Documentation

AHKDb version 0.1 alpha

January 15, 2020

Table of Contents

Introduction

AHKDb is a database library for AutoHotkey (AHK).
It is written entirely in AHK (has no external dependencies), is lightweight (less than 500 kb), fast, powerful and relatively easy to use. AHKDb is also versatile; it contains just over 100 functions, allowing simple and complex tasks alike to be carried out with one function call each.
AHKDb uses the tab-separated database format in plain text.[1] Hence, databases made or modified using AHKDb can be viewed in any normal text editor (such as *Notepad* or *Sublime*) and imported directly by conventional spreadsheet software (such as *Microsoft Excel* and *LibreOffice Calc*). Additionally, databases of comma separated values (.csv) can be converted into tab separated values (see `DatabaseImportCSV`).
An effort has been put into making AHKDb simple to use. Therefore, hopefully all functions are given intuitive names (e.g. `DatabaseGetNumberOfRows`), making it easier to understand what they do, and all functions start with the word "Database" to avoid confusion with other functions. Also, many functions support optional arguments that can take the value *TRUE* or *FALSE*. In those cases, *FALSE* is always the default option. Furthermore, practically each code line of AHKDb has an explaining comment to make the code of AHKDb easier to understand.

Finally, if bugs, bad code, bad explanations, bad naming of functions etc. is discovered, feel free to reach back (with or without solutions/improvements). Any feedback, positive or negative, is appreciated. The goal is to create a great database library for (and written in) AHK.

1. AHKDb does not support tabs inside cells (if necessary, users are suggested to encode tabs using another set of characters). This limitation is not considered problematic, and hence no effort has been put into *solving* this.

Getting Started

Starting to use AHKDb is simple. After downloading the library (AHKDb.ahk) and placing it in the working directory, just add `#include AHKDb.ahk` somewhere in your code and you are good to go. All functions of AHKDb can now be used.

AHKDb identifies rows and columns using numbers. The top row of the database is *Row 1*, the next row is *Row 2* etc. Columns are ordered correspondingly, but from left to right. Occasionally a specified database cell is referred to using *coordinates*. Cell coordinates are written (R,C) where *R* and *C* are replaced by their corresponding column and row numbers (e.g., (2,1) is the cell on the second row and the first column). Also, the size of a database can be written RxC where R is the number of rows in the database, and C is the number of columns. The figure to the right illustrates the structure of a 3x3 database.

	Col 1	Col 2	Col 3
Row 1	(1,1)	(1,2)	(1,3)
Row 2	(2,1)	(2,2)	(2,3)
Row 3	(3,1)	(3,2)	(3,3)

With AHKDb implemented, a new database can be created using `DatabaseAddRow`, and comma-separated databases can be imported using `DatabaseImportCSV`. A large number of functions can then be used to create, modify and get content from a database. And to familiarize oneself with the available functions, it may be helpful to go through the *List of Functions* in the end of this documentation. Also, for someone new to AHKDb, the following functions may be useful:

- `DatabaseCreate`
- `DatabaseGet`
- `DatabaseMatchGetColumn`
- `DatabaseModifyCell`
- `DatabaseSortByColumn`
- `DatabaseView`

Lastly, some functions (e.g. `DatabaseFind`) return *arrays*. For information on how arrays work, see *Object-based Arrays* in the AutoHotkey documentation.

AHKDb Functions

All functions of AHKDb are introduced below with the following information:

- A short **description** of the functions purpose and important information (including relevant limitations).
- A list of all **arguments** the function allows, and whether these are required or optional.
- The content the function **returns** (if any).
- A list of **related functions** (with clickable links if viewed digitally).
- A **code example** designed such that it can be copied and executed without modifications (unless noted). The code examples do not include comments, but are hopefully self-explanatory.

In order to find a function for a given purpose, the list of functions in the end of the documentation may be helpful.

DatabaseAbsoluteValue(DatabaseName, *Row*, *Column*, *SkipFirstRow*)

Description Replaces the content of specified cells with their absolute values.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related

`DatabaseExp`
`DatabaseLog`
`DatabaseModulo`
`DatabaseNaturalLog`
`DatabasePower`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "-5NUMBERS5" )
DatabaseView( "database.txt" )
DatabaseAbsoluteValue( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseAddColumn(DatabaseName, *Row1, Row2, …*)

Description Adds a new column to the right end of database. The content of up to 50 cells (starting for top) can be specified.

Arguments *DatabaseName:* Name of the database.

Row1 (optional): Content of the first cell of the new column.

Row2 (optional): Content of the second cell of the new column.

Return Nothing.

Related `DatabaseAddRow`

`DatabaseInsertColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseAddColumn( "database.txt", "First cell", , "Third cell" )
DatabaseView( "database.txt" )
```

DatabaseAddition(DatabaseName,
Constant,
Row,
Column,
SkipFirstRow)

Description Adds a constant to each cell value.

Arguments *DatabaseName:* Name of the database.

Constant: Constant to add to each cell value.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseDivision`
`DatabaseMultiplication`
`DatabaseSubtraction`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseAddition( "database.txt", 100, , 4 )
DatabaseView( "database.txt" )
```

DatabaseAddNA(DatabaseName, *SkipFirstRow*)

Description Replaces empty cells with "NA".

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Number of replacements.

Related `DatabaseRemoveNA`

Example

```
DatabaseCreateTest( "database.txt", , 3, 3 )
DatabaseModifyColumn( "database.txt", 2, TRUE )
DatabaseView( "database.txt" )
DatabaseAddNA( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseAddNumerationColumn(DatabaseName, *SkipFirstRow*)

Description Adds a new column containing 1, 2, 3 … to the right end of database.

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If set to *TRUE*, the numeration will start on the second row (with the first number being 1).

Return Nothing.

Related `DatabaseAddRandomColumn`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseAddNumerationColumn( "database.txt", TRUE )
DatabaseView( "database.txt" )
```

DatabaseAddRandomColumn(DatabaseName, *RandomMin*, *RandomMax*, *SkipFirstRow*)

Description Adds a new column containing random numbers to the right end of database. For random integers, specify RandomMin and RandomMax using integers. Likewise, for decimal numbers, specify RandomMin and RandomMax using decimal numbers.

Arguments *DatabaseName:* Name of the database.

RandomMin (optional): Smallest possible random number (default: 0).

RandomMax (optional): Largest possible random number (default: 9).

SkipFirstRow (optional): If *TRUE*, no random number will be placed in the first row.

Return Nothing.

Related `DatabaseAddNumerationColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseAddRandomColumn( "database.txt", -5, 3 )
DatabaseAddRandomColumn( "database.txt", -5.0, 3.0 )
DatabaseView( "database.txt" )
```

DatabaseAddRow(DatabaseName, Column1, Column2, …)

Description Adds a new row to the bottom end of database. The content of up to 50 cells (starting for left) can be specified.

Arguments *DatabaseName:* Name of the database.

Column1 (optional): Content of the first cell of the new row.

Column2 (optional): Content of the second cell of the new row.

Return Nothing.

Related `DatabaseAddColumn`

`DatabaseInsertRow`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseAddRow( "database.txt", "First cell", "Second", , "Fourth" )
DatabaseView( "database.txt" )
```

DatabaseArcCos(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with arccosine of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcSin`
`DatabaseArcTan`
`DatabaseCos`

Example

```
DatabaseCreateTest( "database.txt", "-1.0NUMBERS1.0" )
DatabaseView( "database.txt" )
DatabaseArcCos( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseArcSin(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with arcsine of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcCos`
`DatabaseArcTan`
`DatabaseSin`

Example

```
DatabaseCreateTest( "database.txt", "-1.0NUMBERS1.0" )
DatabaseView( "database.txt" )
DatabaseArcSin( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseArcTan(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with arctangent of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcCos`
`DatabaseArcSin`
`DatabaseTan`

Example

```
DatabaseCreateTest( "database.txt", "-1.0NUMBERS1.0" )
DatabaseView( "database.txt" )
DatabaseArcTan( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseBackup(DatabaseName, *Location*, *BackupName*)

Description Backup database using a built-it naming system.

Arguments *DatabaseName:* Name of the database.

Location (optional): Location for the backup file (default location is working directory).

BackupName (optional): Naming system to use. Possible values are *NUMBER* (filename of database and a number), *DATE* (filename of database and the current date), *DATETIME* (filename of database and the current date and time), *ORIGINAL* (same name as the original database). Default naming system is *NUMBER*.

Return Filename of backup, if no problem occurred.
0 if problem occurred.

Related `DatabaseCopy`
`DatabaseDelete`
`DatabaseRecycle`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseBackup( "database.txt", , "DATETIME" )
```

DatabaseCeiling(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with each cell value rounded up (to closest integer).

