- null
```

INET_Ping (Not available in Free version)

INET_Ping(HostName NVARCHAR(4000), PacketSize INT, TimeOut INT, TTL INT, DontFragment BIT, Iterations INT)

RETURNS: TABLE (Num INT, Status NVARCHAR(MAX), RoundTripTime FLOAT, Address NVARCHAR(MAX), BufferSize INT)

NOTES:

- Returns an empty result set if NULL is passed in for @HostName.
- @PacketSize BETWEEN 1 AND 65500
- @TimeOut BETWEEN 1 AND 10240
- @TTL BETWEEN 1 AND 10240
- @Iterations BETWEEN 1 AND 2048
- SocketError returns single row of "num" = -1, "status" = SocketError, and "address" = {message}.

EXAMPLE:
```
SELECT * FROM SQL#.INET_Ping('www.yahoo.com', 300, 1200, 20, 0, 5);
/*
1      Success      37     69.147.114.210     300
2      Success      147    69.147.114.210     300
3      Success      31     69.147.114.210     300
4      Success      141    69.147.114.210     300
5      Success      33     69.147.114.210     300
*/
```

INET_PingTime (Not available in Free version)

INET_PingTime(HostName NVARCHAR(4000), PacketSize INT, TimeOut INT, TTL INT, DontFragment BIT)

RETURNS: FLOAT

NOTES:

- @PacketSize BETWEEN 1 AND 65500
- @TimeOut BETWEEN 1 AND 10240
- @TTL BETWEEN 1 AND 10240
- Return value is Number of Milliseconds
- Returns NULL if NULL passed in.
- Returns -2 for SocketError (typically DNS cannot resolve host).

EXAMPLE:
```
SELECT SQL#.INET_PingTime('www.yahoo.com', 3000, 1200, 200, 0);
-- 43
```

INET_SetConnectionLimitForURI (Not available in Free version)

INET_SetConnectionLimitForURI(@URI NVARCHAR(MAX), @ConnectionLimit INT)

RETURNS: INT

Sets the concurrent connection limit for the specified URI.

NOTES:

- URI = protocol + host name + domain name; it does *not* include path or query string.
- Default connection limit for URIs
    - Prior to SQL# Version 4.1: **2** (.NET default)
    - Starting in SQL# Version 4.1: **20**
- You can change the connection limit using:
    - In SQL# Version 4.0: only INET_SetConnectionLimitForURI (this function)
    - Starting in SQL# Version 4.1: "ConnectionLimit" pseudo-HTTP header for INET_GetWebPages
- If the App Domain gets unloaded after using INET_SetConnectionLimitForURI, then the connection limit for all URIs will go back to the default value (**2** for SQL# Version 4.0, **20** starting with SQL# Version 4.1). To ensure that the connection limit will always be the intended value, call INET_SetConnectionLimitForURI prior to making any network request, or, if using SQL# Version 4.1 or later, pass in the "ConnectionLimit" pseudo-HTTP header when using INET_GetWebPages.
- Return value is the connection limit for the specified URI prior to setting it to @ConnectionLimit.
- If you do not want the return value, try executing the UDF instead of SELECTing it (see example)
- See also: INET_GetConnectionLimitForURI and INET_GetCurrentConnectionCountForURI

EXAMPLE:

```
SELECT SQL#.INET_SetConnectionLimitForURI(N'http://host.site.tld/page.php', 45);
-- 20

EXEC SQL#.INET_SetConnectionLimitForURI N'http://host.domain.tld/page.aspx', 31;
```

INET_SplitIntoFields (Not available in Free version)

INET_SplitIntoFields @URI NVARCHAR(4000) [, @Timeout INT] [, @RegExDelimiter NVARCHAR(4000)] [, @RowsToSkip INT] [, @ColumnNames NVARCHAR(4000)] [, @FileEncoding NVARCHAR(20)] [, @DataTypes NVARCHAR(4000)] [, @FirstRowContainsColumnNames BIT]

PROC: Result set is each row of delimited text returned by @URI, broken into fields based on @RegExDelimiter.

NOTES:

- @URI:
    - Cannot be NULL or empty string
    - Location (file or page) should respond with delimited text
- @Timeout:
    - Optional parameter
    - How many milliseconds to wait for a response before timing out
    - Set to -1 for unlimited
    - Default value = 100,000 (100 seconds)
- @RegExDelimiter:
    - Optional parameter
    - A full Regular Expression (See RegEx section)

 - Default value = `N','`
- @RowsToSkip:
  - Optional parameter
  - Default = 0
  - Use = 1 to ignore header row (also see @FirstRowContainsColumnNames parameter)
- @ColumnNames:
  - Optional parameter
  - Comma-separated list of values that will be used to name the columns of the result set
  - Extra spaces around each name will be trimmed
  - If more fields are in the data than specified in ColumnNames then additional fields will be named as FieldN where N is the field number
  - If more fields are specified in ColumnNames than in the first row of the result set then extra Column Names will be ignored
  - If not set or set to NULL then all field names will be FieldN where N is the field number starting with 1
- @FileEncoding:
  - Optional parameter
  - Value is NOT case-sensitive
  - Value can be:
    - ASCII
    - UNICODE [implied Little Endian]
    - UTF8
    - UTF7
    - UnicodeBigEndian
    - UTF32 [implied Little Endian]
    - Any other value, including NULL, will select your server's system default
  - Default value = NULL
- @DataTypes:
  - Optional parameter
  - Value is NOT case-sensitive
  - Comma-separated list of values that will be used to specify the datatype of the columns of the result set
  - If more fields are in the data than specified in DataTypes then additional fields will be set to NVARCHAR(MAX)
  - If more fields are specified in DataTypes than in the first row of the result set then extra values will be ignored
  - If not set or set to NULL then all field datatypes will be set to NVARCHAR(MAX)
  - Empty value in source data will return empty string for (N)(VAR)CHAR / XML datatypes, 0x00 for (VAR)BINARY, and NULL for number / date datatypes.
  - Currently, the TIME and DATETIMEOFFSET datatypes do not work properly.
- @FirstRowContainsColumnNames:
  - Optional parameter
  - If set to 1:
    - will use first row of file to set column names
    - overrides @ColumnNames parameter if it is also set
    - does not impact rows skipped by @RowsToSkip
    - if @RowsToSkip = 0, then @RowsToSkip will be assumed to be 1 so that the header row isn't used to both set column names and as the first row of data
  - Default value = 0 / false
- Number of fields returned in result set is based on first row of data returned (meaning, if @RowsToSkip = 1 then the first row of data is Row 2)
- After number of fields to return is set, rows with more fields will have the additional fields ignored
- After number of fields to return is set, rows with fewer fields will return empty strings for the missing fields
- See also: File_SplitIntoFields and String_SplitIntoFields

EXAMPLES:
```
EXEC SQL#.INET_SplitIntoFields 'http://www.place.tld/file.csv', -1, ','
-- split a comma-separated response, starting with the first row returned,
-- using "Field"# as the column names, and NVARCHAR(MAX) as all datatypes

EXEC SQL#.INET_SplitIntoFields 'http://www.place.tld/page.aspx', -1, '\t', 2,
'OrderID,OrderDate','INT,DATETIME'
-- split the tab-separated response, starting with the third row returned,
-- using "OrderID" and "OrderDate" for the first two column-names and
-- "Field"# for any remaining column-names, and setting the datatypes for
-- the first two columns to be INT and DATETIME while any remaining
-- columns will be NVARCHAR(MAX)
```

INET_URIDecode

INET_URIDecode(EncodedURI NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Decodes a string, translating URI-encoded characters into their decoded values.

EXAMPLES:
```
SELECT
SQL#.INET_URIDecode('http://www.test.tld/file%20with%20space.aspx?test=one%20two
')
-- http://www.test.tld/file with space.aspx?test=one two
```

INET_URIDecodePlus (Not available in Free version)

INET_URIDecodePlus(EncodedURI NVARCHAR(4000), TrapErrorsInline BIT, ErrorReplacement NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Decodes a string, translating URI-encoded characters into their decoded values. Unlike INET_URIDecode, it will also decode Unicode characters that were encoded with the non-standard %uXXYY encoding. INET_URIDecodePlus also has the option of trapping errors inline and reporting them via the return value as opposed to throwing a hard-error and failing like the regular INET_URIDecode does.

NOTES:

- ErrorReplacement:
    - Only used if TrapErrorsInline is set to 1
    - Will be the return value if EncodedURI produces an error
    - Can be NULL, empty string (''), or any replacement string
    - Can include optional replacement tag of {SQL#ErrorMessage} which will contain the actual error text
    - Replacement tag {SQL#ErrorMessage} is case-sensitive

EXAMPLES:
```
SELECT SQL#.INET_URIDecodePlus(N'bob+%ec', 0, '')
-- SQL error!
SELECT SQL#.INET_URIDecodePlus(N'bob+%8f', 1, 'error: {SQL#ErrorMessage}')
-- error: Invalid URI: There is an invalid sequence in the string.
SELECT SQL#.INET_URIDecodePlus(N'bob+%8f', 1, NULL)
```

```
-- NULL
SELECT SQL#.INET_URIDecodePlus(N'bob+%u00a4', 0, '')
-- bob ¤
```

INET_URIEncode

INET_URIEncode(DecodedURI NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Encodes a URI, translating special characters into their safe / appropriate values except for characters need to make the URI work, such as: **/**, **:**, **?**, **&**, **+**, and **#**.

EXAMPLES:
```
SELECT SQL#.INET_URIEncode('http://www.test.tld/file with space.aspx?test=one
two')
-- http://www.test.tld/file%20with%20space.aspx?test=one%20two
```

INET_URIEncodeData

INET_URIEncodeData(DecodedURIData NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

Encodes a URI data string, translating special characters into their safe / appropriate values including URI special characters such as: **/**, **:**, **?**, **&**, **+**, and **#**.

EXAMPLES:
```
SELECT SQL#.INET_URIEncodeData('http://www.test.tld/file with
space.aspx?test=one two')
-- http%3A%2F%2Fwww.test.tld%2Ffile%20with%20space.aspx%3Ftest%3Done%20two
```

INET_URIGetInfo

INET_URIGetInfo(URI NVARCHAR(4000))

RETURNS: TABLE (AbsolutePath NVARCHAR(4000), AbsoluteUri NVARCHAR(4000), Authority NVARCHAR(4000), DnsSafeHost NVARCHAR(4000), Fragment NVARCHAR(4000), HashCode INT, Host NVARCHAR(4000), HostNameType NVARCHAR(50), IsAbsoluteUri BIT, IsDefaultPort BIT, IsFile BIT, IsLoopback BIT, IsUnc BIT, IsWellFormedOriginalString BIT, LocalPath NVARCHAR(4000), PathAndQuery NVARCHAR(4000), Port INT, Query NVARCHAR(4000), Scheme NVARCHAR(50), UserEscaped BIT, UserInfo NVARCHAR(4000))

Returns various details and sub-parts of a URI.

NOTES:
- A NULL URI value will NOT return a row

EXAMPLES:
```
SELECT AbsolutePath, Query FROM
SQL#.INET_URIGetInfo('http://www.SQLsharp.com/path/page.php?test=1')
-- /path/page.php ?test=1

SELECT tab.TheURI, info.*  FROM
```

```
(
      SELECT 'http://www.place.com/dir/page.aspx?var1=val&var2=somethingElse' AS
[TheURI]
      UNION ALL
      SELECT NULL
      UNION ALL
      SELECT 'https://196.168.12.6:23/'
      UNION ALL
      SELECT 'https://user:pass@localhost/page.htm&var=f%26g'
) tab
CROSS APPLY SQL#.INET_URIGetInfo(tab.TheURI) info
```

INET_URIGetLeftPart

INET_URIGetLeftPart(URI NVARCHAR(4000), PartialType NVARCHAR(50))

RETURNS: NVARCHAR(4000)

Returns the left-most substring ending with the specified PartialType

NOTES:

- PartialType:
  - Values are NOT case-sensitive
  - Valid values are:
    - Authority
    - Path
    - Query
    - Scheme

EXAMPLES:

```
SELECT
SQL#.INET_URIGetLeftPart('http://www.someplace.www/path/page.php?var1=sql&var2=s
harp', 'Scheme')
-- http://
SELECT
SQL#.INET_URIGetLeftPart('http://www.someplace.www/path/page.php?var1=sql&var2=s
harp', 'Authority')
-- http://www.someplace.www
SELECT
SQL#.INET_URIGetLeftPart('http://www.someplace.www/path/page.php?var1=sql&var2=s
harp', 'Path')
-- http://www.someplace.www/path/page.php
SELECT
SQL#.INET_URIGetLeftPart('http://www.someplace.www/path/page.php?var1=sql&var2=s
harp', 'Query')
-- http://www.someplace.www/path/page.php?var1=sql&var2=sharp
```

Miscellaneous

Util_CRC32

Util_CRC32(BaseData VARBINARY(MAX))

RETURNS: BIGINT

Performs the standard CRC32 algorithm over the @BaseData.

EXAMPLES:
```
SELECT SQL#.Util_CRC32(CONVERT(VARBINARY(MAX), some_text_field));

SELECT      photo.LargePhotoFileName,
            SQL#.Util_CRC32(photo.LargePhoto) AS 'CRC32'
FROM        AdventureWorks.Production.ProductPhoto photo
/*
LargePhotoFileName             CRC32
--------------------           -------
no_image_available_large.gif   1776513168
racer02_black_f_large.gif      3582859478
racer02_black_large.gif        1272921550
racer02_blue_f_large.gif       414335965
*/
```

Util_Deflate

Util_Deflate(DataToCompress VARBINARY(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Be sure to test the data first as some file types, such as Zip archives and some image formats, are already compressed and actually become larger when put through this function. You will also want to compare the results with the Util_GZip function as in some cases that can provide better results than Util_Deflate.

EXAMPLES:
```
SELECT SQL#.Util_Deflate(CONVERT(VARBINARY, 'This is a test'));
-- 0xEDBD07601C499625262F6DCA7B7F4AF54AD7E074A10880601324D8904010ECC1...
```

Util_GarbageCollect (Not available in Free version)

Util_GarbageCollect()

RETURNS: BIGINT

NOTES:

- Forces an immediate full garbage collection of just the SQL Server-specific CLR Host.
- Does not affect the main Windows CLR Host.
- Returns the number of bytes of memory that have been released by this particular garbage collection.
- See also: Util_GetTotalMemory

EXAMPLES:
```
SELECT SQL#.Util_GarbageCollect();
-- 27802144
```

Util_GenerateDateTimeRange

Util_GenerateDateTimeRange(StartDateTime DATETIME, EndDateTime DATETIME, Step INT, StepType NVARCHAR(4000))

RETURNS: TABLE (DatetimeNum INT, DatetimeVal DATETIME)

NOTES:

- Step <> 0
- StepType IN (year, month, day, hour, minute, second, millisecond)

EXAMPLES:
```
SELECT * FROM SQL#.Util_GenerateDateTimeRange('1/1/2007', '1/31/2007', 7, 'day')
/*
1      2007-01-01 00:00:00.000
2      2007-01-08 00:00:00.000
3      2007-01-15 00:00:00.000
4      2007-01-22 00:00:00.000
5      2007-01-29 00:00:00.000
*/
```

Util_GenerateDateTimes

Util_GenerateDateTimes(StartDateTime DATETIME, TotalDateTimes INT, Step INT, StepType NVARCHAR(4000))

RETURNS: TABLE (DatetimeNum INT, DatetimeVal DATETIME)

NOTES:

- Step <> 0
- TotalDateTimes >= 0
- StepType IN (year, month, day, hour, minute, second, millisecond)

EXAMPLES:
```
SELECT * FROM SQL#.Util_GenerateDateTimes('1/1/2007', 6, 2, 'week');
/*
1      2007-01-01 00:00:00.000
2      2007-01-15 00:00:00.000
3      2007-01-29 00:00:00.000
4      2007-02-12 00:00:00.000
5      2007-02-26 00:00:00.000
6      2007-03-12 00:00:00.000
*/
```

Util_GenerateFloatRange

Util_GenerateFloatRange(StartNum FLOAT, EndNum FLOAT, Step FLOAT)

RETURNS: TABLE (FloatNum INT, FloatVal FLOAT)

NOTES:

- Step <> 0

EXAMPLES:

```
SELECT * FROM SQL#.Util_GenerateFloatRange(.456, 2.2345, .354);
/*
1      0.456
2      0.81
3      1.164
4      1.518
5      1.872
6      2.226
*/
```

Util_GenerateFloats

Util_GenerateFloats(StartNum FLOAT, TotalNums INT, Step FLOAT)

RETURNS: TABLE (FloatNum INT, FloatVal FLOAT)

NOTES:

- Step <> 0
- TotalNums >= 0

EXAMPLES:

```
SELECT * FROM SQL#.Util_GenerateFloats(4.3, 5, .01231);
/*
1      4.3
2      4.31231
3      4.32462
4      4.33693
5      4.34924
*/
```

Util_GenerateIntRange

Util_GenerateIntRange(StartNum INT, EndNum INT, Step INT)

RETURNS: TABLE (IntNum INT, IntVal INT)

NOTES:

- Step <> 0

EXAMPLES:

```
SELECT * FROM SQL#.Util_GenerateIntRange(-5, 5, 2);
/*
1      -5
2      -3
3      -1
4      1
5      3
*/
```

Util_GenerateInts

Util_GenerateInts(@StartNum INT, @TotalNums INT, @Step INT)

RETURNS: TABLE (IntNum INT, IntVal INT)

NOTES:

- @Step <> 0
- @TotalNums >= 0

EXAMPLES:

```
SELECT * FROM SQL#.Util_GenerateInts(1001, 5, 33)
/*
1      1001
2      1034
3      1067
4      1100
5      1133
*/
```

Util_GetBase2Bits (Not available in Free version)

Util_GetBase2Bits(@Base2Value NVARCHAR(64))

RETURNS: TABLE (BitNum INT, BitVal BIGINT)

Returns both the position and integer value of each bit that is set to "1" (i.e. True / Yes / On / Enabled).

NOTES:

- @Base2Value
  - Passing in NULL, empty string '', or "0" will return an empty result set
  - Any non-"1" characters are treated as being "0" (i.e. they do not cause an error)
  - For strings < 64 characters, assume the value is left-padded with "0"s
- BitNum
  - The bit "number", a value between 1 and 64
  - Bit "number" 1 is on the far right (i.e. "...001"), while "number" 64 is on the far left (i.e. "100...")
- BitVal
  - The bit "value", as an integer, for the corresponding bit "number" (i.e. $2^{(BitNum-1)}$)
  - Bit "number" 1 (far right) = 1, while "number" 63 (2nd from far left) = 4611686018427387904
  - Bit "number" 64 (far left) is detected properly, but due to both the max value allowed in BIGINT *and* the left-most position indicating a negative value in Two's Compliment, its value is: -9223372036854775808

EXAMPLES:

```
SELECT * FROM SQL#.Util_GetBase2Bits(N'0011');
/*
1      1
2      2
*/

SELECT * FROM
SQL#.Util_GetBase2Bits(N'0000000000000000000000000000000000000000000000000000
000000100');
/*
3      4
*/
```

```
SELECT * FROM
SQL#.Util_GetBase2Bits(N'00000000000000000011101110010001101000000000');
/*
9      256
11     1024
12     2048
16     32768
19     262144
20     524288
21     1048576
23     4194304
24     8388608
25     16777216
*/

SELECT * FROM SQL#.Util_GetBase2Bits(N'a1b');
-- 2   2
```

Util_GetCreditCardInfo (Not available in Free version)

Util_GetCreditCardInfo(@CCNumber NVARCHAR(4000))

RETURNS: TABLE (CardType NVARCHAR(50), IsValidNumber BIT)

NOTES:

- {will add detail later}

Util_GetCreditCardType (Not available in Free version)

Util_GetCreditCardType(@CCNumber NVARCHAR(4000), @Validate BIT)

RETURNS: NVARCHAR(50)

NOTES:

- {will add detail later}

Util_GetTotalMemory

Util_GetTotalMemory()

RETURNS: BIGINT

NOTES:

- Returns the total number of bytes being used by the SQL Server-specific CLR Host.
- Does not include the main Windows CLR Host.
- See also: Util_GarbageCollect

EXAMPLES:

```
SELECT SQL#.Util_GetTotalMemory();
-- 29960704
```

Util_GUnzip

Util_GUnzip(DataToDecompress VARBINARY(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Use for data compressed with Util_GZip
- NULL input returns NULL

EXAMPLES:

```
SELECT CONVERT(VARCHAR, SQL#.Util_GUnzip(SQL#.Util_GZip(CONVERT(VARBINARY, 'This
is a test'))))
-- This is a test
```

Util_GZip

Util_GZip(DataToCompress VARBINARY(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Be sure to test the data first as some file types, such as Zip archives and some image formats, are already compressed and actually become larger when put through this function. You will also want to compare the results with the Util_Deflate function as in some cases that can provide better results than Util_GZip.
- NULL or 0x input returns NULL

EXAMPLES:

```
SELECT SQL#.Util_GZip(CONVERT(VARBINARY, 'This is a test'))
-- 0x1F8B0800000000000400EDBD07601C499625262F6DC...
```

Util_Hash

Util_Hash(Algorithm NVARCHAR(50), BaseData VARBINARY(MAX))

RETURNS: NVARCHAR(4000)

NOTES:

- Algorithm = MD5, SHA1, SHA256, SHA384, or SHA512
- Algorithm is NOT case-sensitive
- See also the native T-SQL HashBytes function. It has the following algorithms: MD2, MD4, MD5, SHA, and SHA1. However, it returns a VARBINARY instead of an NVARCHAR.

EXAMPLES:

```
SELECT SQL#.Util_Hash('SHA512', CONVERT(VARBINARY(MAX), some_text_field))

SELECT      photo.LargePhotoFileName,
            SQL#.Util_Hash('sha256', photo.LargePhoto) AS 'SHA256'
FROM        AdventureWorks.Production.ProductPhoto photo
/*
LargePhotoFileName                  SHA256
--------------------                -------
no_image_available_large.gif
      F304005422457A5967287AD82C2E64909093EF5F2FC7E1E41A2D465213E0B969
```

```
racer02_black_f_large.gif
      12FB5792202A5685CDF53A35EC58B12DF1EE1E9366F1C8D78BC322DC7E22111F
racer02_black_large.gif
      B6A9F3B96023BA479AA91D3987D3E38A74055245BD7B83E0387E2982E690730D
racer02_blue_f_large.gif
      82544FDB4615A9DF969B959341A27E5161B1BC2AAA3C6743C0A836F6A8CBC4E9
*/
```

Util_HashBinary

Util_HashBinary(Algorithm NVARCHAR(50), BaseData VARBINARY(MAX))

RETURNS: VARBINARY(8000)

NOTES:

- Algorithm = MD5, SHA1, SHA256, SHA384, or SHA512
- Algorithm is NOT case-sensitive
- See also the native T-SQL HashBytes function. It has the following algorithms: MD2, MD4, MD5, SHA, and SHA1. It also returns a VARBINARY.

EXAMPLES:

```
SELECT SQL#.Util_HashBinary('SHA512', CONVERT(VARBINARY(MAX), 'test!'))
--
0x49CD017D5AFF930CC9636D2BFBA95C9C319C7164A330ECCE35EC23271643C4BD1623BE510F25D5
EFCC4B031C5D68C25F908636A106A41D29F5657A0759CF0687
```

Util_Inflate

Util_Inflate(DataToDecompress VARBINARY(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Use for data compressed with Util_Deflate

EXAMPLES:

```
SELECT CONVERT(VARCHAR, SQL#.Util_Inflate(SQL#.Util_Deflate(CONVERT(VARBINARY,
'This is a test'))))
-- This is a test
```

Util_IsValidCC

Util_IsValidCC(CCNumber NVARCHAR(4000), CCType NVARCHAR(4000))

RETURNS: BIT

Determines if a Credit Card number is valid ONLY FOR the following types of Credit Cards: AmericanExpress, Diners Club, Discover, MasterCard, and Visa.

NOTES:

- This does NOT determine in any way if the Credit Card number is active or anything else about the number outside of it being a possible number that one of those companies would use
- CCNumber can have dashes or just the numbers; it is the same either way
- CCType can be one of the following values: empty string / '', amex, diners, disc, mc, visa

- CCType is not case-sensitive; ‘MC’ and ‘mc’ work just the same
- If CCType is an empty string / ‘’ then it will evaluate the validity of the number based on the first few digits and the length of the number against each of those companies’ rules as each company is different in those respects
- The Visa test number is: 4111111111111111
- If the CCType is not an empty string then the number will be evaluated only for that particular company, and hence is more accurate (meaning, you can know somebody is lying, or just made a mistake, if they enter in a number starting with 4 that is a valid Visa number but yet they selected ‘amex’ as the CCType. If CCType is ‘’ and they enter in a valid Visa number it will pass as valid but that same valid Visa number will fail as invalid if the CCType is ‘amex’ or ‘disc’

EXAMPLES:
```
SELECT SQL#.Util_IsValidCC('4111111111111111', '')
-- 1
SELECT SQL#.Util_IsValidCC('4111111111111111', 'visa')
-- 1
SELECT SQL#.Util_IsValidCC('4111111111111111', 'amex')
-- 0
SELECT SQL#.Util_IsValidCC('6111111111111111', 'visa')
-- 0
SELECT SQL#.Util_IsValidCC('6111111111111111', '')
-- 0
```

Util_IsValidCCNumber

Util_IsValidCCNumber(CCNumber NVARCHAR(4000))

RETURNS: BIT

NOTES:
- {will add details later}

Util_IsValidCheckRoutingNumber

Util_IsValidCheckRoutingNumber(RoutingNumber NVARCHAR(4000))

RETURNS: BIT

Determines if a Check Routing Number (also known as the ABA Routing Number) is valid.

NOTES:
- This does NOT determine in any way if the Routing Number is active or has ever been used; it can only check that it is a number that “could” be used
- RoutingNumber can have dashes, spaces, or just the numbers

EXAMPLES:
```
SELECT SQL#.Util_IsValidCheckRoutingNumber('271972572')
-- 1
SELECT SQL#.Util_IsValidCheckRoutingNumber('371972572')
-- 0
```

Util_IsValidConvert

Util_IsValidConvert(ValueToConvert NVARCHAR(MAX), ConvertToDataTypes NVARCHAR(4000), DecimalPrecision SMALLINT, DecimalScale SMALLINT)

RETURNS: BIT

Determines if a string can be converted to any of the specified datatypes.

NOTES:

- NULL for ValueToConvert returns True / 1
- DecimalPrecision and DecimalScale are only needed if Decimal / Numeric are specified and can be set to NULL otherwise
- ConvertToDataTypes:
    - Allows for one or more datatypes to be tested for possible conversion
    - Accepts both datatype names as well as numeric equivalents separated by pipes |
    - Is NOT case-sensitive
    - Datatype values:
        - BIT = 1
        - TINYINT = 2
        - SMALLINT = 4
        - INT = 8
        - BIGINT = 16
        - DECIMAL / NUMERIC = 32
        - MONEY = 64
        - SMALLMONEY = 128
        - REAL = 256
        - FLOAT = 512
        - SMALLDATETIME = 1024
        - DATETIME = 2048
        - DATETIME2 = 4096
        - DATE = 8192
        - TIME = 16384
        - DATETIMEOFFSET = 32768
        - XML = 65536
        - UNIQUEIDENTIFIER = 131072
        - TIMESTAMP / ROWVERSION = 262144

EXAMPLES:
```
SELECT SQL#.Util_IsValidConvert('12331', 'int|SmallInt|xml', NULL, NULL) -- 1
SELECT SQL#.Util_IsValidConvert('12331', 1, NULL, NULL) -- 1
SELECT SQL#.Util_IsValidConvert('12331', 5, NULL, NULL) -- 1
SELECT SQL#.Util_IsValidConvert('12331', 4 | 8 | 16 | 32, NULL, NULL) -- 1
SELECT SQL#.Util_IsValidConvert('12312.3123123123124', 4 | 8 | 16 | 32, 5, 1) --
0
SELECT SQL#.Util_IsValidConvert('5555-12-01 12:12:12.121', 1024, 5, 1) -- 0
SELECT SQL#.Util_IsValidConvert('1922-12-01 12:12:12.121', 2048 | 32768, 5, 1) -
- 1
SELECT SQL#.Util_IsValidConvert('12312', 524288, 5, 1) -- 0
```

Util_IsValidPostalCode

Util_IsValidPostalCode(CountryCode NVARCHAR(4000), PostalCode NVARCHAR(4000))

RETURNS: BIT

Determines if a Postal Code is valid for the given Country Code.

NOTES:

- This does NOT determine in any way if the PostalCode is active or has ever been used; it can only check that it is a code that “could” be used
- PostalCode is NOT case-sensitive
- CountryCode is NOT case-sensitive
- CountryCode is an ISO two-character Country Code (see LookUp_GetCountryInfo for more information on ISO Country Codes)
- If an unsupported CountryCode is given, a NULL is returned.
- Currently only US and CA are supported Country Codes.
    - US: 5 or 9 (Zip+4) digit Zipcodes; 9 digit can have dash or not
    - CA: 6 character code, or 7 with a space in the middle

EXAMPLES:

```
SELECT SQL#.Util_IsValidPostalCode('us','55555-6794')
-- 1
SELECT SQL#.Util_IsValidPostalCode('US','555556794')
-- 1
SELECT SQL#.Util_IsValidPostalCode('us','55555+6794')
-- 0
SELECT SQL#.Util_IsValidPostalCode('ca','a1a 1a1')
-- 1
SELECT SQL#.Util_IsValidPostalCode('CA','a1a1a1')
-- 1
```

Util_IsValidSSN

Util_IsValidSSN(SSN NVARCHAR(4000))

RETURNS: BIT

Determines if a Social Security Number is valid.

NOTES:

- This does NOT determine in any way if the SSN is active or has ever been used; it can only check that it is a number that “could” be used
- SSN can have dashes or just the numbers; it is the same either way

EXAMPLES:

```
SELECT SQL#.Util_IsValidSSN('123-45-6789')
-- 1
SELECT SQL#.Util_IsValidSSN('123456789')
-- 1
SELECT SQL#.Util_IsValidSSN('1234567890')
-- 0
SELECT SQL#.Util_IsValidSSN('123bob789')
-- 0
SELECT SQL#.Util_IsValidSSN('888-23-4567')
-- 0
```

Util_Paginate (Not available in Free version)

Util_Paginate

```
        @Query NVARCHAR(MAX),
        @RowsToSkip INT,
        @RowsToReturn INT,
        @TotalRows INT = -1 OUTPUT,
        @ResultSetRowNumberHandling TINYINT = 0,
        @ReturnVarcharAsNVarchar BIT = 0,
        @CustomFieldNames NVARCHAR(4000) = NULL
```

PROC: The need to page through a large result set, X rows at a time, is nearly universal. And, often enough, this need also includes the requirement to get the total number of rows of the entire, non-paged result set (in order to determine how many pages exist). One option is to use the OFFSET and FETCH options of the ORDER BY clause. That is a good option for getting one page of the result set, but it doesn't get the total number of rows, and it doesn't apply to anyone using a version prior to 2012. Another option is to dump the entire result set into a temporary table, get the total row count via @@ROWCOUNT, and then just select the subset of rows from the temp table. But, while you don't need to run the query twice, that is a lot of IO writing to the temp table plus the transaction log activity. Also, if you don't have control over the query, but instead just have a Stored Procedure to execute, then you can't use OFFSET and FETCH, and instead need to dump the results to a temp table via INSERT...EXEC. But then you need to be careful not to use the INSERT...EXEC construct within that Stored Procedure as you will get the "INSERT EXEC statement cannot be nested" error.

This procedure executes any query (including a Stored Procedure), and allows for skipping a specified number of rows while returning a specified number of rows. It optionally allows for using an OUTPUT parameter to get the total number of rows for the entire non-paged result set (without executing the query again!). Beyond that, it allows for optionally including one or two extra columns at the end (i.e. far right): the row count within the current page of results and / or the row count within the entire non-paged result set. And if that wasn't enough, there is also an optional parameter to override the field names of the result set.

NOTES:

- @Query
  - Source query; ad hoc or stored procedure
- @RowsToSkip
  - How many rows to read and discard before returning rows of the result set
  - Skip 0 rows for the first page of results
  - Skip 9 rows to start at page 4, assuming 3 rows per page
- @RowsToReturn
  - How many rows to pass back from the actual result set (i.e. page size or rows per page)
- @TotalRows
  - Optional parameter
  - Used to get the total number of rows for the entire result set from the source query. This value can be used to calculate how many pages of results there are at the current page size (meaning: NumPages = @TotalRows / @RowsToReturn).
  - Set to -1 to disable cycling through remaining rows of the internal cursor in order to get the total. This could be slightly faster for some queries.
  - Default value = -1
- @ResultSetRowNumberHandling
  - Optional parameter
  - Adds 1 or 2 "row number" fields to the result set (on the right side)
  - **0** or **NULL** = no extra fields added
  - **1** = add "ImmediateResultsRowNumber" (always starts at 1)
  - **2** = add "OverallResultsRowNumber" (starts at @RowsToSkip + 1)
  - **3** = add both fields to the results
  - Default value = 0
- @ReturnVarcharAsNVarchar
  - Optional parameter

    - When source column is a CHAR / VARCHAR / TEXT type, there is potential for code page conversions, especially if any characters have a value of 128 – 255. If characters in this range exist in the results and are being converted into other characters, then returning the column as NVARCHAR (i.e. Unicode) will preserve the original characters.
    - Default value = 0 / false
- @CustomFieldNames
    - Optional parameter
    - Override field names coming from query or stored procedure
    - Comma-separated list of new names
    - Can be sparsely populated to skip fields that should not be overridden
    - Default value = NULL
- This Stored Procedure does *not* load the entire result set into memory (or anywhere), so it only ever has the current row in memory and thus should not take up any more system resources for a 10 million row result set than it does for a 10 row result set.
- This Stored Procedure will not get the “INSERT EXEC statement cannot be nested” error, so you can execute Stored Procedures that use that construct.

EXAMPLES:
```
-- No extra options; just return a 3-row page of results, starting with row 21.
EXEC SQL#.Util_Paginate
      N'SELECT so.[name], so.[type_desc], so.[schema_id] FROM master.sys.objects so;',
      20, 3;
/*
name             type_desc     schema_id
syscsrowgroups   SYSTEM_TABLE  4
sysextsources    SYSTEM_TABLE  4
sysexttables     SYSTEM_TABLE  4
*/


-- Return a 3-row page of results, starting with row 21, and
--   return the total number of result rows.
DECLARE @TotalRows1 INT;
EXEC SQL#.Util_Paginate
      N'SELECT so.[name], so.[type_desc], so.[schema_id] FROM master.sys.objects so;',
      20, 3, @TotalRows1 OUTPUT;
SELECT @TotalRows1 AS [TotalRows]; -- 110
/*
name             type_desc     schema_id
syscsrowgroups   SYSTEM_TABLE  4
sysextsources    SYSTEM_TABLE  4
sysexttables     SYSTEM_TABLE  4
*/


-- Return a 3-row page of results, starting with row 21, and
--   return the total number of result rows.
-- Add "ImmediateResultsRowNumber" column to returned results.
DECLARE @TotalRows2 INT;
EXEC SQL#.Util_Paginate
      N'SELECT so.[name], so.[type_desc], so.[schema_id] FROM master.sys.objects so;',
      20, 3, @TotalRows2 OUTPUT, 1;
SELECT @TotalRows2 AS [TotalRows]; -- 110
/*
name             type_desc     schema_id  ImmediateResultsRowNumber
syscsrowgroups   SYSTEM_TABLE  4          1
sysextsources    SYSTEM_TABLE  4          2
sysexttables     SYSTEM_TABLE  4          3
*/
```

```
-- Return a 3-row page of results, starting with row 21, and
--   return the total number of result rows.
-- Add "ImmediateResultsRowNumber" and "OverallResultsRowNumber" columns to results.
DECLARE @TotalRows3 INT;
EXEC SQL#.Util_Paginate
       N'SELECT so.[name], so.[type_desc], so.[schema_id] FROM master.sys.objects so;',
       20, 3, @TotalRows3 OUTPUT, 3;
SELECT @TotalRows3 AS [TotalRows]; -- 110
/*
name             type_desc     schema  ImmediateResultsRowNumber  OverallResultsRowNumber
syscsrowgroups   SYSTEM_TABLE  4       1                          21
sysextsources    SYSTEM_TABLE  4       2                          22
sysexttables     SYSTEM_TABLE  4       3                          23
*/


-- Return a 3-row page of results, starting with row 21, and
--   set @TotalRows to -1 to skip additional counting.
-- Add "ImmediateResultsRowNumber" and "OverallResultsRowNumber" columns to results.
EXEC SQL#.Util_Paginate
       N'SELECT so.[name], so.[type_desc], so.[schema_id] FROM master.sys.objects so;',
       20, 3, -1, 3, 1, N'a,,c';
/*
a                type_desc     c  ImmediateResultsRowNumber  OverallResultsRowNumber
syscsrowgroups   SYSTEM_TABLE  4  1                          21
sysextsources    SYSTEM_TABLE  4  2                          22
sysexttables     SYSTEM_TABLE  4  3                          23
*/
```

Util_Print (Not available in Free version)

Util_Print
    @Source NVARCHAR(MAX),
    @WordWrap BIT = 1,
    @MaxLineLength SMALLINT = 4000,
    @WordWrapCharactersOverride NVARCHAR(50) = NULL

PROC: Similar to the T-SQL PRINT command, but accepts an NVARCHAR(MAX) value, will chop the string into 4000 character segments complete with word-wrapping (or fewer characters if specified), and won't split Unicode supplementary characters into two � characters. PRINT is limited to 8000 bytes total per line (which is usually 4000 double-byte characters for NVARCHAR data), can't squeeze the text into less than 4000 characters per line, doesn't handle word-wrapping, and does not properly handle Unicode supplementary characters (it will split them into two � characters).

NOTES:

- @Source:
  - The string to send to the client program (i.e. the Messages tab in SSMS)
- @WordWrap:
  - Whether or not to break the current line at the most recent word-wrap character (see @WordWrapCharactersOverride) if the line reaches the maximum length (see @MaxLineLength)
  - Default value = 1 / true
- @MaxLineLength:
  - The longest a line of text, not broken by a newline character, can be before continuing on the following line
  - Value of NULL, < 1, or > 4000 equate to 4000

  - While the T-SQL PRINT command can do up to 8000 VARCHAR / single-byte characters, the SQLCLR API only allows for NVARCHAR / double-byte strings.
  - Default value = 4000
- @WordWrapCharactersOverride:
  - If not overridden, word-wrap characters are just: [space], - (hyphen), and [tab].
  - If overridden, there are no default characters. Meaning, if you want a fourth character (e.g. a comma as well as [space], -, and [tab]), then you need to set this parameter to all four. This allows you to remove [space] and/or - and/or [tab] by not including them in this parameter.
  - Setting this parameter has no effect if @WordWrap is set to 0 / False.
  - Default value = NULL (this keeps the default characters of [space], -, and [tab])

EXAMPLES:
```
DECLARE @String NVARCHAR(MAX);
SET @String = N'1234567' + NCHAR(13) + NCHAR(10) + N'890 1234567890';
EXEC SQL#.Util_Print @String, 0, 5;
PRINT '~~~~~~~~~~~~~~~~~~~';
EXEC SQL#.Util_Print @String, 1, 5;
PRINT '~~~~~~~~~~~~~~~~~~~';
EXEC SQL#.Util_Print
      @Source = @String,
      @WordWrap = 1,
      @MaxLineLength = 5,
      @WordWrapCharactersOverride = N'4';
```

```
12345
67
890 1
23456
7890
~~~~~~~~~~~~~~~~~~~
12345
67
890
12345
67890
~~~~~~~~~~~~~~~~~~~
1234
567
890 1
234
56789
0
```

Util_ToWords

Util_ToWords(NumericValue FLOAT)

RETURNS: NVARCHAR(4000)

Converts a numerical value into it English word equivalent.

NOTES:
- This can be used for creating checks, much like the function in Crystal Reports.

EXAMPLES:
```
SELECT SQL#.Util_ToWords(91231.67)
--Returns: Ninety One Thousand, Two Hundred Thirty One and 67
```

```
SELECT SQL#.Util_ToWords(47006.6)
--Returns: Forty Seven Thousand, Six and 60
```

Util_UnBitMask (Not available in Free version)

Util_UnBitMask (@BitMask BIGINT)

RETURNS: TABLE (BitNum INT, BitVal BIGINT)

Returns both the position and integer value of each bit that is included in the masked value.

NOTES:

- @BitMask
  - Passing in NULL or 0 will return an empty result set
  - This is not intended to work with negative numbers, but passing in a negative number does not error. However, it might not behave as expected (unless you expect to get back the bits of the Two's Compliment representation of the negative value).
- BitNum
  - The bit "number", a value between 1 and 64
  - Bit "number" 1 is on the far right (i.e. "...001"), while "number" 64 is on the far left (i.e. "100...")
- BitVal
  - The bit "value", as an integer, for the corresponding bit "number" (i.e. $2^{(BitNum-1)}$)
  - Bit "number" 1 (far right) = 1, while "number" 63 (2nd from far left) = 4611686018427387904
  - Bit "number" 64 (far left) is detected properly, but due to both the max value allowed in BIGINT *and* the left-most position indicating a negative value in Two's Compliment, its value is: -9223372036854775808

EXAMPLES:

```
SELECT * FROM SQL#.Util_UnBitMask(3);
/*
1      1
2      2
*/

SELECT * FROM SQL#.Util_UnBitMask(4);
/*
3      4
*/

SELECT * FROM SQL#.Util_UnBitMask(26774784);
/*
9      256
11     1024
12     2048
16     32768
20     524288
21     1048576
24     8388608
25     16777216
*/
SELECT SUM([BitVal]) FROM SQL#.Util_UnBitMask(26774784);
-- 26774784

SELECT * FROM SQL#.Util_UnBitMask(18031994990493696);
/*
```

```
33    4294967296
45    17592186044416
55    18014398509481984
*/
SELECT SUM([BitVal]) FROM SQL#.Util_UnBitMask(18031994990493696);
-- 18031994990493696
```

Date

Date_Age

Date_Age(StartDate DATETIME, EndDate DATETIME, LeapYearHandling NVARCHAR(4000), IncludeDays BIT)

RETURNS: FLOAT

Determine the age assuming that StartDate is a Birth-date or Anniversary-date or some equivalent. The assumption is used to determine how many days from the most recent occurrence of the day component and the EndDate. Meaning, if a Birthday is 2000-05-21 and the EndDate is 2005-02-17, then the whole-value part of the Age is 4 (since 05-21 has not been reached yet) but there are 272 days between 2004-05-21 (the most recent occurrence of 05-21) and 2005-02-17.

NOTES:

- LeapYearHandling
    - Controls how calculations are made when the Month and Day for StartDate is February 29th (Leap Day).
    - Values are NOT case-sensitive
    - Valid values are:
        - 28 / F / Feb = when not in a leap-year, anniversary day is observed on February 28th.
        - 1 / M / March = when not in a leap-year, anniversary day is observed on March 1st.
- IncludeDays:
    - Whether or not to return the number of days from the most recent occurance of the Month and Day portion of StartDate prior to EndDate.
    - If set to True / 1, the number of days as the decimal component (i.e. days / 1000). The values will be between 0 and 365. A value of 365 will occur when the StartDate is on February 29th and EndDate is set to February 28th in a Leap Year while using “28” or “F” or “Feb” as the LeapYearHandling because the most recent occurrence of the anniversary day will have occurred in a non-leap year which is then observed on February 28th but in the current year (as specified in EndDate) there is a February 29th, which is 365 days later than the previous February 28th.
    - If set to False / 0, the number of days will not be calculated (to save time) and will always return .000 so that you do not have to use FLOOR() if you only want the INT value.

EXAMPLES:

```
SELECT SQL#.Date_Age('1996-02-29', '2007-11-23', '28', 1)
-- 11.268
SELECT SQL#.Date_Age('1996-02-29', '2007-11-23', '1', 1)
-- 11.267
SELECT SQL#.Date_Age('1996-02-29', '2007-11-23', '1', 0)
-- 11
```

Date_BusinessDays

Date_BusinessDays(StartDate DATETIME, EndDate DATETIME, ExcludeDaysMask BIGINT)

RETURNS: INT

This function works much like the T-SQL DATEDIFF function except that it excludes a variety of non-Business Days based on the value passed in for ExcludeDaysMask. It is possible to exclude any combination of weekend days and various holidays from the given date-range.

NOTES:

- The time component of StartDate and EndDate are ignored; all are seen as having a time value of: 00:00:00.000
- ExcludeDaysMask is a “bit mask” field that allows for any combination of several options to be sent as a single INT value. Just add up the individual values for the days you want to exclude and use that total value. The chart of combinations is shown below. In the “Selected” column only two options (Saturday and Sunday) are selected. These two options correspond to the values of 1 and 2 respectively. In combination these two are added to come up with the total of 3 that is shown at the bottom. If “Labor Day” was also selected, the value of 1024 would be added to the total to come up with a new total of 1027 (1 + 2 + 1024). If all values are selected the total value will be: 33554431.

Day to Exclude	**Value**	**Selected**
Saturday	1	X
Sunday	2	X
New Year's Day (January 1st) IF Monday - Friday	4	
New Year's Day [observed] (December 31st) IF Jan 1 is on Saturday	8	
New Year's Day [observed] (January 2nd) IF Jan 1 is on Sunday	16	
Martin Luther King, Jr. Birthday [US] (3rd Monday in January)	32	
Memorial Day [US] (Last Monday in May)	64	
Independance Day [US] (July 4th) IF Monday - Friday	128	
Independance Day [US, observed] (July 3rd) IF July 4th is on Saturday	256	
Independance Day [US, observed] (July 5th) IF July 4th is on Sunday	512	
Labor Day [US] (1st Monday in September)	1024	
Thanksgiving [US] (4th Thurday in November)	2048	
Thanksgiving [US] (Friday after)	4096	
Christmas (December 25th) IF Monday - Friday	8192	
Christmas [observed] (December 24th) IF December 25th is on Saturday	16384	
Christmas [observed] (December 26th) IF December 25th is on Sunday	32768	
Friday	65536	
Good Friday (Gregorian calendar -- Western churches)	131072	
Easter (Gregorian calendar -- Western churches)	262144	
Good Friday (Julian calendar -- Eastern churches)	524288	
Easter (Julian calendar -- Eastern churches)	1048576	
Thanksgiving [CANADA] & Columbus Day [US] (2nd Monday in October)	2097152	
Thanksgiving [CANADA] (Friday before)	4194304	
Presidents' Day [US] (3rd Monday in February)	8388608	
Columbus Day [traditional] (October 12th)	16777216	
Veterans Day (November 11th) IF Monday – Friday	33554432	
Veterans Day [observed] (November 10th) IF November 11th is on Saturday	67108864	
Veterans Day [observed] (November 12th) IF November 11th is on Sunday	134217728	
Christmas Eve (December 24th)	268435456	
New Year's Eve (December 31st)	536870912	
Maximum Value =	1073741823	**3**

- The above chart can be found online in XLS format which will automatically calculate the correct value for you after you put an X in each row under "Selected". You can download this at: https://SQLsharp.com/download/SQLsharp_Date_BusinessDays.xls
- The ExcludeDaysMask field can be noted in two more readable ways:
    - Using simple addition (+):
      (1 + 2 + 4) -- Saturday, Sunday, and New Year’s Day

  - Using Bit-wise OR (|):
    (1 | 2 | 4) -- Saturday, Sunday, and New Year's Day
- This function is independent of specific years or dates so it can find Thanksgiving in any year even though it is not the same date between years
- Unlike the DATEDIFF(DAY, StartDate, EndDate) function, Date_BusinessDays() will count ALL days in the range whereas DATEDIFF will return 1 less than the total number of days in order to give a "difference". For example, when comparing the same date, DATEDIFF will return 0 while Date_BusinessDays() will return 1 (assuming that the day isn't excluded for some reason). This difference is due to Date_BusinessDays() giving a true count of the number of Business Days within the given date-range as opposed to the difference in number of days within the date-range.
- Additional holidays will be added over time and can also be done by request

EXAMPLES:
```
SELECT DATEDIFF(DAY, '11/18/2007', '11/25/2007')
-- Sunday, Nov. 18th - Sunday, Nov. 25th
-- 7

SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 0)
-- Sunday, Nov. 18th - Sunday, Nov. 25th
-- 8

SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 1)
-- remove only Saturdays (1 by itself)
-- 7
SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 2)
-- remove only Sundays (2 by itself)
-- 6
SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 3)
-- remove weekend days (1 + 2)
-- 5

SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 2051)
SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', (1 | 2 | 2048))
SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', (1 + 2 + 2048))
-- remove weekend days and Thanksgiving (1 + 2 + 2048)
-- 4
SELECT SQL#.Date_BusinessDays('11/18/2007', '11/25/2007', 6147)
-- remove weekends and typical Thanksgiving (1 + 2 + 2048 + 4096)
-- 3
```

Date_BusinessDaysAdd (not available in Free version)

Date_BusinessDaysAdd(StartDate DATETIME, NumDays INT, ExcludeDaysMask BIGINT)

RETURNS: DATETIME

This function works much like the T-SQL DATEADD function except that it excludes a variety of non-Business Days based on the value passed in for ExcludeDaysMask. It is possible to exclude any combination of weekend days and various holidays from the given date-range.

NOTES:
- Please see the NOTES for Date_BusinessDays for information on ExcludeDaysMask values
- See also: Date_IsBusinessDay

EXAMPLES:

```
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011 13:25:47.678', 5, 0)
-- exclude nothing: 2011-11-27 13:25:47.677
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011 13:25:47.678', 5, 3)
-- exclude Sat, Sun: 2011-11-29 13:25:47.677
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011', 5, 1 + 2 + 2048)
-- exclude Sat, Sun, Thanksgiving [US]: 2011-11-30 00:00:00.000
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011', 5, 1 + 2 + 2048 + 4096)
-- exclude Sat, Sun, Thanksgiving [US], Friday after Thanksgiving [US]:
-- 2011-12-01 00:00:00.000
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011', -1, 1 + 2 + 2048)
-- exclude Sat, Sun, Thanksgiving [US]: 2011-11-21 00:00:00.000
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011', -5, 1 + 2 + 2048)
-- exclude Sat, Sun, Thanksgiving [US]: 2011-11-15 00:00:00.000
SELECT SQL#.Date_BusinessDaysAdd('11/22/2011', 0, 1 + 2 + 2048)
-- exclude Sat, Sun, Thanksgiving [US]: 2011-11-22 00:00:00.000
```

Date_DaysInMonth

Date_DaysInMonth(Year INT, Month INT)

RETURNS: INT

NOTES:

- Takes Leap Years into account
- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.Date_DaysInMonth(2007, 2)
-- 28
SELECT SQL#.Date_DaysInMonth(2008, 2)
-- 29
SELECT SQL#.Date_DaysInMonth(2008, 3)
-- 31
```

Date_DaysInMonthFromDateTime

Date_DaysInMonthFromDateTime(TheDate DATETIME)

RETURNS: INT

NOTES:

- Takes Leap Years into account
- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.Date_DaysInMonthFromDateTime('2014-02-15')
-- 28
SELECT SQL#.Date_DaysInMonthFromDateTime('2012-02-15')
-- 29
```

Date_DaysLeftInMonth (Not available in Free version)

Date_DaysLeftInMonth(TheDate DATETIME)

RETURNS: INT

NOTES:

- Takes Leap Years into account
- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.Date_DaysLeftInMonth('2012-02-01')
-- 28
SELECT SQL#.Date_DaysLeftInMonth('2013-02-01')
-- 27
```

Date_DaysLeftInYear

Date_DaysLeftInYear(TheDate DATETIME)

RETURNS: INT

NOTES:

- Takes Leap Years into account
- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.Date_DaysLeftInYear('02/20/2007')
-- 314
SELECT SQL#.Date_DaysLeftInYear('02/20/2008')
-- 315
```

Date_Extract

Date_Extract(DatePart NVARCHAR(4000), Date DATETIME)

RETURNS: INT

Much like the Microsoft SQL Server built-in DATEPART function, Extract returns a part of the given Date. This is modeled after the PostgreSQL function, Extract, and includes most of the DatePart values that are handled by the built-in DATEPART SQL Server function.

NOTES:

- DatePart:
    - Values are NOT case-sensitive
    - Valid values are:
        - Millennium – 1923 = 1
        - Century – 1923 = 19
        - Decade – 1923 = 192
        - ISOYear – Each year begins on the Monday of the week containing January 4th.
        - Year – 2010-05-03 = 2010
        - DayOfYear – 2010-05-03 = 123
        - Quarter - 2010-05-03 = 2
        - Month – 2010-05-03 = 5
        - Week – 2010-05-03 = 19
        - ISOWeek / ISO_WEEK – WeekNumber can be between 1 and 53, depending on ISOYear calculation
        - ISOWeekDay / ISODOW – Monday = 1, Sunday = 7
        - Weekday – 2010-05-03 = 2

        - Day – 2010-05-03 = 3
        - Hour – 2010-05-03 15:23:46.097 = 15
        - Minute – 2010-05-03 15:23:46.097 = 23
        - Second – 2010-05-03 15:23:46.097 = 46
        - Millisecond – 2010-05-03 15:23:46.097 = 97
    - ISO refers to ISO 8601
    - ISOWeek calculation taken from Rick McCarty: http://personal.ecu.edu/mccartyr/ISOwdALG.txt
- Will return NULL when Date is NULL

EXAMPLES:
```
SELECT SQL#.Date_Extract('ISOYear', '2005-01-03')
-- 2005
SELECT SQL#.Date_Extract('ISOYear', '2005-01-02')
-- 2004
SELECT SQL#.Date_Extract('ISOWeek', '2005-01-02')
-- 53
```

Date_Format

Date_Format(TheDate DATETIME, DateTimeFormat NVARCHAR(4000), Culture NVARCHAR(10))

RETURNS: NVARCHAR(4000)

Returns a string representing the date in the specified format and optional culture.

NOTES:
- DateTimeFormat
    - Standard formats: http://msdn.microsoft.com/en-us/library/az4se3k1(v=vs.80).aspx
    - Custom formats: http://msdn.microsoft.com/en-US/library/8kb3ddd4(v=vs.80).aspx
- Culture
    - Optional, use empty string (‘’) to default to “current culture”
    - Available culture names: http://msdn.microsoft.com/en-US/library/system.globalization.cultureinfo(v=vs.80).aspx
- Essentially the same as the new FORMAT command in SQL Server 2012: http://msdn.microsoft.com/en-us/library/hh213505(v=sql.110).aspx

EXAMPLES:
```
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'D', '')
-- Thursday, December 03, 2009
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'D', 'de')
-- Donnerstag, 3. Dezember 2009
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'D', 'he')
-- 9002 דצמבר 30 יום חמישי
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'D', 'fr-fr')
-- jeudi 3 décembre 2009
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'dd', '')
-- 03
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'dd-MMM', '')
-- 03-Dec
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'dd-MMM', 'he')
-- 03- דצמ
SELECT SQL#.Date_Format('2009-12-03 12:45:56.345', 'tt', 'ja-jp')
-- 午後
```

Date_FormatTimeSpan

Date_FormatTimeSpan(StartDate DATETIME, EndDate DATETIME, OutputFormat NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Returns a formatted string representing the amount of time between the specified DATETIME values. This is like DATEDIFF except with DATEDIFF you can only see the difference expressed in terms of one particular DatePart. For example, you can see the difference between ‘1/1/2008 00:00:00.000’ and ‘1/9/2008 13:42:57.098’ in terms of minutes (12,342) only or in terms of days (8) only or in terms of hours (205) only. However, it is sometimes useful to express that time difference as being:

1 week, 1 day, 13 hours, 42 minutes, 57.097 seconds

NOTES:

- EndDate must be greater-than-or-equal-to StartDate
- OutputFormat works like “printf” in that it can contain both literals that will be returned as they are as well as variables that will be substituted for their particular values. The variables are structured to allow flexibility in terms of offering different text dependant on the numeric value of the TimeSpanPart of the variable. The three options are how to deal with values that are either, 0, 1, or greater-than 1. Meaning, it might be desired to not show anything for a value of 0 (e.g. or maybe even “no minutes” instead of “0 minutes”). It might also be desired to show the difference between “1 minute” and “2 minutes” as opposed to always having the same text and having to show “1 minutes” or even “1 minute(s)”. The variables take the form of:
  %{TimeSpanPart ;; %[width]d text for 0 ;; %[width]d text for 1 ;; %[width]d text for >1}
- Valid TimeSpanParts are:
  - ms = milliseconds
  - ss = seconds (1000 milliseconds)
  - sm = seconds with milliseconds (ss.mmm)
  - mi = minutes (60 seconds)
  - hh = hours (60 minutes)
  - dd = days (24 hours)
  - wk = weeks (7 days)
  - mm = months (30.44 days)
  - yy = years (365.25 days)
- The TimeSpanParts are NOT case-sensitive.
- The %d IS case-sensitive
- The %[width]d will be replaced by the appropriate number for the TimeSpanPart specified.
- The optional [width] value is an INT that will left-pad the resulting time component with zeroes (0).
- You do NOT have to use the %d in any of the 3 text spots. If you do not want the number show (such as might be the case with the zero spot) then it can be left out.
- There is no escape-sequence for the %d; all instances of %d will be translated to the number (e.g. %%d will be %{number} and \%d will be \{number})
- You can specify any number of TimeSpanPart variables and none are required. Meaning, the time spans will always be in terms of days, hours, and minutes, then just specify those three and not years, months, weeks, seconds, milliseconds, or seconds with milliseconds.
- The reason for having the “sm” TimeSpanPart (Seconds With Milliseconds) is to allow for seconds and milliseconds to be expressed with a decimal such as: ss.mmm. If a variable for seconds is used and then a separate one for milliseconds with a literal period between them, such as:
  %{ss;;%d;;%d;;%d}.{%ms;;%d;;%d;;%d} seconds
  then the output might look like:
  1.57 seconds
  when the real value is .057 for milliseconds. The problem is that as a distinct value it cannot have leading zeros. Instead, seconds and milliseconds can be expressed separately in the following manner:

%{ss;;%d seconds;;%d second;;%d seconds} and {%ms;;%d milliseconds;;%d millisecond;;%d milliseconds}

- If a particular TimeSpanPart variable is not used but would still contain a value (i.e. not having a variable for Weeks but measuring a TimeSpan that is over 7 days) then the next lowest TimeSpanPart will NOT contain that missing information. Meaning, if measuring a TimeSpan of 9 days, the "dd" TimeSpanPart variable will contain the number 2 regardless if the TimeSpanPart of "wk" is used in OutputFormat or not. Hence, the days TimeSpanPart variable can never show anything greater than 6 since 7 would translate to 1 week.
- The largest values that the TimeSpanPart variables can show are:
  ms (milliseconds) = 997
  ss (seconds) = 59
  sm (seconds with milliseconds) = 59.997
  mi (minutes) = 59
  hh (hours) = 23
  dd (days) = 30
  wk (weeks) = 4
  mm (months) = 11
  yy (years) = unlimited
- .997 is the largest value for milliseconds that SQL Server can hold before rounding-up to the next second. This has nothing to do with SQL# or the .Net CLR; any T-SQL expression of .998 or .999 milliseconds will automatically round-up to the next second.

EXAMPLES:
```
SELECT SQL#.Date_FormatTimeSpan('1/9/2008 13:00:00.000', '1/9/2008
13:42:57.098',
'%{wk;;;;%d week, ;;%d weeks, }%{dd;;;;%d day, ;;%d days, }%{hh;;;;%d hour, ;;%d
hours, }%{mi;;;;%d minute, ;;%d minutes, }%{sm;;;;%d second;;%d seconds}')
-- 42 minutes, 57.097 seconds

/* %d in middle of text for 1 day slot; no %d for 0 minutes slot */
SELECT SQL#.Date_FormatTimeSpan('1/1/2008 13:00:00.000', '1/3/2008
07:00:42.067',
'%{dd;;;;only %d day, ;;%d days, }%{hh;;;;%d hour, ;;%d hours, }%{mi;;no min.,
;;%d minute, ;;%d minutes, }and %{ss;;;;%d second;;%d seconds}')
-- only 1 day, 18 hours, no min., and 42 seconds

/* "hours" is missing; 0 minutes slot set to empty */
SELECT SQL#.Date_FormatTimeSpan('1/1/2008 13:00:00.000', '1/3/2008
07:00:42.067',
'%{dd;;;;only %d day, ;;%d days, }%{mi;;;;%d minute, ;;%d minutes, }and
%{ss;;;;%d second;;%d seconds}')
-- only 1 day, and 42 seconds
```

Date_FirstDayOfMonth

Date_FirstDayOfMonth(TheDate DATETIME, NewHour INT, NewMinute INT, NewSecond INT, NewMillisecond INT)

RETURNS: DATETIME

Returns a full DATETIME value for the 1st of whatever month is passed in.

NOTES:

- Use NewHour, NewMinute, NewSecond, and NewMillisecond to control the time component of whatever date is returned

- Maximum values are: NewHour = 23, NewMinute = 59, NewSecond = 59, and NewMillisecond = 998
- Any NewMillisecond value between 995 and 998 will show in the returned value as 997; this is just how SQL Server works
- If you use set NewMillisecond to 999 then it will increase the “seconds” by 1

EXAMPLES:

```
DECLARE @DateVal DATETIME
SET @DateVal = '02/14/2008 17:45:32.867'
SELECT SQL#.Date_FirstDayOfMonth(@DateVal, 0, 0, 0, 0) -- first moment
-- 2008-02-01 00:00:00.000
SELECT SQL#.Date_FirstDayOfMonth(@DateVal, 23, 59, 59, 998) -- last moment
-- 2008-02-01 23:59:59.997
SELECT SQL#.Date_FirstDayOfMonth(@DateVal,
                                    DATEPART(HOUR, @DateVal),
                                    DATEPART(MINUTE, @DateVal),
                                    DATEPART(SECOND, @DateVal),
                                    DATEPART(MILLISECOND, @DateVal)
                              ) -- keep the previous time component
-- 2008-02-01 17:45:32.867
```

Date_FromUNIXTime

Date_FromUNIXTime(UNIXDate FLOAT)

RETURNS: DATETIME

NOTES:

- UNIX time is the number of seconds since 12:00 AM, January 1st, 1970
- NULL input returns NULL
- See also Date_ToUNIXTime

EXAMPLES:

```
SELECT SQL#.Date_FromUNIXTime(1195386660)
-- 2007-11-18 11:51:00.000
```

Date_FullDateString

Date_FullDateString(TheDate DATETIME)

RETURNS: NVARCHAR(4000)

NOTES:

- Displays the given date in the format of:
  {Full Day Name}, {Full Month Name} {Day}, {Year}
- Does not display any time information

EXAMPLES:

```
SELECT SQL#.Date_FullDateString('02/20/2007 17:45:10.872')
-- Tuesday, February 20, 2007
```

Date_FullDateTimeString (not available in Free version)

Date_FullDateTimeString(TheDate DATETIME, Separator NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

NOTES:

- Displays the given date in the format of:
  {Full Day Name}, {Full Month Name} {Day}, {Year}
- Displays the given time in the format of:
  {Hour}:{Minute}:{Second} {AM / PM}
- Output = { Date }{ Separator }{ Time }

EXAMPLES:
```
SELECT SQL#.Date_FullDateTimeString('02/20/2007 17:45:10.872', ' at ')
-- Tuesday, February 20, 2007 at 5:45:10 PM
```

Date_FullTimeString

Date_FullTimeString(TheDate DATETIME)

RETURNS: NVARCHAR(4000)

NOTES:

- Displays the given time in the format of:
  {Hour}:{Minute}:{Second} {AM / PM}
- Does not display any date information

EXAMPLES:
```
SELECT SQL#.Date_FullTimeString('02/20/2007 17:45:10.872')
-- 5:45:10 PM
```

Date_GetDateTimeFromIntVals

Date_GetDateTimeFromIntVals(IntDate INT, IntTime INT)

RETURNS: DATETIME

NOTES:

- IntDate should be formatted as: YYYYMMDD
- IntTime should be formatted as: HHMMSS
- NULL input returns NULL
- See also: Date_GetIntDate
- See also: Date_GetIntTime

EXAMPLES:
```
SELECT SQL#.Date_GetDateTimeFromIntVals(20100224, 5)
-- 2010-02-24 00:00:05.000
SELECT SQL#.Date_GetDateTimeFromIntVals(20101204, 4235)
-- 2010-12-04 00:42:35.000

SELECT      SQL#.Date_GetDateTimeFromIntVals(sjh.run_date, sjh.run_time)
            AS [RealDate], *
FROM        msdb.dbo.sysjobhistory sjh
WHERE       SQL#.Date_GetDateTimeFromIntVals(sjh.run_date, sjh.run_time)
            BETWEEN '02/23/2010' AND '05/07/2010'
```

Date_GetIntDate

Date_GetIntDate (TheDate DATETIME)

RETURNS: INT

NOTES:

- Returned value will be formatted as: YYYYMMDD
- See also: Date_GetDateTimeFromIntVals
- See also: Date_GetIntTime

EXAMPLES:

```
SELECT SQL#.Date_GetIntDate('02/17/2010 10:34:23.3')
-- 20100217
```

Date_GetIntTime

Date_GetIntTime (TheDate DATETIME)

RETURNS: INT

NOTES:

- Returned value will be formatted as: HHMMSS
- See also: Date_GetDateTimeFromIntVals
- See also: Date_GetIntDate

EXAMPLES:

```
SELECT SQL#.Date_GetIntTime('02/17/2010 01:34:23.3')
-- 13423
```

Date_IsBusinessDay

Date_IsBusinessDay(TheDate DATETIME, ExcludeDaysMask BIGINT)

RETURNS: BIT

NOTES:

- See NOTES on Date_BusinessDays for an explanation of ExcludeDaysMask
- This function is independent of specific years or dates so it can find Thanksgiving in any year even though it is not the same date between years
- Additional holidays will be added over time and can also be done by request
- See also: Date_BusinessDaysAdd

EXAMPLES:

```
SELECT SQL#.Date_IsBusinessDay('2008-03-21', 131072) --Good Friday
-- 0
```

Date_IsDaylightSavingTime

Date_IsDaylightSavingTime (LocalDate DATETIME)

RETURNS: BIT

NOTES:

- Date input value is assumed to be in same TimeZone as the server running SQL Server.
- NULL input returns NULL

EXAMPLES:
```
SELECT SQL#.Date_IsDaylightSavingTime('2013-01-01')
-- 0
SELECT SQL#.Date_IsDaylightSavingTime('2013-10-01')
-- 2
```

Date_IsLeapYear

Date_IsLeapYear(Year INT)

RETURNS: BIT

NOTES:
- NULL input returns NULL

EXAMPLES:
```
SELECT SQL#.Date_IsLeapYear(2007)
-- 0
SELECT SQL#.Date_IsLeapYear(2008)
-- 1
SELECT SQL#.Date_IsLeapYear(DATEPART(YEAR, GETDATE())) -- current year = 2007
-- 0
```

Date_LastDayOfMonth

Date_LastDayOfMonth(TheDate DATETIME, NewHour INT, NewMinute INT, NewSecond INT, NewMillisecond INT)

RETURNS: DATETIME

Returns a full DATETIME value for the last day of whatever month is passed in.

NOTES:
- Takes Leap Years into account
- Use NewHour, NewMinute, NewSecond, and NewMillisecond to control the time component of whatever date is returned
- Maximum values are: NewHour = 23, NewMinute = 59, NewSecond = 59, and NewMillisecond = 998
- Any NewMillisecond value between 995 and 998 will show in the returned value as 997; this is just how SQL Server works
- If you use set NewMillisecond to 999 then it will increase the “seconds” by 1
- NULL input returns NULL

EXAMPLES:
```
DECLARE @DateVal DATETIME
SET @DateVal = '02/14/2008 17:45:32.867'
SELECT SQL#.Date_LastDayOfMonth(@DateVal, 0, 0, 0, 0) -- first moment
-- 2008-02-29 00:00:00.000
SELECT SQL#.Date_LastDayOfMonth(@DateVal, 23, 59, 59, 998) -- last moment
-- 2008-02-29 23:59:59.997
SELECT SQL#.Date_LastDayOfMonth(@DateVal,
                                DATEPART(HOUR, @DateVal),
                                    DATEPART(MINUTE, @DateVal),
                                    DATEPART(SECOND, @DateVal),
                                    DATEPART(MILLISECOND, @DateVal)
                              ) -- keep the previous time component
-- 2008-02-29 17:45:32.867
```

Date_NewDateTime

Date_NewDateTime(Year INT, Month INT, Day INT, Hour INT, Minute INT, Second INT, Millisecond)

RETURNS: DATETIME

NOTES:

- Result will be NULL if any input parameter is NULL

EXAMPLES:

```
SELECT SQL#.Date_NewDateTime(2000, 5, 10, 16, 2, 3, 7)
-- 2000-05-10 16:02:03.007
SELECT SQL#.Date_NewDateTime(2000, 5, null, 16, 2, 3, 7)
-- NULL
```

Date_NthOccurrenceOfWeekday

Date_NthOccurrenceOfWeekday(Occurrence SMALLINT, Weekday NVARCHAR(10), StartDate DATETIME)

RETURNS: DATETIME

EXAMPLES:

```
SELECT SQL#.Date_NthOccurrenceOfWeekday(20, 'Saturday', '1/1/2009')
-- 2009-05-16 00:00:00.000
SELECT SQL#.Date_NthOccurrenceOfWeekday(3, 'Thursday', '7/1/2009')
-- 2009-07-16 00:00:00.000
```

Date_ToLocalTime (Not available in Free version)

Date_ToLocalTime(UtcDate DATETIME)

RETURNS: DATETIME

NOTES:

- Input is assumed to be UTC time
- Local time is based on the local timezone of the server that SQL Server is running on
- NULL input returns NULL
- This function ***is*** DST aware and reflects the changes that occurred in the US in 2007 (see examples)
- See also Date_ToUniversalTime

EXAMPLES:

```
SELECT SQL#.Date_ToLocalTime('2013-10-01 17:32:12.123')
-- 2013-10-01 13:32:12.123
SELECT SQL#.Date_ToLocalTime('2013-02-01 17:32:12.123')
-- 2013-02-01 12:32:12.123
SELECT SQL#.Date_ToLocalTime('2007-10-31 20:23:45'),
       SQL#.Date_ToLocalTime('2006-10-31 20:23:45')
-- 2007-10-31 16:23:45.000,   2006-10-31 15:23:45.000
```

Date_ToUniversalTime (Not available in Free version)

Date_ToUniversalTime(LocalDate DATETIME)

RETURNS: DATETIME

NOTES:

- Input is assumed to be local time
- Local time is based on the local timezone of the server that SQL Server is running on
- NULL input returns NULL
- This function ***is*** DST aware and reflects the changes that occurred in the US in 2007 (see examples)
- See also Date_ToLocalTime

EXAMPLES:

```
SELECT SQL#.Date_ToUniversalTime('2013-03-01 7:15:49.328')
-- 2013-03-01 12:15:49.327
SELECT SQL#.Date_ToUniversalTime('2013-10-01 7:15:49.328')
-- 2013-10-01 11:15:49.327
SELECT  SQL#.Date_ToUniversalTime('2007-10-31 16:25:30.123'),
        SQL#.Date_ToUniversalTime('2006-10-31 16:25:30.123')
-- 2007-10-31 20:25:30.123    2006-10-31 21:25:30.123
```

Date_ToUNIXTime

Date_ToUNIXTime(SQLDate DATETIME)

RETURNS: FLOAT

NOTES:

- UNIX time is the number of seconds since 12:00 AM, January 1st, 1970
- NULL input returns NULL
- See also Date_FromUNIXTime

EXAMPLES:

```
SELECT SQL#.Date_ToUNIXTime(CONVERT(DATETIME, '2007/11/18 11:51'))
-- 1195386660
```

Date_Truncate

Date_Truncate(DatePart NVARCHAR(4000), Date DATETIME)

RETURNS: DATETIME

Returns the given Date but each piece (Year, Month, Day, Hour, Minute, and Second) reset to the lowest value within the given DatePart. This is modeled after the PostgreSQL function, Trunc.

NOTES:

- DatePart:
  - Values are NOT case-sensitive
  - Valid values are (base date for examples = 2118-08-30 14:23:21.211):

Millennium	=	2000-01-01 00:00:00.000
Century	=	2100-01-01 00:00:00.000
Decade	=	2110-01-01 00:00:00.000

Year	=	2118-01-01 00:00:00.000
Quarter	=	2118-07-01 00:00:00.000
Month	=	2118-08-01 00:00:00.000
Week	=	2118-08-28 00:00:00.000
Day	=	2118-08-30 00:00:00.000
Hour	=	2118-08-30 14:00:00.000
Minute	=	2118-08-30 14:23:00.000
Second	=	2118-08-30 14:23:21.000

- Will return NULL when Date is NULL

EXAMPLES:

```
SELECT SQL#.Date_Truncate('week', '2118-08-30 14:23:21.211')
-- 2118-08-28 00:00:00.000
SELECT SQL#.Date_Truncate('Minute', '2118-08-30 14:23:21.211')
-- 2118-08-30 14:23:00.000
```

Internal

The SQLsharp functions reside in the main SQL# assembly. This assembly needs a security setting of at least EXTERNAL_ACCESS if you are using any functions that access the network. Please see SQLsharp_SetSecurity for both how to see the current settings and how to change them. However, even if you have a security setting of EXTERNAL_ACCESS or UNRESTRICTED, users still will not have access to these functions without being granted permission either explicitly or via SQLsharp_GrantPermissions.

SQLsharp_Download (Not available in Free version)

SQLsharp_Download @LicenseKey NVARCHAR(50), @DownloadPath NVARCHAR(2048)

This is a Stored Procedure. It downloads a new SQLsharp_SETUP_FullVersion.zip file from SQLsharp.com to the location specified by @DownloadPath.

NOTES:

- Requires security setting of EXTERNAL_ACCESS. You might need to use **SQLsharp_SetSecurity** to change the current security setting.
- An error will occur if the LicenseKey is invalid or no longer eligible for updates.
- The only information transmitted to SQLsharp.com is:
    - The License Key
    - The current SQL# version number

SQLsharp_GrantPermissions

SQLsharp_GrantPermissions @GrantTo NVARCHAR(4000) [, @SQLsharpSchema NVARCHAR(4000) = NULL]

This is a Stored Procedure. It GRANTs Permissions for SQL# Functions and Procedures to specified user(s)

NOTES:

- Will NOT grant permissions to the following: SQLsharp_Setup, SQLsharp_Update, SQLsharp_Uninstall, SQLsharp_SetSecurity, SQLsharp_GrantPermissions
- The following characters are removed: *****, **/**, **-**, and **;**
  as there is no reason to use them and filtering them can help reduce unintended use such as SQL Injection.
- SQLsharpSchema is optional and will default to 'SQL#' if not set or set to NULL
- SQLsharpSchema should be set to the Schema name that SQL# is installed as. By default SQL# is installed into a Schema named SQL#.

SQLsharp_Help

SQLsharp_Help

This is a Stored Procedure. It displays list of definitions (signatures) for all SQL# Functions and Procedures

SQLsharp_InstanceSetup

SQLsharp_InstanceSetup [@MaxAllowedAccessLevel TINYINT = { value_at_install_time; default = 3 }]
[[,] @SQLsharpLogin SYSNAME = { value_at_install_time; default = N'SQL#' }]

This is a Stored Procedure. It reinstalls SQL# components to the "**[master]**" Database.

NOTES:

- This is a pure T-SQL Stored Procedure
- The components installed, all in the "**[master]**" Database are:
    - [SQL#Key] Asymmetric Key
    - [SQL#] Login
    - [dbo].[SQLsharp_InstanceUninstall] Stored Procedure
- This Stored Procedure should be used when:
    - The components were not originally installed but are desired now, due to:
        - Change in security policy
        - A Database in which SQL# has been installed is moved (i.e. restored or attached) to a new Instance that has never had SQL# installed in it
    - The components were originally installed but have been removed

SQLsharp_InstanceUninstall

[master].[dbo].SQLsharp_InstanceUninstall

This is a Stored Procedure. It removes SQL# components from the "**[master]**" Database.

NOTES:

- This is a pure T-SQL Stored Procedure
- This Stored Procedure exists in the "**[master]**" Database!!!
- When removing SQL# from an Instance, execute this Stored Procedure *after* executing SQLsharp_Uninstall in each Database where SQL# has been installed.
- The components removed, all from the "**[master]**" Database are:
    - [SQL#Key] Asymmetric Key
    - [SQL#] Login
    - [dbo].[SQLsharp_InstanceUninstall] Stored Procedure (this Stored Procedure)

SQLsharp_IsUpdateAvailable

SQLsharp_IsUpdateAvailable

RETURNS: BIT

NOTES: Requires security setting of EXTERNAL_ACCESS

SQLsharp_SetSecurity

SQLsharp_SetSecurity @PermissionSet INT [, @AssemblyName NVARCHAR(4000)] [, @SetTrustworthyIfNoUser BIT]

This is a Stored Procedure. It sets the specified Assembly PERMISSION_SET. Also, for a setting of either 2 or 3 it will also set the DB is_trustworthy_on setting to 1 (TRUE) if currently 0 (FALSE).

NOTES:

- @PermissionSet:
    - 0 = Display current setting
    - 1 = SAFE,
    - 2 = EXTERNAL_ACCESS
    - 3 = UNRESTRICTED
- @AssemblyName (optional):

- If not set will default to [SQL#]
- Can be set to SQL#, SQL#.OS, SQL#.Twitterizer, SQL#.SgmlReader, SQL#.FileSystem, SQL#.Network, SQL#.DB, or SQL#.DotNetZip

- @SetTrustworthyIfNoUser (optional):
  - If set to 1 will ensure that the Database setting of TRUSTWORTHY is ON if the assembly is not owned by a login that is based on an asymmetric key AND has the appropriate permission granted to it. A login meeting this requirement is created by the installer (as of Version 3.0.x) but if the install is customed to not use that login then this might come in useful.
  - Default = 0

SQLsharp_Setup

SQLsharp_Setup [@SQLsharpSchema NVARCHAR(128)] [, @SQLsharpAssembly NVARCHAR(128)]

This is a Stored Procedure. It creates Functions and Procedures. It is generally not needed after the initial install of SQL# and is called via the installation script.

NOTES:

- @SQLsharpSchema (optional):
  - If not set it will default to whatever @SQLsharpSchema variable towards the top of the install script was defined as (default = SQL#)
  - It should not be necessary to set this explicitly as the default value should always be correct
- @SQLsharpAssembly (optional):
  - If set will it only create the wrapper objects (Stored Procedures / Functions / Types / Aggregates) for the specified assembly.
  - If not set it will create the wrapper objects for all installed SQL# assemblies.
  - If SQL# was installed with an assembly set to skip (i.e. setting the @InstallSQL#____ variable to 0), then it can be installed later by manually running the CREATE ASSEMBLY command manually and then executing this procedure specifying just that assembly name.

SQLsharp_Uninstall

SQLsharp_Uninstall [@SQLsharpAssembly NVARCHAR(128)]

This is a Stored Procedure. It drops the SQL# Functions, Procedures, User-Defined Types, and User-Defined Aggregates wrapper objects for the specified assembly, or all assemblies if one is not specified, and then the assembly itself, or all assemblies if one is not specified.

NOTES:

- @SQLsharpAssembly (optional):
  - If set it will only uninstall the wrapper objects (Stored Procedures / Functions / Types / Aggregates) for the specified assembly and the assembly itself.
  - If not set it will uninstall:
    - the wrapper objects for all installed SQL# assemblies
    - all installed SQL# assemblies
    - the SQL# schema (if no other objects are left in it after the above items have been removed)
- This proc will check for dependent assemblies and, if found, will return an error with the name(s) of those assemblies.

SQLsharp_UnloadAppDomain

SQLsharp_UnloadAppDomain [@DatabaseName SYSNAME]

This is a Stored Procedure. It unloads all App Domains in the current Database, or in the specified Database.

NOTES:

- @DatabaseName
    - Optional parameter
    - Defaults to Database where SQL# is installed if set to NULL or not passed in.
- This works by flipping the Database property of `TRUSTWORTHY` to the opposite state from what it currently is, and then back again: ON -> OFF -> ON, or OFF -> ON -> OFF.
- Whoever runs this will need permission to `ALTER DATABASE` (or use Module Signing)

EXAMPLES:

```
EXEC SQL#.SQLsharp_UnloadAppDomain N'Utility';

EXEC SQL#.SQLsharp_UnloadAppDomain;
```

SQLsharp_Version

SQLsharp_Version()

RETURNS: NVARCHAR(4000)

Displays the version of SQL# currently installed.

SQLsharp_WebSite

SQLsharp_WebSite()

Displays the location (URL) of the SQL# website. Just in case you lose all information including this document ;-).

RETURNS: NVARCHAR(4000)

File (Not available in Free version)

The **File** functions reside mostly in the SQL#.FileSystem assembly, with the GZip functions residing in the SQL#.DotNetZip assembly.

If you use any of the functions that access the file system, then these assemblies will need a security setting of EXTERNAL_ACCESS (Level 2). You can set this by executing the following query:

```
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.FileSystem' -- all except GZip functions
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.DotNetZip' -- needed for GZip functions
```

If you do not want to have these assemblies in your system at all, you can do either of the following:

- Do not install the SQL#.FileSystem and/or SQL#.DotNetZip assemblies by setting the `@InstallSQL#FileSystem` and/or `@InstallSQL#DotNetZip` variables (towards the top of the script) to 0 before installing
- Uninstall either or both of the assemblies by running:
  ```
  EXEC [SQL#].[SQLsharp_Uninstall] N'SQL#.FileSystem'
  EXEC [SQL#].[SQLsharp_Uninstall] N'SQL#.DotNetZip'
  ```

Please note that when accessing the file system, the Operating System user account that will be used is the one that is currently running (i.e. “Log On as”) the main SQL Server process (it might be Local System Account or an account created specifically for SQL Server).

If you will be enabling Long Path Support, you will need to use security level 3 instead of 2 for SQL#.FileSystem and/or SQL#.DotNetZip, depending on which functions will need long path support. While using level 3 (unrestricted) is not ideal, it is a requirement of .NET, unrelated to SQLCLR.

Note about @FileEncoding parameters
Any function or stored procedure, whether in the File category or another category, that has an “Encoding” parameter (typically @FileEncoding, but could be a slightly different name ending with “Encoding”) will accept either Encoding names or Code Page numbers.

The list of Encoding names that can be used are as follows (those ending in “...NoBOM” do not write out the Byte Order Mark / BOM when creating files; multiple entries on the same line are the same base encoding):

- ASCII
- UTF7
- UTF8 / UTF8NoBOM
- UTF16 / UTF16Le / Unicode
- UTF16NoBOM / UTF16LeNoBOM / UnicodeNoBOM
- UTF16Be / BigEndianUnicode
- UTF16BeNoBOM / BigEndianUnicodeNoBOM
- UTF32 / UTF32Le
- UTF32NoBOM / UTF32LeNoBOM
- UTF32Be, UTF32BeNoBOM

The list of possible integer values that can be used can be found here (https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding#remarks – scroll down a little to see the chart). Values can be from the "Code Page" column or the "Name" column.

File_ChangeEncoding

File_ChangeEncoding(FilePath NVARCHAR(4000), CurrentEncoding NVARCHAR(30), FilePathNew NVARCHAR(4000), NewEncoding NVARCHAR(30), OverwriteExistingFile BIT, RemoveOriginalFile BIT)

RETURNS: NVARCHAR(4000)

NOTES:

- FilePath cannot be NULL or empty string
- IF FilePathNew IS NULL or is an empty string, then the file specified by FilePath will be over-written with the new encoding
- Please see the Note on Encoding Parameters for possible values for both “Encoding” parameters.
- SourceEncoding and NewEncoding is NOT case-sensitive
- IF file specified by FilePathNew already exists, an error will be thrown unless OverwriteExistingFile is set to 1 (true)
- OverwriteExistingFile must be set to 1 (true) IF NewFilePath IS NULL or is empty string
- RemoveOriginalFile must be set to 0 (false) IF NewFilePath IS NULL or is empty string
- The return value is an empty string upon success or an error message

EXAMPLES:
```
SELECT SQL#.File_ChangeEncoding('C:\SQL\exported_unicode_file.sql', 'utf8', '',
'Ansi', 1, 0);
-- simply update the file to ANSI / ASCII format
SELECT SQL#.File_ChangeEncoding('C:\some_file.txt', '1255',
'C:\new_path\some_file.txt', 'utf8', 1, 1)
-- convert the file and save in a new location, removing the original
```

File_CheckLongPathSupportRequirements

File_CheckLongPathSupportRequirements

PROC: Displays (and verifies) requirements for enabling long path support. Output is displayed in “Messages” tab of SQL Server Management Studio (SSMS).

NOTES:

- For full details on long path support, please visit:
  Long Path Support in SQL Server: A Case Study in Awesomeness

File_Copy

File_Copy(SourceFilePath NVARCHAR(4000), DestinationFilePath NVARCHAR(4000), OverWrite BIT)

RETURNS: NVARCHAR(4000)

Copies a single file from the Source to the Destination.

NOTES:

- SourceFilePath and DestinationFilePath cannot be empty string or NULL
- SourceFilePath and DestinationFilePath must be absolute paths (including the drive letter or UNC paths) and not relative ones
- If an error occurs it will be returned as the NVARCHAR(4000) else an empty string is returned
- If there is already a file at the DestinationFilePath of the same name as the SourceFilePath filename then it will either throw an error (if OverWrite is set to False / 0) or it will over-write the file (if OverWrite is set to True / 1)

- If the directory / folder portion of DestinationFilePath does not exist, it will NOT be created and an error will be thrown

EXAMPLES:
```
SELECT SQL#.File_CreateDirectory('C:\SQL#Test\Sub1\Sub2')
SELECT SQL#.File_WriteFile('C:\SQL#Test\Sub1\test.txt', 'Hello', 0, '')
SELECT SQL#.File_Copy('C:\SQL#Test\Sub1\test.txt', 'C:\SQL#Test\Sub3\test.txt',
0)
-- error caused since C:\SQL#TestFolder\Sub3 does not exist
SELECT SQL#.File_Copy('C:\SQL#Test\Sub1\test.txt',
'C:\SQL#Test\Sub1\Sub2\test.txt', 0)
-- copies file to Sub2 directory
SELECT SQL#.File_Copy('C:\SQL#Test\Sub1\test.txt',
'C:\SQL#Test\Sub1\Sub2\test.txt', 0)
-- error since file exists from previous command and OverWrite is set to 0
SELECT SQL#.File_Copy('C:\SQL#Test\Sub1\test.txt',
'C:\SQL#Test\Sub1\Sub2\test.txt', 1)
-- now it works since OverWrite is set to 1
```

File_CopyMultiple

File_CopyMultiple(StartingDirectory NVARCHAR(4000), Recursive TINYINT, DirectoryNamePattern NVARCHAR(4000), FileNamePattern NVARCHAR(4000), DestinationDirectory NVARCHAR(4000), OverWrite BIT)

RETURNS: TABLE (Name NVARCHAR(500), OriginalLocation NVARCHAR(1000), NewLocation NVARCHAR(1000), Length BIGINT, Operation NVARCHAR(10), Exception NVARCHAR(1000), Level INT)

Copies any files matching both the DirectoryNamePattern and FileNamePattern to the DestinationDirectory. Unlike File_Copy, File_CopyMultiple will create any directories in the DestinationDirectory that do not already exist rather than throwing an error.

NOTES:
- StartingDirectory cannot be empty string or NULL
- StartingDirectory must be an absolute path (including the drive letter or UNC path) and not a relative one
- Recursive allows for the following three options:
    - **0** = not recursive (StartingDirectory only)
    - **1** = recursive (*skip* empty directories / folders)
    - **2** = recursive (*include* empty directories / folders)
- DirectoryNamePattern and FileNamePattern are full Regular Expressions and if left empty will match everything
- DestinationDirectory cannot be empty string or NULL
- DestinationDirectory must be an absolute path (including the drive letter or UNC path) and not a relative one
- Any Exceptions / Errors will be reported per file and will not stop the rest of the operation
- An error will occur if there is a file of the same name at the Destination location AND OverWrite is set to False / 0
- In the returned table:
    - OriginalLocation and NewLocation are just the directories and do not include the Name of the file as it will not change when copied. If you need to change file names at the Destination then you need to copy them individually using File_Copy
    - Operation is always "COPY"
- In the returned table, Level will always be 0 when Recursive is set to False / 0 and will start with 1 as the base directory when Recursive is set to True / 1

EXAMPLES:
```
SELECT * FROM SQL#.File_CopyMultiple('C:\DBFiles', 0, '', '\.bak$',
'C:\DBBackUps', 1)
-- copy all .bak files (non-recursively) to a backup location,
--    do not overwrite
SELECT * FROM SQL#.File_CopyMultiple('C:\INetPub\WWWRoot', 1, 'images',
'(\.gif|\.jpg)$', 'C:\WebSiteBackUps', 1)
-- copy GIF and JPG files from websites directories (recursively, skip
--    empty folders) to backup location, overwritting existing files
```

File_CreateDirectory

File_CreateDirectory(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Creates the specified directory.

NOTES:
- FilePath cannot be NULL or an empty string
- Will create all Directories in the path if they do not exist
- Nothing (empty string) is returned if successful or the error is returned
- An error will occur if the drive or UNC share path cannot be found

EXAMPLES:
```
SELECT SQL#.File_CreateDirectory('C:\doesnt_exist\sub1')
-- creates C:\doesnt_exist\ and then C:\doesnt_exist\sub1\
```

File_CreateTempFile

File_CreateTempFile

RETURNS: NVARCHAR(4000)

Creates a uniquely-named, empty file and returns the full path to it.

NOTES:
- File is created in the temp directory associated with the user/account that the SQL Server process logs on as.

EXAMPLES:
```
DECLARE @Path NVARCHAR(1000)
SET @Path = SQL#.File_CreateTempFile()
```

File_CurrentEncoding

File_CurrentEncoding(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Gets the current encoding of the specified file.

NOTES:

- FilePath cannot be NULL or an empty string
- Possible return values are:
  - Western European (Windows)
  - Unicode [implied Little-Endian]
  - Unicode (Big-Endian)
  - Unicode (UTF-8)
  - Unicode (UTF-32) [implied Little-Endian]

EXAMPLES:
```
SELECT SQL#.File_ChangeEncoding('c:\one.txt', 'c:\two.txt', 'utf32', 1, 0)
SELECT SQL#.File_CurrentEncoding('C:\two.txt')
-- Unicode (UTF-32)
```

File_Decrypt

File_Decrypt(FilePath NVARCHAR(4000))

Decrypts a file that has been: a) encrypted by the same user account that is trying to Decrypt it, and b) encrypted by either File_Encrypt or anything that calls the general Windows File System Encrypt function (such as right-clicking on a file, going to Properties | Advanced, and checking “Encypt Conents to Secure Data”)

RETURNS: NVARCHAR(4000)

NOTES:
- FilePath cannot be NULL or an empty string
- Nothing (empty string) is returned if successful or the error is returned
- Only the OS user account that Encrypted the file can Decrypt it, in the case of running this within SQL Server it will be the account that SQL Server runs under
- See File_Encrypt for more details

EXAMPLES:
```
SELECT SQL#.File_Decrypt('C:\SQL#Test\test.txt')
```

File_Delete

File_Delete(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Deletes a single file only.

NOTES:
- FilePath cannot be NULL or an empty string
- Nothing (empty string) is returned if successful or the error is returned
- Cannot delete an entire directory; for that use File_DeleteDirectory

EXAMPLES:
```
SELECT SQL#.File_Delete('C:\SQL#Test\test.txt')
```

File_DeleteDirectory

File_DeleteDirectory(FilePath NVARCHAR(4000), Recursive BIT)

RETURNS: NVARCHAR(4000)

Deletes a directory and its contents.

NOTES:

- FilePath cannot be NULL or an empty string
- Nothing (empty string) is returned if successful or the error is returned
- If Recursive is set to 1 / True, then it will delete the entire directory structure starting with FilePath (like “rm -r” in UNIX)
- If Recursive is set to 0 / False, then the directory must be empty (no files or subdirectories) or else an error will occur
- FilePath cannot be a file

EXAMPLES:

```
SELECT SQL#.File_DeleteDirectory('C:\SQL#Test\Sub1\Sub2', 0)
-- causes an error: The directory is not empty.
SELECT SQL#.File_DeleteDirectory('C:\SQL#Test\Sub1\Sub2', 1)
-- all gone!
```

File_DeleteMultiple

File_DeleteMultiple(StartingDirectory NVARCHAR(4000), Recursive BIT, DirectoryNamePattern NVARCHAR(4000), FileNamePattern NVARCHAR(4000))

RETURNS: TABLE (Name NVARCHAR(500), OriginalLocation NVARCHAR(1000), NewLocation NVARCHAR(1000), Length BIGINT, Operation NVARCHAR(10), Exception NVARCHAR(1000), Level INT)

Deletes any files matching both the DirectoryNamePattern and FileNamePattern to the DestinationDirectory. DeleteMultiple will only delete files—not directories—and any directories left empty by deleting all of their files will still remain.

NOTES:

- StartingDirectory cannot be empty string or NULL
- StartingDirectory must be an absolute path (including the drive letter or UNC path) and not a relative one
- DirectoryNamePattern and FileNamePattern are full Regular Expressions and if left empty will match everything
- Any Exceptions / Errors will be reported per file and will not stop the rest of the operation
- If Recursive is set to True / 1 then any directories below StartingDirectory that contain files to be deleted will be traversed
- In the returned table:
  - OriginalLocation is just the directory and does not include the Name of the file
  - NewLocation is always NULL since it is not applicable
  - Operation is always “DELETE”
- In the returned table, Level will always be 0 when Recursive is set to False / 0 and will start with 1 as the base directory when Recursive is set to True / 1

EXAMPLES:

```
SELECT * FROM SQL#.File_DeleteMultiple('C:\DBFiles', 0, '', '\.bak$')
-- delete all .bak files (non-recursively) in C:\DBFiles,
SELECT * FROM SQL#.File_DeleteMultiple('C:\INetPub\WWWRoot', 1, 'images',
'(\.gif|\.jpg)$')
-- delete GIF and JPG files from websites directories (recursively)
```

File_Encrypt

File_Encrypt(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Encrypts files so that only the OS user account that Encrypted it can read it. The file is readable while Encrypted, but cannot be read by any OS account other than the one that Encrypted it unless it is Decrypted by the OS account that Encrypted it.

NOTES:

- FilePath cannot be NULL or an empty string
- Nothing (empty string) is returned if successful or the error is returned
- Only the OS user account that Encrypts the file can Decrypt it, in the case of running this within SQL Server it will be the account that SQL Server runs under
- See File_Decrypt for more details
- Encrypted does not mean unreadable; the file is still fully readable but ONLY by the OS user account that Encrypted it

EXAMPLES:

```
SELECT SQL#.File_Encrypt('C:\SQL#Test\test.txt')
```

File_FileExists

File_FileExists(InputPath NVARCHAR(4000))

RETURNS: INT

NOTES:

- Return values: 0 = Does Not Exist; 1 = Exists
- This is similar to File_PathExists, but only checks for files, not directories.
- If only checking for files, this function is faster than File_PathExists because it does not check each path to see if it might be a Directory.

EXAMPLES:

```
SELECT SQL#.File_FileExists(N'C:\no_path');
-- 0
SELECT SQL#.File_FileExists(N'C:\real_path\no_file.ext');
-- 0
SELECT SQL#.File_FileExists(N'C:\real_path\real_file.ext');
-- 1
```

File_GetCRC32

File_GetCRC32(FilePath NVARCHAR(4000), TrapErrorsInline BIT)

RETURNS: BIGINT

NOTES:

- {will add details later}

File_GetDirectoryListing

File_GetDirectoryListing(StartingDirectory NVARCHAR(4000), Recursive BIT, DirectoryNamePattern NVARCHAR(4000), FileNamePattern NVARCHAR(4000), @IncludeSecurityInfo BIT)

RETURNS: TABLE (Name NVARCHAR(500), Location NVARCHAR(1000), Length BIGINT, CreationTime DATETIME, LastAccessTime DATETIME, LastWriteTime DATETIME, ReadOnly BIT, Hidden BIT, Archive BIT, System BIT, Compressed BIT, Encrypted BIT, Temporary BIT, Type NVARCHAR(5), Level INT, ErrorMessage NVARCHAR(4000), SecurityDescriptor NVARCHAR(4000), OwnerSID NVARCHAR(4000), OwnerName NVARCHAR(4000), GroupSID NVARCHAR(4000), GroupName NVARCHAR(4000))

NOTES:

- If the “Log On As” account for the SQLServer service does not have permissions to enter a directory, that error will be noted in the ErrorMessage field.
- *Retrieving the security info fields has a definite negative impact on performance. Please set @IncludeSecurityInfo to 0 unless you need that information.*
- The “SecurityDescriptor” field is the NTFS ACL information, in SDDL notation, minus the Owner and Group, both of which are broken out into their own pairs of fields for “SID” and “Name”.
- The “OwnerName” and “GroupName” fields always return NULL, and will do so until they can be cached (that lookup is an even larger performance hit). Until then, the SIDs can be translated into names either by using the new OS_TranslateSddlSidToName function, or by using “SUSER_SNAME(SQL#.Convert_SddlSidToBinary(OwnerSID))”.
- If the directory portion of a path (i.e. the Location, including drive letter and “\”s) is 258 or more characters, and does not already start with either the “\\?\” or the “\\?\UNC\” prefix, the appropriate prefix will be automatically added. These prefixes are required for accessing long paths. For more info, please visit:

EXAMPLES:
```
SELECT * FROM SQL#.File_GetDirectoryListing(N'C:\SQL#Test\', 0, '', '', 0)
-- List all files and directories in C:\SQL#Test but not recursively
SELECT * FROM SQL#.File_GetDirectoryListing(N'C:\SQL#Test\', 1, '', '\.htm', 0)
-- List files with .htm* extension, recursively starting in C:\SQL#Test
SELECT * FROM SQL#.File_GetDirectoryListing(N'C:\INetPub\WWWRoot\', 1,
N'^images$', N'(\.gif|\.jpg)', 0);
-- List files with .gif or .jpg extensions, recursively starting in
--   C:\INetPub\WWWRoot\ and in folders named "images"
SELECT * FROM SQL#.File_GetDirectoryListing(N'C:\', 0, N'', N'', 1);
-- get listing from root directory; include security info
SELECT * FROM SQL#.File_GetDirectoryListing(N'\\Server\Share\', 0, N'', N'', 0);
-- get listing from a UNC path
SELECT SUM([Length]) FROM SQL#.File_GetDirectoryListing(N'C:\Windows\Temp', 1,
N'', N'', 0); -- get total size (in bytes) of C:\Windows\Temp directory
```

File_GetDirectoryName

File_GetDirectoryName(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Returns everything to the left of the filename.

EXAMPLES:
```
SELECT SQL#.File_GetDirectoryName('C:\Test\Path\FileName.ext');
-- C:\Test\Path
SELECT SQL#.File_GetDirectoryName('\\SERVER\ShareName\Path\FileName.ext');
-- \\SERVER\ShareName\Path
```

File_GetDriveInfo

File_GetDriveInfo(DriveName NVARCHAR(4000))

RETURNS: TABLE (Name NVARCHAR(500), VolumeLabel NVARCHAR(500), DriveFormat NVARCHAR(50), DriveType NVARCHAR(50), RootDirectory NVARCHAR(500), TotalSize BIGINT, UsedSpace BIGINT, TotalFreeSpace BIGINT, AvailableFreeSpace BIGINT)

NOTES:

- DriveName is NOT case-sensitive
- DriveName can be either a Drive Letter (e.g. ‘C’, ‘C:’, or ‘C:\’) or empty string ‘’ or NULL
- IF DriveName is to be an empty string or NULL, then a security setting of 3 (UNRESTRICTED) is required; see SQLsharp_SetSecurity for more details
- IF DriveName is a drive letter, then a security setting of 2 (EXTERNAL_ACCESS) will work
- In the result set:
    - Format can be: NTFS, CDFS, FAT32, etc
    - DriveType can be: CDRom, Fixed, Unknown, Network, NoRootDirectory, Ram, Removable

EXAMPLES:

```
SELECT * FROM SQL#.File_GetDriveInfo('');
/*
C:\          NTFS   Fixed  C:\   120023252992   55667834880   64355418112   64355418112
D:\   music  CDFS   CDRom  D:\   650801152      650801152     0             0
*/
SELECT * FROM SQL#.File_GetDriveInfo('c');
/*
C:\          NTFS   Fixed  C:\   120023252992   55667834880   64355418112   64355418112
*/
```

File_GetFile

File_GetFile(FilePath NVARCHAR(4000), SplitLines BIT, FileEncoding NVARCHAR(30))

RETURNS: TABLE (LineNum INT, ContentEncoding NVARCHAR(50), ContentLength BIGINT, Content NVARCHAR(MAX), LineLength BIGINT)

NOTES:

- If FilePath does not exist, LineNum / ContentLength / LineLength will all be -1, ContentEncoding will be NULL, and the Content field will be the system error message.
- If SplitLines = 0
    - One row is returned.
    - LineNum = the total number of lines.
    - ContentLength = total characters in the file (including newlines and/or returns).
    - LineLength = ContentLength.
    - Entire file is loaded into memory so that it can be returned without splitting.
- If SplitLines = 1
    - One row per line of the file is returned.
    - LineNum = actual line number of the file.
    - ContentLength = cumulative number of characters (excluding newlines / returns) read so far, inclusive of the current line.
    - LineLength = the number of characters (excluding newlines / returns) of the current line.
    - Stream rows from file to SQL Server; only 1 row of the file in memory at a time.
- FileEncoding
    - Please see the Note on Encoding Parameters for possible values.

EXAMPLES:
```
SELECT * FROM SQL#.File_GetFile('C:\nofile', 0);
-- -1 NULL  -1    Could not find file 'C:\nofile'.    -1
SELECT * FROM SQL#.File_GetFile('C:\Boot.ini', 0);
-- 6  utf-8 211   [boot loader]  timeout=30...  211
SELECT * FROM SQL#.File_GetFile('C:\Boot.ini', 1);
/*
1     utf-8 13    [boot loader]
2     utf-8 10    timeout=30
3     utf-8 51    default=multi(0)disk(0)rdisk(0)partition(2)\WINDOWS
4     utf-8 19    [operating systems]
...
*/
```

File_GetFileBinary

File_GetFileBinary(FilePath NVARCHAR(4000))

RETURNS: VARBINARY(MAX)

Reads the contents of a binary file. File_GetFileBinary can read both binary and text files whereas File_GetFile can only read text files.

NOTES:

- FilePath cannot be NULL or an empty string
- This is a scalar-valued function unlike File_GetFile which is a table-valued function

EXAMPLES:
```
SELECT SQL#.File_GetFileBinary('C:\Temp\manual.pdf');
```

File_GetFileInfo

File_GetFileInfo(FilePath NVARCHAR(4000))

RETURNS: TABLE (Name NVARCHAR(500), Location NVARCHAR(1000), Length BIGINT, CreationTime DATETIME, LastAccessTime DATETIME, LastWriteTime DATETIME, ReadOnly BIT, Hidden BIT, Archive BIT, System BIT, Compressed BIT, Encrypted BIT, Temporary BIT, Type NVARCHAR(5), Level INT)

EXAMPLES:
```
SELECT * FROM SQL#.File_GetFileInfo('c:\boot.ini');
```

File_GetFileName

File_GetFileName(FilePath NVARCHAR(4000), RemoveExtension BIT)

RETURNS: NVARCHAR(4000)

Returns just the filename.

EXAMPLES:
```
SELECT SQL#.File_GetFileName('C:\Test\Path\FileName.ext', 0);
-- FileName.ext
```

```
SELECT SQL#.File_GetFileName('C:\Test\Path\FileName.ext', 1);
-- FileName
SELECT SQL#.File_GetFileName('\\SERVER\ShareName\Path\FileName.ext', 0);
-- FileName.ext
SELECT SQL#.File_GetFileName('\\SERVER\ShareName\Path\FileName.ext', 1);
-- FileName
SELECT SQL#.File_GetFileName('C:\Test\Path\FileName.ext.zip', 1);
-- FileName.ext
```

File_GetFullPath

File_GetFullPath(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Returns the absolute path, translating any "." and ".." references.

NOTES:

- {will add details later}

EXAMPLES:

```
SELECT SQL#.File_GetFullPath(N'..\test.txt');
-- C:\WINDOWS\test.txt

SELECT SQL#.File_GetFullPath(N'C:\Program Files\.\Microsoft\..\..\test.txt');
-- C:\test.txt

SELECT SQL#.File_GetFullPath(N'\\ALBRIGHT\C$\.\Program Files\Microsoft SQL
Server\..\..\test.txt');
-- \\ALBRIGHT\C$\test.txt
```

File_GetHashBinary

File_GetHashBinary(FilePath NVARCHAR(4000), Algorithm NVARCHAR(50), TrapErrorsInline BIT)

RETURNS: VARBINARY(64)

Returns the specified hash value of the contents of the specified file.

NOTES:

- Algorithm:
  - MD5
  - SHA1
  - SHA256
  - SHA384
  - SHA512
- TrapErrorsLine
  - When 1, error codes returned are:
    - 0x01 = File Not Count
    - 0x08 = Invalid Algorithm

File_GetLineCount

File_GetLineCount(FilePath NVARCHAR(4000), TrapErrorsInline BIT)

RETURNS: INT

Returns the number of lines (i.e. number of newlines / returns).

NOTES:

- Does not load entire file into memory; only one line at a time is loaded. Max memory used is longest line of the file.
- TrapErrorsInline:
    - 0 / False = if an error is encountered, fail with an exception
    - 1 / True = if an error is encountered, return value is an error code denoted by a negative value (see below)
- Error code return values:
    - Only returned when TrapErrorsInline = 1
    - -1 = File Not Found
    - -2 = Directory Not Found
    - -3 = Drive Not Found
    - -4 = File Load Exception
    - -5 = Path Too Long
    - -6 = IO Exception
    - -7 = Unauthorized Access
    - -99 = Other

EXAMPLES:

```
SELECT SQL#.File_GetLineCount(N'C:\Windows\system.ini', 0);
-- 13
SELECT SQL#.File_GetLineCount(N'C:\pagefile.sys', 0);
-- System.IO.IOException
SELECT SQL#.File_GetLineCount(N'C:\pagefile.sys', 1);
-- -6
```

File_GetRandomFileName

File_GetRandomFileName()

RETURNS: NVARCHAR(4000)

NOTES:

- Return value is a cryptographically strong, random string that can be used as either a folder name or a file name
- Returned File or Directory name is in standard 8.3 format

EXAMPLES:

```
SELECT SQL#.File_GetRandomFileName();
-- nfuiogeq.vt2
-- w5wkauub.bzz
```

File_GetRootDirectory

File_GetRootDirectory(FilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Returns just the Root Directory or UNC Server and Share Name.

EXAMPLES:
```
SELECT SQL#.File_GetRootDirectory('C:\Test\Path\FileName.ext');
-- C:\
SELECT SQL#.File_GetRootDirectory('\\SERVER\ShareName\Path\FileName.ext');
-- \\SERVER\ShareName
```

File_GetTempPath

File_GetTempPath()

RETURNS: NVARCHAR(4000)

NOTES:

- Returns the path/directory of the system's Temp directory
- Can be used in conjunction with File_GetRandomFileName() for creating temporary files that should have unique and non-predictable names.

EXAMPLES:
```
SELECT SQL#.File_GetTempPath();
-- C:\WINDOWS\TEMP\
```

File_GUnzip

File_GUnzip(FilePath NVARCHAR(4000), OverwriteExistingFile BIT, RemoveOriginalFile BIT)

RETURNS: NVARCHAR(4000)

NOTES:

- FilePath cannot be NULL or empty string
- FilePath filename must end in ".gz" else an error will be thrown
- IF filename of FilePath without the ".gz" extension already exists, an error will be thrown unless OverwriteExistingFile is set to 1 (true)
- Return value is empty string upon success or any errors generated
- This function can handle files of any size as it only operates on 8k at a time as opposed to Util_GUnzip which has to read the entire VARBINARY(MAX) into memory first.
- This function resides in the SQL#.DotNetZip assembly, not in SQL#.FileSystem.

EXAMPLES:
```
SELECT SQL#.File_GUnzip('C:\Manual.PDF.gz', 1, 1);
```

File_GZip

File_GZip(@FilePath NVARCHAR(4000), OverwriteExistingFile BIT, RemoveOriginalFile BIT)

RETURNS: NVARCHAR(4000)

NOTES:

- FilePath cannot be NULL or empty string
- IF filename of FilePath with the ".gz" extension already exists, an error will be thrown unless OverwriteExistingFile is set to 1 (true)

- Return value is empty string upon success or any errors generated
- This function can handle files of any size as it only operates on 8k at a time as opposed to Util_GZip which has to read the entire VARBINARY(MAX) into memory first.
- This function resides in the SQL#.DotNetZip assembly, not in SQL#.FileSystem.

EXAMPLES:
```
SELECT SQL#.File_GZip('C:\Manual.PDF', 1, 1);
```

File_Move

File_Move(SourceFilePath NVARCHAR(4000), DestinationFilePath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Move or rename one file. Moving can be done across volumes.

NOTES:
- SourceFilePath and DestinationFilePath cannot be empty string or NULL
- SourceFilePath and DestinationFilePath can only be a full filenames, *not* directories
- Source and Destination path do *not* need to have identical roots; Move *will* work across volumes.
- Use **File_Move** to rename a file by keeping it in the same directory but specifying a new name in the DestinationFilePath
- SourceFilePath and DestinationFilePath must be absolute paths (including the drive letter or UNC paths) and not relative paths
- If an error occurs it will be returned as the NVARCHAR(4000) else an empty string is returned
- If there is already a file at the DestinationFilePath of the same name as the SourceFilePath filename then it will throw an error
- If the directory / folder portion of DestinationFilePath does not exist, it will NOT be created and an error will be thrown
- If SourceFilePath and DestinationFilePath are the same, the operation will be skipped and a message will be returned.

EXAMPLES:
```
SELECT SQL#.File_Move('C:\SQL#Test\test.txt', 'C:\SQL#Test\Sub1\t2.txt');
-- move the test.txt file to another directory with a new name
SELECT SQL#.File_Move('C:\SQL#Test\test.txt', 'C:\SQL#Test\t2.txt');
-- rename the test.txt file to t2.txt
SELECT SQL#.File_Move('C:\SQL#Test\test.txt', 'D:\Temp\test.txt');
-- move the test.txt file to the Temp directory on the D: drive
SELECT SQL#.File_Move('C:\SQL#Test\test.txt', 'D:\Temp\t3.txt');
-- move the test.txt file to the Temp directory on the D: drive and rename it
```

File_MoveDirectory

File_MoveDirectory(SourcePath NVARCHAR(4000), DestinationPath NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Move or rename an entire directory structure, or a single file. Move cannot be to a different volume.

NOTES:
- SourcePath and DestinationPath cannot be empty string or NULL
- SourcePath can be either a full filename or just a directory
- If SourcePath is a filename, then DestinationPath must also be a filename

- If SourcePath is a directory, then DestinationPath will be taken as a directory name
- Use **File_MoveDirectory** to rename a directory or file by keeping it in the same directory but specifying a new name in the DestinationPath
- SourcePath and DestinationPath must be absolute paths (including the drive letter or UNC paths) and not relative paths
- Source and Destination path must have identical roots; MoveDirectoy will *not* work across volumes.
- If an error occurs it will be returned as the NVARCHAR(4000) else an empty string is returned
- If there is already a file at the DestinationFilePath of the same name as the SourceFilePath filename then it will throw an error
- If the directory / folder portion of DestinationFilePath does not exist, it will NOT be created and an error will be thrown
- If SourcePath and DestinationPath are the same, the operation will be skipped and a message will be returned.
- To move only certain files (i.e. filter on name of file and/or directory), or to move files / directories to a different volume, please see: File_MoveMultiple

EXAMPLES:
```
SELECT SQL#.File_MoveDirectory(N'C:\SomeDirectory', N'C:\RenamedDirectory');
-- Rename a directory
SELECT SQL#.File_MoveDirectory(N'C:\SomeFolder', N'C:\OtherPath\SomeFolder');
-- Move a directory
SELECT SQL#.File_MoveDirectory(N'C:\SomeFolder', N'C:\OtherPath\NewFolderName');
-- Move and rename a directory
SELECT SQL#.File_MoveDirectory(N'C:\Folder\a.txt', N'C:\NewPath\Folder\a.txt');
-- Move a file
```

File_MoveMultiple

File_MoveMultiple(StartingDirectory NVARCHAR(4000), Recursive TINYINT, DirectoryNamePattern NVARCHAR(4000), FileNamePattern NVARCHAR(4000), DestinationDirectory NVARCHAR(4000))

RETURNS: TABLE (Name NVARCHAR(500), OriginalLocation NVARCHAR(1000), NewLocation NVARCHAR(1000), Length BIGINT, Operation NVARCHAR(10), Exception NVARCHAR(1000), Level INT)

NOTES:
- StartingDirectory and DestinationDirectory cannot be empty string or NULL
- StartingDirectory and DestinationDirectory must be absolute paths (including the drive letter or UNC path) and not a relative paths
- Recursive allows for the following three options:
    - **0** = not recursive (StartingDirectory only)
    - **1** = recursive (*skip* empty directories / folders)
    - **2** = recursive (*include* empty directories / folders)
- This operation *does* work across volumes.
- DirectoryNamePattern and FileNamePattern are full Regular Expressions, and if left empty will match everything
- Any Exceptions / Errors will be reported per file and will not stop the rest of the operation
- An error will occur if there is a file of the same name at the Destination location
- *Only the files at the OriginalLocation will be deleted; the source directory structure will not be deleted*.
- In the returned table:
    - OriginalLocation and NewLocation are just the directories and do not include the Name of the file as it will not change when moved. If you need to change file names at the Destination then you need to move them individually using File_Move
    - Operation is always "MOVE"

    - Level will always be 0 when Recursive is set to False / 0 and will start with 1 as the base directory when Recursive is set to either 1 or 2.

EXAMPLES:
```
SELECT * FROM SQL#.File_MoveMultiple('C:\SQL#Test', 0, '', '\.txt$', 'C:\Temp');
-- move .txt files from C:\SQL#Test (but not its subfolders) to C:\Temp
SELECT * FROM SQL#.File_MoveMultiple('C:\INetPub', 1, 'images', '', 'C:\Temp');
-- move all files from folders named "images" within C:\INetPub to C:\Temp
```

File_PathExists

File_PathExists(InputPath NVARCHAR(4000))

RETURNS: INT

NOTES:
- Return values: 0 = Does Not Exist; 1 = Is A Directory; 2 = Is A File
- This can be used in conjunction with File_WriteFile() to determine if the file already exists as File_WriteFile() will over-write an existing file as opposed to throwing an error.

EXAMPLES:
```
SELECT SQL#.File_PathExists('C:\no_path');
-- 0
SELECT SQL#.File_PathExists('C:\');
-- 1
SELECT SQL#.File_PathExists('C:\Boot.ini');
-- 2
```

File_SplitIntoFields

File_SplitIntoFields @FilePath NVARCHAR(4000) [, @RegExDelimiter NVARCHAR(4000)] [, @RowsToSkip INT] [, @ColumnNames NVARCHAR(4000)] [, @FileEncoding NVARCHAR(20)] [, @DataTypes NVARCHAR(4000)] [, @FirstRowContainsColumnNames BIT]

PROC: Result set is each row of file specified by @FilePath broken into fields based on @RegExDelimiter

NOTES:
- @FilePath:
    - Cannot be NULL or empty string
    - IF @FilePath does not exist an error will be thrown
- @RegExDelimiter
    - Optional parameter
    - A full Regular Expression (See RegEx section)
    - Default value = `N','`
- @RowsToSkip:
    - Optional parameter
    - Default = 0
    - Use = 1 to ignore header row (also see @FirstRowContainsColumnNames parameter)
- @ColumnNames:
    - Optional parameter
    - Comma-separated list of values that will be used to name the columns of the result set
    - Extra spaces around each name will be trimmed
    - If more fields are in the data than specified in ColumnNames then additional fields will be named as FieldN where N is the field number

    - If more fields are specified in ColumnNames than in the first row of the result set then extra Column Names will be ignored
    - If not set or set to NULL then all field names will be FieldN where N is the field number starting with 1
- @FileEncoding:
    - Optional parameter
    - Value is NOT case-sensitive
    - Value can be:
        - ASCII
        - UNICODE [implied Little Endian]
        - UTF8
        - UTF7
        - UnicodeBigEndian
        - UTF32 [implied Little Endian]
        - Any other value, including NULL, will select your server's system default
    - Default value = NULL
- @DataTypes:
    - Optional parameter
    - Value is NOT case-sensitive
    - Comma-separated list of values that will be used to specify the datatype of the columns of the result set
    - If more fields are in the data than specified in DataTypes then additional fields will be set to NVARCHAR(MAX)
    - If more fields are specified in DataTypes than in the first row of the result set then extra values will be ignored
    - If not set or set to NULL then all field datatypes will be set to NVARCHAR(MAX)
    - Empty value in source data will return empty string for (N)(VAR)CHAR / XML datatypes, 0x00 for (VAR)BINARY, and NULL for number / date datatypes.
    - Currently, the TIME and DATETIMEOFFSET datatypes do not work properly.
- @FirstRowContainsColumnNames:
    - Optional parameter
    - If set to 1:
        - will use first row of file to set column names
        - overrides @ColumnNames parameter if it is also set
        - does not impact rows skipped by @RowsToSkip
        - if @RowsToSkip = 0, then @RowsToSkip will be assumed to be 1 so that the header row isn't used to both set column names and as the first row of data
    - Default value = 0 / false
- Number of fields returned in result set is based on first row of data returned (meaning, if @RowsToSkip = 1 then the first row of data is Row 2)
- After number of fields to return is set, rows with more fields will have the additional fields ignored
- After number of fields to return is set, rows with fewer fields will return empty strings for the missing fields
- See also: String_SplitIntoFields and INET_SplitIntoFields

EXAMPLES:
```
INSERT INTO dbo.ImportTable (ProductNo, ProductName, SKU, Qty)
      EXEC SQL#.File_SplitIntoFields 'C:\test.txt', '\t', 1
-- this assumes that the "test.txt" file is tab-delimited, has
-- 4 columns, and that the first row contains the column names
```

File_Touch

File_Touch(FilePath NVARCHAR(4000), WhichTime NVARCHAR(20), NewAbsoluteTime DATETIME, NewRelativeTime BIGINT, RelativeFilePath NVARCHAR(4000), SkipFileCreation BIT)

RETURNS: NVARCHAR(4000)

Emulates the UNIX “touch” command.

NOTES:

- WhichTime:
  - Values are NOT case-sensitive
  - Values are:
    - “Both” or empty string ‘’ = Update both Last Access and Last Write times
    - “Access” = only update the Last Access time
    - “Write” = only update the Last Write time
- NewRelativeTime can be used to modify, in MilliSeconds, either the current system time or the time of the file specified by RelativeFilePath
- NewRelativeTime can be positive or negative
- RelativeFilePath points to an optional file to get the Last Access and/or Last Write time(s) from
- SkipFileCreation, if set to 1 / true, will NOT create a file that does not already exist (the default behavior of “touch” is to create a file that does not exist) and will return a message of “File does not exist” but will not error
- If NewAbsoluteTime is specified, both NewRelativeTime and RelativeFilePath must be NULL
- If NewAbsoluteTime is NULL, NewRelativeTime and/or RelativeFilePath can be specified
- If both NewAbsoluteTime and RelativeFilePath are NULL then time used is the current system time

EXAMPLES:

```
SELECT SQL#.File_Touch('C:\Test1.txt', 'Both', NULL, NULL, NULL, 1)
-- File does not exist
SELECT SQL#.File_Touch('C:\Test1.txt', 'Both', NULL, NULL, NULL, 0)
-- {this mirrors the default behavior of the "touch" command}
SELECT SQL#.File_Touch('C:\Test1.txt', '', '12/12/2001', NULL, NULL, 0)
SELECT SQL#.File_Touch('C:\Test2.txt', 'Write', NULL, NULL, 'C:\Test1.txt', 0)
SELECT SQL#.File_Touch('C:\Test2.txt', 'Both', NULL, 300000, 'C:\Test1.txt', 0)
SELECT SQL#.File_Touch('C:\Test3.txt', 'Both', NULL, -600000, NULL, 0)
```

File_WriteFile

File_WriteFile(FilePath NVARCHAR(4000), FileData NVARCHAR(MAX), AppendData BIT, FileEncoding NVARCHAR(20))

RETURNS: NVARCHAR(4000)

NOTES:

- FilePath must be an existing directory/folder but the filename does not need to already exist (e.g. C:\ExistingDirectory\NewFileName.txt)
- AppendData = 1 will append to existing file; AppendData = 0 will overwrite an existing file
- FileEncoding
  - Only applies if exporting to a file
  - Value is NOT case-sensitive
  - Value can be:
    - ASCII
    - UNICODE [implied Little Endian]
    - UTF8
    - UTF7
    - UnicodeBigEndian
    - UTF32 [implied Little Endian]
    - Any other value, including NULL, will select your server’s system default

- Return value is empty string '' for Success OR error message
- Since this command will always over-write an existing file (or append to it) but not throw an error that the file already exists, if you need to protect an already existing file then use File_PathExists() first before attempting to write the file.

EXAMPLES:

```
CREATE TABLE #TempCLR (CLRFunction SYSNAME)
INSERT INTO #TempCLR (CLRFunction)
      SELECT      SPECIFIC_NAME
      FROM  INFORMATION_SCHEMA.ROUTINES

SELECT SQL#.File_WriteFile('C:\clr_functions.txt', SQL#.String_Join('SELECT
CLRFunction FROM #TempCLR', CHAR(13)+CHAR(10), 1), 0, '')
DROP TABLE #TempCLR

-- to see the effect of the above, issue the following:
SELECT * FROM SQL#.File_GetFile('C:\clr_functions.txt', 1)
/*
1     utf-8 1095  File_GetTempPath
2     utf-8 1095  File_PathExists
3     utf-8 1095  File_WriteFile
4     utf-8 1095  File_GetFile
5     utf-8 1095  Agg_GeometricAvg
*/
```

File_WriteFileBinary

File_WriteFileBinary(FilePath NVARCHAR(4000), FileData VARBINARY(MAX), FileMode NVARCHAR(20), FileEncoding NVARCHAR(20))

RETURNS: NVARCHAR(4000)

NOTES:

- FilePath cannot be NULL or an empty string
- FileMode is NOT case-sensitive
- FileMode = Create (File created if it doesn't exist or over-written if it does exist); CreateNew (Error throw if file already exists); Append (File created if it doesn't exist or appended to if it does exist)
- FileEncoding
  - Only applies if exporting to a file
  - Value is NOT case-sensitive
  - Value can be:
    - ASCII
    - UNICODE [implied Little Endian]
    - UTF8
    - UTF7
    - UnicodeBigEndian
    - UTF32 [implied Little Endian]
    - Any other value, including NULL, will select your server's system default

EXAMPLES:

```
SELECT SQL#.File_WriteFileBinary('C:\SQL#Test\test.txt', 0x48656C6C6F,
'CreateNew', '')
-- creates the file
SELECT * FROM SQL#.File_GetFile('c:\sql#Test\test.txt', 0)
--    LineNum     ContentEncoding    ContentLength     Content
--    1            utf-8              5                 Hello
SELECT SQL#.File_WriteFileBinary('C:\SQL#Test\test.txt', 0x48656C6C6F,
'CreateNew', '')
-- error since it already exists; could use "Create" instead
```

Database

The **Database** functions reside in the SQL#.DB assembly. The following assemblies need to be installed in order to use the DB functions: SQL#.Network.

If you use any of the functions that access the file system or network, then this assembly will need a security setting of EXTERNAL_ACCESS (2). You can set this by executing the following query:

```
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.DB'
```

If you do not want to have this assembly in your system at all, you can do either of the following:

- Do not install the SQL#.DB assembly by setting the `@InstallSQL#DB` variable (towards the top of the script) to 0 before installing
- Uninstall the assembly by running:
  `EXEC [SQL#].[SQLsharp_Uninstall] N'SQL#.DB'`

Please note that when accessing the file system, the Operating System user account that will be used is the service account (i.e. "log on as") that is currently running the main "SQL Server (instance_name)" process (instance_name = MSSQLSERVER service for a "default" instance, or something else for a named instance). The account might be Local System Account or an account created specifically for SQL Server.

DB_BulkCopy

```
DB_BulkCopy   [ @SourceType NVARCHAR(4000) = NULL, ]
                [ @SourceConnection NVARCHAR(4000) = NULL, ]
                @SourceQuery NVARCHAR(4000),
                [ @DestinationConnection NVARCHAR(4000) = NULL, ]
                @DestinationTableName NVARCHAR(4000),
                [ @BatchSize INT = 0, ]
                [ @NotifyAfterRows INT = 0, ]
                [ @TimeOut INT = 30, ]
                [ @ColumnMappings NVARCHAR(4000) = NULL, ]
                [ @BulkCopyOptionsList NVARCHAR(4000) = NULL, ]
                [ @SourceCommandTimeout INT = 30, ]
                [ @RowsCopied BIGINT = -1 OUTPUT ]
```

PROC: Uses the .NET SqlBulkCopy class to emulate bcp.exe and the T-SQL `BULK INSERT` command. It can connect natively to Microsoft SQL Server and Oracle. It can also be used with Linked Servers to connect to any DB type that is supported by Linked Servers.

NOTES:

- @SourceType:
  - Is not case-sensitive
  - Can be either MSSQL, Oracle, or NULL
  - Defaults to MSSQL if not specified or set to NULL
  - Specifying "Oracle" uses native Oracle drivers but initial testing has found that using a MSSQL SourceType and a Linked Server to Oracle in the SourceQuery was actually faster.
- @SourceConnection:
  - Connection string to source server
  - Defaults to current connection (i.e. "Context Connection") if not specified or set to NULL
- @SourceQuery:
  - Query to get the source data
  - Can be any query including one that uses a Linked Server

- @DestinationConnection:
  - Connection string to destination server
  - If specified must be a Microsoft SQL Server
  - Defaults to “Data Source=(local); Integrated Security=true;;” if not specified or set to NULL
- @DestinationTableName:
  - The name of the Table that the data will import into
- @BatchSize:
  - Number of Rows per batch
  - Setting of 0 will use a single batch
  - Default is 0
- @NotifyAfterRows:
  - The number of rows copied after which a notification is sent showing the user how many rows total have been copied
  - Setting of 0 will not notify
  - Default is 0
- @TimeOut:
  - Number of seconds for the bulk copy operation to complete before it times out
  - Setting of 0 will wait indefinitely
  - Default is 30
- @ColumnMappings:
  - Only required if the SourceQuery result rows do not match in number and/or position with the DestinationTable
  - Pipe-delimited list of comma-separated pairs of column mappings
  - Each mapping takes the form of: SourceColumn,DestinationColumn
  - Columns can be referred to by name or position
  - If using column names it IS case-sensitive
  - Mappings must be either all names or all positions; you cannot mix specifying names and positions (even though MSDN says you can)
  - Basic example of 3 columns:
    IDField,TargetID|NameField,TargetName|Width,ItemWidth
    1,2|2,3|3,1
- @BulkCopyOptionsList:
  - Optional
  - Pipe-delimited list of options
  - Options are NOT case-sensitive
  - Options are:
    - **KeepIdentity** = Preserve source identity values. When not specified, identity values are assigned by the destination.
    - **CheckConstraints** = Check constraints while data is being inserted. By default, constraints are not checked.
    - **TableLock** = Obtain a bulk update lock for the duration of the bulk copy operation. When not specified, row locks are used.
    - **KeepNulls** = Preserve null values in the destination table regardless of the settings for default values. When not specified, null values are replaced by default values where applicable.
    - **FireTriggers** = When specified, cause the server to fire the insert triggers for the rows being inserted into the database.
    - **UseInternalTransaction** = When specified, each batch of the bulk-copy operation will occur within a transaction.
- @SourceCommandTimeOut:
  - Number of seconds to wait for the command (i.e. @SourceQuery) to execute
  - Setting of 0 will wait indefinitely
  - Default is 30
- @RowsCopied:
  - OUTPUT parameter

- Default value: -1
- If set to -1 (the default), it will not track the number of rows copied as it might add drag to the process. Please keep set to -1 if not using.

DB_BulkExport (Not available in Free version)

```
DB_BulkExport  @Query NVARCHAR(MAX),
               @TextQualifier NVARCHAR(4000) = N'',
               @TextQualifyAllColumns BIT = 0,
               @ColumnHeaderHandling NVARCHAR(4000) = N'Always',
               @BitHandling NVARCHAR(4000) = N'Word',
               @FirstRow INT = 1,
               @LastRow INT = 0,
               @OutputFilePath NVARCHAR(4000) = NULL,
               @FieldTerminator NVARCHAR(4000) = NULL,
               @RowTerminator NVARCHAR(4000) = NULL,
               @FileEncoding NVARCHAR(4000) = NULL,
               @AppendFile BIT = 0,
               @RowsExported INT = -1 OUTPUT,
               @ConnectionString NVARCHAR(500) = N'Context Connection = true;',
               @TextQualifierEscape NVARCHAR(50) = NULL,
               @CommandTimeout INT = 30,
               @OutputFormats NVARCHAR(4000) = N'',
               @OutputFormatsDelimiter NVARCHAR(100) = N'|'
```

PROC: Generates a data-dump much in the same way that BCP, SSIS, and Export Data wizard do. One of the problems with SSIS is that the Data Flow tasks store the column info (names, datatypes, position, etc.) which makes generating a dynamic result set almost impossible. SSIS also has the problem of not accepting a variable for the Flat File Destination if you want to dynamically assign the output filename. SSIS does, however, support text-qualification and column headers: both are important if the file should be easily readable and importable. BCP on the other hand does support dynamic queries as well column-headers. But doing text-qualification requires a format-file and doing so when generating a dynamic query is no easy task. And SQLCMD can do column-headers but not text-qualification. Hence, this procedure combines the benefits of SSIS with the benefits of BCP into a procedure that can do column-headers and text-qualification (like SSIS) but also supports dynamic queries and output filenames (like BCP). It also supports FirstRow and LastRow (like BCP).

NOTES:

- @Query:
  - Can be any query, including an EXEC procedure call
  - Value of NULL or empty string simply exits
- @TextQualifier:
  - Can be any character or set of characters or even an empty string
  - Cannot be NULL
  - Default value = empty string
- @TextQualifyAllColumns:
  - If set to True (1) then all fields are enclosed in the @TextQualifer
  - If set to False (0) then only the followings fields are enclosed in the @TextQualifier: CHAR, VARCHAR, TEXT, NCHAR, NVARCHAR, NTEXT, DATE, SMALLDATETIME, DATETIME, DATETIME2, DATETIMEOFFSET, TIME, UNIQUEIDENTIFIER, SQL_VARIANT, and XML
  - Default value = 0 / False
- @ColumnHeaderHandling:
  - Value is NOT case-sensitive
  - Value can be:

    - ‘Always’, NULL, or empty string ‘’: Always display the Column Headers whether there are results or not
    - ‘Results’: Only display the Column Headers if there is at least one result row
    - ‘Never’: Do not display the Column Headers no matter what
  - Fields that are to be text-qualified will also have their respective column-header text-qualified
  - Default value = N’Always’
- @BitHandling:
  - How to handle the display of BIT fields
  - Value is NOT case-sensitive
  - Only three possible values:
    - ‘Word’: Translate as a text-qualified ‘True’ or ‘False’. (This is how SSIS handles exporting BIT fields)
    - ‘Letter’: Translate as a text-qualified ‘T’ or ‘F’
    - ‘Number’: Translate as a non-text-qualified 1 or 0
  - Default value = N’Word’
- @FirstRow:
  - The first result row to export
  - Set to 0 or 1 to ignore (start with first row)
  - Default value = 1
- @LastRow:
  - The last result row to export
  - Set to 0 to ignore (no limit)
  - Default value = 0
- @OutputFilePath:
  - The full path to the export file including the filename and extension.
  - If set to empty string ‘’ or NULL then the output is sent as a regular query result set
  - If set then the file will be created with the exported data
  - Behavior if the output file already exists determined by @AppendFile parameter (see below)
  - For very large sets of data consider dumping directly to a file and not a result set
  - If this field is set you must have `EXTERNAL_ACCESS` set by doing:
    EXEC SQL#.SQLsharp_SetSecurity 2, ‘SQL#.DB’;
  - Default value = NULL
- @FieldTerminator:
  - Only applies if exporting to a file
  - Can be any character or set of characters including empty string ‘’
  - Value of NULL = tab (\t)
  - Default value is a tab (\t)
- @RowTerminator:
  - Only applies if exporting to a file
  - Can be any character or set of characters including empty string ‘’
  - Value of NULL = Carriage Return – Line Feed / CRLF (\r\n)
  - Default value is a Carriage Return – Line Feed / CRLF (\r\n)
- @FileEncoding:
  - Only applies if exporting to a file
  - Value is NOT case-sensitive
  - Value can be:
    - ASCII
    - UNICODE [implied Little Endian]
    - UTF7
    - UTF8
    - UnicodeBigEndian
    - UTF32 [implied Little Endian]
    - Any other value, including NULL, will select your server’s system default
  - Default value = NULL (i.e. your server’s system default)
- @AppendFile:

  - If file already exists and @AppendFile = 1, exported rows will be appended to the end of it
  - If file already exists and @AppendFile = 0, the file will be replaced
  - Default value = 0 / False
- @RowsExported:
  - OUTPUT variable
  - Returns the total number of records / rows exported
  - Default value = -1 (so that it is not required to be passed in)
- @ConnectionString:
  - A full Connection String allowing connection to a remote instance and/or as another Login.
  - When using an external connection that is using "Trusted_Connection = true", impersonation will automatically be enabled to ensure that a restricted user does not use this proc to come back in as a more priveldged user (which is essentially what happens when *not* using impersonation). Because impersonation is mandatory on external, trusted connections, users that are not based on a Windows Login cannot use external connections without specifying the "User ID" and "Password" connection string options.
  - If set to NULL or empty string '' it will use the in-process / internal Context Connection
  - Default value = 'Context Connection = true;'
- @TextQualifierEscape:
  - String that is prefixed to any embedded characters matching the @TextQualifier character in text-qualified fields, if a @TextQualifier is specified
  - Value of NULL will use whatever value @TextQualifier is set to (meaning: duplicate the embedded text qualifier, just like embedded single-quotes in a T-SQL string)
  - Value of empty string '' = no escape character / do not escape embedded text qualifiers
  - Default value = NULL
- @CommandTimeout:
  - Number of seconds before the executing query times out
  - Default value = 30
- @OutputFormats:
  - Controls the output format, per each column, including minimum width (to support fixed-width exports).
  - This is a delimited list of .NET composite format strings:
    - https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting
    - https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types#format-strings-and-net-types
  - This list can be sparsely populated.
  - The list does not need to contain definitions for all column; columns not in the list will be given the default / standard format.
  - You can add literal values to the left and/or right of each column's Composite Format string (do not use this ability to added quotes for text-qualification as that will not properly escape embedded quotes; text-qualification is handled internally by DB_BulkExport if a value is passed in for the @TextQualifier parameter).
  - General syntax:
    - Default delimiter is: `N'|'` (but can be controlled using @OutputFormatsDelimiter)
    - General syntax: `N'[[pre]{index[,[-]width][:format]}[post]][|[...]]'`
    - Syntax for single column: `N'[pre]{index[,[-]width][:format]}[post]'`
    - Index = 1 for the actual field value
    - Index = 0 for empty string (if you wanted to ignore the field for some reason)
    - Only "index" is required; pre, post, width, and format are all optional
    - "width" is a *minimum* field size, not maximum (meaning: if the data for the field is longer than a specified "width", that row will be longer than rows with data that fit within the "width"; there is currently no truncation being performed by DB_BulkExport)
    - Default format string for each field, by data type (if not overridden via @OutputFormats):
      - DATE: `{1:yyyy-MM-dd}`
      - SMALLDATETIME: `{1:yyyy-MM-dd HH:mm:ss}`

            - DATETIME: `{1:yyyy-MM-dd HH:mm:ss.fff}`
            - DATETIME2: `{1:yyyy-MM-dd HH:mm:ss.fffffff}`
            - DATETIMEOFFSET: `{1:yyyy-MM-dd HH:mm:ss.fffffff zzz}`
            - All other data types: `{1}`
    - Examples:
        - For N-field export, set first field (DATETIME) to a minimum width of 40 characters, left justified, having a format of "DayName, MonthName day, year 12-hour time", and leave any other fields with their default formatting: `N'{1,-40:dddd, MMMM dd, yyyy hh:mm:ss tt}'`
        - For 2-field export (MONEY, INT), set first field to currency (showing "$" on the left and only using 2 decimal places instead of 4) and the second field to hex, both fields being right-justified: `N'{1:C}|{1:X4}'`
        - For 5+ field report, set fields 2 and 5: `N'|{1:something}|||{1:something}'`
    - Column headers, if included, and per each field:
        - will always be left-justified
        - will *not* include any "pre" or "post" literal values, if any are specified
        - will have a minimum width of a "width" specified in the Composite Format string plus the total width of "pre" and "post" literal values, if either are included
    - If you are going to specify field widths in Composite Format strings to produced a fixed-width export, be sure to set the @FieldTerminator parameter to empty string – `N''` – if the exported data will be used as an import into another system. Or, you could set @FieldTerminator to "pipe" – `N'|'` – to have a dividing line between columns on a report.
    - When overriding the format of a column having any of the DATE* data types, you will need to manually specify the proper (small)date(time(2|offset)) format, as shown above, in order to get the default (i.e. BCP-style) format. This is due to the true .NET default format for date-based types not including the milliseconds, and other minor differences. For example, consider having an actual DATETIMEOFFSET value of: `'2018-10-18 22:32:36.7850229 -04:00'`.
        - BCP output will be: `2018-10-18 22:32:36.7850229 -04:00`
        - .NET output, by default, will be: `10/18/2018 10:32:36 PM -04:00`
        - Specifying "`~{1,-32}~`" will output: `~10/18/2018 10:32:36 PM -04:00   ~`
        - Specifying "`~{1,-35:yyyy-MM-dd HH:mm:ss.fffffff zzz}~`" will output: `~2018-10-18 22:32:36.7850229 -04:00 ~`
    - Default value = empty string
- @OutputFormatsDelimiter:
    - Only used if @OutputFormats has a value
    - Delimiter used to separate column format specifications
    - Default value = '|'

EXAMPLES:

```
-- export the Employee table from the AdventureWorks DB
-- export as a standard result set (good for testing)
-- do NOT text-qualify all columns, do NOT include column headers
-- include all rows, translate BIT fields to their native 0 or 1
-- use empty text-qualifier to effectively NOT text-qualify any column
EXEC SQL#.DB_BulkExport 'SELECT * FROM AdventureWorks.HumanResources.Employee',
'', 0, 'results', 'number', 0, 0, NULL, NULL, NULL, NULL;

-- export the Employee table from the AdventureWorks DB
-- export as a file (yes, I have done EXEC SQL#.SQLsharp_SetSecurity 2)
-- text-qualify ALL columns, include column headers
-- export only rows 10 - 50, translate BIT fields as 'True' or 'False'
-- use ASCII character 170 as text-qualifier (logical "not" symbol) as
-- it can be used in Import Wizard (use left ALT key and number pad 170)
-- use default RowTerminator (tab) and non-default comma FieldTerminator
```

```
EXEC SQL#.DB_BulkExport 'SELECT * FROM AdventureWorks.HumanResources.Employee',
'¬', 1, 'always', 'Word', 10, 50, 'C:\TestExport.txt', ',', NULL, 'Unicode';

EXEC SQL#.DB_BulkExport
          @Query = N'SELECT * FROM dbo.ExportTable;',
          @TextQualifier = N'"',
          @OutputFilePath = N'C:\temp\ExportTableData.txt',
          @ConnectionString = 'server=REMOTE-INST; trusted_connection = true;',
          @TextQualifierEscape = N'\';

-- Fixed-width test:
EXEC SQL#.DB_BulkExport
       @Query = N'
SELECT TOP (4) [name], [database_id] AS [db_id], [create_date], [service_broker_guid]
FROM [master].sys.databases;',
       @OutputFilePath = N'C:\TEMP\SQL#Test_BulkExport.txt',
       @FieldTerminator = N' | ', -- column inside border
       @FileEncoding = N'UTF8',
       @OutputFormats = N'{1,-7}|{1,7}|{1,-28:MMM d, yyyy @ HH:mm:ss.fff}|{1,38:b}';
/*
name    | db_id   | create_date                  | service_broker_guid
master  |       1 | Apr 8, 2003 @ 09:13:36.390   | {00000000-0000-0000-0000-000000000000}
tempdb  |       2 | Nov 13, 2018 @ 10:05:15.567  | {2b8af6ee-cf93-4af5-b26e-484cfcc23ccd}
model   |       3 | Apr 8, 2003 @ 09:13:36.390   | {00000000-0000-0000-0000-000000000000}
msdb    |       4 | Aug 22, 2017 @ 19:39:22.887  | {d01ee5be-f571-4cfe-b8e0-fc9ae6d15e2f}
*/
```

DB_CreateOrAlterQueryInfoTables (Not available in Free version)

DB_CreateOrAlterQueryInfoTables
    @TableNamePrefix NVARCHAR(100) = N'##QueryInfo',
    @TableScript NVARCHAR(MAX) OUTPUT

PROC: Creates tables needed to store the output of DB_GetQueryInfo or alters existing tables to the correct structure. DDL is either executed immediately, returned as an OUTPUT parameter to run later, or both. Two stored procedures are also created to make interacting with the tables and data easier: DeleteTest and GetBasicStats.

NOTES:

- For use with DB_GetQueryInfo
- @TableNamePrefix:
  - Set to NULL or empty string to get '##QueryInfo'
  - The value will be prefixed to the four table names:
  - Values can be:
    - #
    - #AnyPrefixString
    - ##
    - ##AnyPrefixString
    - AnyPrefixString
    - SchemaName.
    - SchemaName.AnyPrefixString
    - DatabaseName.SchemaName.
    - DatabaseName.SchemaName.AnyPrefixString
  - If the value starts with a single pound sign "#" (either '#' or '#AnyPrefixString') then the 4 tables must be created ahead of time and the only operation available is to Alter; Create is

not an option as local temporary tables created in a sub-process will not exist once that process (i.e. the Stored Procedure) ends. In this case, just run the four statements below, making sure to replace **{prefix}** with your desired prefix or nothing:

```
CREATE TABLE #{prefix}ExecutionContext (QueryInfoRemove INT);
CREATE TABLE #{prefix}ExecutionPlans (QueryInfoRemove INT);
CREATE TABLE #{prefix}StatsIO (QueryInfoRemove INT);
CREATE TABLE #{prefix}StatsTime (QueryInfoRemove INT);
```

  - Default value = '##QueryInfo'
- @TableScript:
  - OUTPUT parameter
  - Contains the SQL needed to create or alter the tables used by DB_GetQueryInfo
  - Set to empty string '' (no need for OUTPUT keyword) to run immediately
  - Set to a variable (which needs to be NULL and which it is upon declaration) with the OUTPUT keyword to save the script to that variable and to als prevent immediate execution.
- If creating the tables ahead of time and just needing to Alter, make sure each table has just one column which is named 'QueryInfoRemove' (the datatype is irrelevant).
- A stored procedure is auto-generated, using the same @TableNamePrefix value, to make it easy to remove test runs. Typically it is a good idea to remove the first test run as the additional time it took to load the CLR objects for this testing stored procedure should not negatively bias the test results. A good practice is to remove the first test run of any set where the query has changed.
  - Name: @TableNamePrefix + 'DeleteTest'
  - Example (assuming default @TableNamePrefix): ##QueryInfoDeleteTest
  - Parameters: @FirstQueryInfoID INT [, @LastQueryInfoID INT = NULL]
  - If @LastQueryInfoID is unspecified or set to NULL, only @FirstQueryInfoID will be deleted.
  - If @LastQueryInfoID is set to a positive value, all tests between @FirstQueryInfoID and @LastQueryInfoID will be deleted.
  - If @LastQueryInfoID is set to -1, all tests starting at @FirstQueryInfoID will be deleted.
- A stored procedure is auto-generated, using the same @TableNamePrefix value, that provides aggregated and sorted results for the output captures into the QueryInfo tables. The data is grouped by the [QueryGroup] field which is populated with the value of the @QueryGroup input parameter. The results are then ordered by Average Logical Reads.
  - Name: @TableNamePrefix + 'GetBasicStats'
  - Example (assuming default @TableNamePrefix): ##QueryInfoGetBasicStats
  - Parameters: None

EXAMPLES:

```
-- use default prefix of '##QueryInfo', run immediately
EXEC SQL#.DB_CreateOrAlterQueryInfoTables '', ''

-- use 'SchemaName.' prefix and capture the query to run later
DECLARE @Script NVARCHAR(MAX);
EXEC SQL#.DB_CreateOrAlterQueryInfoTables
      @TableNamePrefix = N'SchemaName.',
      @TableScript = @Script OUTPUT;
EXEC SQL#.Util_Print @Script;

-- use '#' prefix for local temp table, run CREATE TABLE manually, then Alter
CREATE TABLE #ExecutionContext (QueryInfoRemove INT);
CREATE TABLE #ExecutionPlans (QueryInfoRemove INT);
CREATE TABLE #StatsIO (QueryInfoRemove INT);
CREATE TABLE #StatsTime (QueryInfoRemove INT);

EXEC SQL#.DB_CreateOrAlterQueryInfoTables N'#', ''
```

DB_CurrentSQLStatement (Not available in Free version)

DB_CurrentSQLStatement(SQLText NVARCHAR(MAX), StatementStartOffset INT, StatementEndOffset INT)

RETURNS: NVARCHAR(MAX)

NOTES:

- For use with `sys.dm_exec_sql_text()` Dynamic Management Function (which returns [text] to be used as SQLText here) and any of the Dynamic Management objects (which return statement_start_offset and statement_end_offset): sys.dm_exec_query_stats, sys.dm_exec_requests, sys.dm_exec_cursors, sys.dm_exec_xml_handles, sys.dm_exec_query_memory_grants, sys.dm_exec_connections
- Short-hand for the following T-SQL expression:

```
SUBSTRING([text], (statement_start_offset / 2) + 1,
            (
                  (CASE statement_end_offset
                        WHEN -1 THEN DATALENGTH([text])
                        ELSE statement_end_offset
                  END - statement_start_offset) / 2
            ) + 1
      )
```

EXAMPLES:

```
SELECT SQL#.DB_CurrentSQLStatement(stext.[text], ereq.statement_start_offset,
       ereq.statement_end_offset) AS [CurrentStatement],
       ereq.*
FROM   sys.dm_exec_requests ereq
CROSS APPLY sys.dm_exec_sql_text(ereq.[sql_handle]) stext
```

DB_DescribeResultSets (Not available in Free version)

DB_DescribeResultSets
@TheQuery NVARCHAR(MAX),
@RowNumberToGetValuesFrom INT = 1,
@ResultSetNumberToDescribe INT = 0,
@ShowHiddenFields BIT = 0,
@ResultSet XML OUTPUT

RETURNS: TABLE (ResultSetNumber INT, FieldNumber INT, FieldName NVARCHAR(128), Value NVARCHAR(MAX), DataType NVARCHAR(150), DataTypeName NVARCHAR(128), ColumnSize INT, MaxLength INT, Precision SMALLINT, Scale SMALLINT, IsNullable BIT, IsAliased BIT, IsExpression BIT, IsIdentity BIT, IsKey BIT, IsReadOnly BIT, IsHidden BIT, BaseDatabaseName NVARCHAR(128), BaseSchemaName NVARCHAR(128), BaseTableName NVARCHAR(128), BaseColumnName NVARCHAR(128))

PROC: Gets result set meta-data and one-row of returned data for a submitted query.

NOTES:

- Similar to sp_describe_first_result_set which first appeared in SQL Server 2012
- Unlike sp_describe_first_result_set:
    - DB_DescribeResultSets actually runs the submitted query. If the query is not read-only / SELECT-only and you don't want the side-effect(s), just wrap the call to DB_DescribeResultSets in a BEGIN TRAN / ROLLBACK TRAN.
    - DB_DescribeResultSets does *not* have certain limitations such as not working with queries that use temporary tables

    - DB_DescribeResultSets does *not* guarantee that the described result set will always be returned by the submitted query; it just describes what was returned for that execution
    - DB_DescribeResultSets describes all returned result sets, not just the first one
    - DB_DescribeResultSets will include sample data from a single row for each result set
- @RowNumberToGetValuesFrom:
    - Values < 1 equate to 1
    - If value > rows returned for a particular result set, the [Value] field for each column of that result set will be set to "<no row>"
    - Default value = 1
- @ResultSetNumberToDescribe:
    - Which result set, if there are multiple, to describe
    - Set to 0 for ALL
    - Default value = 0
- @ShowHiddenFields:
    - If set to 1 / True, fields that are hidden will be returned with a value of 1 in the [IsHidden] field
    - If set to 0 / False, hidden fields will not be returned and all rows will show 0 in [IsHidden]
    - Submit the following query to see an example: `N'SELECT * FROM sys.objects'`
    - Default value = 0 / False
- @ResultSet:
    - OUTPUT parameter
    - Contains XML of the same fields and values as the Result Set of this proc if you need to do further processing on the results (you don't need to create a table and then INSERT...EXEC)
    - If this output is not needed, just pass in empty string '' and without the OUTPUT keyword.

EXAMPLES:
```
-- Get row 30 from all result sets, showing hidden fields, discard output param
EXEC SQL#.DB_DescribeResultSets N'SELECT * FROM sys.objects', 30, 0, 1, '';

-- Get row 11 from all result sets, no hidden fields, capture output to @Out
DECLARE @Out XML;
EXEC SQL#.DB_DescribeResultSets
      @TheQuery = N'SELECT * FROM sys.objects ; SELECT * FROM msdb.dbo.sysjobs',
      @RowNumberToGetValuesFrom = 11,
      @ResultSetNumberToDescribe = 0,
      @ShowHiddenFields = 0,
      @ResultSet = @Out OUTPUT;
SELECT @Out;
```

DB_DeserializeResults (Not available in Free version)

```
DB_DeserializeResults  @SerializedResults VARBINARY(MAX)
                       [ , @QueryToGetSerializedResults NVARCHAR(4000) = NULL ]
                       [ , @ConnectionString NVARCHAR (500) = NULL ]
```

PROC: Transforms one or more serialized result set chunks into one or more result sets.

NOTES:

- Multiple result sets of the same structure (column name and datatype per position) can be combined
    - Combining requires that result sets of the same structure not be separated by a result set of a different structure. Meaning, if we have 4 result sets, 2 of structure A and 2 of structure B:
        - If they are ordered as A, A, B, and B the result will be 2 result sets: A and B
        - If they are ordered as A, B, A, B the result will be 4 result sets: A, B, A, and B
- @SerializedResults VARBINARY(MAX)
    - Can accept a single VARBINARY value generated by either DB_SerializeResults or DB_SerializeResultsInChunks

    - Pass in NULL if not using
    - Cannot be defaulted
    - This field can be used at the same time that @QueryToGetSerializedResults is being used
        - Same rules apply for combining similar result sets
        - Result sets from this value are deserialized *before* any result sets can be deserialized from the results of the @QueryToGetSerializedResults query
- @QueryToGetSerializedResults NVARCHAR(4000)
    - Default value (if set to NULL or empty string '' or not specified) ={no query}
    - If a query is supplied it needs to return a single VARBINARY field
    - Any additional fields will be ignored, but the first field must be VARBINARY
    - Any number of rows can be read
    - Values returned in the VARBINARY field can be a mix of values generated by DB_SerializeResults and values generated by DB_SerializeResultsInChunks
    - Rows will be processed in order (please see note above about combining result sets)
    - This field can be used at the same time that @SerializedResults is being used
        - Same rules apply for combining similar result sets
        - Values returned by this query are returned *after* all result sets have been deserialized from the @SerializedResults value
- @ConnectionString NVARCHAR(500)
    - Default value (if set to NULL or empty string '' or not specified) = "Context Connection = true;"
    - If using a regular connection with Integrated Security:
        - Impersonation is automatically applied to prevent a low-priveleged user from using this as a means to come back in as a privileged user to run restricted commands.
        - Using impersonation might cause errors if the current security context is already impersonated or has no association to a Windows SID.
    - If set to "Context Connection = true;"
        - Current security context is used
        - Standard function restrictions apply except read-only stored procedures can be called
- See also: DB_SerializeResults and DB_SerializeResultsInChunks

```
-- The following global temp table is to be used in all of the following examples
SELECT *
INTO ##Results
FROM SQL#.DB_SerializeResultsInChunks(
   N'USE [master]; SELECT TOP 42 DB_NAME() AS [DatabaseName], * FROM sys.objects;
     SELECT TOP 437 * FROM sys.columns;
     USE [msdb]; SELECT TOP 333 DB_NAME() AS [DatabaseName], * FROM sys.objects;',
   20, NULL);

SELECT * FROM ##Results;
--------------
-- Once the global temp table exists, run each of the following examples by themselves

-- Get all result sets as they were originally sent
EXEC SQL#.DB_DeserializeResults
   @SerializedResults = NULL,
   @QueryToGetSerializedResults = N'SELECT [Results] FROM ##Results ORDER BY
SequenceNumber;';

-- Get all result sets in the reversed order from how they were originally sent
EXEC SQL#.DB_DeserializeResults
   @SerializedResults = NULL,
   @QueryToGetSerializedResults = N'SELECT [Results] FROM ##Results ORDER BY
ResultSetNumber DESC, ChunkNumber ASC;';
```

```
-- Extract a specific result set (#2 of 3)
EXEC SQL#.DB_DeserializeResults
   @SerializedResults = NULL,
   @QueryToGetSerializedResults = N'SELECT [Results] FROM ##Results WHERE
ResultSetNumber = 2 ORDER BY ChunkNumber ASC;';

-- Extract result sets (#1 and #3 of 3); they combine into a single result set
-- due to being the same structure (name and datatype per column)
EXEC SQL#.DB_DeserializeResults
   @SerializedResults = NULL,
   @QueryToGetSerializedResults = N'SELECT [Results] FROM ##Results WHERE
ResultSetNumber IN (1, 3) ORDER BY ResultSetNumber ASC, ChunkNumber ASC;';

-- Pass in single VARBINARY value containing two results sets (25 rows from [model]
-- and 10 rows from [tempdb]); they combine into a single result set due to being
-- the same structure (name and datatype per column)
DECLARE @ModelObjects VARBINARY(MAX);
SET @ModelObjects = SQL#.DB_SerializeResults(N'USE [model]; SELECT TOP 25 DB_NAME() AS
[DatabaseName], * FROM sys.objects;
USE [tempdb]; SELECT TOP 10 DB_NAME() AS [DatabaseName], * FROM sys.objects;', NULL);

EXEC SQL#.DB_DeserializeResults
   @SerializedResults = @ModelObjects
GO

-- Extract result sets (#1 and #3 of 3); AND pass in single VARBINARY value containing
-- two results sets (25 rows from [model] and 10 rows from [tempdb]); they all combine
-- into one result set due to being the same structure (name and datatype per column)
DECLARE @ModelObjects VARBINARY(MAX);
SET @ModelObjects = SQL#.DB_SerializeResults(N'USE [model]; SELECT TOP 25 DB_NAME() AS
[DatabaseName], * FROM sys.objects;
USE [tempdb]; SELECT TOP 10 DB_NAME() AS [DatabaseName], * FROM sys.objects;', NULL);

EXEC SQL#.DB_DeserializeResults
   @SerializedResults = @ModelObjects,
   @QueryToGetSerializedResults = N'SELECT [Results] FROM ##Results WHERE
ResultSetNumber IN (1, 3) ORDER BY ResultSetNumber ASC, ChunkNumber ASC;';
```

DB_DumpData (Not available in Free version)

```
DB_DumpData  @DBPattern NVARCHAR(4000),
                    @SchemaPattern NVARCHAR(4000),
                    @TablePattern NVARCHAR(4000),
                    @IncludeViews BIT,
                    @IncludeComputedColumns BIT,
                    @IdentityHandling NVARCHAR(4000),
                    @DBNameHandling NVARCHAR(4000),
                    @SchemaNameHandling NVARCHAR(4000),
                    @TableAndColumnNameQualifierLeft NVARCHAR(4000),
                    @TableAndColumnNameQualifierRight NVARCHAR(4000),
                    @StringAndDateQualifier NVARCHAR(4000),
                    @DateFormat SMALLINT,
                    @OutputFilePath NVARCHAR(4000),
                    @FileEncoding NVARCHAR(4000),
                    @LinkedServerName NVARCHAR(4000),
```

@DisableConstraints BIT,
@DisableTriggers BIT

PROC: Generates INSERT statements to recreate data. This procedure can work across all user databases on a server / instance or a filtered subset, all schemas or a filtered subset, and all tables or a filtered subset.

NOTES:

- Only generates INSERT statements to populate data; does NOT create tables or generate any DDL
- DB_DumpData works similar to the MySQL utility mysql_dump except that it does not generate any DDL
- Each INSERT ends with a semicolon (;) for compatibility with other RDBMS's
- Columns of datatype TIMESTAMP / ROWVERSION are not included as they cannot be inserted into directly
- @DBPattern:
  - A Regular Expression that can be used to filter which Databases are dumped
  - If left empty ('') then it will match all Databases
  - Pattern is NOT case-sensitive
  - System databases (master, model, msdb, and tempdb) will never match and cannot be dumped
- @SchemaPattern:
  - A Regular Expression that can be used to filter which Schemas are dumped
  - If left empty ('') then it will match all Schemas
  - Pattern is NOT case-sensitive
- @TablePattern:
  - A Regular Expression that can be used to filter which Tables are dumped
  - If left empty ('') then it will match all Tables
  - Pattern is NOT case-sensitive
- @IncludeViews:
  - Whether or not to include views as if they were tables
  - Only set to 1 if the destination DB has a table that should get this data and not a view of the same name
  - If included, Views are generated after all of the Tables
  - If included, Views will be marked with "(VIEW)" after the View name in the comment before the INSERT statements for the View
- @IncludeComputedColumns:
  - Whether or not to include column and data for Computed Columns
  - Only set to 1 if the destination DB has a table with a non-computed column definition for fields that are computed (i.e. formulas) in the source DB
  - If included, any Computed Columns in a table will be noted in the comments before the INSERT statements for that table
- @IdentityHandling:
  - How to handle IDENTITY fields
  - Valid values are: INSERT, INCLUDE, and EXCLUDE
  - Values are NOT case-sensitive
  - INSERT:
    - Include the column and its data
    - Use SET IDENTITY_INSERT [ON | OFF]
    - Use when putting data back into a table that has the same IDENTITY field AND you want to keep the same ID numbers
  - INCLUDE
    - Include the column and its data
    - Do NOT use SET IDENTITY_INSERT
    - Use when putting data back into a table that did not specify IDENTITY for this field
  - EXCLUDE
    - Do NOT include the column and its data

        - Use when putting data back into a table that has the same IDENTITY field but you want to generate new ID numbers
    - If Included or Inserted, the IDENTITY field in a table will be noted in the comments before the INSERT statements for that table
- @DBNameHandling:
    - How to format the DB Name in the INSERT statement
    - A %s variable will be replaced by the DBName if used
    - The %s is not required
    - The %s IS case-sensitive
    - Leave empty if you do not want any specification of DBName
    - If you want to hard-code a DB name then just specify it literally
    - If specifying a DBName either via %s or literal, be sure to include the trailing period (.)
    - For SQL Server, typical usage = '[%s].'
- @SchemaNameHandling:
    - How to format the Schema Name in the INSERT statement
    - A %s variable will be replaced by the Schema Name if used
    - The %s is not required
    - The %s IS case-sensitive
    - Leave empty if you do not want any specification of Schema Name
    - If you want to hard-code a Schema name then just specify it literally
    - If specifying a Schema Name either via %s or literal, be sure to include the trailing period (.)
    - For SQL Server, typical usage = '[%s].'
- @TableAndColumnNameQualifierLeft:
    - What table and column names are prefixed with (e.g. [, ", nothing, etc.)
    - For SQL Server use a left square-bracket ([)
- @TableAndColumnNameQualifierRight:
    - What table and column names are appended with (e.g.], ", nothing, etc.)
    - For SQL Server use a right square-bracket (])
- @StringAndDateQualifier:
    - What String (CHAR, VARCHAR, VARCHAR(MAX), TEXT, NCHAR, NVARCHAR, NVARCHAR(MAX), NTEXT, UNIQUEINDETIFIER, XML, and SQL_VARIANT) and Date (DATETIME and SMALLDATETIME) fields are enclosed in
    - For SQL Server use a single-quote (') which is represented by specifying two single-quotes ('')
- @DateFormat:
    - How the date values are converted into text
    - For SQL Server use a value of 121 or 101
    - DataFormat of 121 = yyyy-mm-dd hh:mi:ss.mmm(24h)
    - DateFormat of 101 = mm/dd/yyyy
- @OutputFilePath:
    - Is not required; can be left as NULL or empty string ('')
    - If set will dump all output (except errors) to the file specified
    - If set will require a setting of 2 or 3 for SQLsharp_SetSecurity where a setting of 2 equates to a DB setting of "SAFE" for the assembly (see discussion regarding Security in the Introduction on Page 5 as well as the SQLsharp_SetSecurity procedure)
    - If the file already exists it will be appended to
- @FileEncoding:
    - Only applies if exporting to a file
    - Value is NOT case-sensitive
    - Value can be:
        - ASCII
        - UNICODE [implied Little Endian]
        - UTF7
        - UTF8
        - UnicodeBigEndian

        - UTF32 [implied Little Endian]
        - Any other value, including NULL, will select your server's system default
- @LinkedServerName:
    - Is not required; can be left as NULL or empty string ('')
    - If set the LinkedServer needs to be SQL Server 2005 (or beyond)
- @DisableConstraints:
    - Is not required; is defaulted to 0 / False
    - If set to 1 / True will add the command:
      ALTER TABLE {TableName} NOCHECK CONSTRAINT ALL;
      before each Table and the command:
      ALTER TABLE {TableName} CHECK CONSTRAINT ALL;
      after each Table
- @DisableTriggers:
    - Is not required; is defaulted to 0 / False
    - If set to 1 / True will add the command:
      ALTER TABLE {TableName} DISABLE TRIGGER ALL;
      before each Table and the command:
      ALTER TABLE {TableName} ENABLE TRIGGER ALL;
      after each Table
- Output:
    - Directly to file by setting @OutputFilePath
        - No column data length problems as opposed to the other methods so this is ideal for dumping tables that make use of TEXT, NTEXT, VARCHAR(MAX), and NVARCHAR(MAX) fields
        - Requires a setting of 2 / SAFE or 3 / UNRESTRICTED for SQLsharp_SetSecurity
    - bcp:
        - Unfortunately, this does not work due to a bug in the bcp.exe utility that is fixed in HotFix 3 for SQL Server 2005 Service Pack 2 (http://support.microsoft.com/kb/939537) that you can request from Microsoft. Service Pack 3 might address this.
        - Once this bug is fixed, bcp will be a viable option for exporting directly to a file without having to set the security level to 2 / SAFE or 3 / UNRESTRICTED (which is required when setting @OutputFilePath)
    - SQL Server Management Studio (SSMS):
        - While this option allows for exporting directly to a file without having to set the security level to 2 / SAFE or 3 / UNRESTRICTED (which is required when setting @OutputFilePath), it does have the problem of not being able to return more than 8192 or 65535 characters per INSERT statement (which is the total for the full row, not just the data for each field).
        - Results to Text | Results to File:
            - Tools | Options | Query Results | SQL Server | Results to Text | Maximum number of characters displayed in each column = 8192
            - Results to File is quick, even if millions of rows of data, but can only show 8192 characters total including the INSERT with Table Name and Column List. Hence this will not work for tables that have 8000 or more bytes of data.
        - Results to Grid:
            - Tools | Options | Query Results | SQL Server | Results to Grid | Maximum Characters Retrieved / Non-XML Data = 65535
            - Tools | Options | Query Results | SQL Server | Results to Grid | Include column headers when copying or saving the results = NOT checked
            - This method can display more data per row than Results to Text and Results to File, but might take more memory if several million rows (or more) are returned
            - After results are returned, right click inside the results grid and select "Save Results As...". After file is saved, change extension from .csv to .sql

EXAMPLES:
```
/* ALL user DBs, no views, no computed columns, use IDENTITY_INSERT */
EXEC SQL#.DB_DumpData '','','', 0, 0, 'insert', '[%s].', '[%s].', '[', ']',
'''', 121, 'C:\PopulateData.sql'

/* we have a read-only DB that has Sales related data from AdventureWorks for
reporting: DBs starting with "adv", "sales" schema only, include computed
columns, include IDENTITY as regular field since app will not insert here, make
sure insert into AdventureWorksSales DB */
EXEC SQL#.DB_DumpData '^adv','^sales$','', 0, 1, 'include',
'[AdventureWorksSales].', '[%s].', '[', ']', '''', 121, NULL, NULL, NULL, 1, 1
```

DB_ForEach (Not available in Free version)

DB_ForEach [@DBPattern NVARCHAR(4000),]
[@DBExcludePattern NVARCHAR(4000),]
[@TablePattern NVARCHAR(4000),]
[@TableExcludePattern NVARCHAR(4000),]
[@PreTableQuery NVARCHAR(4000),]
[@ForEachTableQuery NVARCHAR(4000),]
[@PostTableQuery NVARCHAR(4000)]

PROC: Executes commands on matching DB and/or Tables. Emulates sp_MSforeachdb and sp_MSforeachtable combined, but gives full Regular Expressions for including and excluding Databases and Tables. Since all @__Query parameters are optional, this Proc can be used as a Database-only ForEach, a Table-only ForEach, or a Database and Table ForEach.

NOTES:
- @DBPattern:
  - Regular expression for which Databases to include
  - Not case-sensitive
  - Passing in NULL or empty string '' includes all Databases
  - If not specified, defaults to all Databases
- @DBExcludePattern:
  - Regular expression for which Databases to exclude
  - Not case-sensitive
  - If not specified or passing in NULL, translates to: ^(master|tempdb|model|msdb|resource|distribution|reportserver|reportservertempdb)$
  - Passing in empty string '' does not exclude any Databases
- @TablePattern:
  - Regular expression for which Tables to include
  - Not case-sensitive
  - Passing in NULL or empty string '' includes all Tables
  - If not specified, defaults to all Tables
- @TableExcludePattern:
  - Regular expression for which Databases to exclude
  - Not case-sensitive
  - If not specified or passing in NULL, does not exclude any Tables
  - Passing in empty string '' does not exclude any Tables
- @PreTableQuery:
  - Query to run for each Database that matches @DBPattern and does not match @DBExcludePattern, before any tables are processed

    - Current Database when running DB_ForEach is the same for the session in which DB_ForEach is called. If the query needs to be run in another Database, @PreTableQuery could be set to:
      'USE [{SQL#DBName}]'
- @ForEachTableQuery:
    - Query to run for each Table that matches @TablePattern and does not match @TableExcludePattern
    - Current Database is not automatically set to the Database in which the Table is found. If the query requires that the current Database be the one for the current Table, then be sure to execute a USE statement in either the @PreTableQuery or the beginning of the @ForEachTableQuery
- @PostTableQuery
    - Query to run for each Database that matches @DBPattern and does not match @DBExcludePattern, after all tables are processed
- Database, Schema, and Table name replacement tags are available for use in @PreTableQuery, @ForEachTableQuery, and @PostTableQuery
- Replacement tags are: {SQL#DBName}, {SQL#SchemaName}, {SQL#TableName}, and {SQL#FullTableName}
- Replacement tags ARE case-sensitive
- {SQL#SchemaName} and {SQL#TableName} are not contained in [and]
- {SQL#FullTableName} translates to: [SchemaName].[TableName]
- Replacement tag {SQL#DBName} is available in all three TableQuery parameters
- Replacement tags {SQL#SchemaName}, {SQL#TableName}, and {SQL#FullTableName} are only available in @ForEachTableQuery

EXAMPLES:

```
-- the following works like ForEachDB and separates the commands in case
-- two commands cannot be in the same Query since each Query is a single
-- batch and cannot have GOs
EXEC SQL#.DB_ForEach @PreTableQuery = 'USE {SQL#DBName}',
            @PostTableQuery = 'CHECKPOINT'

-- the following removes the default DBExcludePattern so that system DBs
-- are included.  It then shows how the replacement tags work.
EXEC SQL#.DB_ForEach NULL, '', NULL, NULL,
'USE {SQL#DBName}; PRINT DB_NAME()',
'PRINT ''-- {SQL#SchemaName}, {SQL#TableName}, {SQL#FullTableName}''', ''

-- rebuild indexes on all tables, including system tables
EXEC SQL#.DB_ForEach NULL, '', NULL, '', 'USE {SQL#DBName}',
            'ALTER INDEX ALL ON {SQL#FullTableName} REBUILD', ''
```

DB_GetQueryInfo (Not available in Free version)

DB_GetQueryInfo
    @Query NVARCHAR(MAX)
    [, @ExecutionMode NVARCHAR(15) = N'E']
    [, @ConnectionString NVARCHAR(500) = NULL]
    [,@ResultSetContent NVARCHAR(30) = N'Statistics']
    [, @QueryInfoTableNamePrefix NVARCHAR(100) = NULL]
    [, @QueryGroup NVARCHAR(100) = '']
    [, @CaptureExecutionPlans BIT = 1]

RETURNS: TABLES
(ExecutionPlan XML),

(ServerName NVARCHAR(128), LoginName NVARCHAR(128), UserName NVARCHAR(128), UsedImpersonation BIT, UsedContextConnection BIT, LocalServerTime DATETIME, UtcServerTime DATETIME, InfoType NVARCHAR(30), QueryBatch NVARCHAR(MAX), Messages NVARCHAR(MAX), ErrorMessage NVARCHAR(MAX), QueryGroup NVARCHAR(100) NULL),

(LocalTime DATETIME, UtcTime DATETIME, DatabaseName NVARCHAR(128), TimeType NVARCHAR(30), CpuMilliseconds INT, ElapsedMilliseconds INT)

(LocalTime DATETIME, UtcTime DATETIME, DatabaseName NVARCHAR(128), TableName NVARCHAR(128), ScanCount INT, LogicalReads INT, PhysicalReads INT, ReadAheadReads INT, LobLogicalReads INT, LobPhysicalReads INT, LobReadAheadReads INT)

PROC: Gets XML Query Plans and optionally output of STATISTICS IO and STATISTICS TIME.

NOTES:

- Results for all three types of information can be saved to tables, including additional info about the query execution, such as ExecutionTime, Login, Messages, Errors, etc.
- Tables to store the output, and stored procedures to help interact with the output, can be created directly by DB_CreateOrAlterQueryInfoTables or by a script that that Stored Procedure returns.
- @ExecutionMode:
    - Values:
        - E / Estimated / NULL
            - Returns only XML query plans
            - Results are from SET SHOWPLAN_XML
            - Empty result sets for IO stats and TIME stats
            - Default connection string: "context connection=true;"
            - Connection string can be overridden to use an external connection
        - A / Actual / empty string ‘’
            - Returns XML query plans, IO stats, and TIME stats
            - Results are from:
                - STATISTICS XML
                - STATISTICS IO
                - STATISTICS TIME
            - Default connection string: "trusted_connection=true;"
            - Connection string CANNOT use the Context Connection
    - Values are NOT case-sensitive
    - Default value = ‘E’
- @ConnectionString:
    - See @ExecutionMode for details on how the default is affected by that parameter
    - Default value = NULL
- @ResultSetContent:
    - Values:
        - None / N – no result sets returned
        - Query / Q:
            - Result sets, if any, are only from the submitted query
            - This option is only available for ExecutionMode of Actual / A as Estimated mode does not actually run the query
        - Statistics / S – result sets are the statistics based on @ExecutionMode
        - Anything else equates to ‘Statistics’
    - Values are NOT case-sensitive
    - Default value = ‘Statistics’
- @QueryInfoTableNamePrefix:
    - Set to NULL or empty string ‘’ to NOT save to tables

    - Value should match value used in DB_CreateOrAlterQueryInfoTables, where a NULL or empty string there would equate to a value of '##QueryInfo' here
    - Default value = NULL
- @QueryGroup:
    - Allows for labeling a query to easily group repeated tests / executions
    - The value is placed in the <TableNamePrefix>ExecutionContext table
    - Use in GROUP BY and/or WHERE clauses
    - Make sure that tests that are different queries get different values for @QueryGroup
    - Default value = empty string / ''
- @CaptureExecutionPlans:
    - Allows for discarding execution plans rather than returning them in the result set or saving them to the <TableNamePrefix>ExecutionPlan table.
    - When @ExecutionMode is set to 'Actual' or 'A', testing loops can produce a lot of execution plans, and this can take up both memory (all three types of stats are collected in memory while the submitted @Query is running) and increase the amount of time that it takes for the stored proc to complete.
    - If you aren't going to be looking though the execution plans then you are probably better of setting @CaptureExecutionPlans to 0 (especially if testing a loop that will iterate many times). They can always be turned on for one or two executions and then turned off again.
    - Default value = 1 / true
- Logical Reads (and possibly some other stats) might not be reported correctly for User-Defined Functions, especially Scalar UDFs and Multistatement TVFs. This is an issue with SQL Server and has nothing to do with SQL#.
- See also: http://sqlperformance.com/2012/10/t-sql-queries/beware_statistics_io

EXAMPLES:
```
-- run query, return stats instead of query results, don't save stats
DECLARE @SQL NVARCHAR(MAX);
SET @SQL = N'
       SELECT *
       FROM sys.objects so
       INNER JOIN sys.columns sc
                     ON sc.object_id = so.object_id
       ORDER BY so.name, sc.name';
EXEC SQL#.DB_GetQueryInfo @Query = @SQL, @ExecutionMode = N'a'

-- run query, return query results, save stats to global temp tables
EXEC SQL#.DB_CreateOrAlterQueryInfoTables '', '';
DECLARE @SQL NVARCHAR(MAX);
SET @SQL = N'
       SELECT *
       FROM sys.objects so
       INNER JOIN sys.columns sc
                     ON sc.object_id = so.object_id
       ORDER BY so.name, sc.name';

EXEC SQL#.DB_GetQueryInfo
       @Query = @SQL,
       @ExecutionMode = N'a',
       @ResultSetContent = N'Query',
       @QueryInfoTableNamePrefix = N'##QueryInfo'

SELECT * FROM ##QueryInfoStatsIO;
SELECT SUM(LogicalReads) AS [TotalLogicalReads] FROM ##QueryInfoStatsIO;

SELECT * FROM ##QueryInfoStatsTime;
SELECT SUM(CpuMilliseconds) AS [TotalCpuMilliseconds]
```

```
FROM ##QueryInfoStatsTime
WHERE TimeType = N'Execution';
```

DB_HTMLExport (Not available in Free version)

```
DB_HTMLExport(
                    @Query NVARCHAR(4000),
                    @ColumnHeaderHandling NVARCHAR(4000),
                    @BitHandling NVARCHAR(4000),
                    @NullReplacement NVARCHAR(4000),
                    @FirstRow INT,
                    @LastRow INT,
                    @PreTable NVARCHAR(MAX),
                    @PreHeaderRow NVARCHAR(4000),
                    @PreHeaderColumn NVARCHAR(4000),
                    @PostHeaderColumn NVARCHAR(4000),
                    @PostHeaderRow NVARCHAR(4000),
                    @PreDataRow NVARCHAR(4000),
                    @PreDataColumn NVARCHAR(4000),
                    @PostDataColumn NVARCHAR(4000),
                    @PostDataRow NVARCHAR(4000),
                    @PostTable NVARCHAR(MAX),
                    @OutputFilePath NVARCHAR(4000),
                    @FileEncoding NVARCHAR(4000) = NULL,
                    @EncodeHTML NVARCHAR(4000) = N'',
                    @AppendOutput BIT = 0
             )
```

RETURNS: NVARCHAR(MAX)

Generates an HTML report from the given Query. The final output is configurable via the "Pre" and "Post" variables. Since the structure is user-defined, this function can also be used to generate XML.

NOTES:

- @Query:
  - Can be any query, including an EXEC procedure call
- @ColumnHeaderHandling:
  - Value is NOT case-sensitive
  - Value can be:
    - Always, NULL, or empty string '': Always display the Column Headers whether there are results or not
    - Results: Only display the Column Headers if there is at least one result row
    - Never: Do not display the Column Headers no matter what
  - Fields that are to be text-qualified will also have their respective column-header text-qualified
- @BitHandling:
  - How to handle the display of BIT fields
  - Value is NOT case-sensitive
  - Only three possible values:
    - Word (Default): Translate as a text-qualified 'True' or 'False'. (This is how SSIS handles exporting BIT fields)
    - Letter: Translate as a text-qualified 'T' or 'F'
    - Number: Translate as a non-text-qualified 1 or 0
- @NullReplacement:
  - The string to replace any NULL value with
- @FirstRow:

  - The first result row to export
  - Set to 0 to ignore (start with first row)
- @LastRow:
  - The last result row to export
  - Set to 0 to ignore (no limit)
- @PreTable:
  - Any text before the results
  - If set to NULL will be: "`\t<table border="1">\n`"
  - String of "{SQL#Query}" will be replaced with the Query
- @PreHeaderRow:
  - Any text before the header row
  - If set to NULL will be: "`\t<tr>\n`"
- @PreHeaderColumn:
  - Any text before EACH header column
  - If set to NULL will be: "`\t\t<th>`"
  - String of "{SQL#Column}" will be replaced with the Column name
- @PostHeaderColumn:
  - Any text after EACH header column
  - If set to NULL will be: "`</th>\n`"
  - String of "{SQL#Column}" will be replaced with the Column name
- @PostHeaderRow:
  - Any text after the header row
  - If set to NULL will be: "`\t</tr>\n`"
- @PreDataRow:
  - Any text before EACH data row
  - If set to NULL will be: "`\t<tr>\n`"
- @PreDataColumn:
  - Any text before EACH data column
  - If set to NULL will be: "`\t\t<td>` "
  - String of "{SQL#Column}" will be replaced with the Column name
- @PostDataColumn:
  - Any text after EACH data column
  - If set to NULL will be: "`</td>\n`"
  - String of "{SQL#Column}" will be replaced with the Column name
- @PostDataRow:
  - Any text after EACH data row
  - If set to NULL will be: "`\t</tr>\n`"
- @PostTable:
  - Any text before the results
  - If set to NULL will be: "`\t</table>\n`"
  - String of "{SQL#Query}" will be replaced with the Query
- @OutputFilePath:
  - The full path to the export file including the filename and extension.
  - If this field is empty string '' or NULL then the output is sent as a regular query result set
  - If this field is set then the file will be created with the exported data
  - If the output file already exists, then the @AppendOutput parameter will control if it will be over-written or appended to.
  - For very large sets of data consider dumping directly to a file and not a result set
  - If this field is set you must have EXTERNAL_ACCESS (level 2) set by doing:
    ```
    EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.FileSystem';
    ```
  - If this field is set then the function's return value will be empty
- @FileEncoding:
  - Only applies if exporting to a file
  - Value is NOT case-sensitive

    - Please see the Note on Encoding Parameters for possible Values.
- @EncodeHTML:
    - Value cannot be NULL
    - Value is NOT case-sensitive
    - Value can be:
        - Empty string '' – does not encode any text into HTML entities
        - None – encodes HTML entities but no spaces or returns will be translated
        - Spaces – encodes HTML entities and spaces but not returns
        - Returns – encodes HTML entities and returns but not spaces
        - Both – encodes HTML entities including spaces and returns
    - See INET_HTMLEncode for examples
- @AppendOutput:
    - 1 = If file (set by @OutputFilePath) exists, append resulting value to it, else create it.
    - 0 = If file (set by @OutputFilePath) exists, overwrite it with the resulting value, else create it.
    - Default value is 0 (overwrite).

EXAMPLES:

```
-- Basic report using default value of including the Column Headers,
-- translate BIT values into words, replace NULL values with "-NULL-",
-- do not limit any rows, and take all default HTML values. Overwrite
-- file if it exists. This can easily be included in an email.
DECLARE @HTMLOutput NVARCHAR(MAX)

SELECT @HTMLOutput =
       SQL#.DB_HTMLExport('SELECT TOP 1 * FROM
AdventureWorks.HumanResources.Employee',
       '', 'word', '-NULL-', 0, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
       NULL, NULL, NULL, NULL, NULL, '')

PRINT @HTMLOutput

       <table border="1">
       <tr>
              <th>EmployeeID</th>
              <th>NationalIDNumber</th>
              <th>ContactID</th>
              <th>LoginID</th>
              <th>ManagerID</th>
              <th>Title</th>
              <th>BirthDate</th>
              <th>MaritalStatus</th>
              <th>Gender</th>
              <th>HireDate</th>
              <th>SalariedFlag</th>
              <th>VacationHours</th>
              <th>SickLeaveHours</th>
              <th>CurrentFlag</th>
              <th>rowguid</th>
              <th>ModifiedDate</th>
       </tr>
       <tr>
              <td>1</td>
              <td>14417807</td>
              <td>1209</td>
              <td>adventure-works\guy1</td>
              <td>16</td>
              <td>Production Technician - WC60</td>
              <td>5/15/1972 12:00:00 AM</td>
              <td>M</td>
              <td>M</td>
              <td>7/31/1996 12:00:00 AM</td>
              <td>False</td>
              <td>21</td>
```

```
            <td>30</td>
            <td>True</td>
            <td>aae1d04a-c237-4974-b4d5-935247737718</td>
            <td>7/31/2004 12:00:00 AM</td>
      </tr>
      </table>

-- This example wraps what could be a complex query into a temporary
-- Stored Procedure for a simple call within DB_HTMLExport.  Custom HTML
-- is used for nicer looking output that includes CSS as well as
-- displaying the Query before the results and even hiding the Query
-- in an HTML comment after the results.

-- EXEC SQL#.SQLSharp_SetSecurity 2

IF (OBJECT_ID('tempdb..#TempProc') IS NOT NULL)
BEGIN
      DROP PROCEDURE #TempProc
END
GO

CREATE PROCEDURE #TempProc (
      @WhatPercent      TINYINT
) AS


SELECT      TOP (@WhatPercent) PERCENT *
FROM  AdventureWorks.HumanResources.Employee
GO

DECLARE @CRLF NCHAR(2),
      @PreTable NVARCHAR(MAX),
      @PreHeaderRow NVARCHAR(100),
      @PreHeaderColumn NVARCHAR(100),
      @PostHeaderColumn NVARCHAR(50),
      @PostHeaderRow NVARCHAR(50),
      @PreDataRow NVARCHAR(100),
      @PreDataColumn NVARCHAR(100),
      @PostDataColumn NVARCHAR(50),
      @PostDataRow NVARCHAR(50),
      @PostTable NVARCHAR(MAX),
      @HTMLOutput NVARCHAR(MAX)

SET @CRLF = CHAR(13) + CHAR(10)

SELECT
      @PreTable = '<html>
<head>
      <title>Report Title</title>
      <style>
      .SQLTable   {border:2px solid black; font-family:verdana;
                        background:white;}
      .SQLHeader  {color:white; background:black; text-align:center;}
      .SQLRow           {background:white;}
      TH                {padding: 2px;}
      TD                {border-right:1px dashed black;
                        border-bottom:1px dashed black;}
      TH.Title    {color: red; font-weight: bold;}
      TD.Title    {color: blue; font-weight: bold;}
      </style>
</head>
<body bgcolor="#FFFFFF">

Query was: <b>{SQL#Query}</b><br><br>

      <table class="SQLTable" border="0" cellpadding="0">' + @CRLF,
      @PreHeaderRow = ' <tr class="SQLHeader">' + @CRLF,
      @PreHeaderColumn = '          <th class="{SQL#Column}">',
      @PostHeaderColumn = '</th>' + @CRLF,
      @PostHeaderRow = '      </tr>' + @CRLF,
      @PreDataRow = '   <tr class="SQLRow">' + @CRLF,
      @PreDataColumn = '          <td class="{SQL#Column}">',
      @PostDataColumn = '</td>' + @CRLF,
      @PostDataRow = '  </tr>' + @CRLF,
      @PostTable = '    </table>

<!-- {SQL#Query} -->

</body>
</html>' + @CRLF

SELECT SQL#.DB_HTMLExport('EXEC #TempProc 20', 'results', 'letter', '',
      0, 0, @PreTable, @PreHeaderRow, @PreHeaderColumn, @PostHeaderColumn,
      @PostHeaderRow, @PreDataRow, @PreDataColumn, @PostDataColumn,
      @PostDataRow, @PostTable, 'c:\table.html', 'unicode', '')


-- another example
DECLARE @HTMLOutput NVARCHAR(MAX)

SELECT @HTMLOutput =
      SQL#.DB_HTMLExport('SELECT TOP 2 * FROM
AdventureWorks.Production.ProductReview',
'', 'word', '-NULL-', 0, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, 'returns')

PRINT @HTMLOutput
```

DB_NewID (Not available in Free version)

DB_NewID()

RETURNS: UNIQUEIDENTIFIER

Allows for easily generating a GUID value from within a T-SQL function.

NOTES:

- SQL Server does not allow for calling NEWID in any T-SQL function: Scalar, Inline TVF, or Multistatement TVF
- The builtin NEWID function can be used in a View, which in turn can be selected from in a T-SQL function
    - The View method is slightly faster, so better for creating many GUIDs in a single SELECT
    - The View method *might* get cached and return the same value more than once (need to find reference!)

  - The DB_NewID function, while slightly slower, will never return the same value

EXAMPLES:
```
SELECT SQL#.DB_NewID()
-- 0E81C095-9E33-4CEB-BBEB-87B86988DDA5
```

DB_SerializeResults (Not available in Free version)

DB_SerializeResults(@Query NVARCHAR(MAX), @ConnectionString NVARCHAR(500))

RETURNS: VARBINARY(MAX)

Transforms one or more result sets from a query into a single binary representation. The binary value can be stored and/or transported and Deserialized later.

NOTES:
- Unlike DB_SerializeResultsInChunks, multiple results are not separated and are all included in the scalar result (though still separate within that single VARBINARY value).
- @Query NVARCHAR(MAX):
  - If set to NULL, a NULL will be returned
- @ConnectionString NVARCHAR(500)
  - Default value (if set to NULL or empty string '') = "Integrated Security=true; Enlist=false;"
  - If using a regular connection with Integrated Security:
    - Impersonation is automatically applied to prevent a low-priveleged user from using this as a means to come back in as a privileged user to run restricted commands.
    - Using impersonation might cause errors if the current security context is already impersonated or has no association to a Windows SID.
  - If set to "Context Connection = true;"
    - Current security context is used
    - Standard function restrictions apply except read-only stored procedures can be called
- See also: DB_DeserializeResults

EXAMPLES:
```
SELECT SQL#.DB_SerializeResults(N'SELECT * from sys.objects;', NULL);
-- 0x0001000000FFFFFFFF01000000000000000C020000B3696F6E3D332E332E38342E302C2043756C7...

SELECT SQL#.DB_SerializeResults(N'SELECT * from sys.objects; SELECT * FROM sys.views;',
N'Context Connection = true;');
-- 0x0001000000FFFFFFFF01000000000000000C020000004B535144422C205665296D332E332E2E302...
```

DB_SerializeResultsInChunks (Not available in Free version)

DB_SerializeResultsInChunks(Query NVARCHAR(MAX), MaximumRowsPerChunk INT, ConnectionString NVARCHAR(500))

RETURNS: TABLE(SequenceNumber INT, ResultSetNumber INT, ChunkNumber INT, StartingRowNumber INT, EndingRowNumber INT, TotalRows INT, SerializedLength INT, Results VARBINARY(MAX))

Transforms one or more result sets from a query into a binary representation that is spread out among one or more rows, per result set. The binary value(s) can be stored and/or transported and Deserialized later.

NOTES:
- @Query NVARCHAR(MAX):
  - If set to NULL, a NULL will be returned

- @MaxRowsPerChunk INT
  - A chunk is a set of one or more rows.
  - One or more chunks make up a result set
  - Setting to a value to >= 1
    - No more than this many rows will be in any chunk
    - There might be fewer than this many rows if there are not enough rows left
  - Setting to a value to < 1 or **DEFAULT**
    - Rows are not separated into chunks
    - One row per result set with all rows for that result set in the VARBINARY value
    - This setting makes it very easy to pick out one or more result sets from all result sets
- @ConnectionString NVARCHAR(500)
  - Default value (if set to NULL or empty string '') = "Integrated Security=true; Enlist=false;"
  - If using a regular connection with Integrated Security:
    - Impersonation is automatically applied to prevent a low-priveleged user from using this as a means to come back in as a privileged user to run restricted commands.
    - Using impersonation might cause errors if the current security context is already impersonated or has no association to a Windows SID.
  - If set to "Context Connection = true;"
    - Current security context is used
    - Standard function restrictions apply except read-only stored procedures can be called
- Unlike DB_SerializeResults, multiple results are always separated and cannot be combined.
- Separated result sets allow for:
  - Picking out one or more specific result sets to keep and discarding the others
  - Combining result sets of the same structure (field names and datatypes, per position)
  - Both of the above at the same time
- See also: DB_DeserializeResults

EXAMPLES:

```
SELECT * FROM SQL#.DB_SerializeResultsInChunks(N'SELECT * from sys.objects; SELECT *
FROM sys.columns', 0, '');
```

```
SequenceNumber ResultSetNumber ChunkNumber StartingRowNumber EndingRowNumbr TotalRows SerializedLength Results
1              1               1           1                 420            420       53130
            0x0001000000FFFFFFFF01000000000000000C020000004B53514C232E44422C2056657273696F6E3D332E332E38342E3022
2              2               1           1                 1613           1613      214382
            0x0001000000FFFFFFFF01000000000000000C020000004B53514C232E44422C2056657273696F6E3D332E332E38342E302C
```

```
SELECT * FROM SQL#.DB_SerializeResultsInChunks(N'SELECT * from sys.objects; SELECT *
FROM sys.columns', 200, '');
```

```
SequenceNum ResultSetNum ChunkNum StartingRowNum EndingRowNum TotalRows SerializedLength Results
1           1            1        1              200          200       26063            0x00010
2           1            2        201            400          200       27010            0x00010
3           1            3        401            420          20        4757             0x00010
4           2            1        1              200          200       28032            0x00010
5           2            2        201            400          200       28254            0x00010
6           2            3        401            600          200       28260            0x00010
7           2            4        601            800          200       29927            0x00010
8           2            5        801            1000         200       30320            0x00010
9           2            6        1001           1200         200       30638            0x00010
10          2            7        1201           1400         200       30705            0x00010
11          2            8        1401           1600         200       29968            0x00010
12          2            9        1601           1613         13        5222             0x00010
```

DB_ThrowException (Not available in Free version)

DB_ThrowException(@ErrorMessage NVARCHAR(2048), @Location NVARCHAR(1000) = ‘’, @LineNumber INT = 0)

RETURNS: SQL_VARIANT

Allows forcing an error in T-SQL Scalar User-Defined Functions and Multistatement Table-valued Functions.

NOTES:

- T-SQL does not allow for using either RAISERROR or THROW, but you can call a function
- If any of the three input parameters are set to NULL, a NULL is returned and *no* error is thrown
- Call via SELECT or EXEC (you can’t EXEC a Stored Procedure within a Function, but you can EXEC a scalar User-Defined Function)
- @Location:
    - Optional parameter
    - Additional info to denote in what section of code the error occurred
    - When not passing in a value:
        - If calling via SELECT, pass in the keyword DEFAULT or an empty string ‘’
        - If calling via EXEC, do not pass in the parameter or pass in an empty string ‘’
- @LineNumber
    - Optional parameter
    - Additional info to denote what line the error occurred on
    - When not passing in a value:
        - If calling via SELECT, pass in the keyword DEFAULT or 0
        - If calling via EXEC, do not pass in the parameter or pass in 0

EXAMPLES:

```
SELECT SQL#.DB_ThrowException(N'This did not work!', DEFAULT, DEFAULT)

Msg 6522, Level 16, State 2, Line 1
A .NET Framework error occurred during execution of user-defined routine or aggregate
"DB_ThrowException":
DB+DatabaseException:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This did not work!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DB+DatabaseException:

SELECT SQL#.DB_ThrowException(N'This did not work!', N'TestFunction', DEFAULT)

SELECT SQL#.DB_ThrowException(N'This did not work!', DEFAULT, 123)

SELECT SQL#.DB_ThrowException(N'This did not work!', N'TestFunction', 6454)

Msg 6522, Level 16, State 2, Line 1
A .NET Framework error occurred during execution of user-defined routine or aggregate
"DB_ThrowException":
DB+DatabaseException:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This did not work!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DB+DatabaseException:
   at: TestFunction ; Line: 6454

EXEC SQL#.DB_ThrowException N'This did not work!'

Msg 6522, Level 16, State 1, Procedure DB_ThrowException, Line 0
```

```
A .NET Framework error occurred during execution of user-defined routine or aggregate
"DB_ThrowException":
DB+DatabaseException:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This did not work!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DB+DatabaseException:
```

DB_TryCatch (Not available in Free version)

DB_TryCatch(@SQL NVARCHAR(MAX))

RETURNS: TABLE (ErrorHasOccurred BIT, ErrorMessage NVARCHAR(4000), ErrorNumber INT, ErrorSeverity TINYINT, ErrorState TINYINT, ReturnValue NVARCHAR(MAX))

Allows catching errors in T-SQL Scalar User-Defined Functions and Multistatement Table-valued Functions.

NOTES:

- T-SQL does not allow for using TRY...CATCH, but you can call a function
- Passing in NULL for @SQL returns an empty result set
- "Context Connection" (the internal / in-process connection) is used, meaning:
    - Even though random SQL can be executed, there are no security concerns as the security context is that of whoever is selecting from this function
    - No DML or DDL is allowed, or anything that can change state (i.e. side-effecting)
- Unlike with T-SQL functions:
    - you *can* execute stored procedures, *but* all of the other restrictions still apply (such as no SET commands, no DDL or DML, etc)
    - you *can* do Dynamic sql (whatever is passed into the @SQL parameter is by definition Dynamic SQL)
- The [ErrorHasOccurred] field is a direct indication of an error happening so that whether or not the operation failed does not need to be indirectly determined by checking any of the error detail fields.
- If no error occurs:
    - [ErrorHasOccurred] field will be set to 0.
    - [ErrorMessage] will be NULL
    - [ErrorNumber], [ErrorSeverity], and [ErrorState] will be 0
    - [ReturnValue] can be populated by setting the local variable @SQL#Output:
        - There is no need to declare this variable as it already exists
        - The variable is NVARCHAR(MAX)
        - Anything can be passed back.
        - Multiple values, tables, multiple tables, etc can be passed back as XML. Just SELECT the variable(s) and/or table(s) using `FOR XML`. By default the `FOR XML` clause outputs an NVARCHAR(MAX) containing XML and is not an XML datatype unless the "`, TYPE`" option is added (but don't as that will cause an error).
- If an error does occur:
    - [ErrorHasOccurred] field will be set to 1.
    - [ErrorMessage], [ErrorNumber], [ErrorSeverity], and [ErrorState] will be set appropriately.
    - [ReturnValue] field will be NULL
- Pass values in by concatenating them directly into the SQL that is being passed in to execute.
    - Just like with the ReturnValue output, passing in multiple items (including one or more tables) can be done by packaging everything in XML.

EXAMPLES:

```
-- simple query / no error / no return
SELECT * FROM SQL#.DB_TryCatch(N'DECLARE @T INT; SET @T = 5; SELECT @T / 2.0;');
```

```
ErrorHasOccurred    ErrorMessage ErrorNumber  ErrorSeverityErrorState    ReturnValue
0                   NULL         0            0            0             NULL

-- simple query / no error / return val
SELECT * FROM SQL#.DB_TryCatch(N'DECLARE @T INT; SET @T=5; SET @SQL#Output = (@T/2.0)');

ErrorHasOccurred    ErrorMessage ErrorNumber  ErrorSeverityErrorState    ReturnValue
0                   NULL         0            0            0             2.500000

-- simple query / 2 errors
SELECT * FROM SQL#.DB_TryCatch(N'select 1 / 0; EXEC t; SET @SQL#Output = 5;');

ErrorHasOccurred    ErrorMessage ErrorNumber  ErrorSeverityErrorState    ReturnValue
1                   Divide by    8134         16           1             NULL
                    zero error
                    encountered.

