Version 17

JSL Syntax Reference

"The real voyage of discovery consists not in seeking new landscapes, but in having new eyes."

Marcel Proust

JMP Statistical Discovery LLC
SAS Campus Drive
Cary, North Carolina 27513-2414

17.1

The correct bibliographic citation for this manual is as follows: JMP Statistical Discovery LLC 2022–2023. *JMP® 17 JSL Syntax Reference*. Cary, NC: JMP Statistical Discovery LLC

JMP® 17 JSL Syntax Reference

Copyright © 2022–2023, JMP Statistical Discovery LLC, Cary, NC, USA

All rights reserved. Produced in the United States of America.

JMP Statistical Discovery LLC, SAS Campus Drive, Cary, North Carolina 27513-2414.

October 2022

March 2023

JMP® and all other JMP Statistical Discovery LLC product or service names are registered trademarks or trademarks of JMP Statistical Discovery LLC in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

JMP software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with JMP software, refer to http://support.sas.com/thirdpartylicenses.

Get the Most from JMP

Whether you are a first-time or a long-time user, there is always something to learn about JMP.

Visit JMP.com to find the following:

- live and recorded webcasts about how to get started with JMP
- video demos and webcasts of new features and advanced techniques
- details on registering for JMP training
- schedules for seminars being held in your area
- success stories showing how others use JMP
- the JMP user community, resources for users including examples of add-ins and scripts, a forum, blogs, conference information, and so on

jmp.com/getstarted

Contents

JSL Syntax Reference

1 Learn about JMP 9

Documentation and Additional Resources

Formatting Conventions in JMP Documentation 11
JMP Help 12
JMP Documentation Library 12
Additional Resources for Learning JMP 19
Search JMP 19
JMP Tutorials 19
Sample Data Tables 20
Learn about Statistical and JSL Terms 20
Learn JMP Tips and Tricks 20
JMP Tooltips 20
JMP User Community 21
Free Online Statistical Thinking Course 21
JMP New User Welcome Kit 21
Statistics Knowledge Portal 21
JMP Training 22
JMP Books by Users 22
The JMP Starter Window 22
JMP Technical Support 22

2 JSL Functions, Operators, and Messages 23

Summary of Commands

Assignment Functions 25
Character Functions 28
Character Pattern Functions 43
Comment Functions 53
Comparison Functions 54
Conditional and Logical Functions 59
Constant Functions 71
Date and Time Functions 72
Discrete Probability Functions 81
Display Functions 87

Expression Functions . . . 121
File Functions . . . 123
Financial Functions . . . 145
Graphics Functions . . . 150
HTTP Functions . . . 170
List Functions . . . 170
MATLAB Integration Functions . . . 177
MATLAB JSL Function Interfaces . . . 177
Matrix Functions . . . 184
Numeric Functions . . . 209
Optimization Functions . . . 212
Probability Functions . . . 216
Programming Functions . . . 246
Python Integration Functions . . . 263
R Integration Functions . . . 269
Random Functions . . . 274
Row Functions . . . 285
Row State Functions . . . 291
SAS Integration Functions . . . 294
SQL Functions . . . 312
Statistical Functions . . . 315
Transcendental Functions . . . 329
Trigonometric Functions . . . 336
Utility Functions . . . 339

3 JSL Messages . . . 371
Summary of Messages for Objects and Display Boxes

Alpha Shape Messages . . . 374
Associative Array Messages . . . 374
Class Messages . . . 375
Data Table Messages . . . 377
General Data Table Messages . . . 377
Column Messages . . . 407
Row Messages . . . 414
Data Filter Messages . . . 416
Data Feed Messages (Windows Only) . . . 419
Display Box Messages . . . 422
All Display Box Messages . . . 422
Axis Box Messages . . . 432
Border Box Messages . . . 437

Data Browser Box Message . . . 438
Data Filter Source Box Message . . . 438
Frame Box Messages . . . 438
Display 3D Box Message . . . 440
Excerpt Box Message . . . 441
Filter Col Selector Message . . . 441
Global Box Message . . . 441
Hier Box Message . . . 441
Matrix Box Messages . . . 441
Nom Axis Box Messages . . . 442
Number Col Box Messages . . . 443
Number Col Edit Box Messages . . . 446
Number Edit Box Message . . . 446
Outline Box Messages . . . 447
Panel Box Messages . . . 448
Plot Col Box Messages . . . 448
Slider Box and Range Slider Box Messages . . . 449
String Col Box Messages . . . 450
Tab Box Messages . . . 451
Table Box Messages . . . 452
Text Box Messages . . . 455
Tree Node and Tree Box Messages . . . 456
Triangulation Messages . . . 458
Window Messages . . . 460
Dynamic Link Library (DLL) Messages . . . 462
Image Messages . . . 464
Interactive HTML Messages . . . 467
Web Report Messages . . . 467
JMP Application Messages . . . 468
JMP App Messages . . . 469
JMP App Module Messages . . . 470
JMP App Module Instance Messages . . . 471
MATLAB Messages . . . 471
Namespace Messages . . . 475
Platform Messages . . . 477
Bubble Plot Messages . . . 482
DOE Messages . . . 483
Partition Messages . . . 483
Response Screening Messages . . . 484
Tabulate Messages . . . 484

Python Integration Messages ... 485
R Integration Messages ... 488
SAS Integration Messages ... 492
Metadata Server Object Messages ... 492
SAS Server Object Messages ... 493
Stored Processes Messages ... 502
SAS Results Messages ... 508
Schedule Messages ... 510
Segment Message ... 511
Sockets Messages ... 512
SQL Messages ... 515
Messages for Other Objects ... 519
Zip Archives ... 519
Journals ... 519

A SQL Functions Available for JMP Queries ... 521

Numeric SQL Functions ... 523
Date-Time SQL Functions ... 524
String SQL Functions ... 527
System SQL Functions ... 528
Aggregate SQL Functions ... 528

B References ... 531

C Technology License Notices ... 533

Chapter 1

Learn about JMP

Documentation and Additional Resources

Learn about JMP documentation, such as book conventions, descriptions of each JMP document, the Help system, and where to find additional support.

Contents

Formatting Conventions in JMP Documentation . . . 11
JMP Help . . . 12
JMP Documentation Library . . . 12
Additional Resources for Learning JMP . . . 19
Search JMP . . . 19
JMP Tutorials . . . 19
Sample Data Tables . . . 20
Learn about Statistical and JSL Terms . . . 20
Learn JMP Tips and Tricks . . . 20
JMP Tooltips . . . 20
JMP User Community . . . 21
Free Online Statistical Thinking Course . . . 21
JMP New User Welcome Kit . . . 21
Statistics Knowledge Portal . . . 21
JMP Training . . . 22
JMP Books by Users . . . 22
The JMP Starter Window . . . 22
JMP Technical Support . . . 22

Formatting Conventions in JMP Documentation

These conventions help you relate written material to information that you see on your screen:

- Sample data table names, column names, pathnames, filenames, file extensions, and folders appear in Helvetica (or sans-serif online) font.
- Code appears in `Lucida Sans Typewriter` (or monospace online) font.
- Code output appears in *`Lucida Sans Typewriter`* italic (or monospace italic online) font and is indented farther than the preceding code.
- **Helvetica bold** formatting (or bold sans-serif online) indicates items that you select to complete a task:
  - buttons
  - check boxes
  - commands
  - list names that are selectable
  - menus
  - options
  - tab names
  - text boxes
- The following items appear in italics:
  - words or phrases that are important or have definitions specific to JMP
  - book titles
  - variables
- Features that are for JMP Pro only are noted with the JMP Pro icon JMP PRO. For an overview of JMP Pro features, visit jmp.com/software/pro.

Note: Special information and limitations appear within a Note.

Tip: Helpful information appears within a Tip.

JMP Help

JMP Help in the Help menu enables you to search for information about JMP features, statistical methods, and the JMP Scripting Language (or *JSL*). You can open JMP Help in several ways:

- Search and view JMP Help on Windows by selecting **Help > JMP Help**.
- On Windows, press the F1 key to open the Help system in the default browser.
- Get help on a specific part of a data table or report window. Select the Help tool from the **Tools** menu and then click anywhere in a data table or report window to see the Help for that area.
- Within a JMP window, click the **Help** button.

Note: The JMP Help is available for users with Internet connections. Users without an Internet connection can search all books in a PDF file by selecting **Help > JMP Documentation Library**. See "JMP Documentation Library" for more information.

JMP Documentation Library

The Help system content is also available in one PDF file called *JMP Documentation Library*. Select **Help > JMP Documentation Library** to open the file. You can also download the Documentation PDF Files add-in if you prefer searching individual PDF files of each document in the JMP library. Download the available add-ins from community.jmp.com.

The following table describes the purpose and content of each document in the JMP library.

Document Title	Document Purpose	Document Content
Discovering JMP	If you are not familiar with JMP, start here.	Introduces you to JMP and gets you started creating and analyzing data. Also learn how to share your results.
Using JMP	Learn about JMP data tables and how to perform basic operations.	Covers general JMP concepts and features that span across all of JMP, including importing data, modifying columns properties, sorting data, and connecting to SAS.

Document Title	Document Purpose	Document Content
Basic Analysis	Perform basic analysis using this document.	Describes these Analyze menu platforms: • Distribution • Fit Y by X • Tabulate • Text Explorer Covers how to perform bivariate, one-way ANOVA, and contingency analyses through Analyze > Fit Y by X. How to approximate sampling distributions using bootstrapping and how to perform parametric resampling with the Simulate platform are also included.
Essential Graphing	Find the ideal graph for your data.	Describes these Graph menu platforms: • Graph Builder • Scatterplot 3D • Contour Plot • Bubble Plot • Parallel Plot • Cell Plot • Scatterplot Matrix • Ternary Plot • Treemap • Chart • Overlay Plot The book also covers how to create background and custom maps.
Profilers	Learn how to use interactive profiling tools, which enable you to view cross-sections of any response surface.	Covers all profilers listed in the Graph menu. Analyzing noise factors is included along with running simulations using random inputs.

Document Title	Document Purpose	Document Content
Design of Experiments Guide	Learn how to design experiments and determine appropriate sample sizes.	Covers all topics in the DOE menu.
Fitting Linear Models	Learn about Fit Model platform and many of its personalities.	Describes these personalities, all available within the Analyze menu Fit Model platform: • Standard Least Squares • Stepwise • Generalized Regression • Mixed Model • Generalized Linear Mixed Model • MANOVA • Loglinear Variance • Nominal Logistic • Ordinal Logistic • Generalized Linear Model

Document Title	Document Purpose	Document Content
Predictive and Specialized Modeling	Learn about additional modeling techniques.	Describes these Analyze > Predictive Modeling menu platforms: • Neural • Partition • Bootstrap Forest • Boosted Tree • K Nearest Neighbors • Naive Bayes • Support Vector Machines • Model Comparison • Model Screening • Make Validation Column • Formula Depot Describes these Analyze > Specialized Modeling menu platforms: • Fit Curve • Nonlinear • Functional Data Explorer • Gaussian Process • Time Series • Matched Pairs Describes these Analyze > Screening menu platforms: • Explore Outliers • Explore Missing Values • Explore Patterns • Response Screening • Predictor Screening • Association Analysis • Process History Explorer

Document Title	Document Purpose	Document Content
Multivariate Methods	Read about techniques for analyzing several variables simultaneously.	Describes these Analyze > Multivariate Methods menu platforms: • Multivariate • Principal Components • Discriminant • Partial Least Squares • Multiple Correspondence Analysis • Structural Equation Models • Factor Analysis • Multidimensional Scaling • Multivariate Embedding • Item Analysis Describes these Analyze > Clustering menu platforms: • Hierarchical Cluster • K Means Cluster • Normal Mixtures • Latent Class Analysis • Cluster Variables

Document Title	Document Purpose	Document Content
Quality and Process Methods	Read about tools for evaluating and improving processes.	Describes these Analyze > Quality and Process menu platforms: • Control Chart Builder and individual control charts • Measurement Systems Analysis (EMP and Type 1 Gauge) • Variability / Attribute Gauge Charts • Process Screening • Process Capability • Model Driven Multivariate Control Chart • Legacy Control Charts • Pareto Plot • Diagram • Manage Limits • OC Curves
Reliability and Survival Methods	Learn to evaluate and improve reliability in a product or system and analyze survival data for people and products.	Describes these Analyze > Reliability and Survival menu platforms: • Life Distribution • Fit Life by X • Cumulative Damage • Recurrence Analysis • Degradation • Repeated Measures Degradation • Destructive Degradation • Reliability Forecast • Reliability Growth • Reliability Block Diagram • Repairable Systems Simulation • Survival • Fit Parametric Survival • Fit Proportional Hazards

Document Title	Document Purpose	Document Content
Consumer Research	Learn about methods for studying consumer preferences and using that insight to create better products and services.	Describes these Analyze > Consumer Research menu platforms: • Categorical • Choice • MaxDiff • Uplift • Multiple Factor Analysis
Genetics	Learn about methods that are available in JMP to help you analyze your genetic data and use that data to simulate a breeding program to predict the optimum genetic crosses to make.	Describes these Analyze > Genetics menu platforms: • Marker Statistics • Marker Simulation
Scripting Guide	Learn about taking advantage of the powerful JMP Scripting Language (JSL).	Covers a variety of topics, such as writing and debugging scripts, manipulating data tables, constructing display boxes, and creating JMP applications.
JSL Syntax Reference	Read about many JSL functions on functions and their arguments, and messages that you send to objects and display boxes.	Includes syntax, examples, and notes for JSL commands.

Additional Resources for Learning JMP

In addition to reading JMP help, you can also learn about JMP using the following resources:

- "Search JMP"
- "JMP Tutorials"
- "Sample Data Tables"
- "Learn about Statistical and JSL Terms"
- "Learn JMP Tips and Tricks"
- "JMP Tooltips"
- "JMP User Community"
- "Free Online Statistical Thinking Course"
- "JMP New User Welcome Kit"
- "Statistics Knowledge Portal"
- "JMP Training"
- "JMP Books by Users"
- "The JMP Starter Window"

Search JMP

If you are not sure where to find a statistical procedure, do a search across JMP. Results are tailored to the window that you launch the search from, such as a data table or report.

1. Click **Help** > **Search JMP**. Or, press Ctrl+comma.
2. Enter your search text.
3. Click the result that contains the procedure that you want.

   On the right, you can see a description and the location of the procedure.
4. Click the corresponding button to open or go to a result.

JMP Tutorials

You can access JMP tutorials by selecting **Help > Tutorials**.

If you are not familiar with JMP, start with the **Beginners Tutorial**. It steps you through the JMP interface and explains the basics of using JMP.

The rest of the tutorials help you with specific aspects of JMP, such as designing an experiment and comparing a sample mean to a constant.

Sample Data Tables

All of the examples in the JMP documentation suite use sample data. Select **Help > Sample Data Folder** to open the sample data directory.

To view an alphabetized list of sample data tables or view sample data within categories, select **Help > Sample Index**.

Sample data tables are installed in the following directory:

On Windows: C:\Program Files\SAS\JMP\17\Samples\Data

On macOS: \Library\Application Support\JMP\17\Samples\Data

In JMP Pro, sample data is installed in the JMPPRO (rather than JMP) directory.

To view examples using sample data, select **Help > Sample Index** and navigate to the Teaching Resources section.

Learn about Statistical and JSL Terms

For help with statistical terms, select Help > Statistics Index. For help with JSL scripting and examples, select **Help > Scripting Index**.

Statistics Index Provides definitions of statistical terms.

Scripting Index Lets you search for information about JSL functions, objects, and display boxes. You can also edit and run sample scripts from the Scripting Index and get help on the commands.

Learn JMP Tips and Tricks

When you first start JMP, you see the Tip of the Day window. This window provides tips for using JMP.

To turn off the Tip of the Day, clear the **Show tips at startup** check box. To view it again, select **Help > Tip of the Day**. Or, you can turn it off using the Preferences window.

JMP Tooltips

JMP provides descriptive tooltips (or *hover labels*) when you hover over items, such as the following:

- Menu or toolbar options
- Labels in graphs
- Text results in the report window (move your cursor in a circle to reveal)

- Files or windows in the Home Window
- Code in the Script Editor

Tip: On Windows, you can hide tooltips in the JMP Preferences. Select **File > Preferences > General** and then deselect **Show menu tips**. This option is not available on macOS.

JMP User Community

The JMP User Community provides a range of options to help you learn more about JMP and connect with other JMP users. The learning library of one-page guides, tutorials, and demos is a good place to start. And you can continue your education by registering for a variety of JMP training courses.

Other resources include a discussion forum, sample data and script file exchange, webcasts, and social networking groups.

To access JMP resources on the website, select **Help > JMP on the Web > JMP User Community** or visit https://community.jmp.com.

Free Online Statistical Thinking Course

Learn practical statistical skills in this free online course on topics such as exploratory data analysis, quality methods, and correlation and regression. The course consists of short videos, demonstrations, exercises, and more. Visit jmp.com/statisticalthinking.

JMP New User Welcome Kit

The JMP New User Welcome Kit is designed to help you quickly get comfortable with the basics of JMP. You'll complete its thirty short demo videos and activities, build your confidence in using the software, and connect with the largest online community of JMP users in the world. Visit jmp.com/welcome.

Statistics Knowledge Portal

The Statistics Knowledge Portal combines concise statistical explanations with illuminating examples and graphics to help visitors establish a firm foundation upon which to build statistical skills. Visit jmp.com/skp.

JMP Training

SAS offers training on a variety of topics led by a seasoned team of JMP experts. Public courses, live web courses, and on-site courses are available. You might also choose the online e-learning subscription to learn at your convenience. Visit jmp.com/training.

JMP Books by Users

Additional books about using JMP that are written by JMP users are available on the JMP website. Visit jmp.com/books.

The JMP Starter Window

The JMP Starter window is a good place to begin if you are not familiar with JMP or data analysis. Options are categorized and described, and you launch them by clicking a button. The JMP Starter window covers many of the options found in the Analyze, Graph, Tables, and File menus. The window also lists JMP Pro features and platforms.

- To open the JMP Starter window, select **View** (**Window** on macOS) **> JMP Starter**.
- To display the JMP Starter automatically when you open JMP on Windows, select **File > Preferences > General**, and then select **JMP Starter** from the Initial JMP Window list. On macOS, select **JMP > Preferences > General > Initial JMP Starter Window**.

JMP Technical Support

JMP technical support is provided by statisticians and engineers educated in SAS and JMP, many of whom have graduate degrees in statistics or other technical disciplines.

Many technical support options are provided at jmp.com/support, including the technical support phone number.

Chapter 2

JSL Functions, Operators, and Messages

Summary of Commands

This topic provides abbreviated descriptions for many of JMP's functions, operators, and general object messages. For complete information about functions, see the JMP Scripting Index. In JMP, select **Help > Scripting Index**.

For information about platform messages, see the *Scripting Guide*.