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseFloor`

`DatabaseRound`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS10.0" )
DatabaseView( "database.txt" )
DatabaseCeiling( "database.txt", 2 )
DatabaseView( "database.txt" )
```

DatabaseCheck(DatabaseName)

Description	Inspects database for two types of constructional errors (first, if database has rows with different number of cells, and second, if the last database row with cells end with a linebreak).
Arguments	*DatabaseName:* Name of the database.
Return	1 if no problem was found. 0 if at least one problem was found.
Related	Nothing.

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
if DatabaseCheck( "database.txt" )
    MsgBox, No problem detected.
else
    MsgBox, Problem detected.
```

DatabaseColumnSplitDelimiter(DatabaseName, Column, Delimiter, *SkipFirstRow*)

Description Splits a column, at the instance of a specified delimiter, into two columns. The delimiter itself is removed upon splitting the column. If a cell contains multiple instances of the delimiter, the column is split at the first instance of the delimiter (the remaining delimiters are kept).

Arguments

DatabaseName: Name of the database.

Column: Column number to split (1 is the leftmost column).

Delimiter: Delimiter at which each cell is split.

SkipFirstRow (optional): If *TRUE*, the first row will not be split (the first cell of the new column will be empty).

Return Nothing.

Related

`DatabaseColumnSplitLeft`

`DatabaseColumnSplitRight`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseColumnSplitDelimiter( "database.txt", 3, ", ", TRUE )
DatabaseView( "database.txt" )
```

DatabaseColumnSplitLeft(DatabaseName, Column, SplitPosition, *SkipFirstRow*)

Description Splits a column after a specified number of characters (starting from left) into two columns.

Arguments *DatabaseName:* Name of the database.

Column: Column number to split (1 is the leftmost column).

SplitPosition: Number of characters (from the left) after which to split column cells (the number is the last character that is kept in the left column).

SkipFirstRow (optional): If *TRUE*, the first row will not be split (the first cell of the new column will be empty).

Return Column number of new column.

Related `DatabaseColumnSplitDelimiter`

`DatabaseColumnSplitRight`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseConcatenateColumns( "database.txt", 1, 2 )
DatabaseView( "database.txt" )
DatabaseColumnSplitLeft( "database.txt", 6, 1 )
DatabaseView( "database.txt" )
```

DatabaseColumnSplitRight(DatabaseName, Column, SplitPosition, *SkipFirstRow*)

Description Splits a column after a specified number of characters (starting from right) into two columns.

Arguments

DatabaseName: Name of the database.

Column: Column number to split (1 is the leftmost column).

SplitPosition: Number of characters (from the right) after which to split column cells.

SkipFirstRow (optional): If *TRUE*, the first row will not be split (the first cell of the new column will be empty).

Return Column number of new column.

Related `DatabaseColumnSplitDelimiter`

`DatabaseColumnSplitLeft`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseConcatenateColumns( "database.txt", 1, 2 )
DatabaseView( "database.txt" )
DatabaseColumnSplitRight( "database.txt", 6, 1 )
DatabaseView( "database.txt" )
```

DatabaseCompare(DatabaseName1, DatabaseName2, *Row*, *Column*, *CaseSensitive*, *SkipFirstRow*)

Description Compares content of two databases.

Arguments

DatabaseName1: Name of the first database.

DatabaseName2: Name of the second database.

Row (optional): If specified, only cells on this row are compared.

Column (optional): If specified, only cells on this column are compared.

CaseSensitive (optional): If *TRUE*, case-sensitive comparison.

SkipFirstRow (optional): If *TRUE*, first row is not compared.

Return 1 if databases contain the same content.
0 if databases do not contain the same content.

Related `DatabaseCompareDimensions`

Example

```
DatabaseCreateTest( "database1.txt" )
DatabaseCreateTest( "database2.txt" )
if DatabaseCompare( "database1.txt", "database2.txt" )
    MsgBox, Databases are identical.
else
    MsgBox, Databases are different.
```

DatabaseCompareDimensions(DatabaseName1, DatabaseName2, *SkipRows*, *SkipColumns*, *SkipFirstRowDb1*, *SkipFirstRowDb2*)

Description Compares the number of rows and columns between two databases.

Arguments *DatabaseName1 (and 2):* Name of the database to compare.

SkipRows (optional): If *TRUE*, the number of rows are not compared.

SkipColumns (optional): If *TRUE*, the number of columns are not compared.

SkipFirstRowDb1 (and 2) (optional): If *TRUE*, the first database row is ignored.

Return 1 if the number of rows/columns are the same for both databases.
0 if the number of rows/columns are not the same for both databases.

Related `DatabaseCompare`

Example

```
DatabaseCreateTest( "database1.txt" )
DatabaseCreateTest( "database2.txt" )
if DatabaseCompareDimensions( "database1.txt", "database2.txt" )
    MsgBox, Database dimensions are identical.
else
    MsgBox, Database dimensions are different.
```

DatabaseConcatenateColumns(DatabaseName, Column1, Column2, *SkipFirstRow*)

Description	Concatenates two database columns. New column is based on the right end of database.
Arguments	*DatabaseName:* Name of the database. *Column1 (and 2):* Column to concatenate (1 is leftmost column). *SkipFirstRow (optional):* If *TRUE*, the first row is not concatenated.
Return	Return column number of new (concatenated) column.
Related	`DatabaseConcatenateRows`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Column := DatabaseConcatenateColumns( "database.txt", 2, 3 )
MsgBox, The concatenated column has column number %Column%.
DatabaseView( "db.txt" )
```

DatabaseConcatenateRows(DatabaseName,
Row1,
Row2)

Description	Concatenates two database rows. New row is placed in the bottom of database.
Arguments	*DatabaseName:* Name of the database. *Row1 (and 2):* Row to concatenate (1 is topmost row).
Return	Return row number of new (concatenated) row.
Related	`DatabaseConcatenateColumns`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Row := DatabaseConcatenateRows( "database.txt", 2, 3 )
MsgBox, The concatenated row has row number %Row%.
DatabaseView( "database.txt" )
```

DatabaseCopy(DatabaseName,
NewDatabaseName,
Row,
Column,
Overwrite,
SkipFirstRow)

Description Store a copy of a database (or a part of a database) as a new file.

Arguments

DatabaseName: Name of the database.

NewDatabaseName: Name of database copy.

Row (optional): If specified, copy only the specified row.

Column (optional): If specified, copy only the specified column.

Overwrite (optional): If *TRUE*, overwrite the file if filename exists.

SkipFirstRow (optional): If *TRUE*, do not copy the first row.

Return 1 if no indication of failure.
0 if problem occurred.

Related

`DatabaseBackup`
`DatabaseDelete`
`DatabaseRecycle`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseCopy( "database.txt", "newdatabase.txt", , 2 )
DatabaseView( "newdatabase.txt" )
```

DatabaseCos(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with cosine of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcCos`
`DatabaseSin`
`DatabaseTan`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseCos( "database.txt", , 3 )
DatabaseView( "database.txt" )
```

DatabaseCreate(DatabaseName,
Rows,
Columns,
Overwrite,
ContentByColumn,
Cell1, Cell2, …)

Description

Creates a new database file. If cell content is specified (*Cell1*, *Cell2* etc.), database cells are by default filled row-by-row (can be changed by *ContentByColumn*). It is also possible to create a new database using *DatabaseAddRow*.

Arguments

DatabaseName: Name of the database.

Rows: Number of rows in new database.

Columns: Number of columns in new database.

Overwrite (optional): If *TRUE*, overwrite if file exists.

ContentByColumn (optional): If *TRUE*, cell content is ordered by column.

Cell1 (optional): Content of the first cell.

Cell2 (optional): Content of the second cell.

Return

1 if no indication of failure.
0 if problem occurred.

Related

`DatabaseCreateTest`

`DatabaseAddRow`

Example