-- proc call / error
SELECT * FROM SQL#.DB_TryCatch(N'EXEC sp_who2;');

ErrorHasOccurred  ErrorMessage         ErrorNumber ErrorSeverity ErrorState ReturnValue
1                 Invalid use of a     443         16            2          NULL
                  side-effecting
                  operator 'SET ON/OFF'
                  within a function.

-- proc call / no error / return val
CREATE PROCEDURE #Proc (@Param1 INT, @Param2 INT OUTPUT) AS -- can't do SET NOCOUNT ON
       SET @Param2 = (@Param1 * 10);
GO

SELECT * FROM SQL#.DB_TryCatch(N'EXEC #Proc @Param1=35, @Param2 = @SQL#Output OUTPUT;');

ErrorHasOccurred    ErrorMessage ErrorNumber  ErrorSeverityErrorState    ReturnValue
0                   NULL         0            0            0             350

-- simple query / no error / return val as XML (to pass back a table)
SELECT *, DATALENGTH(ReturnValue) AS [OutputBytes], CONVERT(XML, ReturnValue) AS [InXML]
FROM   SQL#.DB_TryCatch(N'SET @SQL#Output = (SELECT TOP 19 * FROM sys.objects FOR XML RAW);');

ErrorHasOccurred ErrorMessage ErrorNumbr ErrorSeverity ErrorState ReturnValue BytesReturned TheXML
0                NULL         0          0             0
<row name="sysrscols" object_id="3" schema_id="4" parent_object_id="0" type="S "
type_desc="SYSTEM_TABLE" create_date="2012-02-10T20:16:00.707" modify_date="2012-02-
10T20:16:00.713" is_ms_shipped="1" is_published="0" is_schema_published="0"/><row
name="sysrowsets" ... />
9252
<row name="sysrscols" object_id="3" schema_id="4" parent_object_id="0" type="S "
type_desc="SYSTEM_TABLE" create_date="2012-02-10T20:16:00.707" modify_date="2012-02-
10T20:16:00.713" is_ms_shipped="1" is_published="0" is_schema_published="0" /><row
name="sysrowsets" ... />

-- used in scalar function (Dynamic SQL + TryCatch + ThrowException)
CREATE FUNCTION dbo.TestFunc (@DatabaseName SYSNAME)
RETURNS INT
AS
BEGIN
```

```
        DECLARE @SQL NVARCHAR(MAX),
                        @ReturnValue NVARCHAR(10),
                        @ErrorHasOccured BIT,
                        @ErrorNumber INT,
                        @ErrorMessage NVARCHAR(4000),
                        @NumObjects INT;

        SET @SQL = N'SET @SQL#Output = (SELECT COUNT(*) FROM ['
                      + @DatabaseName + N'].sys.objects);';

        SELECT  @ErrorHasOccured = tc.ErrorHasOccurred,
                        @ErrorMessage = tc.ErrorMessage,
                        @ErrorNumber = tc.ErrorNumber,
                        @ReturnValue = tc.ReturnValue
        FROM    SQL#.DB_TryCatch(@SQL) tc;

        IF (@ErrorHasOccured = 0)
        BEGIN
                SET @NumObjects = CONVERT(INT, @ReturnValue);
        END;
        ELSE
        BEGIN
                IF (@ErrorNumber = 208)
                BEGIN
                        SET @NumObjects = -1;
                END;
                ELSE
                BEGIN
                        DECLARE @ProcCall NVARCHAR(500);
                        SET @ProcCall = N'dbo.TestFunc(N''' + @DatabaseName +  N''')';
                        SET @ErrorMessage = @ErrorMessage + NCHAR(10) + NCHAR(10) + NCHAR(9) + @SQL;
                        EXEC SQL#.DB_ThrowException @ErrorMessage, @ProcCall, 18;
                END;
        END;

        RETURN @NumObjects;
END;
GO

-- no error
SELECT dbo.TestFunc(N'master') AS [NumObjects];
-- 91

-- handled error (in DB name; @ErrorNumber = 208)
SELECT dbo.TestFunc(N'nothere') AS [NumObjects];
-- -1

-- Using the [TestFunc] function just created, cause an error and catch it
CREATE PROCEDURE #Temp2 AS
BEGIN TRY
        SELECT dbo.TestFunc(N'bad]name') AS [NumObjects], 1 AS [y];
END TRY
BEGIN CATCH
        SELECT 1 AS [Error];
        PRINT '----------------------';
        PRINT ERROR_MESSAGE();
        PRINT '----------------------';
        PRINT ERROR_PROCEDURE();
        PRINT '----------------------';
```

```
END CATCH;
GO

EXEC #Temp2;

NumObjects    y

Error
1

Messages tab shows:
A .NET Framework error occurred during execution of user-defined routine or aggregate
"DB_ThrowException":
FunctionHelpers+DatabaseException:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Incorrect syntax near ']'.

        SET @SQL#Output = (SELECT COUNT(*) FROM [bad]name].sys.objects);
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FunctionHelpers+DatabaseException:
   at: dbo.TestFunc(N'bad]name') ; Line: 15
.
---------------------
#Temp2___________________________________________________________________________________000094D1
---------------------

-- used in Table-Valued function (Dynamic SQL + TryCatch + ThrowException)
CREATE FUNCTION dbo.TestTVF (@TableNamePattern NVARCHAR(100))
RETURNS TABLE
AS RETURN
WITH cte AS
(
        SELECT  sd.[name] AS [DatabaseName], CONVERT(XML, tc.ReturnValue) AS [TableNames]
        FROM    sys.databases sd
        CROSS APPLY     SQL#.DB_TryCatch(N'SET @SQL#Output = (SELECT st.[name] FROM ['
                                + sd.[name] + N'].sys.tables st FOR XML RAW);') tc
        WHERE   sd.[name] NOT IN (N'master', N'model', N'msdb', N'tempdb')
        AND             tc.ErrorHasOccurred = 0
)
SELECT  cte.DatabaseName, CASE
        WHEN SQL#.RegEx_IsMatch4k(ca.TableName, @TableNamePattern, 1, N'IgnoreCase') = 1
                        THEN SQL#.DB_ThrowException(N'Bad table name: ' + ca.TableName,
cte.DatabaseName, DEFAULT)
                ELSE ca.TableName
        END AS [TableName]
FROM    cte
CROSS APPLY     (SELECT t.c.value(N'./@name[1]', N'SYSNAME') AS [TableName] FROM
cte.TableNames.nodes(N'/row') t(c)) ca;
GO

SELECT * FROM dbo.TestTVF(N'info');

DatabaseName                TableName
CaseSensitiveCollation      test1
CaseSensitiveCollation      test2
Database2                   Table1
Database2                   Table2
ReportServer                Keys
ReportServer                History
```

```
ReportServer                  DBUpgradeHistory

Messages tab shows:
Msg 6522, Level 16, State 2, Line 2
A .NET Framework error occurred during execution of user-defined routine or aggregate
"DB_ThrowException":
FunctionHelpers+DatabaseException:
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Bad table name: ConfigurationInfo
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
FunctionHelpers+DatabaseException:
   at: ReportServer
.
```

DB_WaitForDelay

DB_WaitForDelay(@MillisecondsToSleep INT, @NonDeterminiser VARBINARY(4000))

RETURNS: BIT

Allows for easily pausing / sleeping from within a T-SQL function.

NOTES:

- *WARNING: Be VERY careful when using this function in Production code. Outside of debugging, there are very few (if any) scenarios in which you want to keep a Transaction open longer than necessary. This function is great for debugging certain situations, but could easily destroy performance – lots of blocking and deadlocks – if used in set-based operations!!*
- SQL Server does not allow for calling WAITFOR DELAY in any T-SQL function: Scalar, Inline TVF, or Multistatement TVF
- @MillisecondsToSleep
    - 1000 milliseconds == 1 second
- @NonDeterminiser
    - This parameter can, in most cases (especially non-set-based uses), be set to a constant such as 0x00. In set-based uses, it can be set to NEWID() which helps reduce the chances of the query optimizer caching the return value and not executing it each time as expected.
- If placed in a SELECT list or WHERE / JOIN clause it will run per each row in the result set. If you have a 1000 millisecond delay and there are 8 rows returned:
    - the total delay will be 8 seconds, but
    - the delay is spread out as each row is returned. There will be a 1 second delay between the actual processing of each row.
- *WARNING (repeat of top bullet point): Be VERY careful when using this function in Production code. Outside of debugging, there are very few (if any) scenarios in which you want to keep a Transaction open longer than necessary. This function is great for debugging certain situations, but could easily destroy performance – lots of blocking and deadlocks – if used in set-based operations!!*

EXAMPLES:

```
SELECT SQL#.DB_WaitForDelay(1000, NEWID())
FROM   msdb.sys.objects so
WHERE  so.[name] LIKE N'sysjob%'
-- 8 rows returned, taking just over 8 seconds


DECLARE @Dummy BIT;
SELECT @Dummy = SQL#.DB_WaitForDelay(1000, NEWID())
FROM   msdb.sys.objects so
WHERE  so.[name] LIKE N'sysjob%';
```

```
-- Same 8 second delay, but no result set


GO
-- Call via EXEC in a UDF to ignore the return value
CREATE FUNCTION dbo.TestDelay(@MilliSeconds INT)
RETURNS BIT
AS
BEGIN
       EXEC SQL#.DB_WaitForDelay @MilliSeconds, 0x00; -- 2nd param doesn't matter here
       RETURN 1;
END;
GO
```

Convert

Convert functions allow for transforming data into a different representative that can be converted back (unlike Hash functions).

Convert_Base2ToBase10

Convert_Base2ToBase10(@Base2Value NVARCHAR(64))

RETURNS: BIGINT

Converts a Base 2 value (a string of "0"s and "1"s) into the equivalent integer value.

NOTES:

- Converting either NULL or empty string '' will return NULL
- A @Base2Value containing characters other than 0 or 1 will *not* error; it will return a value based on the relative positions of any "1" characters, or it will return 0 if there are no "1" characters.
- Positive numbers:
    - 1 – 64 bits / characters can be passed in
    - Leading / left-most bits / characters not passed-in are assumed to be "0".
- Negative numbers:
    - Uses Two's Compliment (http://www.exploringbinary.com/twos-complement-converter/)
    - All 64 bits / characters must be passed in, with "1" as the leading / left-most digit
- Also see: Convert_Base10ToBase2

EXAMPLES:

```
SELECT SQL#.Convert_Base2ToBase10(N'0011');
-- 3

SELECT
SQL#.Convert_Base2ToBase10(N'0001000000000001000000000000000000000000000');
-- 549890031616

SELECT SQL#.Convert_Base2ToBase10(N'a1b');
-- 2

SELECT
SQL#.Convert_Base2ToBase10(N'111111111011111111101111111111110000000000000000000
0000000000000');
-- -18031994990493696
```

Convert_Base10ToBase2

Convert_Base10ToBase2(@Base10Value BIGINT)

RETURNS: NVARCHAR(64)

Converts an integer value into the equivalent Base 2 value (a string of "0"s and "1"s).

NOTES:

- Converting a NULL will return NULL

- Negative numbers use Two's Compliment (http://www.exploringbinary.com/twos-complement-converter/)
- Also see: Convert_Base2ToBase10

EXAMPLES:
```
SELECT SQL#.Convert_Base10ToBase2(3);
-- 0000000000000000000000000000000000000000000000000000000000000011

SELECT SQL#.Convert_Base10ToBase2(549890031616);
-- 0000000000000000000000001000000000001000000000000000000000000000

SELECT SQL#.Convert_Base10ToBase2(5764607523034234888); -- Bit #s: 4, 61, 63
-- 0101000000000000000000000000000000000000000000000000000000001000

SELECT SQL#.Convert_Base10ToBase2(-18031994990493696);
-- 1111111110111111110111111111111000000000000000000000000000000000
```

Convert_BinarySidToSddl

Convert_BinarySidToSddl(@UnencodedValue VARBINARY(100))

RETURNS: NVARCHAR(100)

NOTES:
- Security Identifiers (SIDs) have two forms: Binary and Security Descriptor Definition Language (SDDL).
- This function converts the Binary form into the SDDL form as shown below.
- Also see: Convert_SddlSidToBinary

EXAMPLES:
```
SELECT [sid], SQL#.Convert_BinarySidToSddl([sid]) AS [SDDL]
FROM   sys.server_principals
WHERE  [name] = N'NT Service\MSSQLSERVER';
-- 0x010600000000000550000000E20F4FE7B15874E48E19026478C2DC9AC307B83E
-- S-1-5-80-3880718306-3832830129-1677859214-2598158968-1052248003
```

Convert_BinaryToHexString

Convert_BinaryToHexString(BinaryValue VARBINARY(MAX))

RETURNS: NVARCHAR(MAX)

NOTES:
- Starting in SQL Server 2008, the CONVERT function can accomplish this same functionality. Use a "style" setting of 1 to include the "0x" on the left or a setting of 2 to not include the "0x", just as this SQL# function does:
  ```
  SELECT CONVERT(VARCHAR(50), 0x12A5, 2)
  ```

EXAMPLES:
```
SELECT SQL#.Convert_BinaryToHexString(0x48656c6c6f20576f726c6421)
-- 48656C6C6F20576F726C6421
```

Convert_DateTimeToMSIntDate

Convert_DateTimeToMSIntDate(RealDate DATETIME)

RETURNS: INT

NOTES:

- Same as: CONVERT(INT, DATEADD(HOUR, -12, @RealDate))
- Microsoft Int Date Epoch (Day 0) = 1900-01-01
- See also: Convert_MSIntDateToDateTime

EXAMPLES:

```
SELECT SQL#.Convert_DateTimeToMSIntDate('03/15/2010')
-- 40250
```

Convert_FromBase64

Convert_FromBase64(EncodedValue NVARCHAR(MAX))

RETURNS: VARBINARY(MAX)

EXAMPLES:

```
SELECT SQL#.Convert_FromBase64('SGVsbG8gV29ybGQh')
-- 0x48656C6C6F20576F726C6421
```

Convert_HexStringToBinary

Convert_HexStringToBinary(HexStringValue NVARCHAR(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Starting in SQL Server 2008, the CONVERT function can accomplish this same functionality. Use a “style” setting of 1 if the string has the “0x” on the left (the “0x” is required if using a “style” of 1) or a setting of 2 if it does not include the “0x”, just as this SQL# function does:
  ```
  SELECT CONVERT(VARBINARY, 12A5, 2)
  ```

EXAMPLES:

```
SELECT SQL#.Convert_HexStringToBinary('48656C6C6F20576F726C6421')
-- 0x48656C6C6F20576F726C6421
```

Convert_HtmlToXml

Convert_HtmlToXml(Document NVARCHAR(MAX), DocumentUri NVARCHAR(MAX), CaseFolding NVARCHAR(50))

RETURNS: NVARCHAR(MAX)

Converts HTML to well formed XML by adding missing quotes, empty attribute values, ignoring duplicate attributes, case folding on tag names, adding missing closing tags based on SGML DTD information, and so on.

NOTES:

- Document is any HTML text

- DocumentUri is the location of any HTML page
- CaseFolding:
    - Values are NOT case-sensitive
    - Values:
        - ToUpper – upper-cases all tags
        - ToLower – lower-cases all tags
        - {anything else} – doesn't change tag casing
- Sometimes requires having External Access permissions set on the SQL#.SgmlReader Assembly, especially if using DocumentUri:
  `EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.SgmlReader'`
- Either Document or DocumentUri needs to have a value.
- If both Document and DocumentUri have a value, DocumentUri will be used
- This is not a replacement for INET_GetWebPages as this function modifies the document being retrieved

EXAMPLES:
```
PRINT SQL#.Convert_HtmlToXml('
<Html>
<body>
<P class=MsoNormal dir=ltr
style="MARGIN: 0pt;" align=left><?xml:namespace
prefix = st1 ns = "urn:schemas-microsoft-com:office:smarttags"
/><ST1:PERSONNAME></ST1:PERSONNAME></P>
</body>
</html>
', NULL, 'ToLower')
/*
<html>
<body>
<p class="MsoNormal" dir="ltr" style="MARGIN: 0pt;" align="left"><?namespace
prefix = st1 ns = "urn:schemas-microsoft-com:office:smarttags"
?><st1:personname xmlns:st1="#unknown"></st1:personname></p>
</body>
</html>
*/
```

Convert_MSIntDateToDateTime

Convert_MSIntDateToDateTime (MSIntDate INT)

RETURNS: DATETIME

NOTES:
- Same as: CONVERT(DATETIME, @MSIntDate)
- Microsoft Int Date Epoch (Day 0) = 1900-01-01
- See also: Convert_DateTimeToMSIntDate

EXAMPLES:
```
SELECT SQL#.Convert_MSIntDateToDateTime(40250)
-- 2010-03-15 00:00:00.000
```

Convert_ROT13

Convert_ROT13(TextValue NVARCHAR(MAX))

RETURNS: NVARCHAR(MAX)

NOTES:

- ROT13 simply shifts the English alphabet characters 13 places and since there are 26 letters, applying it twice to the same string will bring everything back to where it started. Hence, the ROT13 algorithm decodes what it has already encoded.

EXAMPLES:
```
SELECT SQL#.Convert_ROT13('25) This is a test.')
-- 25) Guvf vf n grfg.
SELECT SQL#.Convert_ROT13('25) Guvf vf n grfg.')
-- 25) This is a test.
```

Convert_SddlSidToBinary

Convert_SddlSidToBinary(@UnencodedValue NVARCHAR(100))

RETURNS: VARBINARY(100)

NOTES:

- Security Identifiers (SIDs) have two forms: Binary and Security Descriptor Definition Language (SDDL).
- This function converts the SDDL form into the Binary form as shown below.
- Also see: Convert_BinarySidToSddl

EXAMPLES:
```
SELECT [sid], SQL#.Convert_BinarySidToSddl([sid]) AS [SDDL]
FROM   sys.server_principals
WHERE  [name] = N'NT Service\MSSQLSERVER';
-- 0x010600000000000550000000E20F4FE7B15874E48E19026478C2DC9AC307B83E
-- S-1-5-80-3880718306-3832830129-1677859214-2598158968-1052248003
```

Convert_ToBase64

Convert_ToBase64(UnencodedValue VARBINARY(MAX), Base64FormattingOption NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- Base64FormattingOption = InsertLineBreaks or None
- Base64FormattingOption is NOT case-sensitive

EXAMPLES:
```
SELECT SQL#.Convert_ToBase64(0x48656c6c6f20576f726c6421,
       'InsertLineBreaks')
-- SGVsbG8gV29ybGQh
```

Convert_UUDecode

Convert_UUDecode(EncodedValue NVARCHAR(MAX))

RETURNS: VARBINARY(MAX)

NOTES:

- Do not include the “begin” and “end” lines, just the actual encoded lines

EXAMPLES:

```
DECLARE      @UUSource VARBINARY(MAX),
             @UUEncoded NVARCHAR(MAX)
SET @UUSource = 0x325B2F99D4F578BD4207A84CE31200D3FF73
SET @UUEncoded = SQL#.Convert_UUEncode(@UUSource)
SELECT @UUEncoded, SQL#.Convert_UUDecode(@UUEncoded)
-- 2,ELOF=3U>+U"!ZA,XQ(`T_]S        0x325B2F99D4F578BD4207A84CE31200D3FF73
```

Convert_UUEncode

Convert_UUEncode(EncodedValue VARBINARY(MAX))

RETURNS: NVARCHAR(MAX)

NOTES:

- Does not include the “begin ### -” and “end” lines

EXAMPLES:

```
DECLARE      @UUSource VARBINARY(MAX),
             @UUEncoded NVARCHAR(MAX)
SET @UUSource = 0x325B2F99D4F578BD4207A84CE31200D3FF73
SET @UUEncoded = SQL#.Convert_UUEncode(@UUSource)
SELECT @UUEncoded, SQL#.Convert_UUDecode(@UUEncoded)
-- 2,ELOF=3U>+U"!ZA,XQ(`T_]S        0x325B2F99D4F578BD4207A84CE31200D3FF73
```

DB System Info (Not available in Free version)

Server-wide views of system objects.

Sys_AllAssemblies

Sys_AllAssemblies()

RETURNS: TABLE ([database_name] SYSNAME, [database_id] INT, [name] SYSNAME, [principal_id] INT, [assembly_id] INT, [clr_name] NVARCHAR(4000), [permission_set] TINYINT, [permission_set_desc] NVARCHAR(60), [is_visible] BIT, [create_date] DATETIME, [modify_date] DATETIME, [is_user_defined] BIT, [principal_name] SYSNAME, [principal_type] NVARCHAR(1), [principal_sid] VARBINARY(85))

Returns one row for each Assembly across all *accessible* Databases.

NOTES:

- The "**microsoft.sqlserver.types**" Assembly (`assembly_id` = 1) is only returned once in this TVF, residing in the "**[master]**" Database, even though it is returned for each Database when querying the "sys.assemblies" system catalog view. This is because just like Stored Procedures with names starting with "`sp_`", the object only exists in the "**[master]**" Database but appears to exist in all Databases.

EXAMPLES:

```
SELECT * FROM SQL#.Sys_AllAssemblies() ORDER BY [assembly_id];
```

Sys_AssemblyName

Sys_AssemblyName(DatabaseID INT, AssemblyID INT)

RETURNS: SYSNAME

Returns the name of the specified Assembly within the specified Database.

NOTES:

- Works similarly to the OBJECT_NAME() and OBJECT_SCHEMA_NAME() built-in functions in that you do not need to be in the database where the item exists in order to get the correct result
- When using one of the Microsoft CLR types (GEOMETRY, GEOGRAPHY, and HIERARCHYID), the Assembly does not show as loaded, but the AppDomain is created in the "**[master]**" Database
- For use with Dynamic Management objects that return the `[assembly_id]` but not the `[name]`, such as `sys.dm_clr_loaded_assemblies`

EXAMPLES:

```
SELECT  DB_NAME(dca.[db_id]) AS [DatabaseName],
        SQL#.Sys_AssemblyName(dca.[db_id], dcla.[assembly_id])
              AS [AssemblyName],
        dca.*,
        N' █ ' AS [ █ ],
        dcla.*
FROM    sys.dm_clr_appdomains dca
LEFT JOIN  sys.dm_clr_loaded_assemblies dcla
       ON  dca.[appdomain_address] = dcla.[appdomain_address]
WHERE dca.[db_id] <> 32767;
```

Sys_IndexName

Sys_IndexName(DatabaseID INT, ObjectID INT, IndexID INT)

RETURNS: SYSNAME

Returns the name of the specified Index on the specified Object within the specified Database.

NOTES:

- Works similarly to the OBJECT_NAME() and OBJECT_SCHEMA_NAME() built-in functions in that you do not need to be in the database where the item exists in order to get the correct result
- For use with Dynamic Management objects that return all three input values, such as `sys.dm_db_index_usage_stats`

EXAMPLES:

```
SELECT DB_NAME(stat.database_id) AS [DatabaseName],
       OBJECT_NAME(stat.[object_id], stat.database_id) AS [TableName],
       SQL#.Sys_IndexName(stat.database_id, stat.[object_id], stat.index_id)
            AS [IndexName],
       *
FROM   sys.dm_db_index_usage_stats stat;
```

Sys_LockResource

Sys_LockResource(ResourceType NVARCHAR(60), ResourceSubtype NVARCHAR(60), ResourceDatabaseID INT, ResourceDescription NVARCHAR(256), ResourceAssociatedEntityID BIGINT)

RETURNS: NVARCHAR(1000)

Returns the name of the specified Lock Resource given the Type, Subtype, DatabaseID, Description, and AssociatedEntityID.

NOTES:

- Returns name and some properties of the lock resource
- Works similarly to the OBJECT_NAME() and OBJECT_SCHEMA_NAME() built-in functions in that you do not need to be in the database where the item exists in order to get the correct result
- Currently not all lock resource types and subtypes can be translated. Hopefully over time additional types and subtypes will be able to be translated.
- Lock Resource Types handled: ALLOCATION_UNIT, COMPRESSED_ROW, DATABASE, FILE, FULLTEXT_INDEX, METADATA (ASSEMBLY, AUDIT, AUDIT_SPECIFICATION {Server and Database}, DATA_SPACE, DATABASE, DATABASE_PRINCIPAL, DB_MIRRORING_SESSION, INDEXSTATS, PASSWORD_POLICY, SCHEMA, SERVER_PRINCIPAL, STATS), OBJECT
- Lock Resource Types with partial support: HOBT, KEY, PAGE, RID
- For use with Dynamic Management objects that do not return the lock resource name, such as `sys.dm_tran_locks`

EXAMPLES:

```
SELECT tl.[request_session_id], db.[name] AS [DatabaseName],
       SQL#.Sys_LockResource(tl.[resource_type], tl.[resource_subtype],
            tl.[resource_database_id], tl.[resource_description],
            tl.[resource_associated_entity_id]) AS [LockResource],
       tl.[request_mode], tl.[request_type], tl.[request_status],
       tl.[resource_type], tl.[resource_subtype], tl.[resource_description],
       tl.[request_owner_id]
FROM      sys.dm_tran_locks tl
LEFT JOIN sys.databases db
       ON db.[database_id] = tl.[resource_database_id]
ORDER BY  tl.[request_session_id], db.[name], [LockResource];
```

Sys_Objects

Sys_Objects(DBNamePattern NVARCHAR(MAX), IncludeSystemDatabases BIT)

RETURNS: TABLE (database_name SYSNAME, database_id INT, name SYSNAME, object_id INT, principal_id INT, schema_id INT, parent_object_id INT, type NCHAR(2), type_desc NVARCHAR(60), create_date DATETIME, modify_date DATETIME, is_ms_shipped BIT, is_published BIT, is_schema_published BIT, schema_name SYSNAME, parent_name SYSNAME, parent_schema_id INT, parent_type NCHAR(2), parent_type_desc NVARCHAR(60), parent_schema_name SYSNAME)

Returns information from sys.objects from all databases matching the DBNamePattern with additional schema and parent_object information.

NOTES:

- DBNamePattern
  - is a full regular expression. If you want to match all databases, pass in empty string ''
  - is NOT case-sensitive
- IncludeSystemDatabases when set to 0 will exclude: master, model, msdb, tempdb, and any database that is a distributor (only on replication distributor nodes)

EXAMPLES:

```
SELECT * FROM SQL#.Sys_Objects('', 1) WHERE [type] IN ('p', 'pc');
-- return all procs (SQL and CLR) from ALL DBs, even system DBs

SELECT * FROM SQL#.Sys_Objects('^Customer|\d{4}$', 0);
-- all objects from DBs starting with "Customer" OR ending with 4 digits
```

XML (Not available in Free version)

Functions that operate on XML data.

XML_EscapeContent

XML_EscapeContent(StringValue NVARCHAR(MAX))

RETURNS: XML

Returns the StringValue with any XML-specific characters properly escaped.

NOTES:

- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.XML_EscapeContent(N'<a>Hello&amp;"Goodbye"?</a>')
-- &lt;a&gt;Hello&amp;amp;"Goodbye"?&lt;/a&gt;
```

XML_SaveToFile

XML_SaveToFile(FilePath NVARCHAR(4000), XmlData XML, ContentNewLineHandling TINYINT, XmlDeclarationHandling TINYINT, IndentNodes BIT, IndentWith NVARCHAR(4000), OneAttributePerLine BIT, CustomNewLineAfterNodes NVARCHAR(4000), AppendData BIT, FileEncoding NVARCHAR(30), TrapErrorsInline BIT)

RETURNS: NVARCHAR(4000)

Saves XML data to a file, including options for formatting and including a properly specified XML declaration.

NOTES:

- {will add details later}

XML_Transform

XML_Transform(SourceXML XML, SourceXMLPath NVARCHAR(4000), SourceXSL XML, SourceXSLPath NVARCHAR(4000), XSLTParameters SQL#.Type_HashTable, OutputFilePath NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

Returns the XML, supplied directly or from a file, transformed according to the XSL, supplied directly or from a file. It can be returned either directly or to a file. Optionally XSLT parameters can be supplied.

NOTES:

- XML must be supplied either directly via SourceXML or from a file via SourceXMLPath.
- If both SourceXML and SourceXMLPath are supplied, SourceXMLPath will be used.
- XSL must be supplied either directly via SourceXSL or from a file via SourceXSLPath.
- If both SourceXSL and SourceXSLPath are supplied, SourceXSLPath will be used.
- When not supplying XSLT Parameters, pass in NULL
- XSLT Parameters are useful for run-time substitutions, especially when getting XML and XSL from tables and/or files.
- If OutputFilePath is specified it will be used and NULL will be returned from the function
- If OutputFilePath is empty string or NULL, the transformed XML will be returned from the function

- See the following for some output options: http://www.w3schools.com/xsl/el_output.asp

EXAMPLES:
```
-- Run all of the following together, including the two SELECTs to see the difference
DECLARE @SourceXML XML, @SourceXSL XML
DECLARE @Params SQL#.Type_HashTable
SET @Params = @Params.AddItem('reportedby', 'Solomon')
SET @Params = @Params.AddItem('hourlyrate', '100')
SELECT @Params.Count;

SET @SourceXML = N'<!-- Project List -->
<projects>
  <project ID="123">
    <title>Website</title>
    <hours>23.50</hours>
  </project>
  <project ID="456">
    <title>Marketing Brochure</title>
    <hours>11.75</hours>
  </project>
</projects>
'

SET @SourceXSL = N'<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="reportedby"/>
  <xsl:param name="hourlyrate"/>
  <xsl:output omit-xml-declaration="no" encoding="windows-1251" method="text" />
  <xsl:template match="/">
    <invoice> MyCompany, Inc.
      <xsl:variable name="totalhours" select="sum(//hours)"/>
         Projects worked on: <xsl:value-of select="count(//project)"/>

         Hours worked:       <xsl:value-of select="$totalhours"/>
         Rate:               $<xsl:value-of select="$hourlyrate"/> / hr
         Total Due:          $ <xsl:value-of select="$totalhours*$hourlyrate"/>

      --------------
      Reported by: <xsl:value-of select="$reportedby"/>
     </invoice>
  </xsl:template>
</xsl:stylesheet>
'
SELECT SQL#.XML_Transform(@SourceXML, '', @SourceXSL, '', @Params, '')
SELECT SQL#.XML_Transform(@SourceXML, '', @SourceXSL, '', NULL, '')
```

```
MyCompany, Inc.

          Projects worked on: 2

          Hours worked:       35.25
          Rate:               $100 / hr
          Total Due:          $ 3525

      --------------
      Reported by: Solomon
```

XML_UnescapeContent

XML_UnescapeContent(XMLValue XML)

RETURNS: NVARCHAR(MAX)

Returns the XML, as NVARCHAR, with any XML-specific characters properly unescaped.

NOTES:

- NULL input returns NULL

EXAMPLES:

```
SELECT SQL#.XML_UnescapeContent(N'&lt;a&gt;Hello&amp;amp;"Goodbye"?&lt;/a&gt;')
-- <a>Hello&amp;"Goodbye"?</a>
SELECT SQL#.XML_UnescapeContent(N'&quot;Who Knew?&quot;')
-- "Who Knew?"
```

LookUp

LookUp functions provide commonly used static data that far too many applications duplicate in tables.

LookUp_GetCountryInfo

LookUp_GetCountryInfo(SearchCode NVARCHAR(4000))

RETURNS: TABLE (NumericCode NCHAR(3), TwoLetterCode NCHAR(2), ThreeLetterCode NCHAR(3), Name NVARCHAR(50), FlagImage VARBINARY(MAX))

Provides ISO-based information on countries. All countries can be returned at once (to create a drop-down list perhaps) or a single country's information can be returned based on the SearchCode passed in. There are 244 countries listed.

NOTES:

- SearchCode can be either the Numeric, TwoLetterCode, or ThreeLetterCode; if either Two or Three LetterCode, then it is NOT case-sensitive
- If SearchCode is empty '' or NULL then all Countries are returned
- By default data is sorted by Name field
- FlagImage column of result set is a PNG picture file of the flag for that country. This can be used on websites rather easily by streaming the binary data to a webpage that is used as the SRC of an IMG tag while changing the mime-type HTTP header to "image/png". If you would rather store the images on disk in actual png files, then you can export all of the flag images into separate files using the following SQL:

  ```
  EXEC SQL#.SQLSharp_SetSecurity 2

  SELECT SQL#.File_CreateDirectory('C:\SQL#CountryFlags')

  SELECT      *, SQL#.File_WriteFileBinary('C:\SQL#CountryFlags\' +
              cinfo.TwoLetterCode + '.png', cinfo.FlagImage,
              'Create', '')
  FROM        SQL#.LookUp_GetCountryInfo('') cinfo
  WHERE       cinfo.FlagImage IS NOT NULL
  ```
- The information is maintained by the ISO (International Standards Organization) and is subject to change, although not frequently. SQL# will be updated if / when it does change.

EXAMPLES:

```
SELECT * FROM SQL#.LookUp_GetCountryInfo('') -- get all Country Info
SELECT * FROM SQL#.LookUp_GetCountryInfo('008') -- get Info for Albania
SELECT * FROM SQL#.LookUp_GetCountryInfo('FI') -- get Info for Finland
SELECT * FROM SQL#.LookUp_GetCountryInfo('usa') -- get Info for US
SELECT * FROM SQL#.LookUp_GetCountryInfo('') ORDER BY TwoLetterCode
-- get all Country Info sorted by the TwoLetterCode
```

LookUp_GetStateInfo

LookUp_GetStateInfo(SearchCode NVARCHAR(4000), USStatesOnly BIT)

RETURNS: TABLE (NumericCode NCHAR(2), TwoLetterCode NCHAR(2), Name NVARCHAR(50), FlagImage VARBINARY(MAX), CountryCode NCHAR(2))

Provides US Postal Service-based information on states and territories. All states can be returned at once (to create a drop-down list perhaps) or a single state's information can be returned based on the SearchCode passed in. The list can also be filtered to show only actual US states. There are 82 states and territories listed.

NOTES:

- SearchCode can be either the Numeric or TwoLetterCode; if TwoLetterCode, then is NOT case-sensitive
- If SearchCode is empty '' or NULL then all States are returned, unless USStatesOnly is True / 1 then all US States are returned
- The numeric code is the FIPS (Federal Information Processing Standard) code, used by various US Govnerment departments
- If USStatesOnly is set to True / 1 then Washington, D.C. is also returned
- By default data is sorted by Name field
- FlagImage column of result set is reserved for future use
- The information is maintained by the United States Postal Service and is subject to change, although not frequently. SQL# will be updated if / when it does change.

EXAMPLES:

```
SELECT * FROM SQL#.LookUp_GetStateInfo('', '') -- get all State Info
SELECT * FROM SQL#.LookUp_GetStateInfo('', 'US') -- get all US States
SELECT * FROM SQL#.LookUp_GetStateInfo('AS', '') -- get only 1 state
SELECT * FROM SQL#.LookUp_GetStateInfo('AS', 'US')
-- returns nothing since 'AS' is not a US state
```

Operating System

The **OS** functions reside in the SQL#.OS assembly.

If you use any of the functions that access the Operating System, then this assembly will need a security setting of EXTERNAL_ACCESS (2), unless indicated otherwise. You can set this by executing the following:

```
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.OS';
```

If you do not want to have this assembly in your system at all, you can do either of the following:

1) Do not install the SQL#.OS assembly by setting the `@InstallSQL#OS` variable (towards the top of the script) to 0 before installing
2) Uninstall the assembly by running:
   ```
   EXEC [SQL#].[SQLsharp_Uninstall] N'SQL#.OS';
   ```

OS_EventLogRead

OS_EventLogRead(LogName NVARCHAR(4000), MachineName NVARCHAR(4000), Source NVARCHAR(4000), EntryType NVARCHAR(4000), InstanceID NVARCHAR(4000), Category NVARCHAR(4000), UserName NVARCHAR(4000), Message NVARCHAR(4000), TimeGeneratedBegin DATETIME, TimeGeneratedEnd DATETIME, IndexBegin INT, IndexEnd INT, RegExOptionsList NVARCHAR(4000))

RETURNS: TABLE (Index INT, Category NVARCHAR(500), EntryType NVARCHAR(50), InstanceId BIGINT, Source NVARCHAR(500), TimeGenerated DATETIME, TimeWritten DATETIME, UserName NVARCHAR(100), Message NVARCHAR(4000), Data VARBINARY(MAX))

NOTES:

- LogName:
  - Needs to be a valid Event Log name such as: System, Application, or Security.
  - Is not case-sensitive.
  - Can also be the Event Log filename, such as: OSession for “Microsoft Office Sessions”
- MachineName:
  - can be set to NULL or empty string ‘’ to mean the local machine
- Source:
  - Is a Regular Expression controlled by RegExOptionsList
- EntryType:
  - Is a Regular Expression controlled by RegExOptionsList
  - Valid Entry Type are: Error, Information, Warning, Failure Audit, and Success Audit
- InstanceID:
  - Underlying value is an INT
  - Parameter is a Regular Expression controlled by RegExOptionsList so that you have more control over what number(s) to filter on.
- Category:
  - Is a Regular Expression controlled by RegExOptionsList
- UserName:
  - Is a Regular Expression controlled by RegExOptionsList
- Message:
  - Is a Regular Expression controlled by RegExOptionsList
- TimeGeneratedBegin:
  - The minimum time in the result set or starting time
  - Set to NULL to mean “no minimum” and to pull from the beginning
- TimeGeneratedEnd:

    - The maximum time in the result set or ending time
    - Set to NULL to mean “no maximum” and to pull until the end
- IndexBegin:
    - The minimum Index number in the result set or starting Index
    - Set to NULL or 0 to mean “no minimum” and to pull from the beginning
- IndexEnd:
    - The maximum Index number in the result set or ending Index
    - Set to NULL or 0 to mean “no maximum” and to pull until the end
- RegExOptionsList:
    - Please see Introduction to Regular Expressions Functions for details.

EXAMPLES:
```
-- read all Events from the System log
SELECT * FROM SQL#.OS_EventLogRead('System', '', '', '', '', '', '', '', NULL,
NULL, 0, 0, '')

-- read only Error and Warning Events from the Application log,
-- ignoring the case of the EventTypes
SELECT * FROM SQL#.OS_EventLogRead('Application', '', '', '(error|warning)', '',
'', '', '', NULL, NULL, 0, 0, 'ignorecase')

-- read all Events from the Application log that came from SQL Server,
-- starting on May 1st, 2009 at 15:30 (or 3:30 PM)
SELECT * FROM SQL#.OS_EventLogRead('Application', '', 'MSSQLSERVER', '', '', '',
'', '', '05/01/2009 15:30', NULL, 0, 0, '')
```

OS_EventLogWrite

OS_EventLogWrite(LogName NVARCHAR(4000), MachineName NVARCHAR(4000), Source NVARCHAR(4000), EntryType NVARCHAR(4000), InstanceID INT, Category SMALLINT, Message NVARCHAR(4000), BinaryData VARBINARY(8000))

RETURNS: NVARCHAR(4000)

NOTES:
- LogName must be a valid Event Log on the system, either Event Log name (e.g. System or Application) or Event Log Filename (e.g. OSession for OSession.evt)
- MachineName can be set to NULL or empty string ‘’ to mean local machine
- Source can be an existing Source for the Event Log or a new one that will be created the first time it is used in the Event Log. Please keep in mind that a Source can only exist in one Event Log so if you create “MyApp” in “Application” then you cannot use “MyApp” as a Source in “System” or any other Even Log. If you try to use a Source that has already been created in another Event Log you will get an error.
- Entry Type can be: Error, Information, Warning, Audit Failure, or Audit Success
- Entry Type is not case-sensitive
- Category is any value you choose
- Message cannot be NULL
- BinaryData can be set to NULL
- Always returns empty string ‘’
- Designed as a Function instead of a Procedure so that it can be used in set-based operations

EXAMPLES:
```
-- write an Informational message to the Application Log with
-- a Source of SQL# and no BinaryData
```

```
SELECT SQL#.OS_EventLogWrite('Application', '', 'SQL#', 'Information', 123, 1,
'Test message', NULL)

-- write a Warning message to the Microsoft Office Sessions Log with
-- a Source of "Microsoft Office 12 Sessions" and some BinaryData
SELECT SQL#.OS_EventLogWrite('OSession', '', 'Microsoft Office 12 Sessions',
'Warning', 7000, -1, 'Test warning', 0x5B327AC4)

-- example of logging a set of errors
CREATE TABLE #TempErrors (Error VARCHAR(20) NOT NULL, BinaryData
      VARBINARY(50))
INSERT INTO #TempErrors VALUES ('error uno', 0xF5D932993B)
INSERT INTO #TempErrors VALUES ('error dos', NULL)
INSERT INTO #TempErrors VALUES ('error tres', 0x53514C2320697320636F6F6C)

DECLARE @DevNull CHAR(1)
SELECT @DevNull = SQL#.OS_EventLogWrite('Application', '', 'SQL#',
      'Error', 12342, 55, err.Error, err.BinaryData)
FROM #TempErrors err
```

OS_GenerateTone

OS_GenerateTone(Frequency INT, Duration INT)

RETURNS: NVARCHAR(4000)

NOTES:

- Requires security level 3 (unrestricted)
- Frequency is between 37 and 32767 hertz
- Duration is in milliseconds
- Always returns empty string ''
- Designed as a Function instead of a Procedure so that it can be used in set-based operations
- Volume is controlled only through the "PC Speaker" volume control and is not affected by the master volume outside of the "mute" function.
- Hearing the tone requires that the service account running the SQL Server process can interact with the desktop. This ability is present when running on older operating systems (such as Windows XP), or when using SQL Server Express LocalDB (which runs as a background User process, not as a service). To hear a tone running a non-LocalDB edition on a newer OS, the service account might need to be granted a permission via Local Group Policy.

EXAMPLES:

```
SELECT  SQL#.OS_GenerateTone(90, 1000);

CREATE TABLE #tones (freq INT, dur INT);
INSERT INTO #tones VALUES (300, 200);
INSERT INTO #tones VALUES (90, 200);
INSERT INTO #tones VALUES (100, 300);
INSERT INTO #tones VALUES (300, 200);
INSERT INTO #tones VALUES (600, 400);
INSERT INTO #tones VALUES (100, 300);

DECLARE     @DevNull NCHAR(1);
SELECT      @DevNull = SQL#.OS_GenerateTone(freq, dur)
FROM        #tones;
```

OS_GetEnvironmentVariable (Not available in Free version)

OS_GetEnvironmentVariable(VariableName NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- {will add details later}

OS_GetEnvironmentVariables (Not available in Free version)

OS_GetEnvironmentVariables()

RETURNS: TABLE (VariableName NVARCHAR(4000), VariableValue NVARCHAR(MAX))

NOTES:

- {will add details later}

OS_GetSystemEnvironmentVariable (Not available in Free version)

OS_GetSystemEnvironmentVariable(VariableName NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- {will add details later}

OS_GetSystemEnvironmentVariables (Not available in Free version)

OS_GetSystemEnvironmentVariables()

RETURNS: TABLE (VariableName NVARCHAR(4000), VariableValue NVARCHAR(MAX))

NOTES:

- {will add details later}

OS_GetUserEnvironmentVariable (Not available in Free version)

OS_GetSystemEnvironmentVariable(VariableName NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- {will add details later}

OS_GetUserEnvironmentVariables (Not available in Free version)

OS_GetUserEnvironmentVariables()

RETURNS: TABLE (VariableName NVARCHAR(4000), VariableValue NVARCHAR(MAX))

NOTES:

- {will add details later}

OS_MachineName

OS_MachineName()

RETURNS: NVARCHAR(4000)

Returns the Computer name of the machine that SQL Server is running on.

NOTES:

- This Function only requires EXTERNAL_ACCESS (2) permissions
- This should be equivalent to the T-SQL function: SERVERPROPERTY('MachineName')

OS_PerfCounterAddData (Not available in Free version)

OS_PerfCounterAddData (CategoryName NVARCHAR(100), CounterName NVARCHAR(100), InstanceName NVARCHAR(100), CounterValue BIGINT)

RETURNS: BIT

NOTES:

- {will add details later}

OS_PerfCounterCreateCategory (Not available in Free version)

OS_PerfCounterAddData (CategoryName NVARCHAR(100), CounterName NVARCHAR(100), InstanceName NVARCHAR(100), IsMultiInstance BIT)

RETURNS: BIT

NOTES:

- {will add details later}

OS_PerfCounterGetSample (Not available in Free version)

OS_PerfCounterGetSample(CategoryName NVARCHAR(100), CounterName NVARCHAR(100), InstanceName NVARCHAR(100))

RETURNS: TABLE (…)

NOTES:

- {will add details later}

OS_PerfCounterGetValue (Not available in Free version)

OS_PerfCounterGetValue (CategoryName NVARCHAR(100), CounterName NVARCHAR(100), InstanceName NVARCHAR(100), SecondsBetweenSamples INT)

RETURNS: FLOAT

NOTES:

- {will add details later}

OS_ProcessGetInfo (Not available in Free version)

OS_ProcessGetInfo(ProcessIDs NVARCHAR(4000))

RETURNS: TABLE (ProcessID INT, ProcessName NVARCHAR(1000), StartTime DATETIME, MainModule NVARCHAR(1000), HandleCount INT, NonPagedSystemMemorySize BIGINT, PagedSystemMemorySize BIGINT, PrivateMemorySize BIGINT, PagedMemorySize BIGINT, VirtualMemorySize BIGINT, PhysicalMemorySize BIGINT, PeakPagedMemorySize BIGINT, PeakVirtualMemorySize BIGINT, PeakPhysicalMemorySize BIGINT, PrivilegedProcessorTime BIGINT, UserProcessorTime FLOAT, TotalProcessorTime FLOAT, Responding BIT)

Gets a list of information for the specified ProcessIDs

NOTES:

- ProcessIDs is a comma-separated list of Process IDs
- You can find general ProcessIDs by going to Task Manager, selecting the View menu, selecting the “Select Columns...” sub-menu, and checking the box for “PID (Process Identifier)”.
- You cannot see information on Processes owned by “SYSTEM” but you should be able to see info for “LOCAL SERVICE” and “NETWORK SERVICE” Processes as well as any Process started by the account running the “SQL Server” Process.
- If you are not allowed to see the Process information the “MainModule” field will display: Access is denied.
- If a Process is taking too long and you notice that the “Responding” field is set to 0, consider using OS_ProcessKill

EXAMPLES:

```
SELECT * FROM SQL#.OS_ProcessGetInfo('2232,2724,0,1632,3648,1356')
```

OS_ProcessKill (Not available in Free version)

OS_ProcessKill(ProcessID INT, ProcessName NVARCHAR(4000))

RETURNS: NVARCHAR(4000)

Kills a process started by OS_ProcessStart.

NOTES:

- ProcessID is the ID of the Process as returned by OS_ProcessStart
- ProcessName is the name of the program running under the ProcessID and is a safe-guard to make sure you do not kill another Process with the same ID. This is just in case the Process you wanted to kill ended and another one started with the same ProcessID (even if that is unlikely to happen).
- If the program started by ProcessStart is a .BAT or .CMD script, then use "CMD" as ProcessName
- ProcessName is NOT case-sensitive
- Return string is a success or error message
- Designed as a Function instead of a Procedure so that it can be used in set-based operations
- ProcessID must be owned / started by the account that runs the SQL Server process; you are not able to kill Processes started by other users or even the main "SQL Server" Process
- ProcessID must be a processes started by the OS_ProcessStart function; you cannot kill the main SQL Server process even though the same user account started that process

EXAMPLES:
```
SELECT SQL#.OS_ProcessKill(1234, 'NotePad');
```

OS_ProcessStart (Not available in Free version)

OS_ProcessStart(FileName NVARCHAR(4000), Arguments NVARCHAR(4000), WorkingDirectory NVARCHAR(4000))

RETURNS: INT

Runs the command specified by FilePath like xp_cmdshell but does so asynchronously so that control returns immediately and proceeds to the next T-SQL command rather than waiting for the Process to complete. Because the process is running separately from the SQL Session, no output from the command is returned unlike with xp_cmdshell.

Notes:

- FilePath can be a full path to a command or a relative path or just a command / program name if it can be found in the PATH environment variable
- Arguments can be NULL, empty string '', or any set of command-line parameters
- If WorkingDirectory is set to NULL or empty string '' it might default to C:\Windows\System32 so it is best to set this value
- ProcessID return value can be used with both OS_ProcessGetInfo and OS_ProcessKill
- Permissions for the Process / Command should be same as Login running the "SQL Server" Process
- Can be used to call DTExec, OSQL, SQLCMD, etc.