Contents

Assignment Functions . . . 25
Character Functions . . . 28
Character Pattern Functions . . . 43
Comment Functions . . . 53
Comparison Functions . . . 54
Conditional and Logical Functions . . . 59
Constant Functions . . . 71
Date and Time Functions . . . 72
Discrete Probability Functions . . . 81
Display Functions . . . 87
Expression Functions . . . 121
File Functions . . . 123
Financial Functions . . . 145
Graphics Functions . . . 150
HTTP Functions . . . 170
List Functions . . . 170
MATLAB Integration Functions . . . 177
    MATLAB JSL Function Interfaces . . . 177
Matrix Functions . . . 184
Numeric Functions . . . 209
Optimization Functions . . . 212
Probability Functions . . . 216
Programming Functions . . . 246
Python Integration Functions . . . 263
R Integration Functions . . . 269
Random Functions . . . 274
Row Functions . . . 285
Row State Functions . . . 291
SAS Integration Functions . . . 294
SQL Functions . . . 312
Statistical Functions . . . 315
Transcendental Functions . . . 329
Trigonometric Functions . . . 336
Utility Functions . . . 339

Assignment Functions

JSL also provides operators for in-place arithmetic, or *assignment operators*. These operations are all done in place, meaning that the result of the operation is assigned to the first argument. The most basic assignment operator is the = operator (or the equivalent function `Assign`). For example, if *a* is 3 and you do `a+=4`, then *a* becomes 7.

The first argument to an assignment function must be something capable of being assigned (an *L-value*). You cannot do something like `3+=4`, because 3 is just a value and cannot be reassigned. However, you can do something like `a+=4`, because *a* is a variable whose value you can set.

`Add To(`*a*`, `*b*`)`

a+=*b*

Description

Adds *a* and *b* and places the sum into *a*.

Returns

The sum.

Arguments

a Must be a variable.

b Can be a variable, a list, a number, or a matrix.

Notes

The first argument must be a variable, because its value must be able to accept a value change. A number as the first argument produces an error.

For `Add To()`: Only two arguments are permitted. If one or no argument is specified, `Add To()` returns a missing value. Any arguments after the first two are ignored.

For `a+=b`: More than two arguments can be strung together. JMP evaluates pairs from right to left, and each sum is placed in the left-hand variable. All arguments except the last must be a variable.

Example

```
a+=b+=c
```

JMP adds *b* and *c* and places the sum into *b*. Then JMP adds *a* and *b* and places the sum into *a*.

`Assign(a, b)`

a=b

Description

Places the value of *b* into *a*.

Returns

The new value of *a*.

Arguments

a Must be a variable.

b Can be a variable, number, or matrix.

Notes

a must be a variable, because it must be able to accept a value change. A number as the first argument produces an error. If *b* is some sort of expression, it's evaluated first and the result is placed into *a*.

`Divide To(a, b)`

a/=b

Description

Divides *a* by *b* and places the result into *a*.

Returns

The quotient.

Arguments

a Must be a variable.

b Can be a variable, number, or matrix.

`Multiply To(a, b)`

a=b*

Description

Multiplies *a* and *b* and places the product into *a*.

Returns

The product.

Arguments

a Must be a variable.

b Can be a variable, number, or matrix.

Notes

The first argument must be a variable, because its value must be able to accept a value change. A number as the first argument produces an error.

For `Multiply To()`: Only two arguments are permitted. If one or no argument is specified, `Multiply To()` returns a missing value. Any arguments after the first two are ignored.

For *a*`*=`*b*: More than two arguments can be strung together. JMP evaluates pairs from right to left, and each product is placed in the left-hand variable. All arguments except the last must be a variable.

Example

```
a*=b*=c
```

JMP multiplies *b* and *c* and places the product into *b*. Then JMP multiplies *a* and *b* and places the product into *a*.

`PostDecrement(`*a*`)`

a`--`

Description

Post-decrement. Subtracts 1 from *a* and places the difference into *a*.

Returns

a-1

Argument

a Must be a variable.

Notes

If *a*`--` or `Post Decrement(`*a*`)` is inside another expression, the expression is evaluated first, and then the decrement operation is performed. This expression is mostly used for loop control.

`Post Increment(`*a*`)`

a`++`

Description

Post-increment. Adds 1 to *a* and places the sum into *a*.

Returns

a+1

Argument

a Must be a variable.

Notes

If `a++` or `Post Increment(a)` is inside another expression, the expression is evaluated first, and then the increment operation is performed. Mostly used for loop control.

`Subtract To(a, b)`

`a-=b`

Description

Subtracts *b* from *a* and places the difference into *a*.

Returns

The difference.

Arguments

a Must be a variable.

b Can be a variable, number, or matrix.

Notes

The first argument must be a variable, because its value must be able to accept a value change. A number as the first argument produces an error.

For `SubtractTo()`: Only two arguments are permitted. If fewer than two or more than two arguments is specified, `SubtractTo()` returns a missing value.

For *a-=b*: More than two arguments can be strung together. JMP evaluates pairs from right to left, and each difference is placed in the left-hand variable. All arguments except the last must be a variable.

Example

```
a-=b-=c
```

JMP subtracts *c* from *b* and places the difference into *b*. Then JMP subtracts *b* from *a* and places the difference into *a*.

Character Functions

Most character functions take character arguments and quoted return character strings, although some take numeric arguments or return numeric data. Arguments that are literal character strings must be enclosed in quotation marks.

`BLOB To Char(blob, <Encoding="enc">)`

Description

Reinterpret binary data as a quoted Unicode string.

Returns

A quoted string.

Required Argument

blob A binary large object.

Optional Argument

Encoding A quoted string that specifies an encoding. The default encoding for the character string is "utf-8". "utf-16le", "utf-16be", "us-ascii", "iso-8859-1", "ascii~hex", "shift-jis", and "euc-jp" are also supported.

Notes

The optional argument "ascii~hex" is intended to make conversions of blobs containing normal ASCII data simpler when the data might contain CR, LF, or TAB characters (for example) and those characters do not need any special attention.

BLOB To Matrix(*blob*, *type*, *bytes*, *endian*, <*nCols*>)

Description

Creates a matrix by converting each byte in the blob to numbers.

Returns

A matrix that represents the blob.

Required Arguments

blob A blob or reference to a blob.

type A quoted string that contains the named type of number. The options are "int", "uint", or "float".

bytes Byte size of the data in the blob. Options are 1, 2, 4, or 8.

endian The quoted endian-ness of your system: "Big" (the first byte is most significant), "Little" (the first byte is the least significant), or "Native" (the machine's native format).

Optional Argument

<*nCols*> The number of columns in the matrix. The default value is 1.

Char(*x*, <*width*>, <*decimal*>, < <<Use Locale(*Boolean*)>)

Description

Converts an expression or numeric value into a quoted character string.

Returns

A quoted string.

Required Argument

x an expression or a numeric value. An expression must be quoted with Expr(). Otherwise, its evaluated value is converted to a quoted string.

Optional Arguments

width A number that sets the maximum number of characters in the quoted string.

decimal A number that sets the maximum number of places after the decimal that is included in the quoted string.

Use Locale(*Boolean*) Preserves locale-specific numeric formatting.

Example

```
Char( Pi(), 10, 4)
   "3.1416"

Char( Pi(), 3, 4)
   "3.1"
```

Notes

The *width* argument overrides the *decimal* argument.

Char To BLOB(*string*, <encoding="*enc*">)

Description

Converts a quoted string of characters into a binary (blob).

Returns

A binary object.

Required Argument

string A quoted string or a reference to a string.

Optional Argument

encoding A quoted string that specifies an encoding. The default encoding for the blob is "utf-8". "utf-16le", "utf-16be", "us-ascii", "iso-8859-1", "ascii~hex", "shift-jis", and "euc-jp" are also supported.

Notes

Converting BLOBs into printable format escapes \ (in addition to ~ " ! and characters outside of the printable ASCII range) into hex notation (~5C for the backslash character).

```
x = Char To BLOB( "abc\def\!n" );
y = BLOB To Char( x, encoding = "ASCII~HEX" );
If(
   y == "abc~5Cdef~0A", "JMP 12.2 and later behavior",
   y == "abc\def~0A", "Pre-JMP 12.2 behavior"
);
   "JMP 12.2 and later behavior" // output
```

Char To Hex(*value*, <*integer*|encoding="*enc*">)

Hex(*value*, <*integer*|encoding="*enc*"|Base(*number*)|Pad To(*number*)>)

Description

Returns the hexadecimal (or other base number system) text corresponding to the given value and encoding, which can be a number a quoted string or a blob. If the value is a number, IEEE 754 64-bit encoding is used unless one of the optional arguments, *integer*, or *Base*, is provided.

Required Argument

value Any number, quoted string, or blob.

Optional Arguments

`integer` A quoted switch that causes the value to be interpreted as an integer.

`encoding` A quoted string that specifies an encoding. The default encoding is `"utf-8"`. `"utf-16le"`, `"utf-16be"`, `"us-ascii"`, `"iso-8859-1"`, `"ascii~hex"`, `"shift-jis"`, and `"euc-jp"` are also supported.

`Base(`*`number`*`)` An integer value between 2 and 36 inclusive. If base is specified, the function returns the text corresponding to the specified number in that base number system instead of hexadecimal.

`Pad To(`*`number`*`)` A value to specify the padded width of the hex output.

`Collapse Whitespace(`*`text`*`)`

Description

Trims leading and trailing whitespace and replaces interior whitespace with a single space. That is, if more than one white space character is present, the `Collapse Whitespace` function replaces the two spaces with one space.

Returns

A quoted string.

Required Argument

`text` A quoted string.

`Concat(`*`a`*`, `*`b`*`)`

`Concat(`*`A`*`, `*`B`*`)`

a||*b*

A||*B*

Description

For quoted strings: Appends the string *b* to the string *a*. Neither argument is changed.

For lists: Appends the list *b* to the list *a*. Neither argument is changed.

For matrices: Horizontal concatenation of two matrices, A and B.

Returns

For quoted strings: A quoted string composed of the string *a* directly followed by the string *b*.

For lists: A list composed of the list *a* directly followed by the list *b*.

For matrices: A matrix.

Arguments

Two or more quoted strings, quoted string variables, lists, or matrices.

Example

```
a = "Hello"; b = " "; c = "World"; a || b || c;
```

```
   "Hello World"
d = {"apples", "bananas"}; e = {"peaches", "pears"}; Concat( d, e );
   {"apples", "bananas", "peaches", "pears"}
A = [1 2 3]; B = [4 5 6]; Concat( A, B );
   [1 2 3 4 5 6]
```

Notes

More than two arguments can be strung together. Each additional quoted string is appended to the end, in left to right order. Each additional matrix is appended in left to right order.

Concat Items

See "Concat Items({string1, string2, ...}, <delimiter>})".

Concat To(*a*, *b*)

Concat To(*a*, *b*)

a||=*b*

A||=*B*

Description

For quoted strings: Appends the string *b* to the string *a* and places the new concatenated string into *a*.

For matrices: Appends the matrix *b* to the matrix *a* and places the new concatenated matrix into *a*.

For lists: Appends the list *b* to the list and places the new concatenated list into *a*.

Returns

For quoted strings: A string composed of the string *a* directly followed by the string *b*.

For matrices: A matrix.

For lists: A list composed of the list *a* directly followed by the list *b*.

Arguments

Two or more quoted strings, quoted string variables, matrices, or lists. The first variable must be a variable whose value can be changed.

Notes

More than two arguments can be strung together. Each additional quoted string, matrix, or list is appended to the end, in left to right order.

Example

```
a = "Hello"; b = " "; c = "World"; Concat To( a, b, c ); Show( a );
   a = "Hello World"
A = [1 2 3]; B = [4 5 6]; Concat To( A, B ); Show( A );
   A = [1 2 3 4 5 6];
```

```
d = {"apples", "bananas"}; e = {"peaches", "pears"}; Concat to(d,e); Show( d
);
d = {"apples", "bananas", "peaches", "pears"};
```

Contains(*whole*, *part*, <*start*>)

Description

Determines whether *part* is contained within *whole*.

Returns

If *part* is found: For lists, quoted strings, and namespaces, the numeric position where the first occurrence of *part* is located. For associative arrays, 1.

If *part* is not found, 0 is returned in all cases.

Required Arguments

whole A quoted string, list, namespace, or associative array.

part For a quoted string or namespace, a string that can be part of the string *whole*. For a list, an item that can be an item in the list *whole*. For an associative array, a key that can be one of the keys in the map *whole*.

Optional Argument

start A numeric argument that specifies a starting point within *whole*. If *start* is negative, contains searches *whole* for *part* backwards, beginning with the position specified by the length of *whole* – *start*. Note that *start* is meaningless for associative arrays and is ignored.

Example

```
nameList={"Katie", "Louise", "Jane", "Jaclyn"};
r = Contains(nameList, "Katie");
```

The example returns a 1 because "Katie" is the first item in the list.

Contains Item(*x*, <*item* | {*list*} | *pattern*>, <*delimiter*>)

Description

Identifies multiple responses by searching for the specified item, list, pattern, or delimiter. The function can be used on columns with the Multiple Response modeling type or column property.

Returns

Returns a Boolean that indicates whether the word (*item*), one of a list of words (*list*), or pattern (*pattern*) matches one of the words in the text represented by *x*. Words are delimited by the characters in the optional quoted delimiter (*delimiter*) string. A comma is the default delimiter. Blanks are trimmed from the ends of each extracted word from the input text string (*x*).

Example

The following example searches for *"pots"* followed by a comma and then outputs the result.

```
x = "Franklin Garden Supply is a leading online store featuring garden decor,
   statues, pots, shovels, benches, and much more.";
b = Contains Item( x, "pots", "," );
If( b,
   Write( "The specified items were found."  ), Write( "No match." )
);
   The specified items were found.
```

Ends With(*string*, *substring*)

Description

Determines whether the quoted `substring` appears at the end of `string`.

Returns

1 if `quoted string` ends with a quoted `substring`, otherwise 0.

Required Arguments

string A quoted string or a quoted string variable. Can also be a list.

substring A quoted string or a quoted string variable. Can also be a list.

Equivalent Expression

`Right(string, Length(substring)) == substring`

Hex(*value*, <*integer*|encoding="*enc*"|Base(*number*)|Pad To(*number*)>)

See "Char To Hex(value, <integer|encoding="enc">)".

Hex To BLOB(*string*)

Description

Converts the quoted hexadecimal ***string*** (including whitespace characters) to a blob (binary large object).

Example

```
Hex To BLOB( "4A4D50" );
   Char To BLOB("JMP", "ascii~hex")
```

Hex To Char(*string*, <*encoding*>)

Description

Converts the quoted hexadecimal ***string*** to its character equivalent.

Example

Hex To Char("30") results in "0".

Notes

The default quoted encoding for character string is "utf-8". "utf-16le", "utf-16be", "us-ascii", "iso-8859-1", "ascii~hex", "shift-jis", and "euc-jp" are also supported.

`Hex To Number(`*`string`*`, <Base(`*`number`*`)>)`

Description

Returns the number corresponding to the hexadecimal (or other base number system) text.

Required Argument

`string` A quoted hexadecimal string.

Optional Argument

`Base(`*`number`*`)` An integer between 2 and 36 inclusive. If base is specified, the text is treated as a quoted string representing the number in that base.

Example

```
Hex To Number( "80" );
  128
```

Notes

- 16-digit hexadecimal numbers are converted as IEEE 754 64-bit floating point numbers. Otherwise, the input is treated as a hexadecimal integer.
- Whitespace between bytes (or pairs of digits) and in the middle of bytes is permitted (for example, `FF 1919` and `F F1919`).

`Insert`

See "Insert(source, item, <position>)".

`Insert Into`

See "Insert Into(source, item, <position>)".

`Item(`*`n`*`|[`*`first last`*`], `*`string`*`, <`*`delimiter`*`>, <Unmatched(`*`result string`*`)>, <Include Boundary Delimiters(`*`Boolean`*`)>)`

Description

Returns the *n*th item or the span from the first to last item of the quoted *`string`* according to the quoted string *`delimiters`* given. If you include a third argument, any and all characters in that argument are taken to be delimiters.

Required Arguments

`n` The position of the word being extracted.

`[`*`first last`*`]` A matrix that defines the beginning and end word range to return.

`string` The quoted string that is evaluated.

Optional Arguments

delimiter The character used as a boundary. If *delimiter* is absent, an ASCII space is used. If *delimiter* is the quoted empty string, each character is treated as a separate word.

Unmatched(*result string*) The quoted string to print if no match is found.

Include Boundary Delimiters(*Boolean*) Determines how delimiters on the front and back of string are treated. The default value is false, which means that the delimiters on these boundaries are ignored. If the value is true, delimiters on these boundaries produce an empty element (similar to consecutive delimiters within a string).

Example

Consecutive delimiters are treated as though they have a word between them. In this example, the delimiters are a comma and a space.

```
Item( 4,"the  quick, brown fox", ", " ); // quick is preceded by two spaces
```

The expression is processed as follows:

the<delim[space]><word2><delim[space]>quick<delim[comma]><word 4><delim[space]>brown<delim[space]>fox

Because word4 is empty, this expression returns a quoted empty string.

Item() is the same as Word() except that Item() treats each delimiter character as a separate delimiter, and Word() treats several adjacent delimiters as a single delimiter.

```
Word( 4,"the  quick, brown fox", ", " ); // quick is preceded by two spaces
```

This expression is processed as follows:

the<delim[2 spaces]>quick<delim[comma + space]>brown<delim[space]>fox

It returns "fox".

Left(*string*, *n*, <*filler*>)

Left({*list*}, *n*, <*filler*>)

Description

Returns a truncated or padded version of the original quoted *string* or *list*. The result contains the left *n* characters or list items, padded with any *filler* on the right if the length of *string* is less than *n*.

Length(*string*)

Description

Returns the length of the given quoted string (in characters), list (in items), associative array (in number of keys), BLOB (in bytes), matrix (in elements), namespace (in number of functions and variables), or class (in number of methods, functions, and variables).

Lowercase(*string*)

Description

Converts any uppercase character found in quoted *string* to the equivalent lowercase character.

Matrix to BLOB(*matrix*, *type*, *bytesEach*, *endian(value)*)

Description

Makes a BLOB from a matrix by converting the matrix elements to 1-byte, 2-byte, or 4-byte signed or unsigned integers or 4-byte or 8-byte floating point numbers.

Required Arguments

matrix The matrix.

type The quoted type of BLOB: `int`, `uint`, or `float`.

bytesEach The number of bytes in each int, uint, or float. Integers can be 1, 2, or 4 bytes each, and floats can be 4 or 8 bytes each.

endian(value) The quoted endian-ness of your system: `"Big"` (the first byte is most significant), `"Little"` (the first byte is the least significant), or `"Native"` (the machine's native format).

Munger(*string*, *start position*, *find|length*, *<replacement string>*)

Description

Computes new quoted character strings from the quoted *string* by inserting or deleting characters. It can also produce substrings, calculate indexes, and perform other tasks depending on how you specify its arguments.

Required Arguments

start position A numeric expression that specifies the starting position to search in the quoted string. If the *start position* is greater than the position of the first instance of the find argument, the first instance is disregarded. If the *start position* is greater than the search string's length, `Munger()` uses the string's length as the *start position*.

find|length Specifies the string or number of characters to find.

Optional Argument

replacement string The quoted replacement string. If *replacement string* is omitted, a substring between start position, and position and length, is returned.

Num(*string*)

Description

Converts a quoted character string into a number.