```
DatabaseCreate( "database.txt", 2, 2, , , "a", "b", "c", "d" )
DatabaseView( "database.txt" )
```

DatabaseCreateTest(DatabaseName, *Type*, *Rows*, *Columns*, *Overwrite*)

Description Creates a test database (useful when testing functions). There are four types available; *A1*, *NUMBERS*, *LETTERS* and *AIRPORTS* (see below).

Arguments

DatabaseName: Name of the database.

Type (optional): Type of database to create. There are four types available:

- *A1 (default):* Each cell contains one letter and one digit.
- *NUMBERS:* Each cell contains a (pseudo-)random number (by default, integers from [0,9]). For integers between -2 and 6, use *2NUMBERS6*. For decimal numbers, specify range using decimals (e.g., *2.0NUMBERS6.0*.
- *LETTERS:* Each cell contains a randomly selected letter. For 3 letters per cell, use *LETTERS3*.
- *AIRPORTS:* Creates a database (of fixed size) containing 18 airports.

Rows (optional): Number of database rows (not supported by type *AIRPORTS*).

Columns (optional): Number of database columns (not supported by *AIRPORTS*)

Overwrite (optional): If *TRUE* an existing database file is overwritten if necessary.

Return 1 if no indication of failure.
0 if problem occurred.

Related `DatabaseCreate`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
```

DatabaseDelete(DatabaseName)

Description Deletes database.

Arguments *DatabaseName:* Name of the database.

Return 1 if database was successfully deleted.
0 if database was not successfully deleted.

Related `DatabaseBackup`
`DatabaseCopy`
`DatabaseRecycle`

Example

```
DatabaseCreateTest( "database.txt" )
if DatabaseDelete( "database.txt" )
    MsgBox, Database deleted.
else
    MsgBox, Database not deleted.
```

DatabaseDivideColumns(DatabaseName, Column1, Column2, *SkipFirstRow*)

Description Adds a new column containing cells of one column divided by another.

Arguments *DatabaseName:* Name of the database.

Column1: Column containing dividends.

Column2: Column containing divisors.

SkipFirstRow (option): If *TRUE*, the first row will not be modified.

Return The column number of the new column.

Related `DatabaseMultiplyColumns`
`DatabaseSubtractColumns`
`DatabaseSumColumns`

Example

```
DatabaseCreateTest( "database.txt", "1NUMBERS" )
DatabaseView( "database.txt" )
Column := DatabaseDivideColumns( "database.txt", 2, 3 )
MsgBox, The result is stored in column %Column%.
DatabaseView( "database.txt" )
```

DatabaseDivision(DatabaseName,
Divisor,
Row,
Column,
SkipFirstRow)

Description Divides specified cells by a specified divisor.

Arguments

DatabaseName: Name of the database.

Divisor: Specified divisor.

Row (option): If specified, only this row will be modified.

Column (option): If specified, only this column will be modified.

SkipFirstRow (option): If *TRUE*, the first row will not be modified.

Return Nothing.

Related

`DatabaseAddition`
`DatabaseMultiplication`
`DatabaseSubtraction`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseDivision( "database.txt", 2, 3 )
DatabaseView( "database.txt" )
```

DatabaseDuplicateColumn(DatabaseName, *Column*)

Description Creates a copy of a column. The duplicate is placed in the rightmost end of database.

Arguments *DatabaseName:* Name of the database.

Column (optional): Column to duplicate. Default is the rightmost column.

Return Column number of new (duplicate) colum.

Related `DatabaseDuplicateRow`

`DatabaseRemoveColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseDuplicateColumn( "database.txt", 3 )
DatabaseView( "database.txt" )
```

DatabaseDuplicateRow(DatabaseName, *Row*)

Description Creates a copy of a row. The duplicate is placed in the bottommost end of database.

Arguments *DatabaseName:* Name of the database.

Row (optional): Row to duplicate. Default is the bottommost row.

Return Row number of new (duplicate) row.

Related `DatabaseDuplicateColumn`

`DatabaseRemoveRow`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseDuplicateRow( "database.txt", 3 )
DatabaseView( "database.txt" )
```

DatabaseExp(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with exp() of each value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseAbsoluteValue`
`DatabaseLog`
`DatabaseModulo`
`DatabaseNaturalLog`
`DatabasePower`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseExp( "database.txt", , 2 )
DatabaseView( "database.txt" )
```

DatabaseExportCSV(InputFile, OutputFile, *Overwrite*)

Description Export a tab-separated database as a comma-separated database (i.e., convert a tab-separated database file to a comma-separated database file).

Arguments *InputFile:* Name of input file (tab-separated database).

OutputFile: Name of output file (comma-separated database).

Overwrite (optional): If *TRUE*, overwrite file if output filename exists.

Return 1 if no indication of failure.
0 if problem occurred.

Related `DatabaseImportCSV`

Example

```
DatabaseCreateTest( "input.txt", "AIRPORTS" )
DatabaseExportCSV( "input.txt", "output.csv" )
```

DatabaseFind(DatabaseName, SearchTerm, *Row*, *Column*, *Occurrence*, *CaseSensitive*, *SkipFirstRow*)

Description Find cell containing a specified search term. The search term is matched with the *entire* cell content. If multiple cells contain the search term, the first cell (from left to right, and up to down) is returned (can be specified using *Occurrence*).

Arguments

DatabaseName: Name of the database.

SearchTerm: Term to search for.

Row (optional): If specified, only cells in this row are searched.

Column (optional): If specified, only cells in this column are searched.

Occurrence (optional): Specify to search for the *n*th match. Default is 1 (i.e., *n*=1).

CaseSensitive (optional): If *TRUE*, the search is case-sensitive. Default is *FALSE*.

SkipFirstRow (optional): If *TRUE*, cells in the first row are not searched.

Return Cell location (as an array) of first match.

Related `DatabaseMatchGetColumn`
`DatabaseMatchGetRowNumber`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
Cell := DatabaseFind( "database.txt", "ORD" )
MsgBox, % "Matching cell: (" . Cell[1] . "," . Cell[2] . ")"
```

DatabaseFloor(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with each cell value rounded down (to closest integer).

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseCeiling`

`DatabaseRound`

Example

```
DatabaseCreateTest( "database.txt", "0.0NUMBERS10.0" )
DatabaseView( "database.txt" )
DatabaseFloor( "database.txt", 2 )
DatabaseView( "database.txt" )
```

DatabaseGet(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Returns a row, column or cell from a database. Both Row and Column cannot be unspecified.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only cells from this row will be returned.

Column (optional): If specified, only cells from this column will be returned.

SkipFirstRow (optional): If *TRUE*, the first row will not be returned.

Return A row (array) is returned if only Row is specified (not Column).
A column (array) is returned if only Column is specified (not Row).
A cell is returned if both Row and Column are specified.

Related Nothing.

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
CellContent := DatabaseGet( "database.txt", 3, 2 )
MsgBox, Cell in row 3 and column 2 contains: %CellContent%
ColumnArray := DatabaseGet( "database.txt", , 3 )
For Index, Value in ColumnArray
    MsgBox, Cell number %Index% in column 3 contains: %Value%
```

DatabaseGetEncoding(DatabaseName)

Description Returns the encoding of the database file.

Arguments *DatabaseName:* Name of the database.

Return The encoding of the database file.

Related Nothing.