EXAMPLES:
```
-- NotePad will not be visible so should not be used normally
-- but works as an example
DECLARE @ProcessID INT
SELECT @ProcessID = SQL#.OS_ProcessStart('NotePad', NULL, 'C:\');
SELECT * FROM SQL#.OS_ProcessGetInfo(CONVERT(NVARCHAR(20), @ProcessID));
SELECT SQL#.OS_ProcessKill(@ProcessID, 'NotePad');
```

OS_ServiceAccount

OS_ServiceAccount()

RETURNS: NVARCHAR(4000)

Returns the name of the Windows account running the SQL Server process.

OS_StartTime (Not available in Free version)

OS_StartTime()

RETURNS: DATETIME

Returns the Date and Time of when the machine was started. This is more consistent than inferring from:

```
DATEADD(MILLISECOND, (SQL#.OS_Uptime() * -1), GETDATE())
```

OS_Uptime

OS_Uptime()

RETURNS: INT

Returns the number of milliseconds since the system started.

Twitter

Twitter functions allow you to get and send message on Twitter.com via simple T-SQL commands. The following assemblies need to be installed in order to use the **Twitter** functions: SQL#, SQL#.JsonFx, SQL#.Twitterizer, and SQL#.TypesAndAggregates.

All Twitter Functions, because they use the Internet, require a security setting of EXTERNAL_ACCESS (2). You can set this by executing the following query:

```
EXEC SQL#.SQLsharp_SetSecurity 2, 'SQL#.Twitterizer'
```

Be sure to note that Twitter.com does enforce “rate limits” and will not allow over a certain amount of calls per hour. Please see http://apiwiki.twitter.com/FAQ for more information regarding “rate limits”, as well as the Twitter Rate Limit Chart.

IMPORTANT: Please see the SQL# Twitter setup guide for details on how to set up your Twitter Application:
https://SQLsharp.com/download/SQLsharp_TwitterSetup.pdf

If you do not want to have this Assembly in your system at all, you can do either of the following:

3) Do not install the SQL#.Twitterizer assembly by setting the `@InstallSQL#Twitterizer` variable (towards the top of the script) to 0 before installing
4) Uninstall the assembly by running:
   `EXEC [SQL#].[SQLsharp_Uninstall] N'SQL#.Twitterizer'`

If you want to use any of the Optional Twitter Parameters, do the following to set the value of the @OptionalParameters input parameter:

```
DECLARE @Params SQL#.Type_HashTable
SET @Params = @Params.AddItem('count', '50');
```

OR, pass in the values directly, formatted as a URL query string (`N'var1=val1[&var2=val2[&...]]'`):

```
N'count=50'
```

Please note, invalid Unicode escape sequences, such as \ud8c3, will display as a question mark (?) since there is no way to translate them.

Twitter_BlockUser

Twitter_BlockUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(4000) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Blocks a user for the authenticating user.

NOTES:

- ScreenName is the user to block

- Returns the blocked user's info
- If you try to Block a user that is already in the authenticating user's "Blocks" list, you will NOT get an error

Twitter_CreateFavorite

Twitter_CreateFavorite(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(4000), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Marks a Status as "Favorite".

Twitter_DestroyDirectMessage

Twitter_DestroyDirectMessage(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), MessageID BIGINT)

RETURNS: NVARCHAR(4000)

Permanently deletes a Twitter direct message.

NOTES:

- Always returns empty string ''
- Designed as a Function instead of a Procedure so that it can be used in set-based operations

Twitter_DestroyFavorite

Twitter_DestroyFavorite(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(4000), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Un-marks a Status as "Favorite".

Twitter_DestroyStatus

Twitter_DestroyStatus(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: NVARCHAR(4000)

Permanently deletes a Twitter Status.

NOTES:

- Always returns empty string ''

- Designed as a Function instead of a Procedure so that it can be used in set-based operations
- UserName must be the author of the message
- If you try to Destroy a message that has already been Destroyed, you will get an exception stating "(404) Not Found"
- There is a time-lag between calling Destroy and the message being deleted so it might show up on GetUserTimeLine for a few minutes after the Destroy

Twitter_FollowUser

Twitter_FollowUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Follows a user for the authenticating user.

NOTES:

- ScreenName is the user to follow
- Returns the followed user's info
- If you try to Follow a user that is already in the authenticating user's "Friends" list, you will get the following error:
  "`<error>Could not follow user: XXXXXXX is already on your list.</error> --->`
  `System.Net.WebException: The remote server returned an error: (403) Forbidden.`"

Twitter_GetBlocks

Twitter_GetBlocks(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Returns the users that the authenticating user has blocked.

Twitter_GetFavorites

Twitter_GetFavorites(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName

NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the top 20 statuses marked as “favorite” by the authenticating user or the User specified in the OptionalParameters.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
    - user_id = The ID of the user for whom to request a list of favorite statuses
    - screen_name = The screen name of the user for whom to request a list of favorite statuses
    - count = The number of results to retrieve. Default = 20 and cannot be over 200.
    - since_id = Returns results with an ID greater than (that is, more recent than) the specified ID
    - max_id = Returns results with an ID less than (that is, older than) or equal to the specified ID.

Twitter_GetFollowers

Twitter_GetFollowers(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Returns the followers of the authenticating user.

Twitter_GetFriends

Twitter_GetFriends(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Returns the friends of the authenticating user.

Twitter_GetHomeTimeline

Twitter_GetHomeTimeline(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the 20 most recent statuses, including retweets if they exist, posted by the authenticating user and the user's they follow. This is the same timeline seen by a user when they login to twitter.com. This method is identical to statuses/friends_timeline, except that this method always includes retweets. This method is can only return up to 800 statuses, including retweets.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
    - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the *since_id*, the *since_id* will be forced to the oldest ID available.
    - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.
    - **count** = The number of records to retrieve. Must be <= 200. Default = 20.
    - **exclude_replies** = This parameter will prevent replies from appearing in the returned timeline. Using *exclude_replies* with the *count* parameter will mean you will receive up-to *count* tweets — this is because the *count* parameter retrieves that many tweets before filtering out retweets and replies..

Twitter_GetMentions

Twitter_GetMentions(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), RateLimit INT, PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the 20 most recent mentions (status containing @username) for the authenticating user.
The timeline returned is the equivalent of the one seen when you view your mentions on twitter.com.
This method can only return up to 800 statuses.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
    - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the *since_id*, the *since_id* will be forced to the oldest ID available.
    - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.

- **count** = Specifies the number of tweets to try and retrieve, up to a maximum of 200. The value of *count* is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the *count* has been applied.

Twitter_GetMessages

Twitter_GetMessages(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the 20 most recent Direct Messages sent to the authenticating user.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
  - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
  - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.
  - **count** = The number of records to retrieve. Must be less than or equal to 200.
  - **page** = the page of results to retrieve.

Twitter_GetMutes

Twitter_GetMutes(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Returns a list of users that the authenticating user has muted.

Twitter_GetRetweetedBy

Twitter_GetRetweetedBy(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Show user objects of up to 100 members who retweeted the status.

Twitter_GetRetweets

Twitter_GetRetweets(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT, OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns up to 100 of the first retweets of a given tweet.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
  - **count** = The number of records to retrieve. Must be less than or equal to 100.

Twitter_GetRetweetsOfMe

Twitter_GetRetweetsOfMe(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the 20 most recent tweets of the authenticated user that have recently been retweeted by others.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
  - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
  - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.

- **count** = The number of records to retrieve. Must be less than or equal to 100.

Twitter_GetSentMessages

Twitter_GetSentMessages(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the 20 most recent Direct Messages sent by the authenticating user. You can request up to 200 direct messages per call, up to a maximum of 800 outgoing DMs.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
  - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
  - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.
  - **count** = The number of records to retrieve. Must be less than or equal to 200.
  - **page** = the page of results to retrieve.

Twitter_GetStatus

Twitter_GetStatus(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns the specified status.

Twitter_GetUser

Twitter_GetUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), UserIDorScreenName NVARCHAR(20))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Returns the specified user.

Twitter_GetUserTimeline

Twitter_GetUserTimeline(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns a collection of the most recent Tweets posted by the user indicated by the *screen_name* or *user_id* parameters. User timelines belonging to protected users may only be requested when the authenticated user either "owns" the timeline or is an approved follower of the owner. The timeline returned is the equivalent of the one seen when you view a user's profile on twitter.com. This method can only return up to 3,200 of a user's most recent Tweets.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
    - **user_id** = The ID of the user for whom to return results for.
    - **screen_name** = The screen name of the user for whom to return results for.
    - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the *since_id*, the *since_id* will be forced to the oldest ID available.
    - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.
    - **count** = Specifies the number of tweets to try and retrieve, up to a maximum of 200 per distinct request. The value of *count* is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the *count* has been applied. We include retweets in the *count*, even if *include_rts* is not supplied.
    - **exclude_replies** = This parameter will prevent replies from appearing in the returned timeline. Using *exclude_replies* with the *count* parameter will mean you will receive up-to *count* tweets — this is because the *count* parameter retrieves that many tweets before filtering out retweets and replies.
    - **include_rts** = When set to *false*, the timeline will strip any native retweets (though they will still count toward both the maximal length of the timeline and the slice selected by the *count* parameter).

EXAMPLE:

```
DECLARE        @ConsumerKey NVARCHAR(100),
               @ConsumerSecret NVARCHAR(100),
               @AccessToken NVARCHAR(100),
               @AccessTokenSecret NVARCHAR(100)

SELECT         @ConsumerKey = 'aaaaaaaaaaa',
            @ConsumerSecret = 'bbbbbbbbbbb',
            @AccessToken = '9999999-ccccccccccc',
            @AccessTokenSecret = 'ddddddddddddddd'

-- Get Timeline for authenticating user
SELECT * FROM SQL#.Twitter_GetUserTimeline(@ConsumerKey, @ConsumerSecret,
@AccessToken, @AccessTokenSecret, NULL)

DECLARE @Params SQL#.Type_HashTable
SET @Params = @Params.AddItem('screen_name', 'sqlsharp')
SET @Params = @Params.AddItem('count', '100')

-- Get Timeline for @sqlsharp
SELECT * FROM SQL#.Twitter_GetUserTimeline(@ConsumerKey, @ConsumerSecret,
@AccessToken, @AccessTokenSecret, @Params)
```

Twitter_MuteUser

Twitter_MuteUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20), UserID BIGINT)

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

Mutes a user for the authenticating user.

NOTES:

- ScreenName OR UserID is the user to un-follow
- ScreenName OR UserID can be NULL, but not both at the same time
- Returns the muted user's info
- If you Mute a user that is currently in the authenticating user's "Mutes" list, you will get that same user's info returned; it will not error.

Twitter_Retweet

Twitter_Retweet(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), StatusID BIGINT)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Retweets a tweet.

Twitter_SearchTweets (Not available in Free version)

Twitter_SearchTweets(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), SearchQuery NVARCHAR(500), OptionalParameters Type_HashTable)

RETURNS: TABLE (StatusID BIGINT, Created DATETIME, InReplyToStatusID BIGINT, InReplyToUserID BIGINT, IsFavorited BIT, IsTruncated BIT, Source NVARCHAR(100), StatusText NVARCHAR(300), RecipientID INT, TimeZone NVARCHAR(100), ScreenName NVARCHAR(100), UserName NVARCHAR(100), UserID BIGINT, Location NVARCHAR(100), PlaceID NVARCHAR(50), PlaceName NVARCHAR(500), PlaceFullName NVARCHAR(500), PlaceType NVARCHAR(500), PlaceCountry NVARCHAR(500), PlaceLatitude FLOAT, PlaceLongitude FLOAT, RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME)

Returns a collection (default = 15) of relevant Tweets matching a specified query. Please note that Twitter's search service is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

NOTES:

- See beginning of Twitter section for example of how to set OptionalParameters
- Optional Parameters ARE case-sensitive!
- Optional Parameters:
    - **geocode** = Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by "latitude,longitude,radius", where radius units must be specified as either "mi" (miles) or "km" (kilometers) (i.e. 5mi).
    - **lang** = Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort.
    - **locale** = Specify the language of the query you are sending (only "*ja*" is currently effective). This is intended for language-specific consumers and the default should work in the majority of cases.
    - **result_type** = Specifies what type of search results you would prefer to receive. The current default is "*mixed*". Valid values include:
        - *mixed*: Include both popular and real time results in the response.
        - *recent*: return only the most recent results in the response
        - *popular*: return only the most popular results in the response.
    - **count** = The number of tweets to return. Must be less than or equal to 100. Default = 15.
    - **until** = Returns tweets generated before the given date. Date should be formatted as YYYY-MM-DD. Keep in mind that the search index may not go back as far as the date you specify here.
    - **since_id** = Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.
    - **max_id** = Returns results with an ID less than (that is, older than) or equal to the specified ID.
- See the following Twitter page for details on using the Search facility:
    - The Search API: https://dev.twitter.com/rest/public/search
    - Working with Timelines: https://dev.twitter.com/rest/public/timelines

Twitter_SendDirectMessage

Twitter_SendDirectMessage(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), Message NVARCHAR(140), Recipient NVARCHAR(20))

RETURNS: BIGINT

Sends a Direct Message (private) to the Recipient from the authenticating user and returns the StatusID of the new message.

Twitter_UnBlockUser

Twitter_UnBlockUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20))

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

UnBlocks a user for the authenticating user.

NOTES:
- ScreenName is the user to UnBlock
- Returns the unblocked user's info
- If you try to UnBlock a user that is not in the authenticating user's "Blocks" list, you will NOT get an error

Twitter_UnFollowUser

Twitter_UnFollowUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20), UserID BIGINT)

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

UnFollows a user for the authenticating user.

NOTES:
- ScreenName OR UserID is the user to un-follow
- ScreenName OR UserID can be NULL, but not both at the same time
- Returns the un-followed user's info
- If you UnFollow a user that is not currently in the authenticating user's "Friends" list, you will get the following error:
  "`<error>You are not friends with the specified user.</error> ---> System.Net.WebException: The remote server returned an error: (403) Forbidden.`"

Twitter_UnMuteUser

Twitter_UnMuteUser(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100), ScreenName NVARCHAR(20), UserID BIGINT)

RETURNS: TABLE (UserID BIGINT, ScreenName NVARCHAR(100), UserName NVARCHAR(100), IsProtected BIT, IsVerified BIT, Description NVARCHAR(4000), CreatedOn DATETIME, Location NVARCHAR(500), TimeZone NVARCHAR(100), UTCOffset INT, ProfileImageUri NVARCHAR(2048), ProfileUri NVARCHAR(2048), FriendsCount INT, NumberOfFollowers INT, NumberOfStatuses INT, StatusText NVARCHAR(300) , RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, Language NVARCHAR(50), NumberOfPublicListMemberships INT, IsGeoEnabled BIT, Following BIT, Muting BIT)

UnMutes a user for the authenticating user.

NOTES:

- ScreenName OR UserID is the user to un-follow
- ScreenName OR UserID can be NULL, but not both at the same time
- Returns the un-muted user's info
- If you UnMute a user that is not currently in the authenticating user's "Mutes" list, you will get an empty result set returned, but not an error.

Twitter_Update

Twitter_Update(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100) , Message NVARCHAR(140), InReplyToStatusID BIGINT, Latitude FLOAT, Longitude FLOAT)

RETURNS: BIGINT

Posts a new Status message for the authenticating user and returns the StatusID of the new message.

NOTES:

- InReplyToStatusID is optional. Pass in NULL if the Update is not a reply.
- InReplyToStatusID will be ignored unless the author of the Status this parameter references is @replied within the Status text. Therefore, you must start the Status with @username, where username is the author of the referenced Status
- Latitude and Longitude should both have a value OR both be NULL
- This Function is subject to update limits. A HTTP 403 will be returned if this limit as been hit.
- Twitter will ignore attempts to perform a duplicate Update. With each Update attempt the application compares the update text with the authenticating user's last successful update and ignores any attempts that would result in duplication. Therefore, a user cannot submit the same Status twice in a row. A duplicate submission will return the StatusID from the previously successful Update if a duplicate has been silently ignored.

Twitter_xAuth

Twitter_xAuth(ConsumerKey NVARCHAR(100), ConsumerSecret NVARCHAR(100), UserName NVARCHAR(100), Password NVARCHAR(100))

RETURNS: TABLE (AccessToken NVARCHAR(100), AccessTokenSecret NVARCHAR(100))

Uses xAuth to translate a UserName and Password into the AccessToken and AccessTokenSecret for the specified Application (as identified by the ConsumerKey and ConsumerSecret).

NOTES:

- This will only work if your Application has been granted access to xAuth by Twitter. Most users / Applications will not need this. If you do need this (to pass in many UserNames and Passwords) then you need to contact Twitter to request xAuth permission at: api@twitter.com

---

Running Totals (Not available in Free version)

Running Totals provides a way for you to quickly and easily add a "running total" field to any query. It is flexible so it is also possible to add multiple Running Total fields so that you can get per-group totals as well as a total for the entire query, or multiple per-group totals and none for the entire query. You can also pre-seed a value (such as an "Opening Balance") and you can still have the value at the end without having to re-run the query. These functions can run in SAFE mode. Be sure to look at the ClearCache function as that will need to be run periodically to clear the memory, although each Running Total (per query) takes up about 100 bytes of memory (plus whatever, if anything, is used for ResetIndicator). If 1000 queries with one RunningTotal run before clearing the cache, that will only take up 100 KB. Running Totals can be shared within a Session (SPID) but not between them.

RunningTotal_Add

RunningTotal_Add(IdentificationLabel NVARCHAR(50), TheValue FLOAT, ResetIndicator NVARCHAR(4000))

RETURNS: FLOAT

Adds TheValue to a variable that starts at 0 and persists between each row of the result set.

NOTES:

- IdentificationLabel:
  - This is only needed if you either:
    - Have more than one Running Total in a query
    - Need to either pre-seed a value OR need the value once the query is done
  - If left blank, a new RunningTotal entry will be created for each run of the query AND it will not be able to be shared across queries
  - If used, be careful to not use a static value as that can be shared between queries in the same SPID or between queries that reuse a SPID. It is safest to create a UNIQUEIDENTIFIER using NEWID() and store that in a variable. This will ensure that the value is unique to that run of the Batch / Stored Proc.
  - See also: RunningTotal_Get
- TheValue:
  - Can be + or –
- ResetIndicator:
  - If the value of ResetIndicator ever changes from one row to the next, the stored value will be reset to zero (0).
  - This should be used only if wanting to show a running total within a grouping
  - If used, it can be set to any field in the query that would change between groups, such as a group ID or name
  - If not used, set to empty string ''
- **At some point you MUST run RunningTotal_ClearCache to clear out the memory or else it will just keep building up (at least until the SQL Server process is restarted)**

EXAMPLES:

```
SELECT ints.IntVal, SQL#.RunningTotal_Add('', ints.IntVal, '') AS [Total]
FROM SQL#.Util_GenerateInts(0, 11, 1) ints
ORDER BY ints.IntVal ASC
----------
DECLARE @RunningTotalID UNIQUEIDENTIFIER, @Dummy FLOAT
SET @RunningTotalID = NEWID()
SET @Dummy = SQL#.RunningTotal_Add(@RunningTotalID, 5, '') -- preset value

SELECT ints.IntVal, SQL#.RunningTotal_Add(@RunningTotalID, ints.IntVal, '') AS
[Total]
FROM SQL#.Util_GenerateInts(0, 11, 1) ints
ORDER BY ints.IntVal ASC
----------
SELECT ints.IntVal, (ints.IntVal /3) AS [Grouping], SQL#.RunningTotal_Add('',
ints.IntVal, (ints.IntVal /3)) AS [Total]
FROM SQL#.Util_GenerateInts(0, 11, 1) ints
ORDER BY ints.IntVal ASC
```

RunningTotal_CacheSize

RunningTotal_CacheSize(SPID INT)

RETURNS: INT

Gets the current size (in bytes) of the memory used by the Running Total cache.

NOTES:

- Pass in @@SPID to get the memory used by the current session only
- Pass in 0 to get the total cache size across ALL sessions

EXAMPLE:

```
SELECT SQL#.RunningTotal_CacheSize(@@SPID) -- 92
SELECT SQL#.RunningTotal_CacheSize(0) -- 213
```

RunningTotal_ClearCache

RunningTotal_ClearCache(MinutesSinceLastAccess INT)

RETURNS: INT

Removes entries from the Running Total cache that have not been accessed (updated or read) within the specified amount of minutes.

NOTES:

- Pass in 0 to clear all values
- Typicallyt, this command should be scheduled via a SQL Agent Job that runs every 30 – 60 minutes and passes in a value of 30 – 60
- Return value is the number of Running Totals that were removed from the cache
- This command MUST be run occassionaly in order to reduce the amount of memory taken up by the Running Total cache as it will remain in memory until this command is called or the SQL Server service is restarted.

EXAMPLE:

```
SELECT SQL#.RunningTotal_ClearCache(30)
```

RunningTotal_Get

RunningTotal_Get(IdentificationLabel NVARCHAR(50))

RETURNS: FLOAT

Retrieves the value denoted by IdentificaionLabel from RunningTotal cache

NOTES:

- This only works if an IdentificationLabel was used when creating the Running Total
- See also: RunningTotal_Add

EXAMPLE:

```
DECLARE @RunningTotalID UNIQUEIDENTIFIER
SET @RunningTotalID = NEWID()

SELECT ints.IntVal, SQL#.RunningTotal_Add(@RunningTotalID, ints.IntVal, '') AS
[Total]
FROM SQL#.Util_GenerateInts(0, 11, 1) ints
ORDER BY ints.IntVal ASC

SELECT SQL#.RunningTotal_Get(@RunningTotalID)
```

User-Defined Aggregates

Creating User-Defined Aggregates started in SQL Server 2005. They act just like standard T-SQL Aggregates (SUM, MIN, MAX, AVG, COUNT) and work over groups of data, typically grouped together via a GROUP BY.

User-Defined Aggregates reside in the [SQL#.TypesAndAggregates] and [SQL#.TypesAndAggregatesPlus] assemblies. The [SQL#.TypesAndAggregates] assembly is required by the following assemblies: [SQL#.Twitterizer] and [SQL#.Network]. This assembly should be able to remain in SAFE mode even if assemblies that require it are set to EXTERNAL_ACCESS or UNRESTRICTED.

Agg_BitwiseAND (Not available in Free version)

Agg_BitwiseAND(BIGINT)

RETURNS: BIGINT

NOTES:

- Returns the result of doing a bitwise AND operation on all values in each group
- T-SQL bitwise AND operator = “&”
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count

EXAMPLES:

```
SELECT 16 & 30 & 24, (16 & 30) & 24, 16 & (30 & 24) -- 16

SELECT tmp.GroupID,
       SQL#.Agg_BitwiseAND(tmp.BitVal) AS [AND]
FROM (
       SELECT 1, 16
       UNION ALL
       SELECT 1, 30
       UNION ALL
       SELECT 1, 24
       UNION ALL
       SELECT 2, NULL
       UNION ALL
       SELECT 3, 23
       ) tmp (GroupID, BitVal)
GROUP BY tmp.GroupID;

GroupID      AND
1            16
2            NULL
3            23
```

Agg_BitwiseOR (Not available in Free version)

Agg_BitwiseOR(BIGINT)

RETURNS: BIGINT

NOTES:

- Returns the result of doing a bitwise OR operation on all values in each group
- T-SQL bitwise OR operator = “|”

- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count

EXAMPLES:

```
SELECT 16 | 30 | 24, (16 | 30) | 24, 16 | (30 | 24) -- 30

SELECT tmp.GroupID,
       SQL#.Agg_BitwiseOR(tmp.BitVal) AS [OR]
FROM (
       SELECT 1, 16
       UNION ALL
       SELECT 1, 30
       UNION ALL
       SELECT 1, 24
       UNION ALL
       SELECT 2, NULL
       UNION ALL
       SELECT 3, 23
       ) tmp (GroupID, BitVal)
GROUP BY tmp.GroupID;

GroupID      OR
1            30
2            NULL
3            23
```

Agg_BitwiseXOR (Not available in Free version)

Agg_BitwiseXOR(BIGINT)

RETURNS: BIGINT

NOTES:

- Returns the result of doing a bitwise XOR (eXclusive OR) operation on all values in each group
- T-SQL bitwise XOR operator = “^”
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count

EXAMPLES:

```
SELECT 16 ^ 30 ^ 24, (16 ^ 30) ^ 24, 16 ^ (30 ^ 24) -- 22

SELECT tmp.GroupID,
       SQL#.Agg_BitwiseXOR(tmp.BitVal) AS [XOR]
FROM (
       SELECT 1, 16
       UNION ALL
       SELECT 1, 30
       UNION ALL
       SELECT 1, 24
       UNION ALL
       SELECT 2, NULL
       UNION ALL
       SELECT 3, 23
       ) tmp (GroupID, BitVal)
GROUP BY tmp.GroupID;

GroupID      XOR
1            22
```

```
2           NULL
3           23
```

Agg_GeometricAvg

Agg_GeometricAvg(FLOAT)

RETURNS: FLOAT

NOTES:

- Returns a geometric average PER GROUP, just like AVG, SUM, etc.
- Formula = $(Val_1 * Val_2 * Val_3 * Val_n)^{1/n}$
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count
- For more info, please see: http://en.wikipedia.org/wiki/Geometric_mean

EXAMPLES:

```
SELECT SQL#.Agg_GeometricAvg(test.col)
FROM (
      SELECT 2 AS 'col'
      UNION ALL
      SELECT 3
      UNION ALL
      SELECT NULL
      UNION ALL
      SELECT 4
) test
-- 2.88449914061482

-- same as:
-- (2 * 3 * 4) = 24; POWER(24.0, (1.0/3.0))
SELECT POWER(CONVERT(FLOAT, 24), (CONVERT(FLOAT, 1)/3))
-- 2.88449914061482
```

Agg_HarmonicMean (Not available in Free version)

Agg_HarmonicMean(Value FLOAT)

RETURNS: FLOAT

NOTES:

- Returns the Harmonic Mean PER GROUP, just like AVG, SUM, etc.
- Formula = $n / (1/X_1 + 1/X_2 + ...1/X_n)$
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count
- For more info, please see: http://en.wikipedia.org/wiki/Harmonic_mean

EXAMPLES:

```
SELECT (1/1 + 1/2.0 + 1/4.0), 3.0 / (1/1 + 1/2.0 + 1/4.0), 12.0/7.0
-- 1.750000          1.714285714285714285 71            1.714285

SELECT tmp.GroupID, SQL#.Agg_HarmonicMean(tmp.val) AS [HarmonicMean]
FROM (
      SELECT 1.00, 1 -- Group 1 is same 3 values shown above to check against
      UNION ALL
      SELECT 2, 1
      UNION ALL
```

```
        SELECT 4, 1
        UNION ALL
        SELECT 72, 2 -- Group 2 is a single value
        UNION ALL
        SELECT 4, 3 -- Group 3 has a NULL and a value
        UNION ALL
        SELECT NULL, 3
        UNION ALL
        SELECT NULL, 4 -- Group 4 is just a NULL
        ) tmp (val, GroupID)
GROUP BY tmp.GroupID;

GroupID       HarmonicMean
1             1.71428571428571
2             72
3             4
4             NULL
```

Agg_Join

Agg_Join(value NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- Produces a comma-separated list of values in the group
- This accomplishes the same thing as String_Join() but over a group rather than over various rows
- Unlike String_Join(), this Agg_Join() does not have the option to ignore empty value
- Duplicate values are included, but NULL values are ignored
- Returns NULL only if all rows in the group are NULL
- Order of items is not guaranteed
- *For SQL Server 2005 only*: Datasets that are over 8000 characters when combined (and including however many commas are needed to separate the values) will work to varying degrees based on how well the data compresses in memory.
- SQL Server 2017 added this functionality via the STRING_AGG() built-in function.
    - STRING_AGG can order the ouput via the "WITHIN GROUP" clause and set a custom delimiter, but cannot have the "DISTINCT" keyword applied to "value"
    - Agg_Join can use the "DISTINCT" keyword (e.g. `Agg_Join(DISTINCT expression)`) but cannot order the output, nor set a custom delimiter.

EXAMPLES:

```
SELECT SQL#.Agg_Join(ROUTINE_NAME) FROM INFORMATION_SCHEMA.ROUTINES
-- File_GetTempPath,File_PathExists,File_WriteFile,File_GetFile,...
```

Agg_JoinDelim

Agg_JoinDelim(Value NVARCHAR(4000), Delimiter NVARCHAR(4000))

RETURNS: NVARCHAR(MAX)

NOTES:

- **This is NOT available on SQL Server 2005 instances!**
- Produces a delimited list of values in the group
- Delimiter can be:

    - empty string
    - NULL (is treated as being empty string)
    - multiple characters
- This accomplishes the same thing as String_Join() but over a group rather than over various rows
- Unlike String_Join(), Agg_JoinDelim() does not have the option to ignore empty value
- Duplicate values are included, but NULL values are ignored.
- Returns NULL only if all rows in the group are NULL.
- Order of items is not guaranteed.
- SQL Server 2017 added this functionality via the STRING_AGG() built-in function.
    - STRING_AGG can order the ouput via the "WITHIN GROUP" clause, but cannot have the "DISTINCT" keyword applied to "value"
    - Agg_JoinDelim can use the "DISTINCT" keyword (e.g. `Agg_JoinDelim(DISTINCT expression, N'delimiter')`) but cannot order the output.

EXAMPLES:
```
SELECT tmp.[Col1], COUNT(*) AS [Count],
       SQL#.Agg_JoinDelim(tmp.[Col2], N'~!~') AS [ConcatenatedValues]
FROM   (VALUES (1, 'a'), (1, 'b'), (2, 'asdas'), (2,'ddfg'), (2, 'hhhhhh'),
               (2, 'z'), (5, 'GH'), (5, 'HHFR'), (3, 'LLL'), (5, 'zz'), (4, NULL),
               (5, NULL)) tmp ([Col1], [Col2])
GROUP BY tmp.[Col1];
/*
Col1    Count    ConcatenatedValues
1       2        b~!~a
2       4        asdas~!~ddfg~!~hhhhhh~!~z
3       1        LLL
4       1        NULL
5       4        zz~!~HHFR~!~GH
*/
```

Agg_JoinPlus (Not available in Free version)

Agg_JoinPlus(Value NVARCHAR(4000), Delimiter NVARCHAR(4000), OrderBy NVARCHAR(4000), Ordering TINYINT, InitialDelimiter NVARCHAR(4000), MinRecordsNeededForInitialDelimiter INT, FinalDelimiter NVARCHAR(4000), MinRecordsNeededForFinalDelimiter INT, NullReplacement NVARCHAR(4000), RemoveEmptyEntries BIT, DuplicateHandling TINYINT, UseCompression BIT)

RETURNS: NVARCHAR(MAX)

NOTES:
- **This is NOT available on SQL Server 2005 instances!**
- This UDA is located in the [SQL#.TypesAndAggregatesPlus] assembly (*which will not be present when installing into a SQL Server 2005 instance*)
- Produces a delimited list of values in the group
- Like Agg_Join, this operation is similiar to String_Join but over a group rather than over various rows.
- Unlike Agg_Join, this "Plus" version allows for customization:
    - Custom delimiter
    - If specified, custom first and/or last delimiter
    - Optional sort by any arbitrary expression
    - Sorting can be case-sensitive or insensitive, and either Ascending or Descending
    - Option to replace NULLs with an actual value, possibly empty string '' (NULLs are otherwise excluded)
    - Option to remove empty entries (empty = empty string '')
    - Option to remove duplicates, and if duplicates are case-sensitive or insensitive
    - Option to use compression (uses less memory but more CPU) to fine-tune operation

- Value NVARCHAR(4000)
  - The string value to be “join”ed
  - Please note that the length is 4000 and not MAX. If you want to concatenate strings that are longer than 4000 characters, using an aggregate function would not be a good approach anyway as values are stored in memory while the “group” is being determined and processed
- Delimiter NVARCHAR(4000)
  - Default value (if set to NULL) = N','
- OrderBy NVARCHAR(4000)
  - Default value (if set to NULL) = {none}
  - Can be set to any valid exression resulting in an NVARCHAR of no more than 4000 chars
  - If you are ordering numerically, then due to this being a string field, you will need to left pad with zeros in order to get the desired sorting:
    ```
    RIGHT(N'00000000' + CONVERT(NVARCHAR(10), col.SomeNumericDataTypeField), 8)
    ```
- Ordering TINYINT
  - default = 0 (i.e. no ordering);
  - values:
    - 1 = Case Insenstive ASC
    - 2 = Case Insenstive DESC
    - 3 = Case Sensitive ASC
    - 4 = Case Senstitive DESC
- InitialDelimiter NVARCHAR(4000)
  - Optional delimiter to be used to separate only the first two items
  - Example with ‘: ’ as InitialDelimeter: “First: Second,Third,Fourth”
  - If used, items after the first two, except for possibly the last two, are separated by @Delimiter
  - Default value (if set to NULL) = empty string / ''
- MinRecordsNeededForInitialDelimiter INT
  - How many records in the group (to be joined) need to exist before @InitialDelimiter, if specified, will be used.
  - A value of less than 2 indicates that an Initial Delimiter will not be used, even if a value is provided for @InitialDelimiter
  - Default value (if set to NULL) = 0
- FinalDelimiter NVARCHAR(4000)
  - Optional delimiter to be used to separate only the last two items
  - Example with ‘ & ’ as FinalDelimeter: “First,Second,Third & Fourth”
  - If used, items before the last two, except for possibly the first two, are separated by @Delimiter
  - Default value (if set to NULL) = empty string / ''
- MinRecordsNeededForFinalDelimiter INT
  - How many records in the group (to be joined) need to exist before @FinalDelimiter, if specified, will be used.
  - A value of less than 2 indicates that a Final Delimiter will not be used, even if a value is provided for @FinalDelimiter
  - Default value (if set to NULL) = 0
- NullReplacement NVARCHAR(4000)
  - NULL entries are filtered out as they cannot be represented in a string
  - This option allows for replacing NULL entries so that they don’t get filtered out
  - NULL replacement occurs *before* removing duplicates and/or removing empty entries, if either is enabled.
  - Default value (if set to NULL) = {no replacement} (i.e. keep as NULL and get filtered out)
- RemoveEmptyEntries BIT
  - If enabled, removes any string that is empty / ‘’
  - No trimming is performed so a value that has a single space is not considered empty. If you need trimming then wrap the expression in an LTRIM()
  - Removal of empty entries, if enabled, occurs *after* Null Replacement but *before* removing duplicates.

        - Default value (if set to NULL) = 0 = {no removal} (i.e. empty string are allowed)
- DuplicateHandling TINYINT
    - If not set to 0, removes duplicates as determined by the value specified for this option
    - Values:
        - 0 = none / disabled / allow duplicates
        - 1 = Remove Case Insensitive duplicates (“A”, “A”, “a”, “a” -> “A” or “a”)
        - 2 = Remove Case Sensitive duplicates (“A”, “A”, “a”, “a” -> “A”, “a”)
    - Default value (if set to NULL) = 0 = {none} (i.e. allow duplicates)
- UseCompression BIT
    - Determines whether or not to compress the data stored in memory as rows are being processed within the groups.
    - Compression was originally added to assist prior to SQL Server 2008 since SQL Server 2005 only allowed for 8000 bytes of memory and hence needed to fit more into that space.
    - Compression is less of an issue starting with SQL Server 2008 as the memory space is now essentially VARBINARY(MAX) instead of VARBINARY(8000)
    - If joined strings are somewhat small and/or memory is plentiful, then try setting to 0 (i.e. disabling).
    - When joining larger strings, or large sets of small to medium size strings and/or memory is less plentiful than extra CPU cycles, then keep as NULL or set to 1 (i.e. enable)
    - Default value (if set to NULL) = 1 (i.e. use compression; err on the side of preserving RAM)

EXAMPLES:
```
-- Get tables with column list (columns order by Ordinal Position)
SELECT
    tab.TABLE_SCHEMA,
    tab.TABLE_NAME,
    SQL#.Agg_JoinPlus(col.COLUMN_NAME, N', ',
         RIGHT(N'00' + CONVERT(NVARCHAR(5), col.ORDINAL_POSITION), 2), 1,
         NULL, NULL, NULL, NULL,
         NULL, 0, 0, NULL) AS [ColumnsInTable]
FROM [msdb].INFORMATION_SCHEMA.TABLES tab
INNER JOIN [msdb].INFORMATION_SCHEMA.COLUMNS col
        ON tab.TABLE_NAME = col.TABLE_NAME
       AND tab.TABLE_SCHEMA = col.TABLE_SCHEMA
WHERE tab.TABLE_TYPE = N'BASE TABLE'
GROUP BY  tab.TABLE_SCHEMA, tab.TABLE_NAME
ORDER BY tab.TABLE_SCHEMA ASC, tab.TABLE_NAME ASC;
```

```
TABLE_SCHEMA  TABLE_NAME                          ColumnsInTable
dbo           log_shipping_monitor_alert          alert_job_id

dbo           log_shipping_monitor_error_detail   agent_id, agent_type, session_id,
database_name, sequence_number, log_time, log_time_utc, message, source, help_url

dbo           log_shipping_monitor_history_detail   agent_id, agent_type, session_id,
database_name, session_status, log_time, log_time_utc, message
```

Agg_Median

Agg_Median(FLOAT)

RETURNS: FLOAT

NOTES:

- Returns the middle value (or average of the two middle values in an even-numbered grouping) PER GROUP, just like COUNT, etc.
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count
- *For SQL Server 2005 only*: Datasets that are over 8000 characters when combined (and including however many commas are needed to separate the values) will work to varying degrees based on how well the data compresses in memory. Natively max number of values is 999 but with compression can be several thousand depending on the values and what order they are in

EXAMPLES:
```
SELECT SQL#.Agg_Median(test.col)
FROM (
      SELECT 2 AS 'col'
      UNION ALL
      SELECT 3
      UNION ALL
      SELECT NULL
      UNION ALL
      SELECT 100
) test
-- 3

SELECT SQL#.Agg_Median(test.col)
FROM (
      SELECT 2 AS 'col'
      UNION ALL
      SELECT 3
      UNION ALL
      SELECT 76
      UNION ALL
      SELECT 100
) test
-- 39.5
```

Agg_Random

Agg_Random(FLOAT)

RETURNS: FLOAT

NOTES:
- Returns a random value from within the grouping
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count
- Uses compression to work over a greater amount of values than is possible natively. Natively max number of values is 999 but with compression can be several thousand depending on the values and what order they are in
- If you use more than once in a single statement, all uses of Agg_Random will pick from the same random row in the set; it will not mix and match random values from different rows

EXAMPLES:
```
SELECT SQL#.Agg_Random(tab.ValOne), SQL#.Agg_Random(tab.ValTwo)
FROM (
            SELECT 1 AS 'ValOne', 100 AS 'ValTwo'
            UNION ALL
            SELECT 2, 200
            UNION ALL
            SELECT 2, 200
            UNION ALL
            SELECT NULL, NULL
            UNION ALL
            SELECT 3, 300
      ) tab
```

Agg_RootMeanSqr

Agg_RootMeanSqr(FLOAT)

RETURNS: FLOAT

NOTES:

- Returns the Root Mean Square (RMS) PER GROUP, just like AVG, SUM, etc.
- Formula = SQRT(($X_1^2 + X_2^2 + X_3^2 + X_n^2$) / n)
- Like the SUM and AVG aggregates, NULL values are ignored but duplicates count

EXAMPLES:

```
SELECT SQL#.Agg_RootMeanSqr(test.col)
FROM (
      SELECT 2 AS 'col'
      UNION ALL
      SELECT 3
      UNION ALL
      SELECT NULL
      UNION ALL
      SELECT 10
) test
-- 6.13731754650732

-- same as:
-- (POWER(2, 2) + POWER(3, 2) + POWER(10, 2)) = 113; SQRT(113 / 3)
SELECT SQRT(CONVERT(FLOAT, 113) / 3)
-- 6.13731754650732
```

User-Defined Types

Creating User-Defined Types started in SQL Server 2005. They act just like standard T-SQL datatypes: they can be used for local variables, they can be used as parameters for Stored Procedures and User-Defined Functions, and they can even be used as columns in tables to be persisted. Regarding persisting in a table, however, it is advised that if this functionality is desired then to instead persist the ToString() output as that can be used as direct input to initialize each Type. The reason for not wanting to persist these in actual tables is the fact that the CLR assembly that contains the definition for the persisted User-Defined Type is then required to exist until the User-Defined Type's are no longer in use (i.e. persisted). This would have the effect of making it impossible to upgrade SQL# or whatever other CLR Assembly holds the persisted type. However, these User-Defined Types are perfect for use with Temp Tables and even Table Variables. The main value in these User-Defined Types is the ability to work with sets of data that are not a Temp Table that has an IDENTITY column that many people use to move away from Cursors. Also, they provide a very easy mechanism for transferring sets of data between Stored-Procedures or User-Defined Functions that is currently only possible with Temp Tables, but Temp Tables need to exist physically, even if only temporarily, in TempDB which can cause additional I/O contention that can lead to blocking or slowing down of other queries or processes on the server that require disk I/O.

User-Defined Types reside in the SQL#.TypesAndAggregates assembly. This assembly is required by the following assemblies: SQL#.Twitterizer and SQL#.Network. This assembly should be able to remain in SAFE mode even if assemblies that require it are set to EXTERNAL_ACCESS or UNRESTRICTED.

Type_FloatArray

DECLARE @Variable SQL#.Type_FloatArray

CONSTRUCTOR:

- Comma-separated list of numbers (INT or FLOAT)
- Spaces are trimmed on both sides before converting values to real numbers
- SET @Type_FloatArrayVariable = ''
- SET @Type_FloatArrayVariable = '1,34,34,98,453'
- SET @Type_FloatArrayVariable = '.02342 ,   5675.4564'

PROPERTIES:

- Count
  The number of items in the Array

METHODS:

- AddData(@Index INT, @InputStrings NVARCHAR(4000))
  RETURNS: Type_FloatArray
  Adds one or more FLOATS, separated by commas (like the Constructor)
  @Index is where to insert the new values
      @Index = 0 will ADD to the end of the Array
      @Index > 1 will INSERT at the @Index point
      @Index > FloatArray.Count or < 0 will cause an error

- Avg()
  RETURNS: FLOAT
  Returns the average of all of the items
  zero-value items do count

- Clear()
  RETURNS: Type_FloatArray
  Removes ALL items from the Array, leaving the Count = 0

- ContainsItem(@SearchFloat FLOAT)
  RETURNS: BIT
  Will return 1 (true) if the @SearchFloat is found anywhere in the array

- GetAt(@Index INT)
  RETURNS: FLOAT
  Returns the value found at the @Index
  @Index < 0 will cause an error
  @Index > FloatArray.Count returns NULL

- IndexOfItem(@SearchFloat, @Index INT)
  RETURNS: INT
  Returns the Index value that matches the first occurrence of @SearchFloat starting at @Index
  If the @SearchFloat value is not found, 0 is returned
  @Index < 1 or > FloatArray.Count will cause an error

- Median()
  RETURNS: FLOAT
  Returns the middle value or the average of the two middle values in an even-numbered array
  0 values do count

- RemoveAt(@Index INT)
  RETURNS: Type_FloatArray
  Removes the value at @Index
  @Index < 1 or > FloatArray.Count will cause an error

- RemoveItem(@InputFloat FLOAT)
  RETURNS: Type_FloatArray
  Removes the first occurance of @InputFloat found in the array

- RemoveRange(@Index INT, @Count INT)
  RETURNS: Type_FloatArray
  Removes @Count number of items from the array starting at @Index
  If @Index + @Count > FloatArray.Count an error will occur

- Reverse()
  RETURNS: Type_FloatArray
  Reverses the order of the items in the array
  This is in essence a DESCending sort by index, not by value

- Sort()
  RETURNS: Type_FloatArray
  This does an ASCending sort of the values by value, not by index
  If you want a DESCending sort of the values, call Reverse() after Sort()

- Sum()
  RETURNS: FLOAT
  Returns a sum of the values

- ToString()
  RETURNS: NVARCHAR(4000)

Returns a comma-separated list of the FLOAT values
If wanting to persist the value of the FloatArray, store the ToString() value and then use that value with the Constructor or AddData().

NOTES:

- Is a 1-based indexed array of FLOATs
- Must be initialized with constructor (=) before using (at least set to = '')
- NULLs (or empty strings or empty values between commas) are NOT allowed
- Property and Method names ARE case-sensitive: Array.Avg() <> Array.AVG()
- Uses compression to work over a greater amount of values than is possible natively. Natively max number of values is 999 but with compression can be several thousand depending on the values and what order they are in

EXAMPLE #1:

```
DECLARE @ArrayVar SQL#.Type_FloatArray
SET @ArrayVar = '100,2,67,3, 13.333 ,-5,55.25'

SELECT @ArrayVar.Count, @ArrayVar.Avg(), @ArrayVar.Median()
-- 7  33.6547142857143  13.333

SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3), @ArrayVar.Sum()
-- 100,2,67,3,13.333,-5,55.25 67    235.583

SET @ArrayVar = @ArrayVar.Reverse()
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3)
-- 55.25,-5,13.333,3,67,2,100 13.333

SET @ArrayVar = @ArrayVar.Sort()
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3), @ArrayVar.Count
-- -5,2,3,13.333,55.25,67,100 3     7

SET @ArrayVar = @ArrayVar.RemoveRange(3,2)
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3), @ArrayVar.Count
-- -5,2,3,67,100   3     5

SET @ArrayVar = @ArrayVar.AddData(0, '98,2,0.0023')
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3), @ArrayVar.Count
-- -5,2,3,67,100,98,2,0.0023   3     8

SET @ArrayVar = @ArrayVar.AddData(3, '101')
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(3), @ArrayVar.Count
-- -5,2,101,3,67,100,98,2,0.0023    101   9

SET @ArrayVar = @ArrayVar.RemoveAt(5)
SELECT @ArrayVar.ToString(), @ArrayVar.GetAt(6), @ArrayVar.Count
-- -5,2,101,3,100,98,2,0.0023 98    8

SET @ArrayVar = @ArrayVar.RemoveItem(100)
SELECT @ArrayVar.ToString(), @ArrayVar.ContainsItem(101),
@ArrayVar.ContainsItem(5)
-- -5,2,101,3,98,2,0.0023     1     0

SELECT @ArrayVar.IndexOfItem(2,1), @ArrayVar.IndexOfItem(2,4),
@ArrayVar.IndexOfItem(2,7)
-- 2  6     0
```

```
SET @ArrayVar = @ArrayVar.Clear()
SELECT '*' + @ArrayVar.ToString() + '*', @ArrayVar.Count
-- ** 0
```

EXAMPLE #2:

```
DECLARE                 @Customers  SQL#.Type_FloatArray,
                        @Index            INT

SET @Customers = SQL#.String_Join('SELECT TOP 10 CONVERT(VARCHAR, ContactID)
FROM AdventureWorks.Person.Contact', ',', 1)
SET @Index = 1

WHILE (@Index <= @Customers.Count)
BEGIN
      PRINT 'Working on CustomerID: ' + CONVERT(VARCHAR,
@Customers.GetAt(@Index))
      /* EXEC Schema.Proc @Customers.GetAt(@Index) */

      SET @Index = @Index + 1
END

PRINT '    -- or --'

SET @Customers = @Customers.Sort() -- just to show sorting; WHILE loop works the
same
WHILE (@Customers.Count > 0)
BEGIN
      PRINT 'Working on CustomerID: ' + CONVERT(VARCHAR, @Customers.GetAt(1))
      /* EXEC Schema.Proc @Customers.GetAt(1) */

      SET @Customers = @Customers.RemoveAt(1)
END

/*
Working on CustomerID: 15696
Working on CustomerID: 11706
Working on CustomerID: 10590
Working on CustomerID: 9998
Working on CustomerID: 337
Working on CustomerID: 11483
Working on CustomerID: 2695
Working on CustomerID: 4144
Working on CustomerID: 10970
Working on CustomerID: 16201
   -- or --
Working on CustomerID: 337
Working on CustomerID: 2695
Working on CustomerID: 4144
Working on CustomerID: 9998
Working on CustomerID: 10590
Working on CustomerID: 10970
Working on CustomerID: 11483
Working on CustomerID: 11706
Working on CustomerID: 15696
Working on CustomerID: 16201
```

```
*/
```

Type_HashTable

DECLARE @Variable SQL#.Type_HashTable

CONSTRUCTOR:

- Ampersand (&)-separated list of Key=Value pairs OR empty string (‘’)
- Both Key and Value are NVARCHAR(4000)
- Accepts URI encoded (i.e. percent-encoded) sequences for both Key and Value (mainly to allow for escaping "&" and "=" characters)
- EXAMPLES:

```
SET @Type_HashTableVar = '';
SET @Type_HashTableVar = 'NC=North Carolina';
SET @Type_HashTableVar = N'City=Chicago&State=IL&Zipcode=60647';
SET @Type_HashTableVar = N'cartoon=Tom %26 Jerry'; -- %26 = "&" character
```

PROPERTIES:

- **Count**
  The number of items in the Array

- **ValuesDataLength**
  The total number of bytes of all of the “Values” combined

- **ValuesLength**
  The number of 2-byte characters across all “Values” in the HashTable.