`Regex(source, pattern, (<replacement string>, <format, "GLOBALREPLACE", "IGNORECASE">>)`

Description

Searches for the quoted *pattern* within the quoted *source* string.

Returns

The matched text as a quoted string or numeric missing if there was no match.

Required Arguments

source A quoted string.

pattern A quoted regular expression.

replacement string The replacement string.

Optional Arguments

format A backreference to the capturing group. The default is \0, which is the entire matched quoted string. \n returns the n^{th} match.

"IGNORECASE" The search is case sensitive, unless you specify "IGNORECASE".

"GLOBALREPLACE" Applies the regular expression to the quoted *source* string repeatedly until all matches are found.

`Remove`

See "Remove(source, position, <n>)".

`Remove From`

See "Remove From(source, position, <n>)".

`Repeat(source, a)`

`Repeat(matrix, a, b)`

Description

Returns a copy of *source* concatenated with itself *a* times. Or returns a matrix composed of *a* row repeats and *b* column repeats. The *source* can be text, a matrix, or a list.

`Reverse`

See "Reverse(source)".

`Reverse Into`

See "Reverse Into(source)".

Right(*string*, *n*, <*filler*>)

Right({*list*}, *n*, <*filler*>)

Description

Returns a truncated or padded version of the original quoted *string* or *list*. The result contains the right *n* characters or list items, padded with any optional *filler* on the left if the length of *string* is less than *n*.

Shift

See "Shift(source, <n>)".

Shift Into

See "Shift Into(source, <n>)".

Starts With(*string*, *substring*)

Description

Determines whether the quoted `substring` appears at the start of the quoted *string*.

Returns

1 if *string* starts with *substring*, otherwise 0.

Arguments

string A quoted string or a reference to one. Can also be a list.

substring A quoted string or a reference to one. Can also be a list.

Equivalent Expression

```
Left(string, Length("substring")) = = "substring"
```

Substitute

See "Substitute(string, "substring", "replacementString", ...)".

Substitute Into

See "Substitute Into(string, substring, replacementString, ...)".

Substr(*string*, *start*, *length*)

Description

Extracts the characters that are the portion of the first argument beginning at the position given by the second argument and ending based on the number of characters specified in the third argument. The first argument can be a character column or value, or an

expression evaluating to same. The starting argument and the length argument can be numbers or expressions that evaluate to numbers.

Example

This example extracts the first name:

```
Substr( "Katie Layman", 1, 5 );
```

The function starts at position 1, reads through position 5, and ignores the remaining characters, which yields "Katie."

`Text Score(`*`text column`*`, `*`text-to-number`*`, <`*`weighting`*`>, <{`*`support vectors`*`}>, <`*`text explorer setup`*`>)`

Description

Used to create scoring formulas in Text Explorer. Not supported for use with the Stem for Combining option.

Returns

Returns a vector of scores.

Required Arguments

`text column` The data table column.

`text-to-number` An associative array that maps lowercase words to numbers.

Optional Arguments

`weighting` The quoted `"Count"`, `"Binary"`, `"Ternary"`, `"LogCount"`, `"LCA"`, or an array of inverse document frequency weights for TFLogIDF. `"Count"` is the default value. The default value is `"Count"`.

`support vectors` A list of vectors that are used in the text scoring. The number and length of the vectors depends on the *`weighting`* argument.

`text explorer setup` An expression that contains a block of Text Explorer setup information.

`Titlecase(`*`string`*`)`

Description

Converts the quoted *`string`* to title case; each word in the string has an initial uppercase character and subsequent lowercase characters.

Returns

A quoted string.

Argument

`string` A quoted string.

Example

The following function capitalizes the name:

```
Titlecase( "veronica layman ")
```

```
"Veronica Layman"
```

`Trim(string,<"Left"|"Right"|"Both">)`

`Trim Whitespace(string,<"Left"|"Right"|"Both">)`

Description

Removes leading and trailing whitespace from the specified string.

Returns

A quoted string.

Required Argument

string A quoted string.

Optional Argument

"Left"|"Right"|"Both" A quoted string that specifies whether whitespace is removed from the left, the right, or both ends of the string. If omitted, whitespace is removed from both ends.

Example

For example, the following command returns "John":

```
Trim( "  John    ", Both )
   "John"
```

`Uppercase(string)`

Description

Converts any lowercase character found in the quoted *string* to the equivalent uppercase character.

`Word(n|[first last], string, <delimiter>, <Unmatched(result string)>)`

Description

Returns the *n*th item of the string, where words are substrings separated by any number of any characters in the *delimiter* argument.

Required Arguments

n The position of the word being extracted.

[*first last*] A matrix that defines the beginning and end word range to return.

string The quoted string that is evaluated.

Optional Arguments

delimiter The character used as a boundary. If *delimiter* is absent, an ASCII space is used. If *delimiter* is the empty quoted string, each character is treated as a separate word.

Unmatched(*result string*) The quoted string to print if no match is found.

Examples

This example returns the last name:

```
Word( 2, "Katie Layman" );
   "Layman"
```

See Also

See "Item(n|[first last], string, <delimiter>, <Unmatched(result string)>, <Include Boundary Delimiters(Boolean)>)" for examples of how Word() differs from Item().

Words

See "Words(string, <delimiter>)".

XPath Query(*xml*, *xpath_expression*)

Description

Runs an XPath expression on an XML document.

Returns

A list.

Required Arguments

xml A valid XML document.

xpath_expression A quoted XPath 1.0 expression.

Example

Suppose that you created a report of test results in JMP and exported important details to an XML document. The test results are enclosed in <result> tags.

The following example stores the XML document in a variable. The XPath Query expression parses the XML to find the text nodes inside the <result> tags. The results are returned in a list.

```
rpt =
"\[<?xml version="1.0" encoding="utf-8"?>
<JMP><report><title>Production Report</title>
<result>November 21st: Pass</result>
<result>November 22nd: Fail</result>
<note>Tests ran at 3:00 a.m.</note></report>
</JMP> ]\";
results = XPath Query( rpt, "//result/text()" );
   {"November 21st: Pass", "November 22nd: Fail"}
```

Character Pattern Functions

Pat Abort()

Description

Constructs a quoted pattern that immediately stops the pattern match. The matcher does not back up and retry any alternatives. Conditional assignments are *not* made. Immediate assignments that were already made are kept.

Returns

0 when a match is stopped.

Argument

none

Pat Altern(*pattern1*, <*pattern2*, ...>)

Description

Constructs a quoted pattern that matches any one of the pattern arguments.

Returns

A quoted pattern.

Argument

One or more patterns.

Pat Any(*string*)

Description

Constructs a quoted pattern that matches a single character in the argument.

Returns

A quoted pattern.

Argument

pattern A quoted string.

Pat Arb()

Description

Constructs a quoted pattern that matches an arbitrary quoted string. It initially matches the null quoted string. It then matches one additional character each time the pattern matcher backs into it.

Returns

A quoted pattern.

Argument

none

Example

```
p = "the beginning" + Pat Arb() >? stuffInTheMiddle + "the end";
Pat Match( "in the beginning of the story, and not near the end, there are
   three bears", p );
Show( stuffInTheMiddle );
   stuffInTheMiddle = " of the story, and not near "
```

Pat Arb No(*pattern*)

Description

Constructs a quoted pattern that matches zero or more copies of *pattern*.

Returns

A quoted pattern.

Argument

pattern A quoted pattern to match against.

Example

```
adjectives = "large" | "medium" | "small" | "warm" | "cold" | "hot" | "sweet";
rc = Pat Match( "I would like a medium hot, sweet tea please",
                Pat Arbno( adjectives | Pat Any(", ") ) >> adj +
                ("tea" | "coffee" | "milk") );
Show( rc, adj );
   rc = 1;
   adj = " medium hot, sweet ";
```

Pat At(*varName*)

Description

Constructs a quoted pattern that matches the null quoted string and stores the current position in the source string into the specified JSL variable (*varName*). The assignment is immediate, and the variable can be used with `expr()` to affect the remainder of the match.

Returns

A quoted pattern.

Argument

varName The name of a variable to store the result in.

Example

```
p = ":" + Pat At( listStart ) + Expr(
   If( listStart == 1,
      Pat Immediate( Pat Len( 3 ), early ),
```

```
        Pat Immediate( Pat Len( 2 ), late )
    )
);
early = "";
late = "";
Pat Match( ":123456789", p );
Show( early, late );
early = "";
late = "";
Pat Match( "   :123456789", p );
Show( early, late );
```

First this is produced:

```
early = "123"
late = ""
```

and later this:

```
early = ""
late = "12"
```

Pat Break(*string*)

Description

Constructs a quoted pattern that matches zero or more characters that are not in its argument; it stops or breaks on a character in its argument. It fails if a character in its argument is not found (in particular, it fails to match if it finds the end of the quoted source string without finding a break character).

Returns

A quoted pattern.

Argument

string A quoted string.

Pat Concat(*pattern1*, *pattern2* <*pattern3*, ...>)

Pattern1 + *Pattern2* + ...

Description

Constructs a quoted pattern that matches each pattern argument in turn.

Returns

A quoted pattern.

Argument

Two or more quoted patterns.

Pat Conditional(*pattern*, *varName*)

Description

Saves the result of the quoted pattern match, if it succeeds, to a variable named as the second argument (*varName*) after the match is finished.

Returns

A quoted pattern.

Arguments

pattern A quoted pattern to match against.

varName The name of a variable to store the result in.

Example

```
type = "undefined";
rc = Pat Match(
   "green apples",
   Pat Conditional( "red" | "green", type ) + " apples"
);
Show( rc, type );
   rc = 1;
   type = "green";
```

Pat Fail()

Description

Constructs a quoted pattern that fails whenever the matcher attempts to move forward through it. The matcher backs up and tries different alternatives. If and when there are no alternatives left, the match fails and `Pat Match` returns 0.

Returns

0 when a match fails.

Argument

none

Pat Fence()

Description

Constructs a pattern that succeeds and matches the quoted null string when the matcher moves forward through it, but fails when the matcher tries to back up through it. It is a one-way trap door that can be used to optimize some matches.

Returns

1 when the match succeeds, 0 otherwise.

Argument

none

Pat Immediate(*pattern*, *varName*)

Description

Saves the result of the pattern match to a variable named as the second argument (*varName*) immediately.

Returns

A quoted pattern.

Arguments

pattern A quoted pattern to match against.

varName The name of a variable to store the result in.

Example

```
type = "undefined";
rc = Pat Match(
   "green apples",
   ("red" | "green") >> type + " pears"
);
Show( rc, type );
   rc = 0
   type = "green"
```

Even though the match failed, the immediate assignment was made.

Pat Len(*int*)

Description

Constructs a quoted pattern that matches *n* characters.

Returns

A quoted pattern.

Argument

int An integer that specifies the number of characters.

Pat Look Ahead(*pattern*, *Boolean*)

Description

A zero-width pattern match after the current position.

Arguments

pattern The quoted pattern.

Boolean 0 (the default) indicates a match. 1 indicates a negative match or non-match.

Pat Look Behind(*pattern*, *Boolean*)

Description

A zero-width quoted pattern match before the current position.

Arguments

pattern The quoted pattern.

Boolean 0 (the default) indicates a match. 1 indicates a negative match or non-match.

Pat Match(*source text*, *pattern*, <*replacement string*>, <"NULL">, <"ANCHOR">, <"MATCHCASE">, <"FULLSCAN">)

Description

Pat Match executes the quoted *pattern* against the *source text*. The pattern must be constructed first, either inline or by assigning it to a JSL variable elsewhere.

Returns

1 if the pattern is found, 0 otherwise.

Required Arguments

source text A quoted string or quoted string variable that contains the text to be searched.

pattern A quoted pattern or pattern variable that contains the text to be searched for.

Optional Arguments

replacement string A quoted string that defines text to replace the pattern in the source text.

"NULL" A placeholder for the third argument if ANCHOR, MATCHCASE, or FULLSCAN are necessary *and* there is no replacement text.

"ANCHOR" Starts the pattern match at the beginning of the quoted string. The following match fails because the pattern, "cream", is not found at the beginning of the string:

```
Pat Match( "coffee with cream and sugar", "cream", NULL, ANCHOR );
```

"MATCHCASE" Optional command to consider capitalization in the match. By default, Pat Match() is case insensitive.

"FULLSCAN" Optional command to force Pat Match to try all alternatives, which uses more memory as the match expands. By default, Pat Match() does not use FULLSCAN, and makes some assumptions that allow the recursion to stop and the match to succeed.

Pat Not Any(*string*)

Description

Constructs a pattern that matches a single character that is not in the argument.

Returns

A quoted pattern.

Argument

string A quoted string.

Pat Pos(*int*)

Description

Constructs patterns that match the quoted null string if the current position is *int* from the left end of the string, and fail otherwise.

Returns

A quoted pattern.

Argument

int An integer that specifies the position in a quoted string.

Pat R Pos(*int*)

Description

Constructs patterns that match the quoted null string if the current position is *int* from the right end of the string, and fails otherwise.

Returns

A quoted pattern.

Argument

int An integer that specifies the position in a quoted string.

Pat R Tab(*int*)

Description

Constructs a quoted pattern that matches up to position *n* from the end of the quoted string. It can match 0 or more characters. It fails if it would have to move backwards or beyond the end of the string.

Returns

A quoted pattern.

Argument

int An integer that specifies a position in a quoted string.

Pat Regex(*string*)

Description

Constructs a quoted pattern that matches the regular expression in the quoted *string* argument.

Returns

A quoted pattern.

Argument

string A quoted string.

Pat Rem()

Description

Constructs a quoted pattern that matches the remainder of the quoted string. It is equivalent to `Pat R Tab(0)`.

Returns

A quoted pattern.

Argument

none

Pat Repeat(*pattern*, *minimum*, *maximum*, <"GREEDY"|"RELUCTANT">)

Description

Matches the quoted *pattern* between *minimum* and *maximum* times.

Returns

A quoted pattern.

Required Arguments

pattern A pattern to match against.

minimum An integer that must be smaller than maximum.

maximum An integer that must be greater than minimum.

Optional Argument

"GREEDY"|"RELUCTANT" If GREEDY is specified, it tries the maximum first and works back to the minimum. If RELUCTANT is specified, it tries the minimum first and works up to the maximum.

Notes

- `Pat Arbno(p)` is the same as `Pat Repeat(p, 0, infinity, RELUCTANT)`
- `Pat Repeat(p)` is the same as `Pat Repeat(p, 1, infinity, GREEDY)`
- `Pat Repeat(p, n)` is the same as `Pat Repeat(p, n, infinity, GREEDY)`
- `Pat Repeat(p, n, m)` is the same as `Pat Repeat(p, n, m, GREEDY)`

Pat Span(*string*)

Description

Constructs a pattern that matches one or more (not zero) occurrences of characters in its argument. It is greedy; it always matches the longest possible quoted string. It fails rather than matching zero characters.

Returns

A quoted pattern.

Argument

string A quoted string.

`Pat String(string)`

Description

Constructs a pattern that matches its quoted *string* argument.

Returns

A quoted pattern.

Argument

string A quoted string.

`Pat Succeed()`

Description

Constructs a pattern that always succeeds, even when the matcher backs into it. It matches the quoted null string.

Returns

1 when the match succeeds.

Argument

none

`Pat Tab(int)`

Description

Constructs a pattern that matches forward to position *int* in the quoted source string. It can match 0 or more characters. It fails if it would have to move backwards or beyond the end of the string.

Returns

A pattern.

Argument

int An integer that specifies a position in a quoted string.

`Pat Test(expr)`

Description

Constructs a pattern that succeeds and matches the quoted null string if *expr* is not zero and fails otherwise.

Returns

A quoted pattern.

Argument

expr An expression.

Notes

Usually the argument is wrapped with `expr()` because the test needs to be made on the current value of variables set by `Pat Immediate`, `Pat Conditional`, and `Pat At`. Without `expr`, the test is based on values that were known when the pattern was constructed, which means the test always succeeds or always fails at pattern execution time, which is probably not what you want.

Example

```
nCats = 0;
whichCat = 3;
string = "catch a catnapping cat in a catsup factory";
rc = Pat Match(
   string,
   "cat" + Pat Test(
      Expr(
         nCats = nCats + 1;
         nCats == whichCat;
      )
   ),
   "dog"
);
Show( rc, string, nCats );
   rc = 1
   string = "catch a catnapping dog in a catsup factory"
   nCats = 3
```

`Regex Match(`*source*`, `*pattern*`, <`*replacement string*`>|<"MATCHCASE">, <"NULL">)`

Description

Executes the pattern match in quoted *pattern* against the quoted *source* string.

Returns

A pattern.

Required Arguments

source A quoted string.

pattern A quoted pattern.

Optional Arguments

replacement string The quoted string that specifies the text to replace the source with.

"MATCHCASE" The search is case insensitive unless you specify MATCHCASE.

"NULL" Indicates that the expression contains MATCHCASE but you don't want to specify a replacement.

Examples

```
Regex Match(
   "person=Fred id=77 friend= favorite=tea", // source
   "(\w+)=(\S*) (\w+)=(\S*) (\w+)=(\S*) (\w+)=(\S*)" // pattern
);
   {"person=Fred id=77 friend= favorite=tea", "person", "Fred", "id", "77",
      "friend", "", "favorite", "tea"}

// case-insensitive, no replacement
Regex Match( "beliEve", "([aeiou])(.*?)(\1)" );
   {"eliE", "e", "li", "E"}
// case-sensitive, no replacement
Regex Match( "beliEve", "([aeiou])(.*?)(\1)", NULL, MATCHCASE );
   {"eliEve", "e", "liEv", "e"}
```

Comment Functions

// comment

Description

Comments to end of line.

Notes

Everything after the // is ignored when running the script.

/* comment */

Description

A comment that can appear in the middle of a line of script.

Notes

Anything between the beginning tag /* and the end tag */ is ignored when running the script. This comment style can be used almost anywhere, even inside lists of arguments. If you place a comment inside a double-quoted string, the comment is treated merely as part of the string and not a comment. You cannot place comments in the middle of operators.

Examples

```
+/*comment*/=
:/*comment*/name
```

are invalid and produce errors. The first comment interrupts += and the second interrupts :name.

```
sums = {(a+b /*comment*/), /*comment*/ (c^2)}
```

is valid JSL; the comments are both ignored.

`//!`

Description

If placed on the first line of a script, this comment line causes the script to be run when opened in JMP without opening into the script editor window.

Notes

You can over-ride this comment when opening the file. Select **File > Open**. Hold the Ctrl key while you select the JSL file and click **Open**. Or right-click the file in the Home Window Recent Files list and select **Edit Script**. The script opens into a script window instead of being executed.

`/*debug step*/`

`/*debug run*/`

Description

If placed on the first line of a script, the script is opened in the debugger when it is run.

Notes

All letters must be lowercase. There must be one space between debug and step or run, and there must be no other spaces present. Only one of these lines can be used, and it must be the first line of the script; a first line that is blank followed by this comment negates the debug command.

Comparison Functions

The comparison operators (<, <=, >, >=) work for numbers, quoted strings, and matrices. For matrices, they produce a matrix of results. If you compare mixed arguments, such as strings with numbers or matrices, the result is a missing value. Comparisons involving lists are not allowed and also return missing values.