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Encoding := DatabaseGetEncoding( "database.txt" )
MsgBox, Database encoding: %Encoding%
```

DatabaseGetLargest(DatabaseName, *Row, Column, Order, SkipFirstRow*)

Description Returns the nth (set by Order) largest value from the specified cells.

Arguments

DatabaseName: Name of the database.

Row (optional): If specified, only this row is considered.

Column (optional): If specified, only this column is considered.

Order (optional): Specify to return the *n*th largest number, where *n* is specified by this argument. *Order* is by default 1.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The nth largest value from the specified cells.

Related

`DatabaseGetSmallest`

`DatabaseMoveRow`

`DatabaseSortByColumn`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
YoungestAirport := DatabaseGetLargest( "database.txt", , 4, , TRUE )
MsgBox, The youngest airport opened %YoungestAirport%.
```

DatabaseGetMean(DatabaseName, *Row, Column, SkipFirstRow*)

Description	Returns the (arithmetic) mean value from the specified cells.
Arguments	*DatabaseName:* Name of the database. *Row (optional):* If specified, only this row is considered. *Column (optional):* If specified, only this column is considered. *SkipFirstRow (optional):* If *TRUE*, the first row will not be considered.
Return	The mean value from the specified cells.
Related	`DatabaseGetMedian`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
Mean := DatabaseGetMean( "database.txt", , 5, TRUE )
MsgBox, The mean airport elevation above sealevel is %Mean% m.
```

DatabaseGetMedian(DatabaseName, *Row, Column, SkipFirstRow*)

Description Returns the median value from the specified cells.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row is considered.

Column (optional): If specified, only this column is considered.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The median value from the specified cells.

Related `DatabaseGetMean`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
Median := DatabaseGetMedian( "database.txt", , 5, TRUE )
MsgBox, The median airport elevation above sea-level is %Median% m.
```

DatabaseGetNumberOfCells(DatabaseName, *SkipFirstRow*)

Description Returns the number of cells in the database.

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The number of database cells.

Related `DatabaseGetNumberOfColumns`
`DatabaseGetNumberOfRows`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
NumberOfCells := DatabaseGetNumberOfCells( "database.txt" )
MsgBox, The database contains %NumberOfCells% cells.
```

DatabaseGetNumberOfColumns(DatabaseName)

Description Returns the number of columns in the database.

Arguments *DatabaseName:* Name of the database.

Return The number of database columns.

Related `DatabaseGetNumberOfCells`
`DatabaseGetNumberOfRows`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
NumberOfColumns := DatabaseGetNumberOfColumns( "database.txt" )
MsgBox, The database contains %NumberOfColumns% columns.
```

DatabaseGetNumberOfRows(DatabaseName, *SkipFirstRow*)

Description Returns the number of rows in the database.

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The number of database rows.

Related `DatabaseGetNumberOfCells`

`DatabaseGetNumberOfColumns`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
NumberOfRows := DatabaseGetNumberOfRows( "database.txt" )
MsgBox, The database contains %NumberOfRows% rows.
```

DatabaseGetSmallest(DatabaseName, *Row*, *Column*, *Order*, *SkipFirstRow*)

Description Returns the *n*th (set by *Order*) smallest value from the specified cells.

Arguments

DatabaseName: Name of the database.

Row (optional): If specified, only this row is considered.

Column (optional): If specified, only this column is considered.

Order (optional): Specify to return the *n*th smallest number, where *n* is specified by this argument. *Order* is by default 1.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The nth smallest value from the specified cells.

Related

`DatabaseGetLargest`

`DatabaseMoveRow`

`DatabaseSortByColumn`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
OldestAirport := DatabaseGetSmallest( "database.txt", , 4, , TRUE )
MsgBox, The oldest airport opened %OldestAirport%.
```

DatabaseGetSum(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Returns the sum of all values in the specified cells.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only cells in this row are considered.

Column (optional): If specified, only cells in this column are considered.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return The sum of all values in the specified cells.

Related Nothing.

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
Sum := DatabaseGetSum( "database.txt", , 5, TRUE )
MsgBox, The sum of all airports elevation above sea-level is %Sum%.
```

DatabaseGetRandomCell(DatabaseName, *Row*, *Column*, *SkipFirstRow*)

Description Returns the content of a randomly selected database cell.

Arguments

DatabaseName: Name of the database.

Row (optional): If specified, a cell is randomly selected from this row.

Column (optional): If specified, a cell is randomly selected from this column.

SkipFirstRow (optional): If *TRUE*, a randomly selected cell will not be located on the first row.

Return Cell content of a randomly selected cell.

Related

`DatabaseGetRandomCellLocation`

`DatabaseGetRandomColumn`

`DatabaseGetRandomColumnNumber`

`DatabaseGetRandomRow`

`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
RandomCell := DatabaseGetRandomCell( "database.txt" )
MsgBox, Content of randomly selected cell: %RandomCell%
```

DatabaseGetRandomCellLocation(*DatabaseName, Row, Column, SkipFirstRow*)

Description Returns the location (as an array) of a randomly selected database cell.

Arguments

DatabaseName: Name of the database.

Row (optional): If specified, a cell is randomly selected from this row.

Column (optional): If specified, a cell is randomly selected from this column.

SkipFirstRow (optional): If *TRUE*, a randomly selected cell will not be located on the first row.

Return Cell location (as an array) of a randomly selected cell.

Related

`DatabaseGetRandomCell`

`DatabaseGetRandomColumn`

`DatabaseGetRandomColumnNumber`

`DatabaseGetRandomRow`

`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
Location := DatabaseGetRandomCellLocation( "database.txt" )
MsgBox, % "Random cell: (" . Location[1] . "," . Location[2] . ")"
```

DatabaseGetRandomColumn(DatabaseName)

Description Returns a randomly selected column (as an array).

Arguments *DatabaseName:* Name of the database.

Return A randomly selected column (as an array).

Related `DatabaseGetRandomCell`
`DatabaseGetRandomCellLocation`
`DatabaseGetRandomColumnNumber`
`DatabaseGetRandomRow`
`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
RandomColumn := DatabaseGetRandomColumn( "database.txt" )
For Index, Value in RandomColumn
    ColumnElements .= Value . " "
MsgBox, Elements of randomly selected column: %ColumnElements%
```

DatabaseGetRandomColumnNumber(DatabaseName)

Description Returns the column number of a randomly selected column.

Arguments *DatabaseName:* Name of the database.

Return The column number of a randomly selected column.

Related

`DatabaseGetRandomCell`
`DatabaseGetRandomCellLocation`
`DatabaseGetRandomColumn`
`DatabaseGetRandomRow`
`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseColumn := DatabaseGetRandomColumnNumber( "database.txt" )
MsgBox, Randomly selected column number: %DatabaseColumn%
```

DatabaseGetRandomRow(DatabaseName, *SkipFirstRow*)

Description Returns a randomly selected row (as an array).

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will selected.

Return A randomly selected row (as an array).

Related

`DatabaseGetRandomCell`
`DatabaseGetRandomCellLocation`
`DatabaseGetRandomColumn`
`DatabaseGetRandomColumnNumber`
`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRow := DatabaseGetRandomRowNumber( "database.txt", TRUE )
MsgBox, Randomly selected row number: %DatabaseRow%
```

DatabaseGetRandomRowNumber(DatabaseName, *SkipFirstRow*)

Description Returns the row number of a randomly selected row.

Arguments *DatabaseName:* Name of the database.

Return The row number of a randomly selected row.

Related

`DatabaseGetRandomCell`
`DatabaseGetRandomCellLocation`
`DatabaseGetRandomColumn`
`DatabaseGetRandomColumnNumber`
`DatabaseGetRandomRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRow := DatabaseGetRandomRowNumber( "database.txt", TRUE )
MsgBox, Randomly selected row number: %DatabaseRow%
```

DatabaseImportCSV(*InputFile, OutputFile, Overwrite*)

Description Import a comma-separated database (i.e., convert a comma-separated file to a tab-separated file).

Arguments *InputFile:* Filename of comma-separated database to import.

OutputFile (optional): Filename for storing tab-separated conversion of database.

Overwrite (optional): If *TRUE*, overwrite file if it exists.

Return 1 if no indication of failure.
0 if problem occurred.

Related `DatabaseExportCSV`

Example

```
; Place a .csv-file ("input.csv") in the working directory.
DatabaseImportCSV( "input.csv", "output.txt" )
DatabaseView( "output.txt" )
```

DatabaseInsertColumn(DatabaseName, Column, *Row1, Row2, ...*)

Description Inserts a column at a specified location. Up to 50 cells of the new column can be specified. No existing column is removed, but moved accordingly to make space for the new column.

Arguments *DatabaseName:* Name of the database.

Column: Column number of the new column.

Row1 (optional): Content of the first cell of the new column.

Row2 (optional): Content of the second cell of the new column.

Return Nothing.

Related `DatabaseAddColumn`

`DatabaseInsertRow`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseInsertColumn( "database.txt", 3, "First", , "Third" )
DatabaseView( "database.txt" )
```

DatabaseInsertRow(DatabaseName,
Row,
Column1, Column2, ...)