METHODS:

- **AddData(**@InputPairs NVARCHAR(4000)**)**
  RETURNS: Type_HashTable
  Adds one or more Key=Value pairs, separated by ampersands (&) (like the Constructor).
  Accepts URI encoded (i.e. percent-encoded) sequences for both Key and Value (mainly to allow for escaping "&" and "=" characters).

- **AddItem(**@InputKey NVARCHAR(4000), @InputValue NVARCHAR(4000)**)**
  RETURNS: Type_HashTable
  Adds a single Key/Value pair. Unlike AddData(), this method / function allows for passing in ampersands (&) and equal signs (=) in the InputKey or InputValue data without needing to use URI encoding.

- **Clear()**
  RETURNS: Type_HashTable
  Removes ALL items from the HashTable, leaving the Count = 0

- **ContainsKey(**@SearchString NVARCHAR(4000)**)**
  RETURNS: BIT
  Returns 1 (true) if @SearchString is found at all amongst the Keys
  @SearchString only matches whole-word and is case-sensitive

- **ContainsValue(**@SearchString NVARCHAR(4000)**)**
  RETURNS: BIT
  Returns 1 (true) if @SearchString is found at all amongst the Values

@SearchString only matches whole-word and is case-sensitive

- **GetTableXML(**@RootName NVARCHAR(4000), @UseElementsInsteadOfAttributes NVARCHAR(4000)**)**
  RETURNS: XML
  - Returns XML document of key-value pairs.
  - If @RootName is NULL then return value will be NULL
  - If @RootName is empty string then return value will contain root element of "<Root>"
  - If @RootName is 'someName' then return value will contain root element of '<someName>'
  - If @UseElementsInsteadOfAttributes = 0 then key-values pairs will be formatted as: `<Item Key="key" Value="val" />`
  - If @UseElementsInsteadOfAttributes = 1 then key-values pairs will be formatted as: `<Item><Key>key</Key><Value>val</Value></Item>`

- **GetValue(**@Key NVARCHAR(4000)**)**
  RETURNS: NVARCHAR(MAX)
  Returns the Value identified by the @Key
  @Key only matches whole-word and is case-sensitive
  If @Key is not found, NULL is returned

- **GetValueByKeyPattern(**@SearchPattern NVARCHAR(4000), @CaseSensitive BIT**)**
  RETURNS: NVARCHAR(MAX)
  Returns the Value identified by the @Key matching the Regular Expression (RegEx) of @SearchPattern which can be set to @CaseSensitive or not
  @SearchPattern can match non-whole-word Keys and is not necessarily Case-Sensitive
  If @SearchPattern matches more than one Key, the first match is used

- **RemovePair(**@InputKey NVARCHAR(4000)**)**
  RETURNS: Type_HashTable
  Removes the Key=Value pair identified by the @Key
  @Key only matches whole-word and case-sensitive

- **ToString()**
  RETURNS: NVARCHAR(MAX)
  Returns an Ampersand (&)-separated list of Key=Value pairs
  If wanting to persist the value of the HashTable, store the ToString() value and then use that value with the Constructor or AddData().
  Ampersands (&) and equal signs (=) are URI encoded (or percent encoded) into %26 and %3D, respectively.

NOTES:
- Is a key/value pair array
- The order of the Keys does not matter
- Key may be an empty string (nothing to the left of the =) but in all cases the Keys must be unique (so only one empty / blank Key can be added)
- Values can also be empty but do NOT need to be unique
- Property and Method names ARE case-sensitive: Clear() <> CLEAR()
- Uses compression to work over a greater amount of values than is possible natively. Natively max number of values is based on all keys and values adding up to 7996 bytes but with compression can be tens of thousands of bytes depending on the values and what order they are in

EXAMPLES:
```
DECLARE             @HashVar     SQL#.Type_HashTable

SET @HashVar = 'City=Chicago&State=IL&Zipcode=60606'
```

```
SELECT @HashVar.ToString(), @HashVar.Count, @HashVar.GetValue('City')
-- Zipcode=60606&City=Chicago&State=IL    3      NULL

SET @HashVar = @HashVar.AddData('County=Cook')
SELECT @HashVar.ToString(), @HashVar.Count, @HashVar.GetValue(' City ')
-- County=Cook&Zipcode=60606&City=Chicago&State=IL    4      NULL

SET @HashVar = @HashVar.RemovePair('State')
SELECT @HashVar.ToString(), @HashVar.Count, @HashVar.ValuesDataLength
-- County=Cook&Zipcode=60606&City=Chicago 3      16

SELECT @HashVar.ContainsKey('County'), @HashVar.ContainsKey('county')
-- 1   0

SELECT @HashVar.ContainsValue('Cook'), @HashVar.ContainsValue('cook')
-- 1   0

SELECT      @HashVar.GetValueByKeyPattern('.*[c].*', 0),
            @HashVar.GetValueByKeyPattern('.*[c].*', 1)
-- Cook      60606
-- first pattern matches Key of 'County' on the first char as it is
--    NOT case-sensitive
-- second pattern matches Key of 'Zipcode' as it IS case-sensitive and
--    is the only Key with a lower-case 'c'

SET @HashVar = @HashVar.Clear()
SELECT '*' + @HashVar.ToString() + '*', @HashVar.Count,
@HashVar.ValuesDataLength
-- ** 0      0
```

Type_NVarcharArray

DECLARE @Varible Type_NVarcharArray

CONSTRUCTOR:

- Comma-separated list of strings (VARCHAR or NVARCHAR)
- Spaces are trimmed on both sides
- SET @Type_NVarcharArrayVariable = ‘’
- SET @Type_NVarcharArrayVariable = ‘Hello’
- SET @Type_NVarcharArrayVariable = ‘One,Two , Three’

PROPERTIES:

- Count
  The number of items in the Array

- DataLength
  The total number of characters of all of the Items combined

METHODS:

- AddData(@Index INT, @InputStrings NVARCHAR(4000))
  RETURNS: Type_NVarcharArray
  Adds one or more Strings (VARCHAR or NVARCHAR), separated by commas (like the Constructor)
  @Index is where to insert the new values

@Index = 0 will ADD to the end of the Array
@Index > 1 will INSERT at the @Index point
@Index > NVarcharArray.Count or < 0 will cause an error

- Clear()
RETURNS: Type_NVarcharArray
Removes ALL items from the Array, leaving the Count = 0

- ContainsItem(@SearchString NVARCHAR(4000))
RETURNS: BIT
Will return 1 (true) if the @SearchString is found anywhere in the array
@SearchString matches only on whole-words and is case-sensitive

- ContainsPattern(@SearchPattern NVARCHAR(4000), @CaseSensitive BIT)
RETURNS: BIT
Returns 1 (true) if any Item matches the Regular Expression (RegEx) of @SearchPattern which can be set to @CaseSensitive or not
@SearchPattern can match non-whole-word Items and is not necessarily Case-Sensitive

- GetAt(@Index INT)
RETURNS: NVARCHAR(4000)
Returns the String found at the @Index
@Index < 0 will cause an error
@Index > NVarcharArray.Count returns NULL

- IndexOfItem(@SeachString NVARCHAR(4000), @Index INT)
RETURNS: INT
Returns the Index value that matches the first occurrence of @SearchFloat starting at @Index
If the @SearchFloat value is not found, 0 is returned
@Index < 1 or > NVarcharArray.Count will cause an error

- IndexOfPattern(@SearchPattern NVARCHAR(4000), @Index INT, @CaseSensitive BIT)
RETURNS: INT
Returns Index of String matching the Regular Expression (RegEx) of @SearchPattern, starting at @Index, which can be @CaseSensitive or not
@SearchPattern can match non-whole-word Items and is not necessarily Case-Sensitive
If more than one Item matches @SearchPattern starting at @Index, the first occurrence is returned

- RemoveAt(@Index INT)
RETURNS: Type_NVarcharArray
Removes the String at @Index
@Index < 1 or > NVarcharArray.Count will cause an error

- RemoveItem(@InputString NVARCHAR(4000))
RETURNS: Type_NVarcharArray
Removes the first occurrence of @InputString found in the array

- RemoveRange(@Index INT, @Count INT)
RETURNS: Type_NVarcharArray
Removes @Count number of Strings from the array starting at @Index
If @Index + @Count > NVarcharArray.Count an error will occur

- Reverse()
RETURNS: Type_NVarcharArray
Reverses the order of the Strings in the array

This is in essence a DESCending sort by Index, not by String

- Sort()
  RETURNS: Type_NVarcharArray
  This does an ASCending sort of the Strings by value, not by index
  If you want a DESCending sort of the values, call Reverse() after Sort()

- ToString()
  RETURNS: NVARCHAR(4000)
  Returns a comma-separated list of the String values
  If wanting to persist the value of the NVarcharArray, store the ToString() value and then use that value with the Constructor or AddData().

NOTES:

- Is a 1-based indexed array of NVARCHAR(4000)s
- Must be initialized with constructor (=) before using (at least set to = '')
- NULLs (or empty strings or empty values between commas) ARE allowed
- Property and Method names ARE case-sensitive: Sort() <> SORT()
- NVarcharArray Properties and Methods work just like matching Properties and Methods of FloatArray and HashTable; see previous examples of FloatArray and HashTable to better understand how NVarcharArray works
- Uses compression to work over a greater amount of values than is possible natively. Natively max number of values is based on all values adding up to 7996 bytes but with compression can be tens of thousands of bytes depending on the values and what order they are in

History

Version 1.0.2 (June 25th, 2006 – Initial Release)

- String_Combine, String_Contains, String_EndsWith, String_Equals, String_IndexOf, String_InitCap, String_LastIndexOf, String_PadLeft, String_PadRight, String_Split, String_StartsWith, String_Trim, String_WordWrap
- RegEx_IsMatch, RegEx_Match, RegEx_Matches, RegEx_Replace, RegEx_Split
- Math_Constants (30 physics constants), Math_Convert (22 measurement conversions), Math_Factorial, Math_IsPrime
- INET_GetWebPages, INET_Ping, INET_PingTime
- Phnx_GenerateDateRange, Phnx_GenerateDates, Phnx_GenerateFloatRange, Phnx_GenerateFloats, Phnx_GenerateIntRange, Phnx_GenerateInts, Phnx_ToWords
- SQLsharp_GrantPermissions, SQLsharp_Help, SQLsharp_IsUpdateAvailable, SQLsharp_SetSecurity, SQLsharp_Setup, SQLsharp_Uninstall, SQLsharp_Update, SQLsharp_Version, SQLsharp_WebSite

Version 1.1.3 (not released)

- Added: INET_FTPDo, INET_FTPGet (BETA), INET_FTPPut (BETA), INET_GetIPAddress, INET_GetHostName

Version 1.1.4 (October 20th, 2006)

- Free Version
- Removed: all INET functions and SQLsharp_Update
- Added: Math_RandomRange
- Changed:
  - Bug in Phnx_ToWords returned Negative Zero for -1
  - Added Lower and Upper bounds checking in Phnx_ToWords
  - Changed return datatype in Math_IsPrime to BIT from INT
  - Added StepTypes of Quarter and Week to Phnx_GenerateDateTimes and Phnx_GenerateDateTimeRange
  - Added abbreviations for StepTypes as paralleled in Books Online under DATEADD and DATEDIFF functions (ex: year = yyyy, yy)

Version 1.1.5 (October 20th, 2006)

- Paid-for Version
- Includes all functions
- Same as Version 1.1.4 except it includes all INET functions and SQLsharp_Update

Version 1.5.6 and 1.5.7 (February, 16th, 2007)

- Fixed installation / setup so that it completes successfully ;-)
- Changed prefix for Miscellaneous functions to be Util_ instead of Phnx_
- Added Math_CompoundAmortizationSchedule (BETA), Util_GZip, Util_GUnzip, Util_Deflate, Util_Inflate

Version 2.0.8 and 2.0.9 (March 18th, 2007)

- Enhanced Installation script
- Added SQL# Schema
- Fixed result column names in Util_Generate* functions and RegEx_* functions
- Enhanced error-handling and success output of SQLsharp_SetSecurity
- Added Optimizer Hints (IsDeterministic and IsPrecise) to all functions
- Removed UseBinaryMode option from INET_FTPGet and INET_FTPPut
- Added **File** functions: GetFile, GetRandomFileName, GetTempPath, PathExists, WriteFile
- Added User-Defined Aggregates: GeometricAvg, Join, Median, RootMeanSqr
- Added User-Defined Types: DoubleArray, HashTable, NVarcharArray

Version 2.1.10 and 2.1.11 (July 16th, 2007)

- Added **Util** Functions: IsValidCC, IsValidSSN
- Added **String** Functions: Count, Newline
- Added User-Defined Aggregate: Random
- Added compression to: Agg_Median, Type_FloatArray, Type_NVarcharArray, Type_HashTable

Version 2.2.12 and 2.2.13 (August 19th, 2007)

- Added **Date** Functions: BusinessDays, DaysInMonth, DaysLeftInYear, FirstDayOfMonth, FullDateString, FullTimeString, IsLeapYear, LastDayOfMonth

Version 2.3.14 and 2.3.15 (September 5th, 2007)

- Added **Date** Functions: IsBusinessDay, FormatTimeSpan
- Updated Date_BusinessDays: added more options for ExcludeDaysMask (Friday, Good Friday [Gregorian Calendar], Easter [Gregorian Calendar], Good Friday [Julian Calendar], Easter [Julian Calendar], Thanksgiving [CANADA], Thanksgiving [CANADA – day 2, Friday before])
- Added **Math** Functions: Cosh, Sinh, Tanh
- Added new **DB** grouping
- Added **DB** Function: DumpData
- Added compression to: Agg_Join

Version 2.3.16 and 2.3.17 (September 10th, 2007)

- Fixed Constructor method for all three User-Defined Types (FloatArray, HashTable, and NVarcharArray); allow for empty string ('') to be passed in to initialize the Type as empty.
- Fixed Math_CompoundAmortizationSchedule: adjusted final payment calculation and minor issues with rounding

Version 2.4.18 and 2.4.19 (October 14th, 2007)

- Added **File** Functions: GetFileBinary, WriteFileBinary, GetDriveInfo, Move, CreateDirectory, DeleteDirectory, Encrypt, Decrypt, GetDirectoryListing, Delete, Copy, DeleteMultiple, CopyMultiple, MoveMultiple
- Added new **LookUp** grouping
- Added **LookUp** Functions: GetCountryInfo, GetStateInfo

Version 2.5.20 and 2.5.21 (November 18th, 2007)

- Added **File** functions: GZip, GUnzip, ChangeEncoding, SplitIntoFields
- Added **INET** functions: IsValidIPAddress, AddressToNumber, NumberToAddress
- Added new **Convert** grouping.
- Added **Convert** functions: ToBase64, FromBase64, ROT13, BinaryToHexString, HexStringToBinary
- Added **Date** functions: ToUNIXTime, FromUNIXTime
- Added **String** function: NthIndexOf, Cut, SplitIntoFields
- Added **Utility** functions: CRC32, Hash(MD5 | SHA1 | SHA256 | SHA384 | SHA512)

Version 2.6.22 and 2.6.23 (May 18th, 2008)

- Updated DB function DumpData:
  - fixed handling of BINARY, VARBINARY, and IMAGE datatypes
  - changed direct query output data-type from TEXT to NVARCHAR(MAX)
  - added optional parameter for @LinkedServer
  - added optional parameter for @FileEncoding that supports: Ascii, UTF8, Unicode, UnicodeBigEndian, and UTF32.
- Added **DB** functions: BulkExport and HTMLExport
- Opened up INET functions AddressToNumber, NumberToAddress, and IsValidIPAddress to Free Version of SQL#
- Added **Util** functions: IsValidCheckRoutingNumber, and IsValidPostalCode
- Updated **SQLsharp** function GrantPermissions: added optional second parameter @SQLsharpSchema
- Added **File** function: CurrentEncoding
- Updated **File** function WriteFile: @FileEncoding parameter now accepts: Ascii, UTF8, Unicode, UnicodeBigEndian, and UTF32
- Updated **File** function WriteFileBinary: @FileEncoding parameter now accepts: Ascii, UTF8, Unicode, UnicodeBigEndian, and UTF32

Version 2.7.24 and 2.7.25 (August 5th, 2008)

- Added **INet** functions: FTPGetBinary, FTPGetFile, FTPPutBinary, FTPPutFile, HTMLDecode, HTMLEncode, URIDecode, and URIEncode
- Added **Date** functions: Age, Extract, and Truncate
- Updated **DB** function HTMLExport:
  - Translate {SQL#Column} into the column name
  - Added "EncodeHTML" option
  - Stopped single-quotes from being escaped to double single-quotes
- Updated **DB** procedure DumpData: Added parameters for Disable / Re-enable ALL Constraints and/or Triggers on each table
- Updated **String** function NewLine: Changed <br/> to <br /> for @EOLType = XHTML
- Updated **SQLsharp** procedure SetSecurity: Fixed sending in parameter of 0 when not in a DB named [SQL#]
- Updated **SQLsharp** procedure Update:
  - Fixed error that removed SQL# Schema when calling SQLsharp_Uninstall but did not re-add it
  - Added @ForceUpdate parameter; and function now checks for newer version of SQL# and will error if no newer version and @ForceUpdate is false / 0 or unset

Version 2.8.28 and 2.8.29 / 2.8.30 and 2.8.31(May 27th and 31st, 2009)

- Updated SQL# Installer to not cause the “file has an extremely long line” warning message when opening in Management Studio (SSMS)
- Renamed main Assembly from [SQLsharp] to [SQL#].
- Added two new Assemblies: [SQL#.OS] and [SQL#.Twitterizer]
- Fixed potential memory leak in INET_GetWebPages
- Fixed output of Table-Valued Functions to be properly streaming
- Increased width of NVARCHAR fields in the result sets of **File** functions
- Fixed RegEx functions to not error if the ExpressionToValidate is an empty string
- Compiled RegEx patterns in Util_IsValid* functions for faster execution
- Updated SQLsharp_Setup, SQLsharp_Uninstall, and the installation scripts to allow SQL# to be installed into a user-defined Schema and then handle being uninstalled from that Schema. In order to install into a Schema other than “SQL#” just change the value of the `@SQLsharpSchema` variable.
- Added RegExOptionsList parameter to all **RegEx** Functions
- Added TrapErrorInline BIT parameter to INET_GetWebPages Function to control whether HTTP errors (e.g. 404, 500, etc.) throw an exception or get returned in the result set. Please note that this is a Function signature change that breaks existing uses of the Function since existing calls will not have the new parameter. Passing in NULL (or 0) as the 3rd parameter will cause the Function to work the same as it did previously.
- Added optional @AssemblyName NVARCHAR(4000) input parameter to SQLsharp_SetSecurity so that the various SQL# Assemblies can be dealt with individually.
- Added **DB** Procs: BulkCopy (suggested and tested by DM Unseen [Martijn Evers]) and ForEach
- Added **INET** Function: URIEncodeData
- Added new **OS** group (SQL#.OS Assembly)
- Added **OS** Functions: EventLogRead, EventLogWrite, ProcessStart, ProcessGetInfo, ProcessKill, GenerateTone, MachineName, and Uptime
- Added new **Twitter** group (SQL#.Twitterizer Assembly)
- Added **Twitter** Functions (via Twitterizer library): Update, DestroyMessage, SendDirectMessage, GetSentMessages, GetMessages, GetFriendsTimeline, GetPublicTimeline, GetUserTimeline, and GetReplies

Version 2.8.32 and 2.8.33 (June 6th, 2009)

- Minor fixes in SQLsharp_SetUp for INET_URIEncodeData and FILE_* Table-Valued Functions.
- Minor fix in DB_ForEach and addition of Replacement Tags: {SQL#Schema} and {SQL#FullTableName}
- Added ProcessName parameter to OS_ProcessKill

Version 2.9.39 and 2.9.40 (November 1st, 2009)

- Added **Convert** functions: HtmlToXml (suggested by Mitch Schroeter), UUDecode, and UUEncode
- Added **Twitter** functions: DestroyDirectMessage, FollowUser, GetFollowers, GetFriends, GetMentions, GetStatus, GetUser, UnFollowUser
- Added **Date** function: NthOccurrenceOfWeekday
- Updated File_SplitIntoFields: Changed SkipFirstRow BIT into RowsToSkip INT and fixed potential memory leak. The parameter change is non-breaking as the BIT values of 0 and 1 (representing to not skip any rows and to skip the first row respectively) directly map to the new behavior of how many rows to skip with 0 still meaning not to skip any and 1 meaning to skip 1 row which is the same result as SkipFirstRow = 1. (suggested by Andy Krafft)
- Updated Twitter functions so that StatusID is now a BIGINT instead of INT: DestroyStatus, GetFriendsTimeline, GetMessages, GetPublicTimeline, GetReplies, GetSentMessages, GetUserTimeline, SendDirectMessage, and Update
- Added INET functions: URIGetInfo (suggested by Mitch Schroeter) and URIGetLeftPart
- Updated Type_HashTable: added AddItem(@InputKey NVARCHAR(4000), @InputValue NVARCHAR(4000)) method / function.
- Updated DB_BulkExport: Added @AppendFile BIT = 0, @RowsExported INT = -1 OUTPUT parameters. This should be a non-breaking change since the two new parameters have defaults. Therefore, existing implementations do not need to change.
- Updated INET_FTPGetFile: Changed @OverwriteExistingFile BIT parameter to be @FileHandling TINYINT (2 = Incremental / Restart). This is a non-breaking change since the old parameter BIT values of 0 and 1 directly map (i.e. implicitly convert) to the new parameter datatype of TINYINT and which cause the same behavior. Therefore, existing implementations do not need to change. (suggested by Andy Krafft)
- **BREAKING CHANGE ☹:** Updated INET_FTPGet and INET_FTPGetBinary: Added @ContentOffset BIGINT parameter to support Incremental downloads / resuming. (suggested by Andy Krafft)
- **BREAKING CHANGE ☹:** Updated INET_GetWebPages: Added four new fields to the result set (IsFromCache, LastModified, StatusCode, StatusDescription) and added three new input parameters (@MaximumAutomaticRedirections, @Timeout, @MaximumResponseHeadersLength, @CustomHeaders)
- **IMPORTANT NOTE:** Deprecated Twitter_DestroyMessage and replaced with Twitter_DestroyStatus. It is just a rename as the functionality is the same. Currently DestroyMessage points to DestroyStatus, however, please convert all references to DestroyMessage as it will be removed in the next version.
- Thanks to Mitch Schroeter for the suggestion of adding the MaximumResponseHeadersLength, CustomHeaders, and Method parameters to INET_GetWebPages

Version 2.10.43 and 2.10.44 (January 20th, 2010)

- Added **Twitter** functions: CreateFavorite and DestroyFavorite
- Updated **Twitter** functions to return UserID INT, RateLimit INT, RateLimitRemaining INT, and RateLimitReset DATETIME in the Result Set: GetFriendsTimeLine, GetMentions, GetMessages, GetPublicTimeLine, GetReplies, GetSentMessages, GetStatus, and GetUserTimeLine
- Updated **Twitter** functions to return RateLimit INT, RateLimitRemaining INT, RateLimitReset DATETIME, IsVerified BIT, CreatedOn DATETIME, UTCOffset INT, and NumberOfStatuses INT in the Result Set: FollowUser, GetFollowers, GetFriends, GetUser, UnFollowUser
- Added **DB** function: XOR which takes two BIT fields and does a logical Exclusive-OR on them.
- Updated RegEx_Split: returns StartPos and EndPos fields in the Result Set.
- Updated RegEx_Replace: "Count" input parameter now accepts -1 to mean "unlimited" replacements.
- Updated all RegEx functions to set the "RegularExpression" input parameter to be an NVARCHAR(MAX) instead of an NVARCHAR(4000).
- Updated String_SplitIntoFields: Added optional parameter for ColumnNames that is a comma-separated list of values that will be used to create the column names of the Procs result set. If not set it will default to the prior behavior of using FieldN where N is the field number starting with 1. (suggested by Olivier Moschkowitz)
- Updated Date_Extract: Added DatePart's that are found in the SQL Server built-in function DATEPART to be comprehensive: Year, Quarter, Month, Day, DayOfYear, Weekday, Week, Hour, Minute, Second, and Millisecond. The DatePart of ISO_WEEK was previously available as ISOWEEK but is now also aliased as ISO_WEEK to match SQL Server's DatePart name.
- Updated SQLsharp_Setup to auto-detect if a particular Assembly has been created and if not, Setup will not attempt to create the Procs and/or Functions contained in the missing assembly. This will allow each user to determine if they want to install the SgmlReader and/or Twitterizer and/or OS Assemblies. Thanks to Scott Prugh for requesting optional Assembly loading.
- **BREAKING CHANGE ☹:** Updated INET_GetWebPages: Added ResponseUri NVARCHAR(4000) to Result Set. Also added Method NVARCHAR(10) and PostData NVARCHAR(MAX) input parameters.

Version 2.11.51 and 2.11.52 (June 19th, 2010)

- Added **Date** functions: GetDateTimeFromIntVals, GetIntDate, and GetIntTime
- Updated File_GZip and File_GUnzip: Replaced built-in .Net GZip and GUnzip libraries with external DotNetZip library for better compression and ZIP64 for > 4 GB files.
- Fixed INET_URIDecode to properly translate "+" (plus-sign) into " " (space) which is not done by the built-in .Net library. (suggested by Andy Krafft)
- Updated all RegEx functions to return NULL (or empty result set for the Table-Valued Functions, or 0 for RegEx_IsMatch) if ExpressionToValidate is passed in as NULL. This behavior mirrors more closely the built-in T-SQL string functions. [thanks to Andy Krafft and Jason Pierce]
- Updated RegEx_Split: Fixed output that was misreporting StartPos and EndPos fields when the "part" was empty (nothing between the delimiters).
- Updated all String functions to return NULL (or empty result set for the Table-Valued Functions, or 0 for scalar functions that return a Boolean / BIT) if input string (or SearchValue if applicable) is passed in as NULL. This behavior mirrors more closely the built-in T-SQL string functions.
- Added String_IsNumeric to mirror the built-in T-SQL ISNUMERIC() function but that can handle more than 8000 characters and more numeric formats.
- Added RegEx_CaptureGroup which returns just the specified captured group and not the entire capture as RegEx_Match does. (suggested by Jason Pierce)
- Added Twitter functions: GetFavorites, GetBlocks, BlockUser, UnBlockUser
- Updated File_GetDirectoryListing to skip the “System Volume Information” folder when doing recursive as that always caused an error.
- Added **File** function: GetFileInfo
- **BREAKING CHANGE ☹:** Updated INET_GetWebPages: Added new field for ContentBinary that holds that Content data IF the data is Binary (in which case the regular Content field is NULL). Also added new input parameter for ContentDetection to either hard-code Binary vs Text or Auto-Detect. If using Auto-Detect, then if the ContentType starts with "text/" then the Content field is filled out and ContentBinary is NULL. (suggested by Mitch Schroeter)
- **IMPORTANT:** Updated all Twitter functions to authenticate against OAuth since Basic Auth is being shut down. This is a TEMPORARY fix which keeps the SQL# Twitter API the same for easy transition to the new authentication method. However, Twitter is requiring a full move to OAuth by October so a new version of SQL# will be released in the next two months that will be an API change for ALL Twitter functions. The API change will be that UserName and Password will no longer be passed in to each function but instead a ConsumerKey and ConsumerToken. Each SQL# user who is using the Twitter functions will have to create an application on Twitter which will give you the ConsumerKey and ConsumerToken. More details will be provided as that development occurs. Just be aware that **EVERYONE** using the SQL# Twitter functions will have to upgrade to the next version when it is released!!

Version 2.12.53 and 2.12.54 (September 19th, 2010)

- This release (2.12.x) is a Twitter API ONLY update. No other changes have been made in this release! If you do not use the Twitter functions and are on 2.11.x you do NOT need to upgrade. However, **if you are using the Twitter functions then you MUST upgrade to 2.12.x for the full OAuth implementation changes!** **The Twitter functions in version 2.11.x will no longer work as of Monday, October 18th, 2010.**
- Please see the SQL# Twitter setup guide for details on how to set up your Twitter Application: http://www.SQLsharp.com/download/SQLsharp_TwitterSetup.pdf

Version 2.13.55 and 2.13.56 (November 22nd, 2010)

- Updated OS_EventLogWrite to accept NVARCHAR(MAX) for @Message as opposed to NVARCHAR(4000)
- Updated OS_EventLogRead to return NVARCHAR(MAX) for Message as opposed to NVARCHAR(4000)
- Added **OS** function: StartTime
- Fixed Math_IsPrime as it was falsely reporting some large numbers as Prime that were not
- Updated File_SplitIntoFields to add new parameter for @FileEncoding so that the user has full control over the encoding type. Previously it was set to AutoDetect which did not always produce the correct result.
- Updated File_WriteFile and File_WriteFileBinary to add "UTF7" as a FileEncoding option
- Added **File** functions: CreateTempFile, GetDirectoryName, GetFileName, GetRootDirectory, and Touch

Version 2.14.60 and 2.14.61 (March 13th, 2011)

- Updated String_IsNumeric to allow for "d" to be double-precision as well as not requiring the + or – for scientific notation
- Added **String** functions: Replace, SplitKeyValuePairs, TrimChars, TrimEnd, and TrimStart
- Updated Twitter Status-related Table-Valued Functions to return 7 geo fields: GetUserTimeline, GetPublicTimeline, GetFriendsTimeline, GetReplies, GetMentions, GetFavorites, GetMessages, and GetSentMessages
- **BREAKING CHANGE ☹:** Updated Twitter_Update to accept optional Longitude and Latitude values for geocoding Tweets
- Added Twitter functions: GetHomeTimeline, GetRetweetedBy, GetRetweetedByMe, GetRetweetedToMe, GetRetweets, GetRetweetsOfMe, and Retweet
- **BREAKING CHANGE ☹:** Updated Twitter Status-related Table-Valued Functions to be able to pass in Twitter Optional Parameters:GetFriendsTimeline, GetFavorites, GetMentions, GetMessages, GetReplies, GetSentMessages, and GetUserTimeline as well as new functions GetHomeTimeline, GetRetweetedBy, GetRetweetedByMe, GetRetweetedToMe, GetRetweets, and GetRetweetsOfMe
- Deprecated Twitter_GetReplies in favor of GetMentions.
- Updated DB_BulkExport: added support for UTF7 as well as more datatypes: rowversion, date, time, datetime2, and datetimeoffset
- Updated DB_DumpData: added support for UTF7 as well as more datatypes: rowversion, date, time, datetime2, and datetimeoffset
- Updated DB_HTMLExport: added support for UTF7
- Added new **RunningTotal** group (not available in Free version)
- Added **RunningTotal** functions: Add, Get, CacheSize, and ClearCache
- Added **Util** functions: HashBinary and IsValidConvert
- Added **Date** functions: FullDateTimeString (not available in Free version) and NewDateTime
- Updated Date_FullDateString and Date_FullTimeString to return NULL if input is NULL rather than error
- Updated FILE functions to allow for full streaming and hence use much less memory: CopyMultiple, DeleteMultiple, GetDirectoryListing, and MoveMultiple
- Updated RegEx_Split to stream results out
- Updated Table-Valued **RegEx** functions to return NVARCHAR(MAX) for [Value] instead of NVARCHAR(4000): Match, Matches, and Split
- Added **INET** function: DownloadFile
- Added **Math** functions: CubeRoot, IEEERemainder, NthRoot, and Truncate

- Added **Convert** functions: DateTimeToMSIntDate and MSIntDateToDateTime
- Updated String_Split to fully stream output so it now uses less memory.

Version 2.15.62 and 2.15.63 (August 31st, 2011)

- Added **RegEx** functions: Escape, Index, and Unescape
- Updated installer to remember security settings of previously installed assemblies
- Updated File_SplitIntoFields and String_SplitIntoFields to accept a new, optional parameter for @DataTypes. The @DataTypes parameter allows you to set the specific data type of one or more of the fields in the result set. The default is still to create each field as NVARCHAR(MAX), but if you know that certain fields will always be a particular data type, then you can have all of the SplitIntoFields stored procedures return a more strongly-typed result set (which means doing fewer conversions later).
- Added **INET** function: SplitIntoFields
- Added new **Sys** group.
- Added **Sys** function: Objects (server-wide view of sys.objects)
- Updated File_SplitIntoFields to make @RowsToSkip parameter optional. The default is 0.

Version 2.16.64 and 2.16.65 (October 19th, 2011)

- Fixed error in installer that shows up when the collation setting for tempdb is not the same as the setting for the database in which SQL# is installed
- Fixed minor error with SQLsharp_Uninstall (minor in that the error is reported but the uninstall still completes) that occurs when the database in which SQL# is installed has a case-sensitive collation
- Updated INET_GetIPAddress to return NULL when NULL is passed in rather than error
- Added **INET** function: INET_GetIPAddressList
- Updated Date_BusinessDays and Date_IsBusinessDay to include two new holidays: *Presidents' Day [US] (3rd Monday in February)* (suggested by Claudio Pracilio) and *Columbus Day [traditional] (October 12th)*
- Added **Date** function: Date_BusinessDaysAdd (suggested by Victor Wang)
- Added **RegEx** function: RegEx_CaptureGroups (suggested by Jason Pierce)
- Updated Date_BusinessDays to allow for StartDate to be greater than EndDate which will return a negative number, similar to how DATEDIFF works (suggested by Victor Wang)
- Updated Date_BusinessDays to return NULL when any parameter is NULL rather than error
- Added **String** functions: FixedWidthIndex (suggested by Don Folino) and FixedWidthSplit

Version 2.17.68 and 2.17.69 (May 6th, 2012)

- Added **Date** function: Format (suggested by Dietmar Müller)
- Added INET function: URIDecodePlus to extend the capabilities of URIDecode in two ways: 1) unescape %uXXYY-encoded Unicode characters which otherwise throw an error; and 2) gracefully handle errors, making set-based processing easier (suggested by Andy Krafft).
- Added **SQLsharp** procedure: Download (replaces Update)
- Updated INET_GetWebPages to allow "Content-Type" to be set via @CustomerHeader value. If set, the passed-in "Content-Type" will override the automatic value set when using the POST method (suggested by Michael Kuhl).
- Updated DB_BulkExport: Improved handling of binary fields: a) drastic speed increase, and b) added missing "0x" prefix
- Added **String** function: CompareSplitValues
- Updated Convert_BinaryToHexString: Improved performance
- Moved the following functions to Full version: String_SplitIntoFields, String_FixedWidthSplit, String_FixedWidthIndex, DB_BulkExport, DB_HTMLExport, and DB_ForEach.

- Renamed String_SplitIntoFields to String_SplitResultIntoFields. String_SplitIntoFields is deprecated and will be replaced in the next version or two with a slightly different usage. Please switch any use of String_SplitIntoFields to point to String_SplitResultIntoFields.

Version 3.0.70 and 3.0.71 (March 4th, 2013)

- Added **Math** functions: FormatDecimal, FormatFloat (not available in Free version), and FormatInteger (not available in Free version)
- Add / Remove assemblies:
    - Ability to install / uninstall individual assemblies. This is not automated yet but will someday soon be incorporated into the installer script.
    - Updated SQLsharp_Setup to accept new parameter @SQLsharpAssembly, which if specified, will install the wrapper functions and stored procedures for only the specified assembly. The specified assembly needs to already exist.
    - Updated SQLsharp_Uninstall to accept new parameter @SQLsharpAssembly so that the specified assembly and its wrapper functions and stored procedures can be uninstalled without affecting anything else.
- Security:
    - SQL#-specific database login created from asymmetric key.
    - All SQL# assemblies now owned / authorized by new SQL# login instead of dbo.
    - Updated SQLsharp_SetSecurity to no longer set the DB to TRUSTWORTHY ON when setting an assembly to level 2 or 3 (External Access or Unrestricted).
    - All assemblies can be disallowed from being set to either Unrestricted (but allowed for External Access) or both Unrestricted and External Access.
    - Most functionality requiring External Access for main SQL# assembly has been broken out into separate assemblies: SQL#.DB, SQL#.FileSystem (FILE_* functions), and SQL#.Network (INET_* functions). Not only does SQL# stay as Safe, but if only INET_* functions are being used and not FILE_*, then no need to set SQL#.FileSystem to External Access.
    - If you are upgrading from a pre-3.0.x version and had any of the assemblies' permissions set to level 2 or 3 (External Access or Unrestricted), then the database where SQL# is installed had its TRUSTWORTHY setting set to ON and this might not be necessary anymore. SQL# no longer requires TRUSTWORTHY to be set to ON for External Access or Unrestricted assemblies and if you have no other need for it to be on then please run the following: `ALTER DATABASE [{database_where_SQL#_exists}] SET TRUSTWORTHY OFF`
    - If you don't want any of the assemblies to ever be set to Unrestricted (but still be eligible to be set to External Access), then set the @AllowUnrestrictedAccess variable towards the top of the install script to 0. If you don't want any of the assemblies to ever be set to either External Access or Unrestricted, then set both @AllowUnrestrictedAccess and @AllowExternalAccess variables to 0. *Please note that the ability to restrict the level of permissions for any assembly requires that the database have its TRUSTWORTHY setting set to 0 / OFF*.
- Installer:
    - Uninstall of exisiting SQL# (if it exists) and install of current version now wrapped in a transaction that will rollback if any problem occurs, leaving everything as it was before the install attempt if the install cannot complete successfully.
    - A login, based on an asymmetric key, is created as a means of allowing assemblies to be set to External Access or Unrestricted without the need for the database to have TRUSTWORTHY set to ON.
    - A user is created in the database where SQL# is being installed, based on the new login mentioned just above. This user will own the SQL# assemblies instead of "dbo".
    - New assemblies: SQL#.DB, SQL#.FileSystem (FILE_* functions), SQL#.JsonFx (needed for Twitter_* functions), SQL#.Network (INET_* functions), and SQL#.TypesAndAggregates.
    - Variables towards the top of the install script allow for easy configuration of new SQL# login name and permissions.
    - If upgrading from a version prior to 3.0.x and the SQL# assembly was set to either External Access (2) or Unrestricted (3), then several of the new assemblies will be set to that same

permission level to have no initial change in behavior. Those new assemblies are: SQL#.DB, SQL#.FileSystem, and SQL#.Network.

- Twitter:
    - Updated to use newer Twitter v1.1 JSON API instead of older v1.0 XML API (v1.0 API starts incremental end-of-life process on March 5th, 2013).
    - Requires SQL#.JsonFx and SQL#.TypesAndAggregates assemblies.
    - **BREAKING CHANGE ☹:** Removed functions Twitter_GetRetweetedToMe and Twitter_GetRetweetedByMe as there is no replacement for either in the v1.1 API.
    - **BREAKING CHANGE ☹:** Removed function Twitter_GetPublicTimeline as there is no exact replacement in the v1.1 API, but might replace with new "sample" call that is similar.
    - **BREAKING CHANGE ☹:** Removed function Twitter_GetReplies as it was deprecated a while ago and merely pointed to Twitter_GetMentions.
    - **BREAKING CHANGE ☹:** Removed function Twitter_GetFriendsTimeline as it does not exist in the v1.1 API and Twitter_GetHomeTimeline is nearly identical.
    - Added function: Twitter_SearchTweets (not available in Free version).
    - SQL Server 2005 requires Unrestricted access (level 3) for both SQL#.JsonFx and SQL#.Twitterizer. This is handled automatically in the installer. Also, it is possible that this is not required in SQL Server 2005 Enterprise Edition, but I have no easy way to verify that at the moment.
    - No signature changes in this release! All input parameters and output fields are the same to make upgrading a smoother process. BUT, in the very near future there will be at least a few changes:
        - UserIDs are now BIGINT at Twitter and the SQL# Twitter functions will be updated to reflect that in both input params and result set fields and scalar return values.
        - Most User-based table-valued functions (i.e. those returning a list of users) allow for paging through the list of results but only getting a max of 20 or 100 at a time, depending on the call. The SQL# Twitter functions will be updated to return the "previous" and "next" cursor values so that they can be sent in as Optional Parameters.
        - Twitter functions that currently do not have the @OptionalParameters input parameter where the Twitter call supports optional parameters will have the @OptionalParameters input parameter added to the signature. These include: Update, GetFollowers, GetBlocks, and GetFriends.

Version 3.0.72 and 3.0.73 (April 6th, 2013)

- Fixed minor bug in SQLsharp_SetSecurity.

Version 3.1.79 and 3.1.80 (January 8th, 2014)

New

- **String** functions: LevenshteinDistance, PadBoth, Split4k*, and TryParseToInt*
- **SysInfo** function: IndexName
- **XML** assembly with functions: EscapeContent, Transform (i.e. XSLT) (suggested by Dave Sumlin), and UnescapeContent
- **Date** functions: DaysInMonthFromDateTime*, DaysLeftInMonth, IsDaylightSavingTime, ToLocalTime, and ToUniversalTime
- **DB** function: CurrentSQLStatement

Those marked with (*) are available in the Free version; the rest are only available in the Full version.

Improved

- Date_FormatTimeSpan: each TimeSpanPart can now include an optional width that will be left-padded with zeros if the actual value of that TimeSpanPart has fewer digits than the specified desired width. (suggested by Dave Sumlin)
- All **Twitter** functions: helpful error message displayed if SQL#.Twitterizer assembly permission level is not set to 2
- File_GetDirectoryListing: added new [ErrorMessage] field to result set. If the process does not have permission to list the contents of a directory when doing recursive, the specific error will be noted in the new field and the process will continue with the next directory; previously the process would error if permission was denied on any folder.
- INET_GetWebPages:
    - Added new [ResponseHeaders] XML field to result set (suggested by Dave Sumlin)
    - Allow "Referer" to be sent in for CustomHeaders (sets the HTTP_REFERER header)

Fixed

- DB_DumpData: Fixed issue preventing @LinkedServerName input parameter from working
- Util_GZip: no longer error on input of NULL or empty binary (0x); it now returns NULL in both cases
- Util_GUnzip: no longer error on input of NULL; it now returns NULL
- Twitter_SearchTweets function: fixed problem with OptionalParameters not being sent to Twitter
- **Twitter** functions no longer error on non-standard Unicode escape sequences in statuses (e.g. \ud8c3)
- INET_GetWebPages: no longer error on invalid date_modified in header; instead return 1900-01-01
- All **Twitter** functions that return a StatusText field now define it as NVARCHAR(300) instead of 200.
- All **Twitter** functions that return a UserID, RecipientID, or InReplyToUserID field now define them as BIGINT instead of INT.
- RegEx_Match: properly handles no match; returns empty result set instead of the bogus row
- Date_DaysInMonth, Date_DaysLeftInYear, Date_FromUNIXTime, Date_GetDateTimeFromIntVals, Date_IsLeapYear, Date_LastDayOfMonth, and Date_ToUNIXTime: return NULL when NULL is passed in rather than erroring

Version 3.2.81 and 3.2.82 (July 30th, 2014)

New

- **RegEx** functions: CaptureGroup4k, IsMatch4k, MatchSimple4k, Replace4k, ReplaceIfMatched (suggested by Matt McClellan), and ReplaceIfMatched4k
- **DB** functions / procs: CreateOrAlterQueryInfoTables (used with DB_GetQueryInfo), DescribeResultSets, GetQueryInfo, and ThrowException
- **Math** functions: BitwiseLeftShift and BitwiseRightShift
- **Util** stored procedure: Print
- **File** function: GetLineCount

The RegEx functions are available in the Free version; the rest are only available in the Full version.

Improved

- SQLsharp_Setup (only used by installer script): better error handling and more verbose output.
- Date_BusinessDays, Date_BusinessDaysAdd, and Date_IsBusinessDay: added 3 options for Veterans Day (November 11th) to Holidays list (suggested by David Sumlin)
- Type_FloatArray, Type_HashTable, and Type_NVarcharArray: updated AddData / AddItem methods so that they can be called without first initializing the variable via `SET @TypeVar = '';`
- RegEx_Replace: @RegularExpression of NULL returns NULL instead of erroring
- DB_BulkExport:
    - NULL or empty sting value for @Query exits instead of erroring
    - Added defaults for most input parameters
    - Changed @Query input param from NVARCHAR(4000) to NVARCHAR(MAX)

    - Added input param for @ConnectionString NVARCHAR(500); default = "Context Connection = true;" (i.e. internal / in-process connection)
    - Added input param for @TextQualifierEscape (for embedded TextQualifiers); default = NULL; NULL = @TextQualifier.
- File_GetFile:
    - Stream rows from file to SQL Server when setting @SplitLines = 1 rather than reading the entire contents of the file into memory first.
    - Added “LineLength BIGINT” field to the result set that is the number of characters (excluding newlines / returns) per each line OR all characters (including newlines / returns), if @SplitLines = 0.
    - Changed “ContentLength” field to be cumulative number of characters (excluding newlines / returns) read so far, inclusive of the current line OR all characters (including newlines / returns), if @SplitLines = 0.

Fixed
- Math_Factorial: passing in 0 returns 1 instead of 0.
- INET_GetWebPages: does not error when setting @SplitLines to 1 (issue from the previous release)

Version 3.3.83 and 3.3.84 (November 17th, 2014)

New
- **DB** functions / procs: DeserializeResults (used with SerializeResults and SerializeResultsInChunks), NewID, SerializeResults, SerializeResultsInChunks, and TryCatch
- **String** functions: DamerauLevenshteinDistance, DamerauLevenshteinDistancePlus, and LevenshteinDistancePlus
- **Util** functions: GarageCollect and GetTotalMemory
- **Twitter** functions: GetMutes, MuteUser, and UnMuteUser (suggested by Joe de Silva)
- **Aggregate** functions: BitwiseAND, BitwiseOR, BitwiseXOR, HarmonicMean, and JoinPlus (suggested by: Martijn Evers, Jason Pierce, and Ryanne Turenhout)

Util_GetTotalMemory is available in the Free version; the rest are only available in the Full version.

Improved
- SQLsharp_Setup (only used by installer script) and the installer script: Updated all "SYSNAME" references to be "sysname" to better support case-sensitive servers as "sysname" is an alias that needs to be looked-up in the [master] database.
- Date_BusinessDays, Date_BusinessDaysAdd, and Date_IsBusinessDay:
    - Added 2 holidays—Christmas Eve (December 24th) and New Year's Eve (December 31st)—to Holidays list
    - Changed @ExcludeDaysMask input param to BIGINT from INT
- DB_BulkCopy: Added optional BIGINT OUTPUT param for @RowsCopied that is the number of rows inserted into the Destination Table (suggested by David Sumlin)
- DB_GetQueryInfo:
    - Added input param @QueryGroup NVARCHAR(100) to more easily group repeated tests / makes aggregations to find averages very easy. Default = empty string.
    - Added input param @CaptureExecutionPlans BIT to disable logging of execution plans. they can be large and if testing a loop, each query within each iteration will have its own plan. Default = "true" / 1.
    - Reduced memory consumption
- DB_CreateOrAlterQueryInfoTables:
    - Added new "QueryGroup" field to @TableNamePrefix + "ExecutionContext" table that is populated via new @QueryGroup input param on DB_GetQueryInfo.
    - Added auto-generated Stored Procedure @TableNamePrefix + "DeleteTest" that removes one or more entries from all 4 "QueryInfo" tables, based on QueryInfoID

    - Added auto-generated Stored Procedure @TableNamePrefix + "GetBasicStats" that aggregates MIN, AVG, STDEV, MAX for the captured metrics, grouping on new QueryGroup field
- DB_BulkExport: Force external (i.e. non-"Context Connection = true") connections to use Impersonation to avoid security hole.
- Twitter: Added the following Result Set fields to all User TVFs: "Language NVARCHAR(50)", "NumberOfPublicListMemberships INT", "IsGeoEnabled BIT", "Following BIT", and "Muting BIT"

Fixed

- INET_GetWebPages: Properly encodes XML special characters for ResponseHeaders field to prevent "&" from causing an error.
- DB_DescribeResultSets: No longer errors on SQL 2005, 2008, and 2008 R2 if there was no row available for sample data.
- Math_Convert:
    - Returns NULL if any input parameter is NULL rather than erroring
    - Improved accuracy for Computer/Digital Storage (will finish temperature and distance conversions in the next release)

Version 4.0.93 (March 16th, 2017)

Initial release of 4.0

- Free version only.
- Same as 4.0.95 except:
    - Missing Date_FormatOffset.
    - Minor bug with installer: error if needing to enable CLR.
    - Minor bug with INET_GetWebPages: doesn't send Content-Length header or @PostData if method is "PUT".

Version 4.0.95 and 4.0.96 (March 25th, 2017)

New Approaches / Concepts

- **Group Functionality into Assemblies by Required Permissions / Security Level:**

  Several Assemblies have code that requires a different security level than other code in that same Assembly. For example, the SQL#.Network Assembly has some code that can run in SAFE (Level 1) mode, while most of it requires EXTERNAL_ACCESS (Level 2). Using the code that requires EXTERNAL_ACCESS forces the code that can run in SAFE to be in an Assembly marked as EXTERNAL_ACCESS.