The equality operators (== and !=) work for numbers, quoted strings, matrices, and lists. For matrices, they produce a matrix of results; for lists, they produce a single result. If you *test equality* of mixed results (for example. strings with numbers or matrices) the result is 0 or unequal.

Range check operators let you check whether something falls between two specified values:

```
a = 1;
Show( 1 <= a < 3 );
b = 2;
Show( 2 < b <= 3 );
   1 <= a < 3 = 1;
   2 < b <= 3 = 0;
```

Expressions with comparison operators are evaluated all at once, not in sequence

All the comparison operators are *eliding operators*. That means JMP treats arguments joined by comparison operators as one big clause, as opposed to the way most expressions are evaluated one operator at a time. Evaluating as a single clause produces different results than the more usual method of evaluating in pieces. For example, the following two statements are different:

```
12 < a < 13;
(12 < a) < 13;
```

The first statement checks whether *a* is between 12 and 13, because all three arguments and both operators are read and evaluated together. The second statement uses parentheses to regroup the operations explicitly to evaluate from left to right, which would be the normal way to evaluate most expressions. Thus it first checks whether 12 is less than *a*, returning 1 if true or 0 if false. Then it checks whether the result is less than 13, which is always true because 0 and 1 are both less than 13.

All the comparison operators are elided when they are used in matched pairs or in the unmatched pairs <... <= and <=... <. What this means is that if you want a comparison statement to be evaluated one comparison operator at a time, you should use parentheses ( ) to control the order of operations explicitly.

`Equal(a, b, ...)`

`a==b==...`

Description

Compares all the listed values and tests if they are all equal to each other.

Returns

1 (true) if all arguments evaluate to the same value.

0 (false) otherwise.

Arguments

Two or more variables, references, matrices, or numbers.

Notes

- If more than two arguments are specified, a 1 is returned only if all arguments are exactly the same. This is typically used in conditional statements and to control loops.
- The comparison is case-sensitive for quoted string comparisons.

`Greater(a, b, ...)`

`a>b>...`

Description

Compares all the list values and tests if, in each pair, the left value is greater than the right.

Returns

1 (true) if *a* evaluates strictly greater than *b* (and *b* evaluates strictly greater than *c*, and so on).

0 (false) otherwise.

Arguments

Two or more variables, references, matrices, or numbers.

Notes

If more than two arguments are specified, a 1 is returned only if each argument is greater than the one that follows it. This is typically used in conditional statements and to control loops.

`Greater`, `Less`, `Greater Or Equal`, and `Less Or Equal` can also be strung together (as in `0 < x <= 5`). If you do not group with parentheses, JMP evaluates each pair left to right. You can also use parentheses to explicitly tell JMP how to evaluate the expression.

`Greater or Equal(`*a*`, `*b*`, ...)`

a>=*b*>=...

Description

Compares all the list values and tests if, in each pair, the left value is greater than or equal to the right.

Returns

1 (true) if *a* evaluates strictly greater than or equal to *b* (and *b* evaluates strictly greater than or equal to *c*, and so on).

0 (false) otherwise.

Arguments

Two or more variables, references, matrices, or numbers.

Notes

If more than two arguments are specified, a 1 is returned only if each argument is greater than or equal to the one that follows it. This is typically used in conditional statements and to control loops.

`Greater`, `Less`, `Greater Or Equal`, and `Less Or Equal` can also be strung together (as in `0 < x <= 5`). If you do not group with parentheses, JMP evaluates each pair left to right. You can also use parentheses to explicitly tell JMP how to evaluate the expression.

`Is Missing(`*expr*`)`

Description

Returns 1 if the expression yields a missing value and 0 otherwise.

Less(*a*, *b*, ...)

a<*b*<...

Description

Compares all the list values and tests if, in each pair, the left value is less than the right.

Returns

1 (true) if *a* evaluates strictly less than *b* (and *b* evaluates strictly less than *c*, and so on).

0 (false) otherwise.

Arguments

Two or more variables, references, matrices, or numbers.

Notes

If more than two arguments are specified, a 1 is returned only if each argument is less than the one that follows it. This is typically used in conditional statements and to control loops.

`Greater`, `Less`, `Greater Or Equal`, and `Less Or Equal` can also be strung together (as in `0 < x <= 5`). If you do not group with parentheses, JMP evaluates each pair left to right. You can also use parentheses to explicitly tell JMP how to evaluate the expression.

Less Less Equal(*a*, *b*, *c*, ...)

a<*b*<=*c*<=...

Description

Range check, exclusive below and inclusive above.

Returns

1 (true) if *b* is greater than *a* and less than or equal to *c*.

0 (false) otherwise.

Arguments

a, *b*, *c* Variables, references, matrices, or numbers.

Notes

Returns 1 when two conditions are met: the first argument is less than the second argument, and each remaining argument is less than or equal to its argument on the right. This is typically used in conditional statements and to control loops.

Less or Equal(*a*, *b*, ...)

a<=*b*<=...

Description

Compares all the list values and tests if, in each pair, the left value is less than or equal to the right.

Returns

1 (true) if *a* evaluates strictly less than or equal to *b* (and *b* evaluates strictly less than or equal to *c*, and so on).

0 (false) otherwise.

Arguments

Two or more variables, references, matrices, or numbers.

Notes

If more than two arguments are specified, a 1 is returned only if each argument is less than or equal to the one that follows it. This is typically used in conditional statements and to control loops.

`Greater`, `Less`, `Greater Or Equal`, and `Less Or Equal` can also be strung together (as in `0 < x <= 5`). If you do not group with parentheses, JMP evaluates each pair left to right. You can also use parentheses to explicitly tell JMP how to evaluate the expression.

`Less Equal Less(a, b, c, ...)`

`a<=b<c<...`

Description

A range check, inclusive below and exclusive above.

Returns

1 (true) if *b* is greater than or equal to *a* and less than *c*.

0 (false) otherwise.

Arguments

a, *b*, *c* Variables, references, matrices, or numbers.

Notes

Returns 1 when two conditions are met: the first argument is less than or equal to the second argument, and each remaining argument is less than its argument on the right. This is typically used in conditional statements and to control loops.

`Not Equal(a, b)`

`a!=b`

Description

Compares *a* and *b* and tests if they are equal.

Returns

0 (false) if *a* and *b* evaluate to the same value.

1 (true) otherwise.

Argument

a, b Any variable or number.

Notes

Mostly used for conditional statements and loop control.

Conditional and Logical Functions

And(*a, b*)

a&*b*

Description

The logical And.

Returns

1 (true) if both *a* and *b* are true.

0 (false) if either *a* or *b* is false or if both *a* and *b* are false.

Missing if either *a* or *b* is a missing value or if both *a* and *b* are missing values.

Arguments

Two or more variables or expressions.

Notes

More than two arguments can be strung together. a&b returns 1 (true) only if all arguments evaluate to true.

AndMZ(*a, b*)

a:&*b*

Description

Returns the logical AND of all arguments. Missing values are treated as zeroes.

Returns

1 (true) if both *a* and *b* are true.

0 (false) if either *a* or *b* is false or if both *a* and *b* are false.

0 (false) if either *a* or *b* is a missing value or if both *a* and *b* are missing values.

Arguments

Two or more variables or expressions.

Notes

More than two arguments can be strung together. *a*:&*b* returns 1 (true) only if all arguments evaluate to true.

Break()

Description

Stops execution of a loop completely and continues to the statement following the loop.

Notes

`Break()` works with `For` and `While` loops and also with `For Each Row`.

Choose(*expr*, *r1*, *r2*, *r3*, ..., *resultElse*)

Description

Evaluates *expr*. If the value of *expr* is 1, *r1* is returned; if 2, the value of *r2* is returned, and so on. If no matches are found, the last argument (*resultElse*) is returned.

Returns

The value whose index in the list of arguments matches *expr*, or the value of the last argument.

Arguments

expr An expression or a value.

r1, *r2*, *r3*, ... An expression or a value.

resultElse The argument that is returned when no matches are found.

Continue()

Description

Ends the current iteration of a loop and begins the loop at the next iteration.

Notes

`Continue()` works with `For` and `While` loops, and also with `For Each Row`.

Filter Each(*names*, *container*, *locals*, *body*)

Description

Iterates over a container, which can be a list, an associative array, or a matrix, and returns a subset of the values in the container based on the evaluation of a Boolean expression at each iteration. The value, key, or element, as well as the index number, are all available at each iteration.

- For associative array containers, the key and value can be accessed using a two-item list.
- For matrix containers, a linear index is provided by default, but a two-item list can be used to access the row and column indices.

These symbols are provided within the body of the loop only. A list of local variables can also be provided. If local variables are defined, they are initialized after the first iteration symbols are set.

Returns

A subset from the original container. The return object has the same type as the original container.

Arguments

names Specification of loop control variable names, specified as a list. The form of the list is determined by the type of container. All of the names are optional, so you need to specify them only if you need to refer to them in the *locals* or *body* arguments. If you do not specify any names, the first argument should be an empty list, which can be specified as {} or List().

For a *list* container, the names list contains a name for each value in the list and a name for the index of each value in the list.

For an *associative array* container, the names list contains a two-item list of names and a name for the index of each item in the associative array.

For a *matrix* container, the names list contains a name for each element in the matrix and a second argument for the index of each element in the matrix. The second argument can be specified as a single name or as a two-item list of names that represent symbols for the row and column indices.

When specifying multiple containers using the Across() keyword, the names list contains a list of names that refer to values in each container. The number of names in the first item in the names list must match the number of containers specified in the Across() keyword.

container A list, associative array, or a matrix. The container can be defined in the argument or it can be a reference to a previously defined object.

This argument can also use the Across() keyword, which enables you to use the function across multiple containers. The multiple containers can be specified as separate arguments or as items in a list. The Across() keyword has an optional Count() argument that enables you to specify how containers of different sizes are handled. The available Count() options are: "Shortest", "Longest", "Enforce Equal", and N, where N is a number. If you specify a number, the function iterates through all containers exactly N times; note that the function loops back to the start of containers that have fewer than N items.

locals A list of variables that are local to the function. This is equivalent to other Local variable initializations in JSL. The initialization of the local variables occurs after the first loop control variables have been set, but the local variables do not get initialized again after that.

body Any number of valid JSL expressions, glued together if there are more than one. The result of the JSL expressions should be a Boolean value. If the result of the expression is true, the container value at the current iteration is included in the result; otherwise, the

container value at the current iteration is not included in the result. You can use the `Continue()` function as an equivalent way to return false and skip to the next iteration. You can also use `Break()` function to stop iteration through the loop and proceed to the next expression that follows the loop. See "Break and Continue Functions".

Example

```
values = Filter Each( {x}, {10, 20, 30}, x > 15 );
Show( values );
  values = {20, 30};
```

For(*init*, *while*, *increment*, *body*)

Description

Repeats the statement(s) in the *body* as long as the *while* condition is true. *init* and *increment* control iterations.

Returns

Null.

Arguments

init Initialization of loop control counter.

while Condition for loop to continue or end. As long as the conditional statement while is true, the loop is iterated one more time. As soon as while is false, the loop is exited.

increment Increments (or decrements) the loop counter after while is evaluated every *time* the loop is executed.

body Any number of valid JSL expressions, glued together if there are more than one.

Example

```
mysum = 0; myprod = 1;
For( i = 1, i <= 10, i++, mysum += i; myprod *= i; );
Show( mysum, myprod );
  mysum = 55;
  myprod = 3628800;
```

For Each(*names*, *container*, *locals*, *body*)

Description

Iterates over a container, which can be a list, an associative array, or a matrix, and provides the value, key, or element at each iteration. The index number is also available at each iteration. For associative array containers, the key and value can be accessed using a two-item list. For matrix containers, a linear index is provided by default, but a two-item list can be used to access the row and column indices. These symbols are provided within the body of the loop only. A list of local variables can also be provided. If local variables are defined, they are initialized after the first iteration symbols are set.

Arguments

names Specification of loop control variable names, specified as a list. The form of the list is determined by the type of container. All of the names are optional, so you need to specify them only if you need to refer to them in the *locals* or *body* arguments. If you do not specify any names, the first argument should be an empty list, which can be specified as {} or List().

For a *list* container, the names list contains a name for each value in the list and a name for the index of each value in the list.

For an *associative array* container, the names list contains a two-item list of names and a name for the index of each item in the associative array.

For a *matrix* container, the names list contains a name for each element in the matrix and a second argument for the index of each element in the matrix. The second argument can be specified as a single name or as a two-item list of names that represent symbols for the row and column indices.

When specifying multiple containers using the Across() keyword, the names list contains a list of names that refer to values in each container. The number of names in the first item in the names list must match the number of containers specified in the Across() keyword.

container A list, associative array, or a matrix. The container can be defined in the argument or it can be a reference to a previously defined object.

This argument can also use the Across() keyword, which enables you to use the function across multiple containers. The multiple containers can be specified as separate arguments or as items in a list. The Across() keyword has an optional Count() argument that enables you to specify how containers of different sizes are handled. The available Count() options are: "Shortest", "Longest", "Enforce Equal", and N, where N is a number. If you specify a number, the function iterates through all containers exactly N times; note that the function loops back to the start of containers that have fewer than N items.

locals A list of variables that are local to the function. This is equivalent to other Local variable initializations in JSL. The initialization of the local variables occurs after the first loop control variables have been set, but the local variables do not get initialized again after that.

body Any number of valid JSL expressions, glued together if there are more than one. You can use the Continue() function to skip to the next iteration. You can also use Break() function to stop iteration through the loop and proceed to the next expression that follows the loop. See "Break and Continue Functions".

Example

```
For Each( {value, index}, {10, 20, 30}, Show( value, index ) );
  value = 10;
  index = 1;
  value = 20;
  index = 2;
```

```
value = 30;
index = 3;
```

For Each Row(<*dt*>, *script*)

Description

Repeats the *script* on each row of the data table.

Returns

Null.

Required Argument

script Any valid JSL expressions.

Optional Argument

dt Positional argument that is a reference to a data table. If this argument is not in the form of an assignment, then it is considered a data table expression.

Example

The following example creates data table references and then iterates over each row in Big Class.jmp. If the value of age in a row is greater than 15, the age is printed to the log.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
For Each Row( dt, If( :age > 15, Show( :age ) ) );
```

If(*condition1*, *result1*, <*condition2*, *result2*>, ..., <*elseResult*>)

Description

Evaluates the first of each pair of arguments and returns the evaluation of the *result* expression associated with the first *condition* argument that evaluates to a nonzero result. The *condition* arguments are evaluated in order. If all of the *condition* arguments evaluate to zero, the optional *elseResult* is evaluated and the result is returned. If no *elseResult* is specified, and none of the conditions are true, a missing value is returned. If all of the *condition* arguments evaluate to missing, a missing value is returned.

IfMax(*expr1*, *result1*, *expr2*, *result2*, ... <*all missing results*>)

Description

Evaluates the first of each pair of arguments and returns the evaluation of the result expression (the second of each pair) associated with the maximum of the expressions. If more than one expression is the maximum, the first maximum is returned. If all expressions are missing and a final result is not specified, missing is returned. If all expressions are missing and a final result is specified, that final result is returned. The test expressions must evaluate to numeric values, but the result expressions can be anything.

Returns

The result expression associated with the maximum of the expressions

IfMin(*expr1*, *result1*, *expr2*, *result2*, ... <*all missing results*>)

Description

Evaluates the first of each pair of arguments and returns the evaluation of the result expression (the second of each pair) associated with the minimum of the expressions. If more than one expression is the minimum, the first minimum is returned. If all expressions are missing and a final result is not specified, missing is returned. If all expressions are missing and a final result is specified, that final result is returned. The test expressions must evaluate to numeric values, but the result expressions can be anything.

Returns

The result expression associated with the minimum of the expressions

IfMZ(*condition1*, *result1*, <*condition2*, *result2*>, ..., <*elseResult*>)

Description

Evaluates the first of each pair of arguments and returns the evaluation of the *result* expression associated with the *condition1* argument that evaluates to a nonzero result. The *condition* arguments are evaluated in order. If all of the *condition* arguments evaluate to zero or missing, the optional *elseResult* is evaluated and the result is returned. If no *elseResult* is specified, and none of the conditions are true, a missing value is returned.

Notes

The test arguments are evaluated in order until the first nonzero result. If all test results return zero or missing, the *elseExpr* argument is evaluated.

IfMZ() is equivalent to If() where missing values for evaluated *condition* arguments are treated as zero.

Interpolate(*x*|*xmatrix*|*xlist*, *x1*, *y1*, *x2*, *y2*, ...)

Interpolate(*x*|*xmatrix*|*xlist*, *xmatrix*, *ymatrix*)

Interpolate({*x*, *y*}, *xvector*, *yvector*, *zmatrix*)

Description

Performs linear interpolation for continuous data. There are many ways to specify the function.

In the simplest cases where the first argument is a single numeric value, the *y* value corresponding to a given *x* value between two sets of points or by matrices *xmatrix* and *ymatrix*. The *x* values, (*x1, x2*, ...) or *xmatrix*, must be in ascending order.

If the first argument is a matrix or list of numeric values, the resulting matrix or list is a set of interpolated values.

You can perform *bilinear interpolation* using the four-argument case. Here, the first argument is a list of two points, the second and third arguments are vectors that define the grid of x and y values, and the fourth argument is a matrix of data points. The function then finds the interpolated *z* value within the appropriate quadrant of the *zmatrix*.

Returns

The interpolated value or values. In the three-argument cases, the return object type matches the type of the first argument. In the four-argument case, the return object is a number.

`Is Associative Array(name)`

Description

Returns 1 if the evaluated argument is an associative array, or 0 otherwise.

`Is Empty(global)`

`Is Empty(dt)`

`Is Empty(col)`

Description

Returns 1 if the *global* variable, data table, or data column is undefined or holds the `Empty()` value, or 0 otherwise.

`Is Expr(x)`

Description

Returns 1 if the evaluated argument is an expression, or 0 otherwise.

`Is List`

See "Is List(x)".

`Is Name(x)`

Description

Returns 1 if the evaluated argument is a name, or 0 otherwise.

`Is Namespace(namespace)`

Description

Returns 1 if the namespace argument is a namespace; returns 0 otherwise.

`Is Number(x)`

Description

Returns 1 if the evaluated argument is a number or missing numeric value, or 0 otherwise.

`Is Scriptable(x)`

Description

Returns 1 if the evaluated argument is a scriptable object, or 0 otherwise.

`Is String(x)`

Description

Returns 1 if the evaluated argument is a quoted string, or 0 otherwise.

`Match(x, value1, result1, value2, result2, ..., resultElse)`

Description

If *a* is equal to *value1*, then *result1* is returned. If *a* is equal to *value2*, *result2* is returned, and so on.

Notes

The `Match()` function explicitly checks to see if the compare expression *x* is missing and if the value of *value1* is missing, then it returns the value of `result1`; otherwise it continues to compare the expression *x* to each *valueN* value in each *valueN/resultN* pair, ignoring any missing values. If the expression *x* is equal to any of the *valueN* value, then the corresponding *resultN* value is returned. If no matching *valueN* value is found, then the *resultElse* value is returned.

`MatchMZ(x, value1, expr1, value2, expr2, ..., exprElse)`

Description

Evaluates and returns the *exprN* argument that equals *x* or evaluates and returns the *exprElse* argument if no value equals *x*.

Notes

The `MatchMZ()` function works the same as the `Match()` function except that missing values are treated as 0.

`Not(a)`

`!a`

Description

The logical Not.

Returns

0 (false) if *a*>0.

1 (true) if *a*<=0.

Missing value if *a* is missing.

Argument

a Any variable or number. The variable must have a numeric or matrix value.

Notes

Mostly used for conditional statements and loop control.

`Or(a, b)`

`a|b`

Description

The logical Or.

Returns

1 (true) if either of or both *a* and *b* are true.

0 (false) otherwise.

Missing if either are missing.

Arguments

a, *b* Any variable or number.

Notes

Mostly used for conditional statements and loop control.

`OrMZ(a, b)`

`a:|b`

Description

Returns the logical OR of all arguments with missing values treated as zeroes: 1 if any arguments are nonzero and 0 otherwise.

Returns

1 (true) if either of or both *a* and *b* are true.

0 (false) otherwise.

Arguments

a, *b* Any variable or number.

Notes

– Mostly used for conditional statements and loop control.

– Or() returns missing if any evaluated argument is missing. OrMZ() returns 0 if any evaluated argument is missing.

Return(<expr1>, <expr2>, ..., <exprN>)

Description

Returns an expression value from a user-defined function.

Example

This example returns the evaluation of both expressions in the Return() function as a list. The Return() function can have more than one argument. If only one is present, then the value of the expression is returned. If more than one is present, then the values of all the expressions is returned in a list.