Description Inserts a row at a specified location. Up to 50 cells of the new row can be specified. No existing row is removed, but moved accordingly to make space for the new row.

Arguments *DatabaseName:* Name of the database.

Row: Row number of the new row.

Column1 (optional): Content of the first cell of the new row.

Column2 (optional): Content of the second cell of the new row.

Return Nothing.

Related `DatabaseAddRow`

`DatabaseInsertColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseInsertRow( "database.txt", 3, "First", , "Third" )
DatabaseView( "database.txt" )
```

DatabaseIsNumeric(DatabaseName, *Row*, *Column*, *SkipFirstRow*)

Description Returns 1 if all cells are numeric, 0 otherwise. Using the arguments, the function can be limited to a part of the database.

Arguments

DatabaseName: Name of the database.

Row (optional): If specified, only cells in this row are considered.

Column (optional): If specified, only cells in this column are considered.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return 1 if all cells are numeric.
0 if at least one cell is not numeric.

Related Nothing.

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
if DatabaseIsNumeric( "database.txt" )
    MsgBox, All cells are numeric.
else
    MsgBox, Not all cells are numeric.
```

DatabaseKeepIfEqual(DatabaseName,
Column,
Value,
SkipFirstRow)

Description Keeps only rows for which a column value is equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Value: The value column content shall equal to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfGreater`
`DatabaseKeepIfGreaterOrEqual`
`DatabaseKeepIfLess`
`DatabaseKeepIfLessOrEqual`
`DatabaseKeepIfNotEqual`
`DatabaseRemoveIfEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseKeepIfGreater(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Keeps only rows for which a column value is greater than a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be greater than to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfEqual`
`DatabaseKeepIfGreaterOrEqual`
`DatabaseKeepIfLess`
`DatabaseKeepIfLessOrEqual`
`DatabaseKeepIfNotEqual`
`DatabaseRemoveIfGreater`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfGreater( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseKeepIfGreaterOrEqual(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Keeps only rows for which a column value is greater than or equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be greater than or equal to in order to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfEqual`

`DatabaseKeepIfGreater`

`DatabaseKeepIfLess`

`DatabaseKeepIfLessOrEqual`

`DatabaseKeepIfNotEqual`

`DatabaseRemoveIfGreaterOrEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfGreaterOrEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseKeepIfLess(DatabaseName,
Column,
Threshold,
SkipFirstRow)

Description Keeps only rows for which a column value is less than a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be less than to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfEqual`
`DatabaseKeepIfGreater`
`DatabaseKeepIfGreaterOrEqual`
`DatabaseKeepIfLessOrEqual`
`DatabaseKeepIfNotEqual`
`DatabaseRemoveIfLess`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfLess( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseKeepIfLessOrEqual(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Keeps only rows for which a column value is less than or equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be less than or equal to in order to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfEqual`

`DatabaseKeepIfGreater`

`DatabaseKeepIfGreaterOrEqual`

`DatabaseKeepIfLess`

`DatabaseKeepIfNotEqual`

`DatabaseRemoveIfLessOrEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfLessOrEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseKeepIfNotEqual(DatabaseName,
Column,
Value,
SkipFirstRow)

Description Keeps only rows for which a column value is not equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Value: The value rows shall not equal to remain in database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related `DatabaseKeepIfEqual`
`DatabaseKeepIfGreater`
`DatabaseKeepIfGreaterOrEqual`
`DatabaseKeepIfLess`
`DatabaseKeepIfLessOrEqual`
`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseKeepIfNotEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseLog(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with the (base 10) logarithm of each value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseAbsoluteValue`
`DatabaseExp`
`DatabaseModulo`
`DatabaseNaturalLog`
`DatabasePower`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "1NUMBERS" )
DatabaseView( "database.txt" )
DatabaseLog( "database.txt", , 3 )
DatabaseView( "database.txt" )
```

DatabaseMatchCountRows(DatabaseName, *SkipFirstRow*, *Column1*, *Column2*, …)

Description Counts the number of rows with cell content as specified (Column1, Column2 etc.). A row is a considered a match if all cells of the row are as specified. Requirements can be set for the first 50 columns.

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Column1 (optional): If specified, only rows with this cell content in the first column are counted. If unspecified, no requirement is set for the first column value.

Column2 (optional): Same as Column1, but for the second column.

Return The number of matching rows (i.e., rows with column content as specified by Column1, Column2 etc.).

Related `DatabaseFind`

`DatabaseMatchGetColumn`

`DatabaseMatchGetRowNumber`

`DatabaseMatchSetColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Matches := DatabaseMatchCountRows( "database.txt", , "A1", , "A3" )
MsgBox, Number of matching rows: %Matches%
```

DatabaseMatchGetColumn(DatabaseName, Column, *SkipFirstRow, Column1, Column2, …*)

Description Retrieves column content from a row that matches a given set of column criteria (Column1, Column2 etc.). A row is a considered a match if all cells of the row contain what is specified by arguments. Requirements can be set for the first 50 columns. If multiple rows are meet the specified criteria, column content of the first match will be returned.

Arguments

DatabaseName: Name of the database.

Column: Column from which to return content (upon finding a matching row).

SkipFirstRow (optional): If TRUE, the first row will not be considered.

Column1 (optional): If specified, a row must have this cell content in the first column to not be disqualified as a match. If unspecified, no requirement is set for the first column content.

Column2 (optional): Same as Column1, but for the second column.

Return The cell content of a specified column from the first matching row.

Related

`DatabaseFind`

`DatabaseMatchCountRows`

`DatabaseMatchGetRowNumber`

`DatabaseMatchSetColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Content := DatabaseMatchGetColumn( "database.txt", 4, , "D1", "D2" )
MsgBox, Cell content of matching row: %Content%
```

DatabaseMatchGetRowNumber(DatabaseName, *SkipFirstRow*, *Column1*, *Column2*, …)

Description Retrieves a row number of a row that matches a given set of column criteria (Column1, Column2 etc.). A row is a considered a match if all cells of the row contain what is specified by arguments. Requirements can be set for the first 50 columns. If multiple rows are meet the specified criteria, the row number of the first match will be returned.

Arguments

DatabaseName: Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Column1 (optional): If specified, a row must have this cell content in the first column to not be disqualified as a match. If unspecified, no requirement is set for the first column content.

Column2 (optional): Same as *Column1*, but for the second column.

Return The row number of the first matching row.

Related

`DatabaseFind`

`DatabaseMatchCountRows`

`DatabaseMatchGetColumn`

`DatabaseMatchSetColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
Row := DatabaseMatchGetRowNumber( "database.txt", , "E1", , "E3" )
MsgBox, Row number of matching row: %Row%
```

DatabaseMatchSetColumn(DatabaseName, ColumnToSet, NewContent, *SkipFirstRow*, *Column1*, *Column2*, …)

Description Sets the column content of a specified column from a row that matches a given set of column criteria (Column1, Column2 etc.). A row is a considered a match if all cells of the row contain what is specified by arguments. Requirements can be set for the first 50 columns. If multiple rows are meet the specified criteria, column content of the first match will be returned.

Arguments

DatabaseName: Name of the database.

ColumnToSet: Column to modify (upon finding a matching row).

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Column1 (optional): If specified, a row must have this cell content in the first column to not be disqualified as a match. If unspecified, no requirement is set for the first column content.

Column2 (optional): Same as *Column1*, but for the second column.

Return Cell location of modified cell.

Related

`DatabaseFind`

`DatabaseMatchCountRows`

`DatabaseMatchGetColumn`

`DatabaseMatchGetRowNumber`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseAddNumerationColumn( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseMergeByColumns(Database1, Database2, DatabaseName, *Overwrite*)

Description Merges two databases by columns (i.e., stacks vertically). The two databases that are merged are not deleted, but merged into a new file.