  Ideally, code will never be given a higher security level than it needs. The goal here is to have as much code as possible remain in Assemblies that will only ever be marked as SAFE (Level 1). And then, code requiring EXTERNAL_ACCESS (Level 2) shouldn't be in an Assembly marked as UNSAFE. In order to accomplish this, code in each Assembly that requires a higher security level needs to be moved into a separate Assembly for just that security level. Assemblies will be named the same as the current category, but with a number at the end indiciating what security level they will need.

  This release has one new Assembly, **SQL#_2**, that is just the functionality from the SQL# Assembly that needs security level 2. In the next release, the rest of the Assemblies will be broken out into 1 or 2 additional Assemblies.

- **Easier, More Customizable Installation**

  In prior versions it was possible to disable installation of most Assemblies, as well as tailor the highest allowed security level via the @AllowExternalAccess and @AllowUnrestrictedAccess variables. However, it was not possible to skip the creation of the Asymmetric Key and Key-based Login in [master], even though those objects aren't needed if the Assemblies will always be set to SAFE.

  Now, those two variables have been replaced with @MaxAllowedAccessLevel that will be set to a value of 0 to 3, where 1, 2, and 3 correspond to the same security levels used for SQLsharp_SetSecurity, and all three levels will install the [master] Database objects. A value of 0 for @MaxAllowedAccessLevel skip all server-level configuration, including attempting to enable CLR Integration. This new option of 0 is intended for environments in which you have no server-level permissions and only SAFE Assemblies are allowed. The details of what each level does are fully documented in the installation script itself, towards the top.

  The installer also does more validation now, to identify problems before installation begins.

- **More Thorough Installation / Uninstallation**

  In previous versions of SQL#, the SQLsharp_Uninstall Stored Procedure would uninstall all SQLCLR objects from the local database, as well as the SQL# Schema, but it did not remove the SQL# User from the local Database, nor did it remove the two objects in [master]: the Asymmetric Key and the Key-based Login. Now SQLsharp_Uninstall does remove the SQL# User, and there is a new, pure T-SQL Stored Procedure in [master] that drops the Key-based Login, then the Asymmetric Key, and finally itself. SQLsharp_Uninstall even prints a reminder to the "Messages" tab that you will probably need to execute [master].[dbo].[SQLsharp_InstanceUninstall]. This new Stored Procedure does not run automatically at the end of SQLsharp_Uninstall as there might be other Databases with SQL# installed that need those objects.

  Another issue that one might run into is when restoring a Database that has SQL# installed onto an Instance that has never had SQL# on it. In this scenario, code requiring security levels 2 or 3 will not be able to execute since those elevated permissions require that Key-based Login (and that it have the correct permission). Previously, you had to re-execute the install script to get those objects back into [master]. But now, SQLsharp_InstanceSetup – a new, pure T-SQL Stored Procedue in the local Database – will re-create those 3 objects in [master]. And, this new InstanceSetup Stored Procedure is dynamically created during installation so that it can retain the value used for @SQLsharpLogin.

- **Better Local Help**

  Previously, the SQLsharp_Help stored procedure would display all of the available commands, including their parameter options. This worked well for a relatively well for first 100 to 150 functions, but became much harder to maintain and less useful as SQL# continued to expand (this release – 4.0 – contains 350 objects!). So, starting with this release, the full PDF manual is included in the installer and is imported (as a GZipped binary) into the Database along with the Assemblies. The PDF manual will thus always be available, even if the website isn't for any reason. You won't have to search around for it, and it will be included in every Database backup.

  The embedded binary data includes additional meta-data along with the gzipped PDF file. The meta-data is properties about the PDF, such as the Document Revision Number, which will make it easy to determine if a newer version has been published, and will allow for a PDF manual update script to prevent overwriting a newer version with an older one. There are two new functions that will get this meta-data: a scalar function to get the Document Revision Number, and a TVF to display all of the meta-data. And, there is a new stored procedure that will save the PDF manual to disk, provided a path and optional filename. If no filename is specified, it defaults to the current PDF manual filename.

  Please note that extracting the PDF manual requires EXTERNAL_ACCESS (Security Level 2). This means that it will not be possible to extract if the install was done using a value of either 0 or 1 for @MaxAllowedAccessLevel.

Highlights

- **RegEx cache:** Similar to how queries are compiled and their execution plans get cached, Regular Expressions – the pattern / expression itself – gets compiled (which takes time) and is then cached to save time on subsequent executions. And, just like the plan cache, there is a limited amount of space to use for cached expressions. The default cache size is 15 expressions. If there are more than 15 patterns / expressions that you regularly use, then the system will be spending more time than it should recompiling the expressions that are getting forced out of the cache. This release includes two new functions for managing the expression cache: one to set the cache to whatever size you want (only available in the Full version) and one to display the current size of the cache (also available in the Free version). Increasing the cache size (at least a little, not too high) so that all (or at least most) of your expressions can fit in should make your system more efficient and your queries using **`RegEx_*`** functions complete in less time.

  There is currently no means of having a custom cache size automatically set, and so setting the cache size will need to be done each time the App Domain is loaded. If the App Domain is unloaded due to memory pressure, a custom cache size will reset to the default value of 15. But this is something that could be checked periodically in a SQL Server Agent Job that also sets the value to the desired size if it is lower than the desired size. The next release of SQL# will provide a mechanism for setting a custom cache size each time the App Domain is created.

- **URI host connection limit:** When accessing network resources, .NET throttles connections per each host. The default limit for any particular host is 2. When this limit is reached, additional requests are blocked and will wait until a process completes and frees up its connection. This means that if you use **`INET_GetWebPages`** to hit one or more URIs that are fairly static (the host / machine portion of the URI, that is), then you could be experiencing a lot of latency if you have a call that is made repeatedly across multiple sessions. This release includes three new scalar functions for managing this. There is a function to display the current connection *limit* for a particular host, one to display the current connection *count* to a particular host, and one to set the limit for a particular host. It is unclear what a bad upper-limit would be, but it is clear that 2 is a very low. If you have code that calls an in-house web page or web service and is called by multiple sessions at the same time, then you need to set the connection limit to something higher than 2, and then your process should complete faster.

  There is currently no means of having a custom connection limit automatically set, and so setting a connection limit will need to be done each time the App Domain is loaded. If the App Domain is unloaded due to memory pressure, a custom connection limit will reset to the default value of 2. But this is something that could be checked periodically in a SQL Server Agent Job that also sets the value to the desired size if it is lower than the desired size. The next release of SQL# will provide a mechanism for setting custom connection limits each time the App Domain is created.

- **Splitting to native types:** Quite often (perhaps most often) when needing to split a string, the string in question is a list of IDs (i.e. INT or BIGINT values). In these cases, it is much more efficient to split directly to the numeric type as opposed to splitting to strings to then convert that string into an INT or BIGINT value. This release includes a few new TVFs to allow for splitting into native types: two for splitting into BIGINT (one is only available in the Full version) and one for splitting directly into UNIQUEIDENTFIER / GUIDs (only available in the Full version). The two TVFs that are only available in the Full version allow for determining how they handle empty elements as well as invalid elements: exclude them, return NULL, or error. These same two TVFs also allow for tailoring whether a NULL input value returns an empty result set (as most other TVFs do), or returns a single row of NULL.

- **Environment Variables:** This release includes several functions in the **OS** category (Full version only) that can get, and in some cases set, environment variables. There are TVFs for getting a list of all variables and their values, and UDFs to get the value of a specific variable. You can get Machine / System variables only, User variables only, or Process variables which includes variables specific to the process running **`sqlservr.exe`** as well as the Machine and User variables.

  You can set the value of Process variables, and those values will persist beyond that call, for the life of that process, and will even be available to a subprocess such as **`xp_cmdshell`** (unlike setting an environment variable via xp_cmdshell which does not persist). This only requires EXTERNAL_ACCESS (Level 2). You can also set User environment variables, and those will persist beyond the life of the **`sqlservr.exe`** process. However, this requires UNRESTRICTED access (Level 3) because it writes the values to the user's registry (which is how the value persists beyond the life of the process).

  Please also note that the "User" in question is the service account running the **`sqlservr.exe`** process (and this *might* not work if the process is running as "Local System"). There is no ability to write to the Machine / Server environment variables as that presents too great of a security risk.

- **Performace Counters:** This release includes several new functions in the **OS** (Full version only) category that allow you to interact with all Performance Counters, not just the SQL Server Instance-specific counters available via **`sys.dm_os_performance_counters`**. There is a TVF to get the full details of a sample: RawValue, BaseValue, Frequency, TimeStamp, CounterType, etc. And there is a UDF to get the calculated value of the most recent sample. Another UDF allows for adding a sample value to a counter (no, the system will not allow you to add samples to system counters). Finally, there is a function that allows for creating a new, custom Performance Counter category and counter, but it is a very rough draft / experimental: it only allows for creating unpaired counters (i.e. not requiring an additional "base" counter), and only allows for creating a single counter in the category. It is possible that it will be completed in the next release (and potentially along with an "UpdateCategory" function that allows for adding / removing counters from a category, something which is not directly supported; it requires dropping and recreating the category).

- **JSON:** Even though SQL Server 2016 includes native JSON support, it will be many years before everyone is using that version (there are plenty of installations still running 2005, 2008, and 2008 R2, as well as some still on SQL Server 2000 even). This release (Full version only) includes the first two functions intended to help people still using a version prior to 2016. There is a UDF to convert JSON into XML (as it can be parsed natively starting with SQL Server 2005), and one to convert XML into JSON (as XML can be natively generated using FOR XML, also starting with SQL Server 2005).

  Please keep in mind that JSON allows for anonymous arrays, a concept that does not exist in XML. In order to handle this, the resulting XML will use "<item>" elements to represent these anonymous / unnamed arrays.

- **XML:** While prior versions of SQL# allowed for saving XML data to a file, that XML data first had to be converted to NVARCHAR(MAX). That, however, does not provide a true XML document since the XML datatype does not store either the XML declaration (i.e. the "<?xml ... ?>" node on the first line) or any whitespace. This release includes a new function (Full version only) that can save XML data as actual XML. Not only can you specify any encoding you want (such as UTF-8, the most common Unicode encoding that wasn't supported in SQL Server until 2016), but if you include the XML declaration, you then have the option of having it include the "encoding=" property set to the name of the encoding being used to write the file. There also options for having the output indented as well as optionally placing each attribute on its own line (instead of the default behavior of placing all attributes of an element on the same line).

- **Pagination:** The need to page through a large result set, X rows at a time, is nearly universal. And, often enough, this need also includes the requirement to get the total number of rows of the entire, non-paged result set (in order to determine how many pages exist). One option is to use the OFFSET and FETCH options of the ORDER BY clause. That is a good option for getting one page of the result set, but it doesn't get the total number of rows, and it doesn't apply to anyone using a version prior to 2012. Another option is to dump the entire result set into a temporary table, get the total row count via @@ROWCOUNT, and then just select the subset of rows from the temp table. But, while you don't need to run the query twice, that is a lot of IO writing to the temp table plus the transaction log activity. Also, if you don't have control over the query, but instead just have a Stored Procedure to execute, then you can't use OFFSET and FETCH, and instead need to dump the results to a temp table via INSERT...EXEC. But then you need to be careful not to use the INSERT...EXEC construct within that Stored Procedure as you will get the "INSERT EXEC statement cannot be nested" error.

This release includes a Stored Procedure (Full version only) that addresses this need. It executes any query (including a Stored Procedure), and allows for skipping a specified number of rows while returning a specified number of rows. It optionally allows for using an OUTPUT parameter to get the total number of rows for the entire non-paged result set (without executing the query again!). Beyond that, it allows for optionally including one or two extra columns at the end (i.e. far right): the row count within the current page of results and / or the row count within the entire non-paged result set. And if that wasn't enough, there is also an optional parameter to override the field names of the result set.

This Stored Procedure does *not* load the entire result set into memory (or anywhere), so it only ever has the current row in memory and thus should not take up any more system resources for a 10 million row result set than it does for a 10 row result set. It also will not get the "INSERT EXEC statement cannot be nested" error, so you can execute Stored Procedures that use that construct.

- **Instance-wide Assembly info:** Some DMVs, such as **`sys.dm_db_index_operational_stats`**, report data across the entire Instance, but only provide IDs that need to be translated into names. The OBJECT_NAME built-in function can take an optional parameter for "database_id" so it can map names across the entire instance. The OBJECT_SCHEMA_NAME built-in function also has a parameter for "database_id". But, there is no built-in function for getting names across the entire Instance that are not in **`sys.objects`** or **`sys.schemas`**. The Sys_IndexName function was added in Version 3.1 (early 2014) to fill this void when using the Index DMVs. In this release, two functions (Full version only) have been added to fill a similar void with Assemlby DMVs, namely **`sys.dm_clr_loaded_assemblies`**. One function returns **`sys.assemblies`** across all accessible Databases, and even includes additional properties related to the "principal_id" – name, type, and SID – as they also exist on a per-Database basis. The other function simply returns the Assembly name given the IDs for the Database and Assembly.

- **Credit Card Number validation:** This is one of those subjects where there is more misinformation than correct information floating around related to how to properly determine what type of credit card it is based on the first 6 digits (many posts and RegEx patterns only check the first 4 digits). A good deal of research went into the improvements in this area in this release. The improvements in this release include handling many more card types (even the new 222300-prefixed MasterCard range) as well as a new function to simply validate the number (regardless of card type). There are two additional functions (Full version only): one that returns the type of card (with or without validating the number at the same time), and a TVF that, given a card number, returns the type and whether or not it is a valid number. Please note that "valid" and "invalid" are not in any way related to "active" and "inactive" as it is impossible to determine that active status of a card.

  There will eventually be a blog post detailing the research that went into this.

- **Retrieve large quantities of Twitter Users:** The original functions to get the various lists of Users (i.e. Followers, Friends, Mutes, Blocks, etc) returned 20 by default and could go up to 200 when passing in an @OptionalParameter. The problems here were: a) that not all of the SQL# Twitter functions were able to pass in the @OptionalParameters, b) no SQL# Twitter functions supported cursors, and c) some of the lists are tens of thousands of Users long.

This release includes not only additional properties (i.e. result set columns) for both User and Status TVFs, it also includes: a) the ability to pass in @OptionalParamaters for the remaining functions that can support them, b) the ability to get and pass along Twitter cursor valus (so that pages of data can be stepped through), and c) a SQL#-specific option for the “count” optional parameter whereby a value of “all” will loop through the cursor internally so that the entire list (or as much as can be retrieved within the rate-limit window) can be retrieved in a single execution, and returned as a single result set. Also, new functions (Full version only) have been added to get lists of IDs to be looked up using another new function to get a list of Users based on those IDs. The functions that get the list of IDs are not limited to getting only 200 at a time, and can instead get thousands per each call.

- **String Escape Sequences:** T-SQL does not have the concept of string escape sequences (e.g. \n or \r\n = newline, \t = tab, etc) so instead we have to use CHAR(13) + CHAR(10) for \r\n, CHAR(9) for \t, and so on, and they have to be concatenated in as opposed to the escape sequences that are done inline. This release contains a function (Full version only) that will “un”escape all .NET string escape sequences, including “\x” followed by 1 to 4 hex digits, “\u” followed by 4 hex digits, and others.

  This functionality has been included in the handling of certain input parameters of various functions and stored procedures where it is more common to use string escape sequences. RegEx replacement strings as well as some of the export stored procedures in the DB category (e.g. **`DB_HTMLExport`**) now have this ability built in.

- **Banker’s Rounding:** Typical rounding behavior – the type of rounding that the built-in T-SQL **`ROUND`** function does – is to round “away from zero”. Meaning, positive numbers go up (increase) and negative numbers go down (decrease). But there is another type of rounding called “to even” that, as the name implies, rounds to the closest even number, which might be an increase or decrease, depending on the digit being rounded. Using this method, a value of 8.45, rounded to a single decimal place, would result in a value of 8.4 instead of the 8.5 that would be returned from the **`ROUND`** function. This is commonly referred to as “Banker’s Rounding” due to it being used in finances.

  There are two UDFs that handle this: one using DECIMAL input and return value, and one using FLOAT input and return value. DECIMAL is slower but more precise, and FLOAT is faster and less precise. Ideally, financial calculations should be done using DECIMAL datatypes instead of FLOAT, since their need to be accurate is more important than performance (end-users don’t tend to appreciate faster calculations that are also incorrect).

- **Amortization Schedules:** Ideally there would be one approach to calculating an amortization schedule and everyone would follow it. Unfortunately some programmers use FLOAT (Double in .NET) datatypes and standard rounding instead of banker’s rounding (among other mistakes, such as when rounding is applied). An unfortunate example of these two particular mistakes is SQL#’s original CompoundAmortizationSchedule TVF (but that was added 10 years ago!). Even more unfortunately than that is the fact that its results were validated against an actual amortization schedule given to me as part of a refinance I did through a reputable lender. However, in this release there is a new, alternate CompoundAmortizationSchedule TVF that uses DECIMAL datatypes and banker’s rounding. The original TVF is being kept as its results do match those of software being used by banks. But if there is ever a choice, the new TVF is preferred as it is more accurate (its results have also been validated against another amortization schedule given to me as part of another refinance I did through another reputable lender.

  There will eventually be a blog post detailing the research that went into this.

- **Improved LevenshteinDistance Performance:** The basic algorithm for the four Levenshtein Distance functions compares each character of one sting to each character of the other string. The number of comparisons is the length of one string multiplied by the length of the other. Hence it is very easy for these functions to slow down as the strings get longer. This release introduces a few new shortcuts to reduce time spent doing unnessary comparisons: if the two strings are equal in length then check if they are the same, if one is shorter than the other then check if it is a substring of the longer one, etc. I will test one or two other options for improving performance for the next release.

Additionaly, all four Levenshtein Distance functions now have a @MaxDistance parameter that can be set to a value at which to exit the calculation if that value is reached, rather than proceeding to get the actual distance. The SQL# implementation of the Levenshtein and Damerau-Levenshtein distance algorithms is now one of the very few that truly short-circuit (i.e. exit early) rather than merely reducing how much of the strings to compare.

There will eventually be a blog post detailing the research that went into this.

- **Improved Code Page and Unicode Support:** In prior versions of SQL#, functions that allowed for specifying a file's encoding only allowed for specifying the six named .NET encodings: ASCII, UNICODE [implied Little Endian], UTF7, UTF8, UnicodeBigEndian, and UTF32 [implied Little Endian]. As of this release, aliases have been added to make your code easier to read for those who are not aware of Microsoft using "Unicode" to mean UTF-16 Little Endian, as well as the new ability to specify UTF-32 Big Endian.

  Additionally, there are several new encoding names that allow for saving files *without* a Byte Order Mark (BOM) for the few encodings that have BOMs. By default, the named .NET encodings include the BOM when saving files, but there are times when this behavior is undesirable.

  AND, it is now possible to pass in a number instead of an encoding name. Any value that can be converted to an Int will be interpreted as a Code Page.

New Functionality

Category	***Free Version***	***Additional in Full Version***
DB	WaitForDelay	
File		GetCRC32**,** GetFullPath (suggested by Jason Pierce), GetHashBinary (suggested by Kevin Greiner)
Math	CompoundAmortizationSchedule2, RoundToEvenDecimal, RoundToEvenFloat	
Util	IsValidCCNumber	GetCreditCardInfo, GetCreditCardType, Paginate
OS	ServiceAccount	GetEnvironmentVariable, GetEnvironmentVariables, GetSystemEnvironmentVariable**,** GetSystemEnvironmentVariables**,** GetUserEnvironmentVariable**,** GetUserEnvironmentVariables**,** IsBinarySidAValidWindowsAccount**,** IsSddlSidAValidWindowsAccount**,** PerfCounterAddData**,** PerfCounterCreateCategory**,** PerfCounterGetSample**,** PerfCounterGetValue**,** SetEnvironmentVariable**,** SetUserEnvironmentVariable**,** TranslateSddlSidToName
String	SplitInts (and 4k), ToLowerInvariant (and 4k), ToTitleCase (and 4k), ToUpperInvariant (and 4k), TryParseToDateTime, TryParseToDecimal (suggested by	CompareWithDynamicCollation, SplitIntoGuids (and 4k), SplitIntoIntegers (and 4k) , SplitKeyValuePairs4k, Unescape (and 4k)

	Jason Pierce)	
Twitter	Twitter_GetUserByScreenName	GetApiResponseJSON, GetApiResponseXML, GetBlockedUserIDs, GetFollowerUserIDs, GetFriendUserIDs, GetMutedUserIDs, GetStatusesByStatusIDs, GetUsersByScreenNames, GetUsersByUserIDs, UnRetweet
Sys		AllAssemblies, AssemblyName
RegEx	CaptureGroupCapture (and 4k), GetCacheSize, Index, MatchLength4k	CaptureGroupCaptures, SetCacheSize
Convert	BinarySidToSddl, SddlSidToBinary	
INET		GetConnectionLimitForURI, GetCurrentConnectionCountForURI, SetConnectionLimitForURI
JSON		JSONtoXML, XMLtoJSON
XML		SaveToFile
Date	FormatOffset (suggested by Jason Pierce)	
SQLsharp	InstanceSetup, ManualMetadata, ManualRevisionNumber, SaveManualToDisk, UnloadAppDomain	
Misc.	master.dbo.SQLsharp_InstanceUninstall	

Improved

- GENERAL:
    - Reviewed and cleaned up a lot of the early code / technical debt.
    - Streamlined build process.
    - Most functions requiring elevated permissions now display an error message containing the exact statement to execute to fix the problem. The remaining functionality missing this detailed error message will be addressed in the next release.
    - Reduced size of main SQL# assembly by 47 KB, while at the same time adding functionality.
    - REMOVED DB_XOR as it was unnecessary. Instead use built-in " ^ (Bitwise Exclusive OR)" operator.
    - The @FileEncoding parameter of any Function or Stored Procedure now accepts all encodings found here (https://msdn.microsoft.com/en-us/library/system.text.encoding.aspx#Anchor_5). Values can be from the "Code Page" column or the "Name" column. Additionally, you can use the following values, some being SQL#-specific (use one of the “NoBOM” values if you are writing to a file and need to omit the Byte Order Mark / BOM):
        - ASCII
        - UTF7
        - UTF8 / UTF8NoBOM
        - UTF16 / UTF16Le / Unicode
        - UTF16NoBOM / UTF16LeNoBOM / UnicodeNoBOM
        - UTF16Be / BigEndianUnicode
        - UTF16BeNoBOM / BigEndianUnicodeNoBOM
        - UTF32 / UTF32Le
        - UTF32NoBOM / UTF32LeNoBOM
        - UTF32Be, UTF32BeNoBOM
- Installation Script:
    - Removed “GO” after “USE” statement so that any error with “USE” (which will be a parse error) will fail the entire script. Previously (with the “GO” statement) if there was an error with the “USE” statement, that batch would fail, but the script would proceed to install SQL# into whatever database was current / active.

- Replaced @AllowExternalAccess and @AllowUnrestrictedAccess with @MaxAllowedAccessLevel:
  - Level 3 is same as @AllowUnrestrictedAccess = 1
  - Level 2 is same as @AllowExternalAccess = 1
  - Level 1 is the same as both previous variables being 0
  - Level 0 is a new concept that skips Instance-level configuration (Asymmetric Key, Key-based Login, and new dbo.SQLsharp_InstanceUninstall Stored Procedure in [master]) and doesn't install Assemblies that cannot be used in SAFE mode.
- Validate that the database that SQL# is being installed into has the same owner SID in both sys.databases and {install_db}.sys.database_principals. If there is a mismatch between those two records, then the install script will abort and provide the appropriate steps to remedy the situation, specific to the version of SQL Server (SQL Server 2005 is different than post-2005). The situation is usually caused during a restore operation. When the SID values do not match, then attempting to set any Assembly to either EXTERNAL_ACCESS or UNSAFE will result in error 10314 and/or 33009.
- Validate @SQLsharpLogin to ensure that it isn't a server- or database- principal that already exists and is a non-Asymmetric Key-based principal (e.g. “dbo”, “sa”, etc).

- **(BREAKING CHANGE)** String_LevenshteinDistance, String_LevenshteinDistancePlus, String_DamerauLevenshteinDistance, and String_DamerauLevenshteinDistancePlus:
  - Added @MaxDistance input parameter to allow for better efficiency by stopping processing once max value has been reached (suggested by Kirby Moyers).
- **(BREAKING CHANGE)** String_LevenshteinDistancePlus and String_DamerauLevenshteinDistancePlus:
  - Added input parameter for @LCID (i.e. Locale / Culture) with 0 = database_default and -1 = Invariant Culture (previously behavior used only the Invariant Culture); @CompareOptions param now allows for 'DATABASE_DEFAULT' (value is case-INsensitive) which uses the sensitivity settings associated with the Database's default Collation (empty string still means "None" == everything-sensitive).
- **(Behavior Change)** String_LevenshteinDistance and String_DamerauLevenshteinDistance:
  - Assumes LCID (i.e. Locale / Culture) associated with Database's default Collation (previous behavior used only the Invariant Culture).
  - Compare Options now uses the sensitivity settings associated with the database's default Collation (previous behavior used "Ordinal" / Binary).
- String_LastIndexOf:
  - Returns NULL on NULL input of any parameter rather than erroring if NULL passed into @StartIndex or @ComparisonType.
  - If @StartIndex < 1, start searching at end of @StringValue, rather than erroring.
  - **(Possible Behavior Change)** Uses Locale / LCID of the Database's default Collation instead of the Windows OS LCID (typically these are the same LCID, but they can be different).
  - Added @CompareOption = 0 that uses the sensitivity settings (case, accent, Kana, and width) of the Database's default Collation.
  - Invalid @ComparisonType value will error rather than returning 0 (which was misleading).
- String_Newline:
  - Returns current environment's newline when NULL passed in (used to error)
- Util_CRC32, Util_GenerateDateTimeRange, Util_GenerateDateTimes, Util_Hash, and Util_HashBinary:
  - Better performance.
- Util_GenerateIntRange, Util_GenerateDateTimes, Util_GenerateDateTimeRange, Util_GenerateDateTimes, Util_GenerateFloats, and Util_GenerateFloatRange:
  - Return empty result set if any input param is NULL instead of erroring.
- Util_IsValidCC:
  - More efficient (tremendous improvement!).
  - More accurate (including new Mastercard range).
  - Support several new card types: JCB, Carte Blanche, PayPal, Union Pay, MIR, UATP, Dankort, InterPayment, and Maestro.
  - Returns NULL on NULL input (either parameter) instead of erroring.
- Math_Cosh, Math_Sinh, and Math_Tanh:

  - Returns NULL on NULL input instead of erroring.
- Math_IsPrime and Math_RandomRange:
  - Return NULL on any input param being NULL rather than erroring.
- Math_CompoundAmortizationSchedule:
  - Added result set columns: **`CumulativePrincipal FLOAT, CumulativeAmountPaid FLOAT, and TotalAmountPaid FLOAT`**
- File_GetDirectoryListing and File_GetFileInfo:
  - Added result set columns for NTFS ACL info: **`SecurityDescriptor NVARCHAR, OwnerSID VARBINARY, OwnerName NVARCHAR, GroupSID VARBINARY, GroupName NVARCHAR`**
    (suggested by Tim Chapman)
  - GetDirectoryListing always returns NULL for OwnerName and GroupName (until they can be cached, which should be in the next release). For now, the SIDs can be translated either by using the new OS_TranslateSddlSidToName function, or by using "SUSER_SNAME(SQL#.Convert_SddlSidToBinary(OwnerSID))".
- File_GetDirectoryListing:
  - **(BREAKING CHANGE)** Added @IncludeSecurityInfo input parameter to toggle whether or not ACL info is retrieved as it does hurt performance (don't include unless you need the info).
  - Better performance.
- File_PathExists:
  - Returns NULL if a NULL is passed in rather than erroring.
- File_GetFileBinary and File_WriteFileBinary:
  - Better handling of NULL @FilePath.
- File_Touch:
  - Added @WhichTime options of "Create" and "All" (meaning all 3 times: Create, Access, and Write).
- RegEx (all functions):
  - Added "Compiled" to @RegExOptions parameters
  - Returns NULL (or empty result set for TVFs) if @RegularExpression is NULL (used to error)
  - Better error handling if @StartAt parameter is NULL
- RegEx:
  - **(Possible Behavior Change)** Unescape the following input params (so "\n" becomes an actual newline, and so on):
    - RegEx_CaptureGroup (@NotFoundReplacement)
    - RegEx_Replace (@Replacement)
    - RegEx_ReplaceIfMatched (@Replacement and @NotFoundReplacement)
- RegEx_IsMatch, RegEx_MatchLength, Replace, CaptureGroup, Index, Split, and CaptureGroups:
  - Better performance.
- RegEx_IsMatch:
  - **(Behavior Change)** Returns NULL if @ExpressionToValidate is NULL (** used to return 0 / "false")
- RegEx_MatchLength:
  - Better error handling if @Length is NULL
- RegEx_CaptureGroup:
  - Better error handling if @CaptureGroupNumber is NULL
  - @Length of NULL or < -1 assumed to be -1 (used to error)
- RegEx_Split:
  - @Count of NULL or < -1 assumed to be -1 (used to error)
- RegEx_Replace and RegEx_ReplaceIfMatched:
  - Better error handling if @Replacement is NULL
  - @Count of NULL or < -1 assumed to be -1 (used to error)
- Twitter (all functions):
  - Streamlined Twitter code

  - Added result set columns to all User-based functions:
    `ContributorsEnabled BIT, IsTranslator BIT, FollowRequestSent BIT, NumberOfFavorites INT, ProfileImageUriHttps NVARCHAR(2048), ProfileBackgroundColor NVARCHAR(20), ProfileTextColor NVARCHAR(20), ProfileLinkColor NVARCHAR(20), ProfileSidebarFillColor NVARCHAR(20), ProfileSidebarBorderColor NVARCHAR(20), ProfileBackgroundImageUri NVARCHAR(2048),ProfileBackgroundImageUriHttps NVARCHAR(2048), ProfileUseBackgroundImage BIT, ProfileBackgroundTile BIT, DefaultProfile BIT, DefaultProfileImage BIT, PreviousCursor NVARCHAR(50), NextCursor NVARCHAR(50), RateLimitResetLocalTime DATETIME`
  - Added result set columns to all Status-based functions:
    `PlaceCountryCode NVARCHAR(2), PlaceAttributes XML, PlaceBoundingBox XML, PlaceURL NVARCHAR(4000), FavoriteCount INT, FilterLevel NVARCHAR(50), InReplyToScreenName NVARCHAR(100), Language NVARCHAR(20), PossiblySensitive BIT, QuotedStatusID BIGINT, RetweetCount INT, Retweeted BIT, WithheldCopyright BIT, WithheldScope NVARCHAR(20), WithheldInCountries NVARCHAR(4000), Entities XML, RateLimitResetLocalTime DATETIME`
- Twitter:
  - Added @OptionalParameters input parameter to the following functions:
    - GetFollowers and GetFriends (user_id & screen_name & count & cursor)
    - GetBlocks and GetMutes (count & cursor)
- Twitter_GetBlockedUserIDs, Twitter_GetBlocks, Twitter_GetFollowers, Twitter_GetFollowerUserIDs, Twitter_GetFriends, Twitter_GetFriendUserIDs, Twitter_GetMutedUserIDs, and Twitter_GetMutes:
  - Allow for SQL#-specific Optional Parameter value of "all" for "cursor" parameter. Cycles through list until none left OR Rate Limit has been reached.
- DB_BulkCopy:
  - **(Possible Breaking Change)** Added optional parameter for @SourceCommandTimeout (suggested by Gary Steffen).
  - Added parameter defaults for @BatchSize (0), @NotifyAfterRows (0), and @TimeOut (30).
- DB_BulkExport:
  - Better error messaging (Messages tab and Return Value)
  - More efficient code (at least 10% faster)
  - Added optional parameter for @SourceCommandTimeout (default = 30).
- DB_ForEach:
  - Add replacement tags – {SQL#DBList.System}, {SQL#DBList.MSApp}, and {SQL#DBList.MSDemo} – for use with @DBPattern and @DBExcludePattern.
- DB_HTMLExport:
  - **(Possible Behavior Change)** Unescaped input params
  - Added parameter default values for most input parameters (helps when using via EXEC)
  - **(BREAKING CHANGE)** Added @AppendOutput input parameter (false = overwrite; default is to Overwrite, same as previous behavior).
- DB_DumpData:
  - Added parameter default values for most input parameters
  - Added optional @AppendOutput input parameter (false = overwrite; default is to Append, same as previous behavior).
- Sys_IndexName and Sys_Objects:
  - Improved performance (across repeated calls).
- SQLsharp_GrantPermissions:
  - Now auto-detects the Schema being used for SQL# objects, and so the optional input parameter @SQLsharpSchema has been removed.
  - An optional input parameter has been added for @PrintSqlInsteadOfExecute that will simply print to the "Messages" tab the SQL that would be executed without actually executing it.
- SQLsharp_Help:
  - Replaced outdated / unmaintained list of commands with info on how to extract embedded PDF manual.
- SQLsharp_Download:
  - Better error checking around input parameters.

    - Added default filename of "SQLsharp_SETUP_FullVersion.zip" if only a directory is specified.
    - Check for file and directory existence before attempting download.
- SQLsharp_Uninstall:
    - Ensure that using "dbo" Schema doesn't attempt to drop the Schema.
- SQLsharp_SetSecurity:
    - Added optional input parameter @DoNotPrintSuccessMessage to disable the default "Successfully set Assembly..." message (suggested by Kirby Moyers)
- Date_Format:
    - Return NULL on any input param being NULL rather than erroring.
- INET_GetWebPages:
    - Added result set column:
      `CharacterSet NVARCHAR(100)`

Fixed

- File_GetFile:
    - **(BREAKING CHANGE)** It would only properly handle files encoded as the system default or one of the Unicode encodings (as long as it included the Byte Order Mark / BOM). Added @FileEncoding input parameter to force the encoding when no BOM is present. Leaving @FileEncoding set to NULL will cause it to behave as it previously did (i.e. will auto-detect IF a BOM is present, else will assume system default, which is most likely non-Unicode, 8-bit Extended ASCII).
- File_ChangeEncoding:
    - **(BREAKING CHANGE)** Did not always translate correctly as source encoding was being auto-detected but not all encodings use Byte Order Marks, and the ones that do use them do not require that they be used. Added @CurrentEncoding to explicitly set the encoding of the source file.
    - No longer errors if @FilePathNew is NULL.
- File_CopyMultiple, File_DeleteMultiple, File_GetDirectoryListing, and File_MoveMultiple:
    - Properly handles C:\ (doesn't require "C:\\\") and \\Server\Share (doesn't require "\\\\Server\Share").
- File_CurrentEncoding:
    - Correctly reports encodings.
- String_LevenshteinDistance, String_LevenshteinDistancePlus, String_DamerauLevenshteinDistance, and String_DamerauLevenshteinDistancePlus:
    - Now handle spaces properly when using 'IgnoreSymbols' @ComparisonOption.
- Util_GenerateIntRange, Util_GenerateDateTimeRange, and Util_GenerateFloatRange:
    - Now include the upper-end value.
- RegEx_Split:
    - Fixed minor bug related to @StartAt parameter
    - Fixed minor bug related to @Count parameter
- Twitter (all functions returning a string value or field):
    - Fixed handling of Supplementary Characters (often used for Emoji).
- DB_BulkExport:
    - ColumnHeaderHandling of "always" did not work. No rows returned would not print column headers.
- DB_ForEach:
    - Schema name for each Table was not always correct.
- DB_BulkCopy:
    - Default destination changed: was "(local)", now correctly detects current instance, whether it is a default instance (will use "(local)"), a named instance (will use

"ServerName\InstanceName"), or SQL Server Express LocalDB (will use "np:\\.\pipe\LOCALDB#xxxxxxxx\tsql\query");
  - **(Behavior Change)** Default database was "tempdb" and is now unspecified, hence it will use the Login's default DB (so as to not increase contention on "tempdb").
- SQLsharp_GrantPermissions:
  - Now grants EXECUTE on User-Defined Aggregates (Agg_*) and User-Defined Types (Type_*).
  - User and Role names -- passed into @GrantTo -- with embedded single-quotes are now handled properly (but who would ever do that, right?)
- Math_Truncate:
  - Now handles negative numbers correctly (was rounding down).
- INET_GetWebPages:
  - Properly handles standard HTTP headers (specifically: Content-Type, Referer, User-Agent, Connection, Transfer-encoding, and Range)
  - Properly handles PUT method:
    - Sends @PostData.
    - If @PostData is NULL or empty string, will send Content-Length header as 0.
- INET_HTMLEncode:
  - Encodes apostrophes as "&apos;" instead of "&#039;".
  - Now encodes spaces (as "&nbsp;") instead of ignoring them.
- INET_HTMLDecode:
  - Decodes "&nbsp;" (as a space) instead of ignoring them.
  - Decodes "&apos;" (as an apostrophe) instead of ignoring them.

Version 4.1.98 and 4.1.99 (February 8th, 2018)

New Functionality

Category	*Free Version*	*Additional in Full Version*
Util		GetBase2Bits, UnBitMask
String	Trim4k	PadBoth4k, RemoveDiacritics (and 4k; Suggested by Jason Pierce), TrimChars4k, TrimEnd4k, TrimStart4k
Sys		LockResource
RegEx	Escape4k, Index4k, Match4k, Matches4k, Split4k, Unescape4k	CaptureGroupCaptures4k, CaptureGroups4k
Convert	Base2ToBase10, Base10ToBase2	

Improved

- GENERAL:
  - Greatly reduced size (by approx. 310 kb) of main **SQL#** Assembly by moving LookUp category into its own Assembly: **SQL#.LookUps**. This will improve initial load times and won't waste much memory when not using the LookUp functions.
- Installation Script:
  - Account for security changes related to SQL Server 2017 (i.e. "CLR strict security") using a Certificate (flexible, clean) instead of the new "Trusted Assemblies" (inflexible, messy).
- Networking:
  - Added explicit support for TLS 1.1 and TLS 1.2 protocols
  - Increased default "Connection Limit" for URIs to 20 from the .NET default of 2. This will reduce performance bottlenecks from concurrent access to the same URI.
- Twitter:
  - All functions now have a concurrent connection limit of 25 instead of the .NET default of 2

- INET_DownloadFile:
    - Set "User-Agent" HTTP header (required by some sites)
    - Improved error message when SQL#.Network Assembly wasn't at correct security level.
- INET_GetWebPages:
    - Added support for "Keep-alive" HTTP header
    - Added support for "ConnectionLimit" pseudo-HTTP header to set the URI's Connection Limit
- String_Contains:
    - @SearchValue input parameter is now NVARCHAR(MAX), something LIKE cannot do (please see "Overcome LIKE character length limitation" on DBA.StackExchange for details)
- Convert_BinarySidToSddl:
    - Slight performance improvement
- SQLsharp_Download, SQLsharp_IsUpdateAvailable, and SQLsharp_WebSite:
    - Updated URLs to be "https"

Fixed

- Util_Print:
    - Handles various types of newlines (**\r**, **\n**, and **\r\n**), including accounting for bug in SSMS – discovered while testing Util_Print – related to **\r\n** at the end of the line being printed (see: SSMS ignores final \r\n / CRLF / Carriage Return + Line Feed in PRINT and RAISERROR)
    - Properly handles lines that are the exact same length as @MaxLength
    - Added "-" as a default word-wrap character
    - Works with Supplementary Characters! (better than PRINT and RAISERROR !) Doesn't split the surrogate pair in two when the first half of it is the final "character" of the line.
- Twitter:
    - Support sending of all UTF-8 characters
- File_CopyMultiple, File_GetDirectoryListing, File_DeleteMultiple, and File_MoveMultiple:
    - Fixed error that would occur only if the Database containing SQL# had a Collation that was not case-insenstive, accent-sensitive, and having the same LCID as the OS.

Version 4.2.100 and 4.2.101 (November 13th, 2018)

New Functionality

Category	***Free Version***	***Additional in Full Version***
File		File_CheckLongPathSupportRequirements, File_FileExists, File_MoveDirectory
Types and Aggregates	Agg_JoinDelim	

Improved

- GENERAL:
    - Reduced size of main SQL# assembly.
- Installation Script:
    - Created separate User (Database-level principal) to "own" SQL#.FileSystem and SQL#.DotNetZip Assemblies. This will allow the Assemblies to be set to UNRESTRICTED (level 3) and be isolated in their own App Domain, something that is required if using long-paths (over 259 characters).
- Agg_Join and Agg_Median:
    - Handle more data internally. Previously was limited to 8000 bytes in order to be compatible with SQL Server 2005. Now, SQL Server 2005 version still has that limit, but when installed on SQL Server 2008 or newer, limit will be 2 GB.

- All File functions:
    - Added error messaging to detect if error is due to long paths, and if so, display appropriate info.
- Convert_HtmlToXml:
    - When accessing URLs, automatically handle TLS 1.1 and 1.2 when required.
- DB_BulkExport:
    - Added parameters (@OutputFormats and @OutputFormatsDelimiter) and functionality to control, per-column, the output format, including minimum width (to support fixed-width exports). This is a delimited list of .NET composite format strings. This list can be sparsely populated and does not need to contain definitions for all column; columns not in the list will be given the default / standard format. Default delimiter is "|".
- File_CopyMultiple and File_MoveMultiple:
    - Updated to handle (i.e. create) empty folders. Changed datatype of @Recursive parameter from BIT to TINYINT. This change is backwards compatible: **0** still means “not recursive”, and **1** still means “recursive”. But, now passing in **2** means “recursive, *including empty folders*”.
- File_CurrentEncoding:
    - Returns proper error message if @FilePath is an empty string.
- File_GetDirectoryListing:
    - Checks for folder structures over 259 characters in length. When found, the appropriate long-path prefix is added to the path: "**\\?**" for local paths, and "**\\?\UNC**" for UNC paths.
- File_GetLineCount:
    - Returns proper error message if @FilePath is an empty string.
- File_GUnzip:
    - Improved error message when SQL#.DotNetZip Assembly wasn’t at correct security level.
- File_Move:
    - Now supports ability to move file across volumes (i.e. drives / shares)
    - **(Behavior Change)** No longer supports moving / renaming a directory or directory structure. Please use the new File_MoveDirectory function to do either of those operations.
- File_SplitIntoFields:
    - Added option to set column names based on first row in file: @FirstRowContainsColumnNames input parameter (optional parameter; default = 0 / false for backwards compatibility / to be consistent with prior behavior)
    - Made @RegExDelimiter an optional parameter
- File_WriteFile:
    - Returns proper error message if @FilePath is NULL or empty string.
    - Assumes "false" / 0 if @AppendData is NULL (instead of erroring).
    - Creates empty file if @FileData is either NULL or empty string (instead of erroring).
- File_WriteFileBinary:
    - Creates empty file when passing in NULL for @FileData (instead of erroring).
    - Error message for invalid @FileMode parameter value now mentions the three valid values.
- All INET_FTP* functions
    - Updated to have better error handling / messaging.
- INET_GetHostName:
    - Returns NULL if NULL passed in.
    - Better error handling.
- INET_GetIPAddress:
    - Better error handling.
- INET_GetIPAddressList:
    - Returns an empty result set if NULL is passed in (only for @HostName).
    - Better error handling.
- INET_GetWebPages:
    - Handle AutomaticDecompression header (valid options: gzip, deflate, or both separated by comma or pipe).
    - Handle Authorization header (requires both "username" and "password" headers).
    - Passing in NULL for MaximumResponseHeadersLength no longer causes an error.

    - Changed datatype of @SplitLines input parameter from BIT to SMALLINT. This change is backwards compatible: values 0 and 1 still cause the same behavior as they previously did (0 = do not split text content; 1 = split text content). New option of -1 allows for skipping content download entirely. The effect of this is getting a very quick response containing the status, headers, content-type, response URI, LastModified date, and sometimes the content-length (this value is not always sent). Downloading an 11 MB file, including content (no split) was taking 1200 - 2100 ms. Using the new -1 option to skip downloading the content was returning in 90 - 160 ms. This option allows for:
        - quickly checking status / existence.
        -
    - Returns actual "content_length" instead of "-1" for non-error responses, even if the "Content-Length" header isn't in the response.
- INET_Ping:
    - Returns an empty result set if NULL is passed in (only for @HostName).
    - Increased max allowed TTL to 10240 (to match INET_PingTime).
    - Better error handling.
    - SocketError returns single row of "num" = -1, "status" = SocketError, and "address" = {message}.
- INET_PingTime:
    - Returns NULL if NULL passed in.
    - Better error handling.
    - Returns -2 for SocketError (typically DNS cannot resolve host).
- INET_SplitIntoFields:
    - Added option to set column names based on first row in file: @FirstRowContainsColumnNames input parameter (optional parameter; default = 0 / false for backwards compatibility / to be consistent with prior behavior)
    - Made @Timeout and @RegExDelimiter optional parameters
    - Better error handling.
- Sys_LockResource:
    - Added handling of: COMPRESSED_ROW, FILE, FULLTEXT_INDEX, METADATA (AUDIT, AUDIT_SPECIFICATION {Server and Database}, DATABASE, DB_MIRRORING_SESSION, INDEXSTATS , SERVER_PRINCIPAL, PASSWORD_POLICY, and STATS)
    - Added various properties of the resource to the return value to help identify it more quickly and provide additional insight to hopefully avoid additional manual queries required to trouble-shoot. For example, objects now include OBJECT_TYPE, indexes include INDEX_TYPE, and so on.
    - Schema-based items are now prefixed with the schema name
- Twitter:
    - All functions automatically handle TLS 1.1 and 1.2 when required.
    - All functions updated to send and receive extended tweets (up to 280 characters instead of 140)
    - Better handling of retweets and quoted tweets. New status result set fields: [StatusType] NVARCHAR(20), [QuotedStatus] XML, [QuotedStatusCreatedAt] DATETIME, [QuotedStatusCreatedAtLocalTime] DATETIME, and [CreatedLocalTime] DATETIME.
        - [StatusType] = 'Original', 'Retweet', or 'Quoted'.
        - [QuotedStatus] = XML of quoted status when [StatusType] is 'Quoted'.
        - [QuotedStatusCreatedAt] = UTC time of quoted/retweeted status.
        - [QuotedStatusCreatedAtLocalTime] = local server time of quoted/retweeted status.
        - [CreatedLocalTime] = local server time of current status.
    - At all times, the [StatusID] and [Created] fields describe the current status record. Additionally, when the status is a retweet: a) [QuotedStatusID] is the ID of the original status being retweeted in the current status record, and b) [QuotedStatusCreatedAt] / [QuotedStatusCreatedAtLocalTime] are the creation times of the original status being retweeted in the current status record, and c) the rest of the fields describe the retweeted

status, not the current status. Please see https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json#retweet for additional details.

- Type_HashTable:
  - Added new methods:
    - ValuesLength: returns the number of 2-byte characters across all Values in the HashTable.
    - GetTableXML: returns an XML document containing the contents of the HashTable.
  - Better error handling when passing in empty / white-space-only string for initialization or AddData() method.
  - Added ability to unescape URI encoded values, for initialization and AddData() method, so "&" and "=" characters can be passed in (those two are key-value pair delimiters). This works for both "Key" and "Value".

Fixed

- Agg_Join
  - Now returns NULL instead of an empty string when all rows in a group are NULL
- DB_BulkCopy:
  - “Object not set to a reference of an object” message would be returned instead of actual error message if an error occured in initial connection or SQL execution. Now source error message is returned.
- DB_BulkExport:
  - **(Behavior Change)** Fixed default formatting of certain datatypes to match BCP output (meaning: it can be imported by BCP without modification):
    - DATE, SMALLDATETIME, DATETIME, DATETIME2, and DATETIMEOFFSET now include milliseconds;
    - BINARY, VARBINARY, IMAGE, and ROWVERSION / TIMESTAMP now exclude the "0x" prefix.
- File_GUnzip:
  - Updated to use case-insensitive comparison when checking if file extension is ".gz"
- INET_HTMLDecode:
  - Handle all 2125 HTML 5 named character entity references (not just 102 of them).
  - Handle **&#{decimal_code_point};** and **&#x{hex_code_point};** numeric character references.
  - INET_HTMLEncode will be updated to also handle all 2125 HTML 5 named character entity references in SQL# version 5.0.
- RegEx_CaptureGroup4k:
  - Changed @NotFoundReplacement datatype from NVARCHAR(MAX) to NVARCHAR(4000)
- String_Unescape:
  - Fixed octal escape sequences that are 2 - 3 digits and begin with a 0 (e.g. \0x or \0xx)
- Sys_AllAssemblies
  - No longer errors on "invalid column is_user_defined" on SQL Server 2005
- Twitter:
  - Some Twitter functions getting "error parsing ..." due to newer, larger IDs. Proper handling of BIGINT / Int64 numbers that are large enough to default to DECIMAL instead of Int64 (i.e. long) when implicitly converted.
- Type_HashTable:
  - Now returns NVARCHAR(MAX) instead of NVARCHR(4000) for the following methods: GetValue, GetValueByKeyPattern, and ToString.
  - ValuesDataLength method now returns the total number of bytes instead of the number of characters.
  - URI encode (i.e. percent-encode) "&" and "=" characters when using the ToString() method. These two characters need to be escaped because they are used as key-value pair delimiters.