```
f = Function( {a, b},
  Return( a - b, a + b )
);
{lo, hi} = f( 10, 1 );
Show( lo, hi );
Show( f( 7, 15 ) );
   lo = 9;
   hi = 11;
   f(7, 15) = {-8, 22};
```

Notes

Return() not enclosed by a function, method, or recursive function call causes an error.

Step(x0, x1, y1, x2, y2, ...)

Step(x0, [x1, x2, ...], [y1, y2, ...])

Description

Returns the *y* argument corresponding to the largest *x* argument that is less than or equal to *x0*. The *x* points must be specified in ascending order.

Stop()

Description

Immediately stops a script that is running.

Transform Each(names, container, <Output(type)>, <locals>, body)

Description

Iterates over a container, which can be a list, an associative array, or a matrix, and updates each of the values in the container based on the evaluation of a JSL expression at each iteration. The value, key, or element, as well as the index number, are all available at each iteration. For associative array containers, the key and value can be accessed using a

two-item list. For matrix containers, a linear index is provided by default, but a two-item list can be used to access the row and column indices. These symbols are provided within the body of the loop only. A list of local variables can also be provided. If local variables are defined, they are initialized after the first iteration symbols are set.

Returns

An updated version of the original container. The return object has the same type as the original container, unless the `Output()` keyword is used to designate a different type for the returned container.

Arguments

`names` Specification of loop control variable names, specified as a list. The form of the list is determined by the type of container. All of the names are optional, so you need to specify them only if you need to refer to them in the *`locals`* or *`body`* arguments. If you do not specify any names, the first argument should be an empty list, which can be specified as `{}` or `List()`.

For a *list* container, the names list contains a name for each value in the list and a name for the index of each value in the list.

For an *associative array* container, the names list contains a two-item list of names and a name for the index of each item in the associative array.

For a *matrix* container, the names list contains a name for each element in the matrix and a second argument for the index of each element in the matrix. The second argument can be specified as a single name or as a two-item list of names that represent symbols for the row and column indices.

When specifying multiple containers using the `Across()` keyword, the names list contains a list of names that refer to values in each container. The number of names in the first item in the names list must match the number of containers specified in the `Across()` keyword.

`container` A list, associative array, or a matrix. The container can be defined in the argument or it can be a reference to a previously defined object.

This argument can also use the `Across()` keyword, which enables you to use the function across multiple containers. The multiple containers can be specified as separate arguments or as items in a list. The `Across()` keyword has an optional `Count()` argument that enables you to specify how containers of different sizes are handled. The available `Count()` options are: `"Shortest"`, `"Longest"`, `"Enforce Equal"`, and `N`, where `N` is a number. If you specify a number, the function iterates through all containers exactly N times; note that the function loops back to the start of containers that have fewer than N items.

`Output(type)` Specifies a type for the output. This can be `"List"`, `"Matrix"`, or `"Associative Array"`. By default, the output type matches the type of the input container.

`locals` A list of variables that are local to the function. This is equivalent to other Local variable initializations in JSL. The initialization of the local variables occurs after the

first loop control variables have been set, but the local variables do not get initialized again after that.

body Any number of valid JSL expressions, glued together if there are more than one. The result of the JSL expressions at each iteration is used in the output container. You can use the `Continue()` function to return no value for an iteration and skip to the next iteration. You can also use `Break()` function to stop iteration through the loop and proceed to the next expression that follows the loop. See "Break and Continue Functions".

Example

```
values = Transform Each( {x}, {10, 20}, x + 10 );
Show( values );
   values = {20, 30};
```

`While(expr, body)`

Description

Repeatedly tests the *expr* condition and executes the *body* until the *expr* condition is no longer true.

`Zero Or Missing(expr)`

Description

Returns 1 if *expr* yields a missing value or zero, 0 otherwise.

Constant Functions

JMP provides functions for two useful constant functions.

Note: These functions do not take an argument, but the parentheses are required.

`e()`

Description

Returns the constant *e*, which is 2.7182818284590451...

`Pi()`

Description

Returns the constant π, which is 3.1415926535897931...

Date and Time Functions

Datetime values are handled internally as numbers of seconds since midnight, January 1, 1904.

The expression `x=01Jan1904` sets *x* to zero, since the indicated date is the base date or "zero date" in JMP. If you examine the values of dates, they should be appropriately large numbers (for example, `5oct1998` is 2990390400).

Abbrev Date(*date*)

Description

Converts the provided *date* to a quoted string.

Returns

A quoted string representation of the date.

Argument

date Can be the number of seconds since the base date (midnight, January 1, 1904), or any date-time operator.

Example

```
Abbrev Date( 29Feb2004 );
```

02/29/2004

As Date(*x*)

Description

Formats the number or expression *x* so that it shows as a date or duration when displayed in a text window. Values that represent one year or more are returned as dates. Values that represent less than a year are returned as durations.

Returns

A date that is calculated from the number or expression provided.

Argument

x Number or expression.

Date Difference(*datetime1*, *datetime2*, *interval name*, <*alignment*>)

Description

Returns the difference in intervals of two date-time values.

Returns

A number.

Required Arguments

datetime1, *datetime2* Date-time values.

interval name A quoted string that contains a date-time interval, such as "month", "day", or "hour".

Optional Arguments

alignment A quoted string:

- "Start" includes full or partial intervals.
- "Actual" counts only whole intervals.
- "Fractional" returns fractional differences using averages for "year", "quarter", and "month" intervals.

Date DMY(*day*, *month*, *year*)

Description

Constructs a date value from the arguments.

Returns

The specified date, expressed as the number of seconds since midnight, 1 January 1904.

Arguments

day The numeric day of month, 1-31. Note that there is no error-checking, so you can enter February 31.

month The numeric month, 1-12.

year The year.

Date Increment(*datetime*, *interval name*, <*increment*>, <*alignment*>)

Description

Adds 1 or more intervals to a starting datetime value.

Returns

Returns the new datetime value.

Required Arguments

datetime The starting datetime value.

interval name A quoted string that contains the name of a datetime interval. "year", "quarter", "month", "week", "day", "hour", "minute", and "second" are supported.

Optional Arguments

increment A number that specifies the number of intervals. The default value is 1.

alignment A quoted string that contains a keyword:

- "Start" truncates the date to the nearest interval prior to adding the increment. For example, it removes the time and outputs the date. "start" is the default value.
- "Actual" retains the full input datetime value.

– "Fractional" allows fractional incremental values using averages for the duration of "Year", "Quarter", and "Month" intervals.

Date MDY(*month*, *day*, *year*)

Description

Constructs a date value from the arguments.

Returns

The specified date, expressed as the number of seconds since midnight, 1 January 1904.

Arguments

month The numeric month, 1-12.

day The numeric day of month, 1-31. Note that there is no error-checking, so you can enter February 31.

year The year.

Day(*datetime*)

Description

Determines the day of the month supplied by the *datetime* argument.

Returns

Returns an integer representation for the day of the month of the date supplied.

Arguments

datetime The number of seconds since midnight, 1 January 1904. This can also be an expression.

Example

```
d1 = Date DMY( 12, 2, 2003 );
   3127852800

Day( 3127852800 );
   12
Day( d1 );
   12
```

Day Of Week(*datetime*)

Description

Determines the day of the week supplied by the *datetime* argument.

Returns

Returns an integer representation for the day of the week of the date supplied. Weeks are Sunday–Saturday.

Arguments

datetime The number of seconds since midnight, 1 January 1904. This can also be an expression.

`Day Of Year(datetime)`

Description

Determines the day of the year supplied by the *datetime* argument.

Returns

Returns an integer representation for the day of the year of the date supplied.

Arguments

datetime The number of seconds since midnight, 1 January 1904. This can also be an expression.

`Days In Month(year, month)`

Description

Returns the number of days in the month for the specified *month* and *year*.

Format(*x*, formatString, *width*|<*width*, *decimal places*>, <"Use Thousands Separator">)

Format(*x*, "Best", <*width*>, <"Use Thousands Separator">)

Format(*x*, ("Fixed Dec"|"Percent"), *width*|<*width*, *decimal places*>, <"Use Thousands Separator">)

Format(*x*, "Pvalue", <*width*>)

Format(*x*, ("Scientific"|"Engineering"|"Engineering SI"), <*width*>|<*width*, *decimal places*>)

Format(*x*,"Precision", *width*|<*width*, *decimal places*>, <"Use Thousands Separator">, <"Keep Trailing Zeroes">, <"Keep All Whole Digits">)

Format(*x*, "Currency", <*currency code*>, <*width*>|<*width*, *decimal places*>, <"Use Thousands Separator">, < <<Use Locale(*Boolean*) >)

Format(*x*, *datetime*, <*width*>)

Format(*x*, ("Latitude DDD"|"Latitude DDM"|"Latitude DMS"|"Longitude DDD"|"Longitude DDM"|"Longitude DDM"), *width*|<*width*, *decimal places*>, ("PUN"|"DIR"|"PUNDIR"))

Format(*x*, "Custom", Formula(), <*width*>)

Description

Converts the value *x* into the quoted *format* that you specify in the subsequent arguments.

Returns

Returns the text that corresponds to the number in the specified format.

Arguments

See *Using JMP* for more information about the arguments. The arguments are also shown in a data table column's Column Info window.

Examples

```
Format( x, "Fixed Dec", 10, 2, "Use Thousands Separator");
Format( x, "Currency", "EUR", 20, <<Use Locale(0)); // ignores computer locale
Format( x, "m/d/y", 10 );
Format( x, "Precision", 10, 2, "Keep trailing zeroes", "Keep All Whole Digits"
  );
Format( x, "Latitude DDD", "PUNDIR"); // "PUN" for punctuation, "DIR" for
  direction, PUNDIR for both
Format( x, "Custom", Formula( Abs( value ) ), 15 );
```

Notes

- You must always precede the number of decimal places with the width.
- If the date format is unknown, an error is written to the log.

Format Date(*x*, *datetime*, <*width*>)

Description

Converts the value of *x* into the quoted *datetime* that you specify in the second argument. Format choices are those shown in the data table Column Info window.

Returns

Returns the number in the specified format.

Arguments

See *Using JMP* for more information about the arguments.

Example

```
Format Date( 22Feb2021, "yyyyQq" );
   "2021Q1"
```

Hour(*datetime*, <"12"|"24">)

Description

Determines the hour supplied by the *datetime* argument.

Returns

Returns an integer representation for the hour part of the date-time value supplied.

Arguments

`datetime` The number of seconds since midnight, 1 January 1904. This can also be an expression.

`"12"|"24"` Changes the mode to 12 hours (with am and pm). The default is 24-hour mode.

HP Time()

Description

Returns a high precision time value (in microseconds). This function is only useful relative to another `HP Time()` value. The time value represents the number of microseconds since the start of the JMP session.

Notes

For less precise time values use `Tick Seconds()`.

In Days(*n*)

Description

Returns the number of seconds per *n* days. Divide by this function to express seconds as days.

`In Hours(n)`

Description

Returns the number of seconds per *n* hours. Divide by this function to express seconds as hours.

`In Minutes(n)`

Description

Returns the number of seconds per *n* minutes. Divide by this function to express seconds as minutes.

`In Weeks(n)`

Description

Returns the number of seconds per *n* weeks. Divide by this function to express seconds as weeks.

`In Years(n)`

Description

Returns the number of seconds per *n* years. Divide by this function to express seconds as years.

`Informat(string, format)`

`Parse Date(string, format)`

Description

Parses a quoted *string* of a given quoted *format* and returns a date/time value. The value is expressed as if surrounded by the `As Date()` function, returning the date in "*ddMonyyyy*" format.

Example