Arguments

Database1: First database to merge (top part of new database).

Database2: Second database to merge (bottom part of new database).

DatabaseName: Name of the new database.

Overwrite (optional): Overwrite file if it already exists.

Return 1 if no indication of failure.
0 if problem occurred.

Related `DatabaseMergeByRows`

Example

```
DatabaseCreateTest( "db1.txt" )
DatabaseCreateTest( "db2.txt" )
DatabaseMergeByColumns( "db1.txt", "db2.txt", "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseMergeByRows(Database1, Database2, DatabaseName, *Overwrite*)

Description Merges two databases by rows (i.e., attaches horizontally). The two databases that are merged are not deleted, but merged into a new file.

Arguments

Database1: First database to merge (left part of new database).

Database2: Second database to merge (right part of new database).

DatabaseName: Name of the new database.

Overwrite (optional): Overwrite file if it already exists.

Return 1 if no indication of failure.
0 if problem occurred.

Related `DatabaseMergeByColumns`

Example

```
DatabaseCreateTest( "db1.txt" )
DatabaseCreateTest( "db2.txt" )
DatabaseMergeByRows( "db1.txt", "db2.txt", "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseModifyCell(DatabaseName,
Row,
Column,
NewContent)

Description Modifies the content of a cell.

Arguments *DatabaseName:* Name of the database.

Row: Row number of the cell.

Column: Column number of the cell.

NewContent: New cell content.

Return Nothing.

Related `DatabaseModifyColumn`

`DatabaseModifyRow`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseModifyCell( "database.txt", 2, 3, "New content" )
DatabaseView( "database.txt" )
```

DatabaseModifyColumn(DatabaseName, Column, *RemoveUnspecified*, *Row1*, *Row2*, …)

Description Modifies cells of a column. Up to 50 cells of the column can be specified.

Arguments

DatabaseName: Name of the database.

Column: Column to modify.

RemoveUnspecified (optional): If *TRUE*, unspecified cells will be emptied. The default value is *FALSE*.

Row1 (optional): If specified, the first cell of the column will be set to this content.

Row2 (optional): Same as Row1, but for the second column.

Return Nothing.

Related `DatabaseModifyCell`

`DatabaseModifyRow`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseModifyColumn( "database.txt", 4, , "First cell", , "Third" )
DatabaseView( "database.txt" )
```

DatabaseModifyRow(DatabaseName, Row, *RemoveUnspecified*, *Column1*, *Column2*, …)

Description Modifies cells of a row. Up to 50 cells of the row can be specified.

Arguments

DatabaseName: Name of the database.

Row: Row to modify.

RemoveUnspecified (optional): If *TRUE*, unspecified cells will be emptied. The default value is *FALSE*.

Column1 (optional): If specified, the first cell of the row will be set to this content.

Column2 (optional): Same as Row1, but for the second row.

Return Nothing.

Related

`DatabaseModifyCell`

`DatabaseModifyColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseModifyRow( "database.txt", 4, TRUE, "First", , "Third" )
DatabaseView( "database.txt" )
```

DatabaseModulo(DatabaseName,
Divisor,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with modulo of each value.

Arguments

DatabaseName: Name of the database.

Divisor (optional): Divisor used in the modulo operation.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related

`DatabaseAbsoluteValue`
`DatabaseExp`
`DatabaseLog`
`DatabaseNaturalLog`
`DatabasePower`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "100NUMBERS200" )
DatabaseView( "database.txt" )
DatabaseModulo( "database.txt", 3, , 4 )
DatabaseView( "database.txt" )
```

DatabaseMoveColumn(DatabaseName, OldLocation, NewLocation)

Description Moves a column to a new location within the database. The move is done without removing any other column.

Arguments *DatabaseName:* Name of the database.

OldLocation: Column that shall be moved.

NewLocation: New location of column.

Return Nothing.

Related `DatabaseMoveRow`

`DatabaseSwitchColumns`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseMoveColumn( "database.txt", 1, 4 )
DatabaseView( "database.txt" )
```

DatabaseMoveRow(DatabaseName,
OldLocation,
NewLocation)

Description Moves a row to a new location within the database. The move is done without removing any other row.

Arguments *DatabaseName:* Name of the database.

OldLocation: Row that shall be moved.

NewLocation: New location of row.

Return Nothing.

Related `DatabaseMoveColumn`

`DatabaseSwitchRows`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseMoveRow( "database.txt", 1, 4 )
DatabaseView( "database.txt" )
```

DatabaseMultiplication(DatabaseName,
Factor,
Row,
Column,
SkipFirstRow)

Description Multiplies each cell value by a factor.

Arguments

DatabaseName: Name of the database.

Factor: Factor by which each cell value is multiplied.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related

`DatabaseAddition`
`DatabaseDivision`
`DatabaseSubtraction`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseMultiplication( "database.txt", 5, , 3 )
DatabaseView( "database.txt" )
```

DatabaseMultiplyColumns(DatabaseName, Column1, Column2, *SKipFirstRow*)

Description Multiplies cells of one column by elements of another column, and stores the result in a new column. The new column is placed in the right end of the database.

Arguments

DatabaseName: Name of the database.

Column1: The first column with cell values for multiplication.

Column2: The second column with cell values for multiplication.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Column number of new column containing the result.

Related

`DatabaseDivideColumns`

`DatabaseSubtractColumns`

`DatabaseSumColumns`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS", , 2 )
DatabaseView( "database.txt" )
DatabaseMultiplyColumns( "database.txt", 1, 2 )
DatabaseView( "database.txt" )
```

DatabaseNaturalLog(DatabaseName, *Row*, *Column*, *SkipFirstRow*)

Description Replaces the content of specified cells with the (base e) natural logarithm of each value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseAbsoluteValue`
`DatabaseExp`
`DatabaseLog`
`DatabaseModulo`
`DatabasePower`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "1NUMBERS" )
DatabaseView( "database.txt" )
DatabaseNaturalLog( "database.txt", , 2 )
DatabaseView( "database.txt" )
```

DatabasePower(DatabaseName,
Power
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with each value to the nth power, where n is decided by the argument *Power*.

Arguments

DatabaseName: Name of the database.

Power: The power used in the exponentiation of cell values.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related

`DatabaseAbsoluteValue`
`DatabaseExp`
`DatabaseLog`
`DatabaseModulo`
`DatabaseNaturalLog`
`DatabaseSquareRoot`

Example

```
DatabaseCreateTest( "database.txt", "1NUMBERS" )
DatabaseView( "database.txt" )
DatabasePower( "database.txt", 3, , 2 )
DatabaseView( "database.txt" )
```

DatabaseRecycle(DatabaseName)

Description Recycles database (places the database in the Recycle Bin).

Arguments *DatabaseName:* Name of the database.

Return 1 if database was successfully placed in the Recycle Bin.
0 if database was not successfully placed in the Recycle Bin.

Related `DatabaseBackup`
`DatabaseCopy`
`DatabaseDelete`

Example

```
DatabaseCreateTest( "database.txt" )
if ( DatabaseRecycle( "database.txt" ) )
    MsgBox, Database recycled.
else
    MsgBox, Database not recycled.
```

DatabaseRemoveColumn(DatabaseName, *Column*)

Description Deletes a database column.

Arguments *DatabaseName:* Name of the database.

Column (optional): Column to delete. Default is the rightmost column.

Return Nothing.

Related `DatabaseDuplicateColumn`

`DatabaseRemoveRow`

Example

```
DatabaseCreateTest( "database.txt", , , 3 )
DatabaseView( "database.txt" )
DatabaseRemoveColumn( "database.txt", 2 )
DatabaseView( "database.txt" )
```

DatabaseRemoveDuplicatesByColumn(DatabaseName, Column, *SkipFirstRow*)

Description Remove rows with the same content in a specified column. All but the first occurrence of cell content that occurs multiple times are removed.

Arguments *DatabaseName:* Name of the database.

Column: Column from which row values are compared.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related `DatabaseRemoveDuplicates`

Example