```
Informat( "07152000", "MMDDYYYY" );
```

15Jul2000

Notes

- To see the format options, open the Column Info window on a data table column, select a date/time value for the format, and view the Input Format list.
- If the date format is unknown, an error is written to the log.

See Also

"As Date(x)"

Is Leap Year(*year*)

Description

Returns 1 if the specified *year* is a leap year and 0 otherwise.

ISO Year(*datetime*)

Description

Returns the ISO year of the *datetime* supplied. ISO years correspond to ISO weeks; they begin on the Monday of the first week that contains at least four days.

Long Date(*date*)

Description

Returns a locale-specific quoted string representation for the *date* supplied, formatted like "Sunday, February 29, 2004" or "Wednesday, November 9, 2011".

MDYHMS(*date*)

Description

Returns a quoted string representation for the *date* supplied, formatted like "2/29/04 00:02:20".

Minute(*datetime*)

Description

Determines the minute supplied by the *datetime* argument, 0-59.

Returns

Returns an integer representation for the minute part of the date-time value supplied.

Month(*date*)

Description

Returns an integer representation for the month of the *date* that is supplied.

Nth Day of Week in the Month(*datetime*)

Description

Determines the day of the week of the *datetime* argument and how many instances of that day of the week have occurred in the month of the *datetime* argument.

Returns

Returns an integer that represents the number of instances of the day of the week of the *datetime* argument that have occurred in the month.

Parse Date()

See "Informat(string, format)".

Quarter(*datetime*)

Description

Returns the annual quarter of a *datetime* value as an integer 1-4.

Second(*datetime*)

Description

Determines the second supplied by the *datetime* argument.

Returns

Returns an integer representation for the second part of the date-time value supplied.

Argument

datetime Number of seconds since midnight, 1 January 1904. This can also be an expression.

Short Date(*date*)

Description

Returns a quoted string representation for the *date* supplied in the format MM/DD/YYYY.

Tick Seconds()

Description

Measures the time taken for a script to run, measured down to the 60th of a second.

Notes

For higher time value resolution (for example, microseconds) use the HP Time() function.

Time Of Day(*datetime*)

Description

Returns an integer representation for the time of day of the *datetime* supplied.

Today()

Description

Returns the current date and time expressed as the number of seconds since midnight, 1 January 1904. No arguments are available, but the parentheses are still necessary.

Week Of Year(*date*, <*ruleN*>)

Description

Returns the week of the year that contains a date-time value. Three rules determine when the first week of the year begins.

- With rule 1 (the default), weeks start on Sunday, with the first Sunday of the year being week 2. Week 1 is a partial week or empty.
- With rule 2, the first Sunday begins with week 1, with previous days being week 0.
- With rule 3, the ISO-8601 week number is returned. Weeks start on Monday. Week 1 is the first week of the year with four days in that year. It is possible for the first or last three days of the year to belong to the neighboring year's week number.

Year(*date*)

Description

Returns an integer representation for the year of *date*.

Discrete Probability Functions

Beta Binomial Distribution(*k*, *p*, *n*, *delta*)

Description

Returns the cumulative distribution function (cdf) of the beta binomial distribution. This is the probability that a beta binomially distributed random variable is less than or equal to *k*. The cdf is calculated as the summation of the beta binomial pmf for values of X from 0 to *k*.

Arguments

k The count of interest. *k* must be an integer.

p The probability of success for each trial, which must be between 0 and 1.

n The number of trials, which must be greater than 1.

delta The overdispersion parameter, which must be between Maximum[-*p*/(*n*-*p*-1), -(1-*p*)/(*n*-2+*p*)] and 1. When the overdispersion parameter is zero, the distribution reduces to Binomial(*n*, *p*).

Beta Binomial Probability(*k*, *p*, *n*, *delta*)

Description

Returns the probability mass function (pmf) of the beta binomial distribution. This is the probability that a beta binomially distributed random variable is equal to *k*. The pmf is parameterized as follows:

$$P(X = k;p, n, \delta) = \binom{n}{k} \frac{\Gamma\left(\frac{1}{\delta}-1\right)\Gamma\left[k+p\left(\frac{1}{\delta}-1\right)\right]\Gamma\left[n-k+(1-p)\left(\frac{1}{\delta}-1\right)\right]}{\Gamma\left[p\left(\frac{1}{\delta}-1\right)\right]\Gamma\left[(1-p)\left(\frac{1}{\delta}-1\right)\right]\Gamma\left(n+\frac{1}{\delta}-1\right)}$$

Arguments

k The count of interest. *k* must be an integer.

p The probability of success for each trial, which must be between 0 and 1.

n The number of trials, which must be greater than 1.

delta The overdispersion parameter δ, which must be between Maximum[-*p*/(*n*-*p*-1), -(1-*p*)/(*n*-2+*p*)] and 1. When the overdispersion parameter is zero, the distribution reduces to Binomial(*n*, *p*).

Notes

The beta binomial distribution results from assuming that X|π follows a Binomial(n,π) distribution and π follows a Beta(p(1-δ)/δ,(1-p)(1-δ)/δ) distribution. It is useful when the data are a combination of several Binomial distributions that each have different probabilities of success.

Beta Binomial Quantile(*p*, *n*, *delta*, *cumprob*)

Description

Returns the smallest integer quantile for which the cumulative probability of the Beta Binomial(p, n, delta) distribution is larger than or equal to *cumprob*.

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of trials, which must be greater than 1.

delta The overdispersion parameter δ, which must be between Maximum[-*p*/(*n*-*p*-1), -(1-*p*)/(*n*-2+*p*)] and 1. When the overdispersion parameter is zero, the distribution reduces to Binomial(*n*, *p*).

cumprob The cumulative probability of the quantile desired. *cumprob* must be between 0 and 1.

Binomial Distribution(*p*, *n*, *k*)

Description

Returns the cumulative distribution function (cdf) of the binomial distribution. This is the probability that a binomially distributed random variable is less than or equal to *k*. The cdf is calculated as the summation of the binomial pmf for values of X from 0 to *k*.

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of trials.

k The number of successes, which must be less than or equal to *n*.

Binomial Probability(*p*, *n*, *k*)

Description

Returns the probability mass function (pmf) of the binomial distribution. This is the probability that a binomially distributed variable is equal to *k*. The pmf is parameterized as follows:

$$P(X = k;p, n) = \binom{n}{k} p^k (1-p)^{n-k}$$

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of trials.

k The number of successes, which must be less than or equal to *n*.

Binomial Quantile(*p*, *n*, *cumprob*)

Description

Returns the smallest integer quantile for which the cumulative probability of the Binomial(p, n) distribution is larger than or equal to *cumprob*.

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of trials.

cumprob The cumulative probability of the quantile desired. *cumprob* must be between 0 and 1.

Gamma Poisson Distribution(*k*, *lambda*, *sigma*)

Description

Returns the cumulative distribution function (cdf) of the gamma-Poisson distribution. This is the probability that a gamma-Poisson distributed random variable is less than or equal to *k*. The cdf is calculated as the summation of the gamma-Poisson pmf for values of X from 0 to *k*.

Arguments

k The count of interest. *k* must be an integer.

lambda The shape parameter λ, which much be greater than 0. This is the mean of the distribution.

sigma The overdispersion parameter σ, which must be greater than or equal to 1. When the overdispersion parameter is 1, the distribution reduces to a Poisson(λ) distribution.

Gamma Poisson Probability(*k*, *lambda*, *sigma*)

Description

Returns the probability mass function (pmf) of the gamma-Poisson distribution. This is the probability that a gamma-Poisson distributed random variable is equal to *k*. The pmf is parameterized as follows:

$$P(X = k;\lambda, \sigma) = \frac{\Gamma\left(k + \frac{\lambda}{\sigma - 1}\right)}{\Gamma(k+1)\Gamma\left(\frac{\lambda}{\sigma - 1}\right)}\left(\frac{\sigma - 1}{\sigma}\right)^{k}\left(\sigma^{-\left(\frac{\lambda}{\sigma - 1}\right)}\right)$$

where $\Gamma(\cdot)$ is the Gamma function.

Arguments

k The count of interest. *k* must be an integer.

lambda The shape parameter λ, which much be greater than 0. This is the mean of the distribution.

sigma The overdispersion parameter σ, which must be greater than or equal to 1. When the overdispersion parameter is 1, the distribution reduces to a Poisson(λ) distribution.

Notes

The gamma Poisson distribution results from assuming that $X|\mu$ follows a Poisson(μ) distribution and μ follows a Gamma($\lambda/(\sigma-1)$,$\sigma-1$) distribution. It is useful when the data are a combination of several Poisson(μ) distributions that each have different values of μ.

Gamma Poisson Quantile(*lambda*, *sigma*, *cumprob*)

Description

Returns the smallest integer quantile for which the cumulative probability of the Gamma Poisson(lambda, sigma) distribution is larger than or equal to *cumprob*.

Arguments

lambda The shape parameter λ, which much be greater than 0. This is the mean of the distribution.

sigma The overdispersion parameter σ, which must be greater than or equal to 1. When the overdispersion parameter is 1, the distribution reduces to a Poisson(λ) distribution.

cumprob The cumulative probability of the quantile desired. *cumprob* must be between 0 and 1.

Hypergeometric Distribution(*N*, *K*, *n*, *x*, <*r*>)

Description

Returns the cumulative distribution function (cdf) of the hypergeometric distribution. This is the probability that a hypergeometrically distributed random variable is less than or equal to *x*. The cdf is calculated as the summation of the hypergeometric pmf for values of X from 0 to *x*.

Required Arguments

N The population size.

k The number of items in the category of interest.

n The sample size.

x The count of interest, which must be less than or equal to *n* and *k*.

Optional Argument

r The odds ratio.

Hypergeometric Probability(*N*, *k*, *n*, *x*, <*r*>)

Description

Returns the probability mass function (pmf) of the hypergeometric distribution. This is the probability that a hypergeometrically distributed random variable is equal to *x*. The pmf is parameterized as follows:

$$P(X = x; N, n, k) = \frac{\binom{k}{x}\binom{N-k}{n-x}}{\binom{N}{n}}, \; n - x \le N - k$$

Required Arguments

N The population size.

k The number of items in the category of interest.

n The sample size.

x The count of interest, which must be less than or equal to *n* and *k*.

Optional Argument

r The odds ratio.

Neg Binomial Distribution(*p*, *n*, *k*)

Description

Returns the cumulative distribution function (cdf) of the negative binomial distribution. This is the probability that a negative binomially distributed random variable is less than or equal to *k*. The cdf is calculated as the summation of the negative binomial pmf for values of X from 0 to *k*.

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of successes.

k The number of failures before the n^{th} success.

Neg Binomial Probability(*p*, *n*, *k*)

Description

Returns the probability mass function (pmf) of the negative binomial distribution. This is the probability that a negative binomially distributed random variable is equal to *k*. The pmf is parameterized as follows:

$$P(X = k;p, n) = \binom{n+k-1}{k} p^n (1-p)^k$$

Arguments

p The probability of success for each trial. *p* must be between 0 and 1.

n The number of successes.

k The number of failures before the n^{th} success.

Notes

The return value of the pmf is the probability of observing the n^{th} success after *k* failures have occurred.

Poisson Distribution(*lambda*, *k*)

Description

Returns the cumulative distribution function (cdf) of the Poisson distribution. This is the probability that a Poisson distributed random variable with mean *lambda* is less than or equal to *k*. The cdf is calculated as the summation of the Poisson pmf for values of X from 0 to *k*.

Arguments

k The number of events in a given time interval. *k* must be an integer.

lambda The shape parameter λ, which much be greater than 0. This is the mean of the distribution.

Poisson Probability(*lambda*, *k*)

Description

Returns the probability mass function (pmf) of the Poisson distribution. This is the probability that a Poisson distributed random variable with mean *lambda* is equal to *k*. The pmf is parameterized as follows:

$$P(X = k;\lambda) = \frac{e^{-\lambda}\lambda^{k}}{k!}$$

Arguments

k The number of events in a given time interval. *k* must be an integer.

lambda The shape parameter λ, which must be greater than 0. This is the mean of the distribution.

Poisson Quantile(*lambda*, *cumprob*)

Description

Returns the smallest integer quantile for which the cumulative probability of the Poisson(lambda) distribution is larger than or equal to *cumprob*.

Arguments

lambda The shape parameter λ, which must be greater than 0. This is the mean of the distribution.

cumprob The cumulative probability of the quantile desired. *cumprob* must be between 0 and 1.

Display Functions

Alpha Shape(Triangulation())

Description

Returns the alpha shape for the given triangulation.

Border Box(<Left(*pix*)>, <Right(*pix*)>, <Top(*pix*)>, <Bottom(*pix*)>, <Sides(*Boolean*)>, *db*)

Description

Constructs a bordered display box that contains another display box. Optional arguments (Left, Right, Top, Bottom) add space between the border box and what it contains. The other optional argument (Sides) draws borders around the border box on any single side or combination of sides; draws the border in black or the highlight color; makes the background transparent or white or erases the background of a display box that contains it.

Returns

The display box.

Required Argument

db A display box object (for example, a text box or another border box).

Optional Arguments

Left(*pix*) An integer that measures pixels.

Right(*pix*) An integer that measures pixels.

Top(*pix*) An integer that measures pixels.

Bottom(*pix*) An integer that measures pixels.

Sides(*pix*) An integer that maps to settings for the display box.

Notes

The formula for deriving the integer for Sides is: 1*top + 2*left + 4*bottom + 8*right + 16*highlightcolor + 32*whitebackground + 64*erase. Thus, if you want to just draw a black border on the top and bottom, 1+4 = 5. If you want that same box with a white background, 5+32 = 37.

Box Plot Seg(<*data*>, <*frequency*>, <*weight*>, <Vertical(*Boolean*)>)

Description

Returns a display seg that represents a box plot based on the passed x and y values.

Returns

The display box (a box plot).

Optional Arguments

data The data values within the box plot.

frequency The frequency values within the box plot.

weight The weights for observations on continuous Ys.

Vertical(*Boolean*) A vertical (1) or horizontal (0) box plot.

Example

```
win = New Window( "Box Plot Seg Example",
Graph Box(
   Frame Size( 40, 180 ),
   Y Scale( 0, 100 ),
   Box Plot Seg(
      [20, 30, 40], // data
      [1, 1, 3], // frequencies
      [1, 1, 1], // weights
      1 // vertical
      )
   )
);
```

Busy Light(< <<Automatic(*Boolean*)>, <Size(*x*, *y*)>, < <<Disable>)

Description

Creates a rotating image to indicate a busy process.

Returns

A rotating image.

Optional Argument and Messages

<<Automatic(*Boolean*) Rotates the image.

Size(*x*, *y*) Specifies the size of the image in pixels.

<<Disable Hides the image.

Example

```
win = New Window( "Example",
  blb = Busy Light( <<Automatic( 1 ), Size( 50, 50 ) ) );
```

Button Box(*title*, *script*, < *<<Set Icon(path)*>, < <<Set Icon Location(left|right)>

Description

Constructs a button with the text *title* that executes *script* when clicked.

Returns

The display box (button box).

Required Arguments

title A quoted string or a string variable.

script A quoted string or a reference to a string that specifies a valid JSL script.

Optional Messages

<<Set Icon("*path*") Displays the image in the quoted pathname on the button. Most common graphic formats are supported, such as GIF, JPG, PNG, BMP, TIF. Since the title argument is optional, you can create a button with only a text title, with only an icon, or with both a text title and an icon. In the last case, the icon is placed next to the text title.

<<Set Icon Location("left"|"right") Allows the position of the icon on a button to be either left or right of the text.

Example

The following example shows a simple button box. When the user clicks the button box, "Pressed" is printed to the log.

```
win = New Window( "Simple Example",
  ex = Button Box( "Press me" )
);
ex << Set Script( Print( "Pressed" ) );
```

Notes

Line-break characters are ignored in button boxes.

Calendar Box(*title*, < <<Date>, < <<Min Date>, < <<Max Date>, < <<Show Time>)

Description

Constructs a pop-up calendar with selectable days and time.

Returns

The display box (calendar box).

Required Argument

title A quoted string or a string variable.

Optional Messages

<<Date The currently selected date.

<<Min Date The earliest date that can be selected.

<<Max Date The latest date that can be selected.

<<Show Time The time that can be specified.

Example

The following example creates a calendar with October 5, 1989 initially selected. The minimum date and maximum date are specified, so the user can select only dates in that range.

```
New Window( "Calendar Box Example", cal = Calendar Box() );
date = Date MDY (10, 5, 1989);
cal << Date( date );
cal << Show Time( 0 ); // omit the time

/* earliest date that can be selected is 60 days before 10/5/1989
"start" truncates the value so the time is not included */
cal << Min Date( Date Increment(date, "Day", -60, "start" ) );

// latest date that can be selected is 60 days after 10/5/1989
cal << Max Date( Date Increment(date, "Day",  60, "start" ) );

cal << Set Function( Function( {this}, Print( Abbrev Date(this << Get Date())
  ) ) ); // print the abbreviated date to the log
```

Cell Plot(*Y(column(s))*, *<X(column)>*)

Description

Displays each value in a cell graph.

Check Box({*list*}, <*script*>, < <<Get(*n*)>, < <<Set(*n*, *Boolean*)>, < <<Get Selected>, < <<Enable Item(*n*, *Boolean*)>, < <<Item Enabled(*check box item*)>)

Description

Constructs a display box to show one or more check boxes.

Returns

The display box (Check Box).

Required Argument

list A list of quoted strings or a reference to a list of strings.

Optional Argument

script An optional JSL script.

Optional Messages

<<Get(*n*) Returns 1 if the check box item specified by *n* is selected, or 0 otherwise.

<<Set(*n*, *Boolean*) Sets the check box item specified by *n* as either selected (1) or cleared (0).

<<Get Selected Returns a list of quoted strings that contain the names of the check box items that are selected.

<<Enable Item(*n*, *Boolean*) Sets the check box item specified by *n* as either enabled (1) or disabled (0). The state of a disabled check box cannot be changed.

<<Item Enabled(*check box item*) Returns 0 or 1 depending on whether the specific check box item is enabled.

Example

Create three check boxes labeled “one”, “two”, and “three”. The first check box is selected.

```
New Window( "Example", Check Box( {"one", "two", "three"}, <<Set( 1, 1 ) ) );
```

Col Box(*title*, *display boxes*)

Description

Returns a column box made up of the specified display boxes.

Arguments

title The quoted title of the column.

display boxes Display boxes that hold content within the column box.

Example

```
win = New Window( "Example",
   exx = 1;
   exy = 4;
   exz = 8;
   Table Box(
      String Col Box( "strings", {"x", "y", "z"} ),
      Col Box(
         "boxes",
```

```
            Slider Box( 0, 10, exx, Show( exx ) ),
            Slider Box( 0, 10, exy, Show( exy ) ),
            Slider Box( 0, 10, exz, Show( exz ) )
        )
    );
);
```

`Col List Box(<Data Table(`*name*`)>, <"All"|"Character"|"Numeric">, <width(`*pixels*`)>, <"Grouped">, <MaxSelected(`*n*`)>, <nLines(`*n*`)>, <MaxItems(`*n*`)>, <MinItems(`*n*`)>, <On Change(`*expr*`)>, < <<Modeling Type("Any"|"Continuous"|"Ordinal"|"Nominal"|Multiple Response"|"Unstructured Text"|"Vector"|"None"|"Row State") >, < <<Set Data Type(Any|Numeric|Character)>, <`*script*`>)`

Description

Constructs a display box to show a list box that allows selection of data table columns.

Returns

The display box (Col List Box).

Optional Arguments

`Data Table(`*name*`)` The quoted name of the data table.

`"All" | "Character" | "Numeric"` Adds all columns of the current data table into the list. Omitting `"All"` results in an empty col list box with the "optional" label. To display "optional character", specify `"Character"`. To display "optional numeric", specify `"Numeric"`.

`width(`*pixels*`)` Sets the width of the list box to `pixels`. `pixels` is a number that measures pixels.

`"Grouped"` Displays grouped columns in the box.

`MaxSelected(`*n*`)` Sets whether only one item can be selected. For *n*>1, *n* is ignored.

`nLines(`*n*`)` Sets the length of the list box to n number of lines. *n* is an integer.

script A script.

`MaxItems(`*n*`)` A number that allows only *n* columns to be added to the list.

`MinItems(`*n*`)` A number that only requires at least *n* columns for the list. If *n*=2, the top two slots in the col list box an initial display of "required numeric" (or whatever you set the data type to be).

`On Change(`*expr*`)` Evaluates the expression when the selection in the list changes. Dragging between two column list boxes that have this argument results in both expressions being evaluated. The expression for the target being dragged is evaluated first, then the expression for the source is evaluated.

Optional Messages

`<<Set Tips({`*Tip text 1*`, `*Tip text 2*`, ...})` Quoted strings that set tool tips for items in the list box. A quoted null string or an empty list results in no tips. A list

shorter that the list of items in the list box will use the last tip text for the remaining items in the list and the list box.

`<<Set Tip(Tip text)` A quoted string that overrides any tool tips set using `Set Tips()` function. If there is a tip set for the box, you cannot set tips for each individual item.

Using `Set Tip()` with no arguments clears the list box tip and allows the individual item tool tips to be displayed.

Notes

- The `MaxSelected(n)` argument only affects whether one or more than one item can be selected. It does not enforce a limit greater than 1.
- Specialty modeling types can be used only in a role (determined by the platform) that explicitly accepts columns of the same type.

Col Span Box(*title*, *display box args*)

Description

Creates spanned columns headers inside a table box. The top column header spans two child column headers.

Returns

The display box (a Col Span Box).

Arguments

title The title that appears in the box.

display box args Display boxes.

Example

```
win = New Window( "Col Span Box",
   <<Modal,
   Table Box(
      Col Span Box(
         "Confidence Limits",
         neb = Number Col Edit Box( "Upper limits", [0, 0] ),
         Number Col Edit Box( "Lower limits", [0, 0] )
      )
   )
);
```

Combo Box(*{items <(tip string)>, ...}*, *<script>*)

Description

Constructs a display box to show a drop-down list.

Returns

The display box (Combo Box).

Arguments

items The items that the user can select.

tip string A quoted string that specifies tooltip text.

script An optional JSL script.

`Context Box(display box, ...)`

Description

Defines a scoped evaluation context. Each Context Box is executed independently of each other.

Returns

A display box.

Arguments

Any number of display boxes.

`Contour Seg(Triangulation, [levels], <zColor([colors]|{colors}, <"Cycle Colors"|"Interpolate Colors">)>, <"Fill"|"Fill Between"|"Fill Below"|"Fill Above">, <Transparency([]|t)>)`

Description

Returns a display seg that represents contours of a Triangulation.

Required Arguments

`Triangulation` The columns to include in the Triangulation.

`[levels]` A matrix of values that control the contour levels that are drawn.

Optional Arguments

`zColor([colors]|{colors}` Colors for each level, specified as a list or matrix.

`"Cycle Colors"|"Interpolate Colors Cycle Colors"` alternates the colors (for example, red, green, red, green). With `Interpolate Colors`, the first contour is red, and the last is green. The contours between smoothly blend the colors.

`"Fill Below"|"Fill Between"|"Fill Above"|"Fill" "Fill Below"` fills the region below the lines. `"Fill Between"` fills only the middle region. `"Fill Above"` fills the region above the lines.. `"Fill"` works like `"Fill Above"`, but the default is to display lines if no Fill options are specified.

`Transparency([]|t)` The transparency specified as a number or matrix.

Example

```
dt = Open( "$SAMPLE_DATA/Cities.jmp" );
tri = Triangulation( X( :X, :Y ), Y( :POP ) );
{xx, yy} = tri << Get Points();
win = New Window( "Contour Seg Example",
  g = Graph Box(
     X Scale( Min( xx ) - .1, Max( xx ) + .1 ),
```

```
            Y Scale( Min( yy ) - .1, Max( yy ) + .1 ),
            Contour Seg(
               tri,
               [0, 400, 1000, 2000, 9000],
               zColor( 5 + [64 32 0 16 48] ),
               Transparency( [1, 1, 1, 1, 1] )
            )
         )
      );
```

Notes

The triangulation is computed using the Xs, and the Y is a continuous variable defined at each position. The [levels] in this case defines values of POP that are drawn as lines, one line per level. If any Fill argument is specified, then the filled regions are [level1, level2], [level2, level3], ..., [level-n].

Current Journal()

Description

Gets the display box at the top of the current (topmost) journal.

Returns

Returns a reference to the display box at the top of the current journal.

Current Report()

Description

Gets the display box at the top of the current (topmost) report window.

Returns

Returns a reference to the display box at the top of the current report window.

Current Window()

Description

Returns a reference to the current window.

Data Filter Context Box(*display box*)

Description

Returns a display box that defines the extent of the local data filters that a display tree contains. Data filters and Data Filter Context Boxes can be arranged in a hierarchy and shared among platforms or boxes that the Data Filter Context Boxes contain.

Data Filter Source Box(*display box*)

Description

Defines which graph is the "source" of the selection filter. Selected rows in reports that are within the Data Filter Source box are included for analysis in the other reports that are within a common Data Filter Context Box.

Data Table Box(*data table*)

Description

Returns a table box the represents the specified data table.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
win = New Window( "Example", Data Table Box( dt ) );
```

Data Table Col Box(*column*)

Description

Returns a column box that corresponds to the specified data table column.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
win = New Window( "Example",
  Table Box( Data Table Col Box( :name ), Data Table Col Box( :height ) )
);
```

Dialog(*contents*)

Description

`Dialog` is deprecated. Use `New Window()` with the `Modal` argument instead.

Excerpt Box(*report*, *subscripts*)

Description

Returns a display box containing the excerpt designated by the report held at number *report* and the list of display subscripts *subscripts*. The subscripts reflect the current state of the report, after previous excerpts have been removed.

Expr As Picture(*Expr*(...), <Width(*pixels*)>)

Description

Converts `expr()` to a picture as it would appear in the Formula Editor.

Returns

Reference to the picture.

Required Argument

Expr(...) Place any valid JSL expression that can be displayed as a picture inside expr().

Optional Argument

Width(*pixels*) Sets the width of the box to pix. pix is a number that measures pixels.

Filter Col Selector(<Data Table(*name*)>, <Width(*pixels*)>, <nLines(*n*)>, <*script*>, <OnChange(*expr*)>)

Description

Returns a display box that contains a list of items. Control allows column filtering.

Get Project(*title*|*index*|*display box*|*window*)

Description

Returns a single project.

Examples

The following examples show how to get the window title of various projects.

```
Get Project( 1 ) << Get Window Title;
// first open project
Get Project( "My Project" ) << Get Window Title;
// first project named “My Project”
Get Project( display box ) << Get Window Title;
// parent project of the specified display box
```

Get Project List()

Description

Returns a list of all open projects.

Example

```
Get Project List() << Get Window Title;
// list of the titles of all open projects
```

Get Window(<Project(*title*|*index*|*display box*|*window*)>, <Type(*string*)>, *title*|*index*|*display box*)

Description

Returns a reference to a specific open window by title, index, or display box. When run in a project, Get Window() returns windows from the current project.

Optional Arguments

Project Specifies the title (a quoted string), index, display box, or window from another project.

Type(*string*) To limit the search to particular types of windows, use the Type() argument and one of these quoted strings: "Data Tables", "Journals", "Reports", or "Dialogs".

Examples

The following examples show how to get the window title of various windows.

```
Get Window( 1 ) << Get Window Title;
// first window in the current project

Get Window( "Big Class" ) << Get Window Title;
// Big Class window in the current project

Get Window( ob ) << Get Window Title;
// parent window of specified display box in the current project

Get Window( Project(), 1 ) << Get Window Title;
// first window, no project (global scope)

Get Window( Project( myProject ), "Big Class" << Get Window Title;
// Big Class window in the specified project
```

Get Window List(<Project(*title*|*index*|*display box*|*window*),><Type(*string*)>)

Description

Returns a list of currently open windows. By default, Get Window List() returns references to open windows in the current project. You can return an open window list from something other than the current project by using the Project() argument. To limit the search to particular types of windows, use the Type () argument and one of these quoted strings: "Data Tables", "Journals", "Reports", or "Dialogs".

Optional Arguments

Project Specifies the title, index, display box, or window from another project.

Type To limit the search to particular types of windows, use the Type () argument and one of these quoted strings: "Data Tables", "Journals", "Reports", or "Dialogs".

Examples

```
Get Window List() << Get Window Title;
// list of open windows in the current project
Get Window List( Type( "Reports" ) ) << Get Window Title;
// list of the titles of open reports in the current project
Get Window List( Project( 0 ), Type( "Reports" ) ); // positional arguments
// list of the titles of open reports outside of a project
Get Window List( 2 );
// second window list
```

Global Box(*global*)

Description

Constructs a box for editing *global* value directly.

Graph()

See "Graph Box(properties, script)".

Graph 3D Box(*properties*)

Description

Constructs a display box with 3-D content.

Returns

The display box.

Arguments

properties Properties can include: Frame Size(x, y), Xname("title"), Yname("title"), Zname("title").

Notes

This display box constructor is experimental.

Graph Box(*properties*, *script*)

Graph(*properties*, *script*)

Description

Constructs a graph with axes.

Returns

The display box (Graph Box).

Arguments

properties Named property arguments: Title("title"), XScale(low, high), YScale(low, high), FrameSize(h, v), XName("x"), YName("y"), SuppressAxes.

script Any script to be run on the graph box.

H Center Box(*<child box>*)

Returns a display box that contains the optional child display box argument. The box is centered in the horizontal space defined by the maximum size of that child display box and all of the other siblings of the center box.

H List Box(<Align("center"|"bottom")>, display box, ...)

Description

Creates a display box that contains other display boxes and displays them horizontally.

Arguments

`Align("center"|"bottom")` Specify center or bottom alignment of the contents in the list box. The contents are bottom aligned by default.

`display box` Any number of display box arguments can be contained in the list box.

H Scroll Box(<Size(*h*)>, *display box*)

Description

Returns a display box that positions a larger child box using a horizontal scroll bar.

Arguments

`Size(`*`h`*`)` The horizontal length of the scroll bar.

Notes

The flexible argument is deprecated. Use `Set Stretch` instead. See "V Scroll Box(<Size(v)>, `display box`)" for an example.

H Sheet Box(<<Hold(*report*), *display boxes*)

Description

Returns a display box that arranges the display boxes provided by the arguments in a horizontal layout. The `<<Hold()` message tells the sheet to own the report(s) that is excerpted.

H Splitter Box(<Size(*h*, *v*)>, *display box*, ...)

Description

Returns a display box that arranges the display boxes provided by the arguments in a horizontal layout (or panel). The splitter enables the user to interactively resize the panel.

Required Argument

`display box` Any number of display box arguments can be contained in the splitter box.

Optional Argument

`Size(`*`h`*`, `*`v`*`)` Specifies the size of the splitter box in pixels. This size is for the outer splitter box. Inner display boxes are proportionately sized according to the width and height of the outer splitter box.

Optional Messages

`<<Size(`*`n`*`)` Specifies the proportions of the last panel. `<<Size(.25)` resizes the last panel to 25% the splitter box height (or width, for vertical splitter boxes).

<<Set Sizes({*n*, *n*}) Specifies the proportions of each panel.
db<<Set Sizes({.75, .25}) sizes the first panel to 75% and the second panel to 25% of the splitter box height (or width, for vertical splitter boxes).

<<Close Panel(*n*, <*Boolean*>) Closes the panel that you specify. <<Close Panel(2) closes the second panel. With three or more panels, you must include the second Boolean value. That value indicates which panel expands to fill the space left by the closed panel.

– <<Close Panel(2,0) closes the second panel; the following sibling takes the extra space.

– <<Close Panel(2,1) closes the second panel; the preceding sibling takes the extra space.

<<Open Panel(*n*, *<Boolean>)* Opens the panel that you specify. With three or more panels, you must include the second Boolean value. Works similar to <<Close Panel described above. The panels are initially opened. You use <<Open Panel only after using <<Close Panel.

<<Get Sizes() Returns the proportions of each panel as a list.

Hier Box(text, Hier Box(...), ...)

Description

Constructs a node of a tree (similar to Diagram output) containing text. Hier Box can contain additional Hier Boxes, allowing you to create a tree. The text can be a quoted string or a Text Edit Box.

Hist Seg([*data*], <[*freq column*]>, <[*weight column*]>, <Vertical(*Boolean*)>, <Row States()>

Description

Returns a histogram seg.

Required Argument

data The data in matrix format.

Optional Arguments

freq column The frequency column in matrix format.

weight column The weight column in matrix format.

Vertical(*Boolean*) Displays the histogram vertically by default (or if set to 1). Display the histogram horizontally by setting the value to 0.

Row States Specifies a data table reference or row states.

Icon Box(*name*)

Description

Constructs a display box containing an icon, where the argument is a name such as Popup, Locked, Labeled, Sub, Excluded, Hidden, Continuous, Nominal, or Ordinal. The argument can also be a path to an image.

Argument

name A quoted string that is the name of a JMP icon or the path to an icon.

Example

`Icon Box( "Nominal" )` constructs a display box that contains the Nominal icon.

`Icon Box( "$SAMPLE_IMAGES/pi.gif" )` inserts the pi.gif sample image.

Notes

- Some icons are used on both Windows and macOS. Other icons are platform specific.
- Consider installing the Built-In JMP Icons add-in. The add-in lets you view icons interactively and see what they look like in different contexts. Download the add-in from https://community.jmp.com/t5/JMP-Add-Ins/Built-In-JMP-Icons/ta-p/42251.

If Box(*Boolean*, *display boxes*)

Description

Constructs a display box whose contents are conditionally displayed.

Arguments

Boolean 1 displays the display boxes inside the `If Box`. 0 does not display them.

display boxes Any display box tree.

If Seg(<State(*Boolean*)>)

Description

Returns a display seg that shows or hides display seg children.

Arguments

`State(`*Boolean*`)` Determines whether the display seg children are shown (1) or hidden (0).

Example

```
lines = [30 20 80 70, 10 90 90 10, 40 20 60 30];
win = New Window( "Lines Seg Example",
   g = Graph Box( If Seg( true, <<Append( Lines Seg( lines ) ) ) )
);
```

Journal Box(*string*)

Description

Constructs a display box that displays the quoted string. We recommend that you do not generate the journal text by hand.

Line Seg(*x*, *y*, <Row States(*dt*|*dt*, [*rows*]|dt, {{*rows*}, ...}|{*row states*})>, <Sizes(*s*)>)

Description

Creates a display seg of connected line segments for the given *x* and *y* values. The optional third argument enables row state assignments from either a data table or independently.

Lines Seg([*x1 y1 x2 y2*, ...])

Description

Returns a display seg with a sequence of line segments for the given *x* and *y* values.

Lineup Box(<nCol(*n*)>, <Spacing(*pixels*, <*vspace*>), *display boxes*, ...)

Description

Constructs a display box to show an alignment of boxes in *n* columns.

List Box({*item*, ...}, <Width(*pixels*)>, <maxSelected(*n*)>, <nLines(*n*)>, <*script*>)

Description

Creates a display box to show a list box of selection items (quoted strings). The argument can be a list of two-item lists containing the item name and a quoted string that specifies the modeling type or sorting order. Item names are case sensitive by default. The icon appears next to the corresponding item in the list box.

Marker Seg(*x*, *y*, <Row States(*dt*|*dt*, [*rows*]|*dt*, {{*rows*}, ...}|{*row states*})>, <Sizes(*s*)>)

Description

Creates a display seg with markers for all of the x and y values. The optional third argument enables row state assignments from either a data table or independently.

`Matrix Box(x)`

`Matrix Box(matrix, < <<Column Names(col1, col2, ...)>, < <<Row Names(row1, row2, ...)>)`

Description

Displays the *matrix* given in the usual array form. Column and row names are quoted strings.

`Mouse Box(display box arguments)`

Description

Returns a box that can make JSL callbacks for dragging and dropping, marking, or clicking and tracking mouse actions.

Arguments

display box arguments Specifies the object that the user interacts with, such as a `Text Box` or `Button Box`. See the Scripting Index in the Help menu.

`Move to Project(<Source(project)|Destination(project)>, <Windows({list of windows to move})>)`

Description

Moves one or more windows into a project or out of a project, or between projects.

Arguments

`Source(project)` The project containing the windows that you want to move.

`Destination(project)` The project to which you want to move the windows.

`Windows({list of windows to move})` A list of windows to move to the project. If omitted, all windows will be moved. Note that the data table and all of its dependent reports will be moved. However, you need to specify only the data table name or report name in the `windows` argument to move it.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
report = dt << Run Script( "Bivariate" );
project = New Project();
// move the report and data table to a new project
Move to Project( Destination( project ), Windows( {report} ) );
```

`New Image()`

`New Image(`*`width`*`, `*`height`*`)`

`New Image(`*`filepath`*`)`

`New Image(Open(`*`url`*`) )`

`New Image(`*`picture`*`)`

`New Image(`*`matrix`*`)`

`New Image(rgb|r|rgba, {`*`matrix`*`, ...})`

Description

Creates a new image which can then be edited using JSL. Valid formats are `png`, `bmp`, `jpeg`, `jpg`, `tiff`, `tif`, and `gif`.

Returns

An image.

Arguments

All argument sets are optional, but all arguments within each set are required.

`width, height` Sets the width and height of the image in pixels.

`filepath` The quoted filepath to an image.

`Open(`*`url`*`)` Opens the image at the specified URL path.

`picture` A JSL picture object.

`matrix` The image as a matrix of JSL color pixels.

`rgb|r|rgba, {`*`matrix`*`, ...}` Specifies the channels (`"rgb"`, `"r"`, or `"rgba"`) and provides a matrix of values (0.0-1.0) for each channel. Examples:

```
New Image( "r", [r matrix] );
New Image( "rgb", {[r matrix], [g matrix], [b matrix]} );
New Image( "rgba", {[r matrix], [g matrix], [b matrix], [a matrix]} );
```

`New Project(`*`arguments`*`)`

Description

Creates a project using the specified script.

Arguments

`<<Add Bookmarks({<File(`*`path`*`)>, <Folder(`*`path`*`, Expanded(`*`Boolean`*`)>)>, <Group(`*`name`*`, <Expanded(`*`Boolean`*`)>, {`*`contents`*`}>)` Creates bookmarks for frequently used files in the project. The argument is a list of bookmark items, each of which is specified using `File()`, `Folder()`, or `Group()`. `Group()` accepts `File()`, `Folder()`, and `Group()` as children.

`<<Reset Layout` Sets the project to use the default layout.

`<<Run Script` Specifies the data tables and reports that appear in the project.

<<Save(<*path*>) Saves the project. Include a quoted path and file name to save the project to a specific location. Save As is an alias.

<<Set Bookmarks({<File(*path*)>, <Folder(*path*, Expanded(*Boolean*)>)>, <Group(*name*, <Expanded(*Boolean*)>, {*contents*}>) Sets the bookmarks for the project. The argument is a list of bookmark items, each of which is specified using File(), Folder(), or Group(). Group() accepts File(), Folder(), and Group() as children.

<<Set Layout Sets the window layout of the project.

<<Show Bookmarks Shows or hides the bookmarks.

<<Show Log Shows or hides the log.

<<Show Window List Shows or hides the Window List.

Example

The following example creates a project from BigClass.jmp and two reports.

```
project = New Project();
project << Run Script(
   dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
   dt << Run Script( "Bivariate" );
   dt << Run Script( "Distribution" );
);
```

New Window(*title*, <*arguments*>, *display box*)

Description

Makes a new window with the required quoted title and a display box tree.

Optional Arguments

<<Script(<*script*>) Creates a new script window. The optional quoted string *script* is placed inside the script window.

<<Journal Creates an empty journal.

<<Size Window(*x*, *y*) Creates a new window of the specified height and width.

<<Modal Makes the new window a modal window, which prevents any other actions in JMP until the window is closed. If you do not include an **OK** or **Cancel** button, one is added automatically for you. **Note:** If used, this argument must be the second argument, directly after the window title. Available modal window arguments are:

– <<On Open(*expr*) runs expr when the window is created.

Note: In data tables, On Open (or OnOpen) scripts that execute other programs are never run. Set the Evaluate OnOpen Scripts preference to control when the script is run.

– <<On Close(*expr*) runs expr when the window is closed. Returns 0 if the window fails to close.

– `<<On Validate(expr)` runs `expr` when the **OK** button is pressed. If it returns True, the window is closed otherwise the window remains open.

– `<<Return Result` changes the window's return value when it closes to match that of the deprecated `Dialog()` function.

`Show Toolbars(Boolean)` Shows or hides the toolbar. The default value is 1. (Windows only.)

`Show Menu(Boolean)` Shows or hides the menu bar. The default value is 1. (Windows only.)

`Suppress AutoHide(Boolean)` Suppresses or uses the auto-hide feature for menus and toolbars. The default value is 1. (Windows only).

Notes

`Dialog()` is deprecated. Use `New Window()` with the `Modal` argument instead.

`Number Col Box(title, numbers)`

Description

Creates a column named *title* (a quoted string) with numeric entries given in list or matrix form.

`Number Col Edit Box(title, {numbers}|[numbers])`

Description

Creates a column named *title* (a quoted string) with numeric entries given in list or matrix form. The numbers can be edited.

`Number Edit Box(initial value, <width>)`

Description

Creates an editable number box that initially contains the *initial value* argument.

Returns

The display box object.

Argument

initial value Any number to use as the initial value. If you use a date or time format, a date and time selector window is created.

<width> Sets the width of the box in characters.

`Outline Box(title, display box, ...)`

Description

Creates a new outline named *title* (a quoted string) containing the listed display boxes.

Page Break Box()

Description

Creates a display box that forces a page break when the window is printed.

Panel Box(*title*, *display box*)

Description

Creates a display box labeled with the quoted string *title* that contains the listed display boxes.

Picture Box(Open(*picture*), *format*)

Description

Creates a display box that contains a graphics picture object.

Returns

A reference to the display box.

Argument

Open(*picture*) Opens the directory that contains the picture.

format Specifies the graphic file format. Specifying the format opens the picture in JMP. If you omit this argument, the picture opens in the default graphics program. Valid formats are the quoted strings "png", "bmp", "jpeg", "jpg", "tiff", "tif", and "gif".

Example

```
New Window( "Example",
   Picture Box( Open( "$SAMPLE_IMAGES/pi.gif", gif ) ) );
```

Platform(*data table*, *script*)

Description

Evaluates the specified script in the context of the specified data table.