```
DatabaseCreateTest( "database.txt", , 40 )
DatabaseView( "database.txt" )
Removed := DatabaseRemoveDuplicatesByColumn( "database.txt", 2 )
MsgBox, Number of removed rows: %Removed%
DatabaseView( "database.txt" )
```

DatabaseRemoveDuplicates(DatabaseName, *SkipFirstRow*)

Description Removes any row that is a complete copy (each column content is the same) of another row, and keeps only the first (top) row among duplicates.

Arguments *DatabaseName:* Name of the database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related `DatabaseRemoveDuplicatesByColumn`

Example

```
DatabaseCreateTest( "database.txt", , 40 )
DatabaseView( "database.txt" )
Removed := DatabaseRemoveDuplicates( "database.txt" )
MsgBox, Number of removed rows: %Removed%
DatabaseView( "database.txt" )
```

DatabaseRemoveIfEqual(DatabaseName,
Column,
Value
SkipFirstRow)

Description Removes rows for which a column value is equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Value: The value column content shall equal to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfEqual`
`DatabaseRemoveIfGreater`
`DatabaseRemoveIfGreaterOrEqual`
`DatabaseRemoveIfLess`
`DatabaseRemoveIfLessOrEqual`
`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveIfGreater(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Removes rows for which a column value is greater than a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be greater than to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfGreater`
`DatabaseRemoveIfEqual`
`DatabaseRemoveIfGreaterOrEqual`
`DatabaseRemoveIfLess`
`DatabaseRemoveIfLessOrEqual`
`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfGreater( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveIfGreaterOrEqual(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Removes rows for which a column value is greater than or equal to a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be greater than or equal to in order to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfGreaterOrEqual`
`DatabaseRemoveIfEqual`
`DatabaseRemoveIfGreater`
`DatabaseRemoveIfLess`
`DatabaseRemoveIfLessOrEqual`
`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfGreaterOrEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveIfLess(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Removes rows for which a column value is less than a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be less than to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfLess`
`DatabaseRemoveIfEqual`
`DatabaseRemoveIfGreater`
`DatabaseRemoveIfGreaterOrEqual`
`DatabaseRemoveIfLessOrEqual`
`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfLess( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveIfLessOrEqual(DatabaseName, Column, Threshold, *SkipFirstRow*)

Description Removes rows for which a column value is less than or equal to a specified value.

Arguments

DatabaseName: Name of the database.

Column: The column used to evaluate each row.

Threshold: The value rows shall be less than or equal to in order to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfLessOrEqual`

`DatabaseRemoveIfEqual`

`DatabaseRemoveIfGreater`

`DatabaseRemoveIfGreaterOrEqual`

`DatabaseRemoveIfLess`

`DatabaseRemoveIfNotEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfLessOrEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveIfNotEqual(DatabaseName, Column, Value, *SkipFirstRow*)

Description Removes rows for which a column value is not equal to a specified value.

Arguments *DatabaseName:* Name of the database.

Column: The column used to evaluate each row.

Value: The value rows shall not equal to be removed from database.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseKeepIfNotEqual`
`DatabaseRemoveIfEqual`
`DatabaseRemoveIfGreater`
`DatabaseRemoveIfGreaterOrEqual`
`DatabaseRemoveIfLess`
`DatabaseRemoveIfLessOrEqual`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseRemoveIfNotEqual( "database.txt", 4, 1936, TRUE )
DatabaseView( "database.txt" )
```

DatabaseRemoveNA(DatabaseName, *CaseSensitive*, *SkipFirstRow*)

Description Empties database cells containing *NA*.

Arguments *DatabaseName:* Name of the database.

CaseSensitive (optional): If *TRUE*, only fully capitalized "NA" is removed (and for example "Na" is not).

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed *NA*s.

Related `DatabaseAddNA`

Example

```
DatabaseCreateTest( "database.txt", , 3, 3 )
DatabaseModifyColumn( "database.txt", 2, , "NA", , "NA" )
DatabaseView( "database.txt" )
DatabaseRemoveNA( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseRemoveRow(DatabaseName, *Row*)

Description Deletes a database row.

Arguments *DatabaseName:* Name of the database.

Row (optional): Row to delete. Default is the bottommost column.

Return Nothing.

Related `DatabaseDuplicateRow`

`DatabaseRemoveColumn`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseRemoveRow( "database.txt", 4 )
DatabaseView( "database.txt" )
```

DatabaseReplace(DatabaseName, FindContent, ReplaceWith, *CaseSensitive*, *Row*, *Column*, *SkipFirstRow*)

Description Replaces a specified cell content with another specified cell content.

Arguments

DatabaseName: Name of the database.

FindContent: Cell content to replace.

ReplaceWith: Replacement content.

CaseSensitive (optional): If *TRUE*, cell content search is case-sensitive.

Row (optional): If specified, only cells in this row can be replaced.

Column (optional): If specified, only cells in this column can be replaced.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of replacements.

Related `DatabaseFind`

`DatabaseModifyCell`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseReplace( "database.txt", "1936", "New content" )
DatabaseView( "database.txt" )
```

DatabaseRound(DatabaseName, *DecimalPlaces*, *Row*, *Column*, *SkipFirstRow*)

Description Rounds cell values.

Arguments *DatabaseName:* Name of the database.

DecimalPlaces (optional): Number of decimals of rounded value (default is 0).

Row (optional): If specified, only values in this row are rounded.

Column (optional): If specified, only values in this column are rounded.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Nothing.

Related `DatabaseCeiling`

`DatabaseFloor`

Example

```
DatabaseCreateTest( "database.txt", "0.0NUMBERS10.0" )
DatabaseView( "database.txt" )
DatabaseRound( "database.txt", 1, , 3 )
DatabaseView( "database.txt" )
```

DatabaseSin(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with sine of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcSin`
`DatabaseCos`
`DatabaseTan`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseSin( "database.txt", 2 )
DatabaseView( "database.txt" )
```

DatabaseSortByColumn(DatabaseName, Column, *DecreasingSort*, *AlphabeticSort*, *SkipFirstRow*)

Description Sorts rows of database by content in a specified column.

Arguments

DatabaseName: Name of the database.

Column: Column by which rows are sorted.

DecreasingSort (optional): If *TRUE*, sorting is decreasing (default is *FALSE*).

AlphabeticSort (optional): If *TRUE*, sorting is alphabetic. If *FALSE* (default), sorting is numeric.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Number of removed rows.

Related

`DatabaseGetLargest`

`DatabaseGetSmallest`

`DatabaseMoveRow`

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseSortByColumn( "database.txt", 4, , , TRUE )
DatabaseView( "database.txt" )
```

DatabaseSquareRoot(DatabaseName, *Row*, *Column*, *SkipFirstRow*)

Description Replaces the content of specified cells with the square root of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseAbsoluteValue`
`DatabaseExp`
`DatabaseLog`
`DatabaseModulo`
`DatabaseNaturalLog`
`DatabasePower`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseSquareRoot( "database.txt", , 4 )
DatabaseView( "database.txt" )
```

DatabaseSubString(DatabaseName, *StartingPosition, Length, Row, Column SkipFirstRow*)

Description Replace the content of specified cells with a substring of the original content. This function is based on the `SubStr()` function in AHK, and the AHK documentation for this function may be helpful.

Arguments

DatabaseName: Name of the database.

StartingPosition (optional): Starting position for the substring. The leftmost starting position (the first character) is 1 (default value).

Length (optional): Length of substring (number of characters).

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Nothing.

Related `DatabaseTrimLeft`
`DatabaseTrimRight`

Example

```
DatabaseCreateTest( "database.txt", "LETTERS6" )
DatabaseView( "database.txt" )
DatabaseSubString( "database.txt", , 2, 4 )
DatabaseView( "database.txt" )
```

DatabaseSubtractColumns(DatabaseName, Column1, Column2, *SkipFirstRow*)

Description Adds a new column containing one column subtracted by another column.

Arguments *DatabaseName:* Name of the database.

Column1: Column which is subtracted by Column2.

Column2: Column which is subtracted from Column1.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Column number of the new column (containing the subtracted result).

Related `DatabaseDivideColumns`

`DatabaseMultiplyColumns`

`DatabaseSumColumns`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS", , 2 )
DatabaseView( "database.txt" )
DatabaseSubtractColumns( "database.txt", 1, 2 )
DatabaseView( "database.txt" )
```

DatabaseSubtraction(DatabaseName, SubtractBy, *Row*, *Column*, *SkipFirstRow*)

Description Subtracts a value from the specified cells.

Arguments *DatabaseName:* Name of the database.

SubtractBy: Value by which the cell content is subtracted.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Nothing.

Related `DatabaseAddition`
`DatabaseDivision`
`DatabaseMultiplication`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseSubtraction( "database.txt", 10, , 2 )
DatabaseView( "database.txt" )
```

DatabaseSumColumns(DatabaseName, Column1, Column2, *SkipFirstRow*)

Description Adds a new column containing the sum of (the corresponding cells of) two columns.

Arguments

DatabaseName: Name of the database.

Column1: First column in summation.

Column2: Second column in summation.

SkipFirstRow (optional): If *TRUE*, the first row will not be considered.

Return Column number of the new column (containing the summed result).

Related

`DatabaseDivideColumns`

`DatabaseMultiplyColumns`

`DatabaseSubtractColumns`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS", , 2 )
DatabaseView( "database.txt" )
DatabaseSumColumns( "database.txt", 1, 2 )
DatabaseView( "database.txt" )
```

DatabaseSwitchColumns(DatabaseName, Column1, Column2)

Description Switches the location of two database columns.

Arguments *DatabaseName:* Name of the database.

Column1: First database column to switch.

Column2: Second database column to switch.

Return Nothing.

Related `DatabaseMoveColumn`

`DatabaseSwitchRows`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseSwitchColumns( "database.txt", 1, 5 )
DatabaseView( "database.txt" )
```

DatabaseSwitchRows(DatabaseName,
Row1,
Row2)

Description Switches the location of two database columns.

Arguments *DatabaseName:* Name of the database.

Row1: First row to switch.

Row2: Second row to switch.

Return Nothing.

Related `DatabaseMoveRow`

`DatabaseSwitchColumns`

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
DatabaseSwitchRows( "database.txt", 1, 5 )
DatabaseView( "database.txt" )
```

DatabaseTan(DatabaseName,
Row,
Column,
SkipFirstRow)

Description Replaces the content of specified cells with tangent of each cell value.

Arguments *DatabaseName:* Name of the database.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseArcTan`
`DatabaseCos`
`DatabaseSin`

Example

```
DatabaseCreateTest( "database.txt", "NUMBERS" )
DatabaseView( "database.txt" )
DatabaseTan( "database.txt", , 5 )
DatabaseView( "database.txt" )
```

DatabaseTranspose(DatabaseName)

Description	Transposes database (i.e., turns columns into rows, and rows into columns).
Arguments	*DatabaseName:* Name of the database.
Return	Nothing.
Related	Nothing.

Example

```
DatabaseCreateTest( "database.txt", "AIRPORTS" )
DatabaseView( "database.txt" )
DatabaseTranspose( "database.txt" )
DatabaseView( "database.txt" )
```

DatabaseTrimLeft(DatabaseName,
Characters,
Row,
Column,
SkipFirstRow)

Description Removes a specified number of characters from each cell content (starting from left). Note that this function is based on `StringTrimLeft` of AHK, which is no longer recommended for use.

Arguments

DatabaseName: Name of the database.

Characters: Number of characters to remove (trim) from each cell.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified.

Return Nothing.

Related `DatabaseSubString`
`DatabaseTrimRight`

Example

```
DatabaseCreateTest( "database.txt", "LETTERS6" )
DatabaseView( "database.txt" )
DatabaseTrimLeft( "database.txt", 5, , 4 )
DatabaseView( "database.txt" )
```

DatabaseTrimRight(DatabaseName,
Characters,
Row,
Column,
SkipFirstRow)

Description Removes a specified number of characters from each cell content (starting from right). Note that this function is based on `StringTrimRight` of AHK, which is no longer recommended for use.

Arguments

DatabaseName: Name of the database.

Characters: Number of characters to remove (trim) from each cell.

Row (optional): If specified, only this row will be modified.

Column (optional): If specified, only this column will be modified.

SkipFirstRow (optional): If *TRUE*, the first row will not be modified. Default is *FALSE*.

Return Nothing.

Related `DatabaseSubString`

`DatabaseTrimLeft`

Example

```
DatabaseCreateTest( "database.txt", "LETTERS6" )
DatabaseView( "database.txt" )
DatabaseTrimRight( "database.txt", 5, , 4 )
DatabaseView( "database.txt" )
```

DatabaseView(DatabaseName, *UnPauseScript*)

Description Opens database in the graphical *Database Viewer*. The viewer cannot display more than 50 columns.

Arguments *DatabaseName:* Name of the database.

UnPauseScript (optional): If *TRUE*, script will not be paused when opening the Database Viewer. Default is *FALSE*.

Return Nothing.

Related Nothing.

Example

```
DatabaseCreateTest( "database.txt" )
DatabaseView( "database.txt" )
```

List of Functions

DatabaseAbsoluteValue
DatabaseAddColumn
DatabaseAddition
DatabaseAddNA
DatabaseAddNumerationColumn
DatabaseAddRandomColumn
DatabaseAddRow
DatabaseArcCos
DatabaseArcSin
DatabaseArcTan
DatabaseBackup
DatabaseCeiling
DatabaseCheck
DatabaseColumnSplitDelimiter
DatabaseColumnSplitLeft
DatabaseColumnSplitRight
DatabaseCompare
DatabaseCompareDimensions
DatabaseConcatenateColumns
DatabaseConcatenateRows
DatabaseCopy
DatabaseCos
DatabaseCreate
DatabaseCreateTest
DatabaseDelete
DatabaseDivideColumns
DatabaseDivision
DatabaseDuplicateColumn
DatabaseDuplicateRow
DatabaseExp
DatabaseExportCSV
DatabaseFind
DatabaseFloor
DatabaseGet
DatabaseGetEncoding
DatabaseGetLargest
DatabaseGetMean
DatabaseGetMedian
DatabaseGetNumberOfCells
DatabaseGetNumberOfColumns
DatabaseGetNumberOfRows
DatabaseGetRandomCell
DatabaseGetRandomCellLocation
DatabaseGetRandomColumn
DatabaseGetRandomColumnNumber
DatabaseGetRandomRow
DatabaseGetRandomRow
DatabaseGetSmallest
DatabaseGetSum
DatabaseImportCSV
DatabaseInsertColumn
DatabaseInsertRow
DatabaseIsNumeric
DatabaseKeepIfEqual
DatabaseKeepIfGreater
DatabaseKeepIfGreaterOrEqual
DatabaseKeepIfLess
DatabaseKeepIfLessOrEqual
DatabaseKeepIfNotEqual
DatabaseLog
DatabaseMatchCountRows
DatabaseMatchGetColumn
DatabaseMatchGetRowNumber
DatabaseMatchSetColumn
DatabaseMergeByColumns
DatabaseMergeByRows
DatabaseModifyCell
DatabaseModifyColumn
DatabaseModifyRow
DatabaseModulo
DatabaseMoveColumn
DatabaseMoveRow
DatabaseMultiplication
DatabaseMultiplyColumns
DatabaseNaturalLog
DatabasePower
DatabaseRecycle
DatabaseRemoveColumn
DatabaseRemoveDuplicates
DatabaseRemoveDuplicatesByColumn
DatabaseRemoveIfEqual
DatabaseRemoveIfGreater
DatabaseRemoveIfGreaterOrEqual
DatabaseRemoveIfLess
DatabaseRemoveIfLessOrEqual
DatabaseRemoveIfNotEqual
DatabaseRemoveNA
DatabaseRemoveRow
DatabaseReplace
DatabaseRound
DatabaseSin
DatabaseSortByColumn
DatabaseSquareRoot
DatabaseSubString
DatabaseSubtractColumns
DatabaseSubtraction
DatabaseSumColumns
DatabaseSwitchColumns
DatabaseSwitchRows
DatabaseTan
DatabaseTranspose
DatabaseTrimLeft
DatabaseTrimRight
DatabaseView