Returns

The resulting display box for embedding in a display tree.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
win = New Window( "Platform example",
   H List Box(
      Platform(
         dt,
         Bubble Plot(
            X( :weight ),
            Y( :height ),
```

```
                Sizes( :age ),
                Title Position( 0, 0 )
            )
        ),
        Platform(
            dt,
            Bubble Plot(
                X( :weight ),
                Y( :age ),
                Sizes( :height ),
                Title Position( 0, 0 )
            )
        )
    )
);
```

Plot Col Box(*title*, *numbers*)

Description

Returns a display box labeled with the quoted string *title* to graph the *numbers*. The numbers can be either a list or a matrix.

Poly Seg(*x values*, *y values*)

Description

Returns a display seg that represents a polygon with vertices based on the *x values* and *y values*.

Example

```
x = [10, 50, 90];
y = [10, 90, 10];
win = New Window( "Poly Seg Example",
g = Graph Box( Poly Seg( x, y ) ) );
frame = g[FrameBox( 1 )];
seg = (frame << Find Seg( "Poly Seg" ));
```

Popup Box({*command1*, *script1*, *command2*, *script2*, ...})

Description

Creates a red triangle menu. The single argument is an expression yielding a list of an even number of items alternating between the quoted command string and the expression that you want evaluated when the command is selected. If the command is an empty string, a separator line is inserted.

Notes

Pressing ALT and right-clicking the red triangle menu opens a window with check boxes for the commands.

Radio Box({*item*, ...}, <*script*>)

Description

Constructs a display box to show a set of radio buttons. The items are quoted strings. The optional script is run every time a radio button is selected.

Range Slider Box(*minValue*, *maxValue*, *lowVariable*, *highVariable*, *script*)

Description

`Range Slider Box()` returns a display box that shows a range slider control that ranges from *minValue* to *maxValue*. As the two sliders' positions change, their values are placed into *lowVariable* and *highVariable*, and the script is run.

Returns

The display box (Range Slider Box).

Arguments

minValue, *maxValue* Numbers that set the minimum and maximum value the slider represents.

lowVariable The variable whose value is set and changed by the lower slider.

highVariable The variable whose value is set and changed by the upper slider.

script Any valid JSL commands that are run as the slider is moved.

Report(*obj*)

Description

Returns the display tree of a platform *obj*. This can also be sent as a message to a platform: `obj<<Report.`

Scene Box(*x size*, *y size*)

Description

Creates an *x*-sized by *y*-sized scene box for 3-D graphics.

Scene Display List

Description

Returns a display list for 3-D graphics.

Example

```
ex = Scene Display List();
```

```
ex << Color( .9, .9, .9 );
ex << Text( center, middle, .3, "Hello World" );
exScene = Scene Box( 600, 600 );
exScene << Background Color( 0 );
exScene << Show Arcball( always );
New Window( "See HelloWorld.jsl in sample scripts", exScene );
exScene << Perspective( 45, .2, 20 );
exScene << Translate( 0.0, 0.0, -4.5 );
exScene << Arcball( ex, 1.5 );
exScene << Update;
```

Script Box(<*script*>, <*language*>, <*width*>, <*height*>)

Description

Constructs an editable box that contains the quoted string *script*. The editable box is a script window and can both be edited and run as JSL.

Optional Arguments

script A quoted string that appears in the script box.

language A quoted string that provides syntax highlighting for the specified language. Valid languages are "JSL", "Text", "SAS", "SAS Output", "SASLog", "R", "MATLAB", "JavaScript", "C", "SQL", "Python", "JSON", and "XML".

width An integer that sets the width of the script box.

height An integer that sets the height of the script box.

Example

```
// JSON
New Window( "JSON",
   Script Box(
      "{\!"a\!":1,\!"b\!":\!"test\!"}",
      "JSON"
   )
);
```

Scroll Box(<Size(*h*,*v*)>, *display box*, ...)

Description

Creates a display box that positions a larger child box using scroll bars.

Returns

A reference to the scroll box object.

Required Argument

display box Any number of display box arguments can be contained in the scroll box.

Optional Argument

Size(*h*,*v*) The *h* and *v* arguments specify the size of the box in pixels.

Notes

You can send a scroll box object a message to set the background color:

```
<<Set Background Color( {R, G, B} | <color> )
```

The `Flexible` argument is deprecated. Use the `Set Stretch` message instead. See "V Scroll Box(<Size(v)>, `display box`)" for an example.

You can set the Boolean flags for horizontal (h) and vertical (v) scrolling to enable (1) or disable (0) the scroll bars. If scrolling is disabled in a given direction, the Scroll Box will behave as a regular container in that direction.

```
<<Set Scrollers (h, v)
```

To return the flags for scrolling, use the following message:

```
<<Get Scrollers
```

To set the horizontal (h) and vertical (v) positions (in pixels) for the scrollers on the scroll bar:

```
<<Set Scroll Position (h,v)
```

To return the flags for scroll position, use the following message:

```
<<Get Scroll Position
```

To return the maximum positions for horizontal and vertical scrolling, use the following message:

```
<<Get Scroll Extents
```

Example

The following example shows a window containing a scroll box with the specified settings.

```
win = New Window( "Example",
   sb = Scroll Box(
      Size( 150, 75 ),
      List Box(
         {"First Item", "Second Item",
         "Third Item", "Fourth Item",
         "Fifth Item"},
         width( 200 ),
         max selected( 2 ),
         nLines( 6 )
      )
   )
);
win << Set Window Size( 300, 200 );
sb << Set Scrollers( 1, 1 ); // enable both scroll bars
// position the scrollers on the scroll bar
sb << Set Scroll Position( 0, 20 );
```

Shape Seg({Path(*path*), ...}, <Row States(*dt*|*dt*,[*rows*]|dt,{{*rows*}, ...}|{*row states*})>)

Description

Returns a display seg with a collection of shapes.

Required Argument

Path Specifies the path with an Nx3 matrix or with a text representation. A path matrix has three columns for x, y, and flags for each point in the path. The flag values are 0 for control, 1 for move, 2 for a line segment, 3 for a cubic Bézier segment, and are negative if the point also closes the path. Path text supports SVG syntax.

Optional Arguments

Row States Specifies a data table reference and optionally rows or the actual row state.

Example

```
win = New Window( "Shape Seg Example",
   Graph Box(
      Shape Seg(
         {Path( [10 10 1, 10 70 0, 70 70 0, 70 10 -3] ),
         Path( "M20,20 C20,60 60,60 60,20 Z" )},
         Row States( {Selected State( 1 ), Color State( "red" )} )
      )
   )
);
```

Sheet Box(<<Hold(*rpt*), *display box*, ...)

Description

Returns a display box that can organize other display boxes vertically or horizontally.

Sheet Panel Box(*title*, *display box*)

Description

Specifies whether the Sheet Box should be horizontal or vertical.

Slider Box(*minValue*, *maxValue*, *variable*, *script*, <Set Width(*n*)>, <Rescale Slider(*min*, *max*)>)

Description

Creates an interactive slider control.

Returns

The display box (Slider Box).

Required Arguments

minValue, *maxValue* Numbers that set the minimum and maximum value the slider represents.

variable The variable whose value is set and changed by the slider box.

script Valid JSL commands that are run as the slider box is moved.

Optional Arguments

Set Width(*n*) Specifies the width of the slider box in pixels.

Rescale Slider(*minValue*, *maxValue*) Resets the maximum and minimum values for the slider box.

Notes

You can send Set Width and Rescale Slider as messages to a slider object. For example:

```
ex = .6;
New Window( "Example", mybox = Slider Box( 0, 1, ex, Show( ex ) ) );
mybox << Set Width( 200 ) << Rescale Slider( 0, 5 );
```

Spacer Box(<Size(*h*, *v*)>, <Color(*color*)>)

Description

Creates a display box that can be used to maintain space between other display boxes, or to fill a cell in a LineUp Box.

Returns

A reference to the display box.

Optional Arguments

Size(*h*, *v*) The *h* and *v* arguments specify the size of the box in pixels.

Color(*color*) Sets the color of the box to the JSL color argument.

Spin Box(*script*)

Description

Returns a display box that shows a button with up and down controls.

Argument

script Invoked with an argument that indicates the direction of the arrow clicked. Negative is down, and positive is up. A value of 1 indicates a single click, while larger values indicate a repeating action.

Example

```
win = New Window( "Example",
   Lineup Box(
      2,
      nb = Number Edit Box( 3 ),
      sb = Spin Box( Function( {value}, nb << Increment( value ) ) )
   )
);
nb << Set Increment( 1 );
```

Splitter Box(<Size(*x*, *y*)>, *display box*, ...)

Description

Returns a display box that can organize other display boxes horizontally or vertically with interactive control of sizes. Child sizes are specified as a proportion of the width or height of the splitter box. The optional *Size* argument is used only for the top-most splitter box. Lower level display boxes are sized like any other child box.

Use `H Splitter Box()` or `V Splitter Box()`.

String Col Box(*title*, {*string*, ...})

Description

Creates a column in the data table containing the quoted *string* items. The column is named after the quoted *title*.

String Col Edit Box(*title*, {*string*, ...})

Description

Creates a column in the data table containing the quoted *string* items. The string boxes are editable. The column is named after the quoted *title*.

Notes

To retrieve the data, use the following message:

```
data = obj << Get;
```

Tab Box(Tab Page Box(Title(*page title 1*), <*options*>, *contents of page 1*), Tab Page Box(Title(*page title 2*), <*options*>, *contents of page 2*), ...);

Description

(Previously called `Tab List Box`.) Creates a tabbed window pane. The arguments are an even number of items alternating between the name of a tab page and the contents of the tab page. The page titles must be quoted.

Arguments

`Tab Page Box` Returns a display box that can be used in a tab box or as a stand-alone container with a title.

- `Title(page title #)` specifies the title of page 1.
- Options include <<`Closeable`(*Boolean*) (specifies whether the box can be closed), <<`Tip`(*string*) (a quoted string that specifies a tooltip), and <<`Icon`(*string*) (a quoted string that specifies the icon).
- *contents of page #* is a quoted string that specifies the text on the tab.

Example

```
New Window( "Example",
```

```
Tab Box(
   t1 = Tab Page Box( Title( "alpha" ), Panel Box( "panel", Text Box( "text"
) ) ),
   t2 = Tab Page Box( Title( "beta" ), Popup Box( {"x", ex = 1, "y", ex = 2}
) ),
)
);
```

Notes

Certain messages that you can send to `Tab Page Box` have been renamed:

- `Set Title` is called `Title`.
- `Set Tip` is called `Tip`.
- `Set Icon` is called `Icon`.
- `Set Closeable` is called `Closeable`.

Tab List Box(*title*, *tabExpr1*, ...)

See "Tab Box(Tab Page Box(Title(page title 1), <options>, contents of page 1), Tab Page Box(Title(page title 2), <options>, contents of page 2), ...);".

Tab Page Box(Title(*string*), <*options*>, *contents*)

Description

Returns a display box that can be used in a tab box or as a stand-alone container with a title.

Required Argument

`Title` A quoted string that specifies the title on the tab.

Optional Messages

`<<Tip(`*string*`)` A quoted string that specifies a tooltip.

`<<Closeable(`*Boolean*`)` Specifies whether the page can be closed.

`<<Icon(`*string*`)` A quoted string that specifies the icon.

`<<Set Font(font name, <size>, <"bold"|"italic"|"underline"|strikeout">, <angle>` Specifies the font properties.

`<<Set Font Name("`*font name*`")` Specifies the name of the font.

`<<Set Font Scale(f)` Specifies the scale factor for the font. The scale factor is applied to the size that is determined by the base font and point size.

`<<Set Font Size(n)` Specifies the font size in pixels.

`<<Set Font Style("plain"|"bold")` Specifies the font style.

Notes

Certain messages that you can send to `Tab Page Box` have been renamed:

- `Set Title` is called `Title`.

- Set Tip is called Tip.
- Set Icon is called Icon.
- Set Closeable is called Closeable.

Table Box(*display box, ...*)

Description

Creates a report table with the *display boxes* listed as columns.

Text Box(*text, <arguments>*)

Description

Constructs a box that contains the quoted string *text*.

Arguments

<<Justify Text(*position*) Justifies the text left, center, or right as specified in quotes.

<<Set Wrap(*pixels*) Sets the point at which text wraps.

Text Edit Box(*text, <arguments>*)

Description

Constructs an editable box that contains the quoted string *text*.

Arguments

<<Password Style(*Boolean*) Displays asterisks in the box rather than the password.

<<Set Script Runs the specified script after the text is edited.

<<Set Width(*pixels*) Sets the point at which text wraps.

This Project()

Description

Gets the current project when a JSL script is run from that project.

Example

The following example gets the window title of the current project.

```
project = New Project();
project << Save( "$DOCUMENTS/Test Project.jmpprj" );
project << Run Script(
   New Window( "Project Title",
      Text Box(This Project() << Get Window Title())
   );
);
```

Tree Box(<{*rootnodes*}>, <Size(*width*, *height*)>, <MultiSelect(*Boolean*)>)

Description

Constructs a box to show a hierarchical tree composed of tree nodes.

Arguments

{*rootnodes*} Specifies the names for the root nodes created by `Tree Node()` which the box contains.

`Size(`*width*, *height*`)` Specifies the width and height (in pixels) of the box.

`MultiSelect(`*Boolean*`)` Indicates that more than one item in the tree can be selected.

Tree Node(<*data*>)

Description

Creates a node for display in a Tree Box display. Tree Node is used for both parent and child nodes.

Notes

If you send a root node that contains one or more nodes with the `Set Node Select Script` defining a collapse message, then macOS runs the script twice. Windows doesn't run the script. This behavior on macOS doesn't just affect increments. Any script runs twice. It will print to the log twice, create a column twice, try to delete something twice, and so on.

Triangulation(<*dt*>, X(*col1*, *col1*), <Y(*col*)>)

Description

Returns an object containing the Delaunay triangulation of the given point set. The optional Y will be averaged for duplicate points, and all points in the output will be unique.

Examples

```
tri = Triangulation(
  X( [0 0 1 1], [0 1 0 1] ),
  Y( [0 1 2 3] )
);
dt = Open( "$SAMPLE_DATA/Cities.jmp" );
tri = Triangulation( X( :X, :Y ), Y( :POP ) );
```

V Center Box(*display box...*)

Returns a display box that contains the child display box argument. The box is centered in the vertical space defined by the maximum size of that child display box and all of the other siblings of the center box.

V List Box(<Align("Center"|"Right")>, *display box*, ...)

Description

Creates a display box that contains other display boxes and displays them vertically.

Arguments

`Align` Specify right or center alignment of the contents in the list box. The contents are center aligned by default.

`display box` Any number of display box arguments can be contained in the list box.

V Scroll Box(<Size(*v*)>, *display box*)

Description

Returns a display box that places a scroll bar on the bottom and right if the contents are bigger than the size of the scroll box.

Arguments

`Size(`*`v`*`)` The vertical length of the scroll bar.

`display box` Any number of display box arguments can be contained in the scroll box.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
New Window( "stretchable",
  H Splitter Box(
     Size( 400, 200 ),
     Scroll Box(
        Size( 200, 200 ),
        dt <<Run Script( "Distribution" ),
        <<Set Stretch( "Window", "Window" )
     ),
     Scroll Box(
        Size( 200, 200 ),
        dt <<Run Script( "Bivariate" ),
        <<Set Stretch( "Window", "Window" )
     ),
     <<Set Stretch( "Window", "Window" )
  )
);
```

Notes

The flexible argument is deprecated. Use `Set Stretch` instead.

V Sheet Box(<<Hold(*report*), *display box, ...*)

Description

Returns a display box that arranges the display boxes provided by the arguments in a vertical layout. The <<Hold() message tells the sheet to own the report(s) that is excerpted.

V Splitter Box(<Size(*h*,*v*)>, *display box...*, <*arguments*>)

Description

Returns a display box that arranges the display boxes provided by the arguments in a vertical layout (or panel). The splitter enables the user to interactively resize the panel.

Arguments

Size(*v*) The vertical length of the scroll bar.

display box Any number of display box arguments can be contained in the splitter box.

Optional Arguments

For more information about the optional arguments, see "H Splitter Box(<Size(h, v)>, display box, ...)".

Web Browser Box(*url*)

Description

Creates a display box that contains a web page. Available only on Windows in Internet Explorer.

Returns

A reference to the web browser box object.

Argument

url A quoted string containing the URL to the web page to display.

Example

The following example creates a splitter box with the web browser box on the left and the bubble plot on the right.

```
dt = Open( "$SAMPLE_DATA/PopAgeGroup.jmp" );
New Window( "Example",
   H Splitter Box(
      Size( 800, 300 ),
      wb = Web Browser Box( "http://www.jmp.com" ),
      dt << Run Script( "Bubble Plot Region" )
   )
);
wb << Set Stretch( "Window", "Window" ); // auto stretch horizontally and
   vertically
wb << Set Max Size( 10000, 10000 ); // maximum size in pixels
```

Notes

The <a href> target "_blank" opens the web page in a new Internet Explorer window.
The <a href> target "_new" opens the web page in the active Internet Explorer tab.

`Window(<string|int>)`

Returns

Either a list of references to all open windows, or a reference to an explicitly named window.

Arguments

string A quoted string containing the name of a specific open window.

int The number of a specific open window.

Notes

If no argument is provided, a list of all open windows is returned.

If the argument (either a window name or number) does not exist, an empty list is returned.

`Wrap List Box(display box, ...)`

Description

Creates a list box that contains other display boxes and displays them horizontally, but wraps them when printing.

Arguments

display box Any number of display box arguments can be contained in the list box.

Expression Functions

`Arg(expr, i)`

`Arg Expr(expr, i)`

Description

Finds the argument numbered by *i* within the given expression.

Returns

The *i*th argument within the expression *expr*.

`Empty()` if that argument does not exist or is not specified.

Arguments

expr An expression.

i An integer that indicates which argument to return.

Notes

`Arg Expr()` is deprecated. Use `Arg()` instead.

`Eval Expr(`*`expr`*`)`

Description

Evaluates any expressions within *`expr`*, but leaves the outer expression unevaluated.

Returns

An expression with all the expressions inside *`expr`* evaluated.

Argument

`expr` A valid expression.

`Expr(`*`expr, i`*`)`

Description

Returns the argument unevaluated (expression-quoting).

Returns

The argument, unevaluated.

`Empty()` if that argument does not exist or is not specified.

Argument

`expr` An expression.

`i` An integer that indicates which argument to return.

`Extract Expr(`*`expr`*`, `*`pattern`*`)`

Description

Find *`expr`* matching *`pattern`*.

Returns

A pattern that matches the specified *`pattern`*.

Arguments

`expr` An expression.

`pattern` Any pattern.

`Head(`*`exprArg`*`)`

`Head Expr(`*`exprArg`*`)`

Description

Returns the head of the evaluated expression, without its arguments.

Notes

`Head Expr()` is deprecated. Use `Head()` instead.

Head Name(*expr*)

Head Name Expr(*expr*)

Description

Returns the head of the evaluated expression as a quoted string.

Notes

Head Name Expr() is deprecated. Use Head Name() instead.

N Arg(*exprArg*)

Description

Returns the number of arguments of the evaluated expression head.

N Arg Expr(*exprArg*)

Description

Returns the number of arguments of the expression head.

Notes

N Arg Expr() is deprecated. Use N Arg() instead.

Name Expr(*x*)

Description

Returns the unevaluated expression of *x* rather than the evaluation of *x*.

File Functions

Close(<*dt*|*query*>, <*noSave*|*Save*(*path*)>)

Description

Closes a data table, query, or JSON file. If no arguments are specified, the current file is closed. If the file has been changed, it is automatically saved. All dependent windows are also closed (for example, report windows that are based on the data table).

Returns

Void.

Arguments

dt An optional reference to a data table, query, or JSON file.

nosave|Save(*path*) An optional switch to either save the file to the specified quoted path before closing or to close the file without saving it.

`Close All(<Project(title|index|display box|window)>, type, <"Invisible"|"Private">, <NoSave|Save>)`

Description

Closes all open resources of *type*.

Required Argument

type A named argument that defines the type of resources that you want to close. The allowable types are: `Data Tables`, `Reports`, and `Journals`.

Optional Argument

`Project(title|index|display box|window)` Closes the specified project window.

`"Invisible"` Closes all invisible data tables.

`"Private"` Closes all private data tables.

`NoSave or Save` Saves the specified types of windows before closing or to close without saving.

`Close Database Connection(db connection handle)`

Description

Closes a database connection returned from Create Database Connection.

`Close Log(Boolean)`

Description

Closes the log window.

`Convert File Path(path, <"Absolute"|"Relative">, <"POSIX"|"Windows">, <Base(path)>, <Search>)`

Description

Converts a file path according to the arguments.

Returns

The converted path.

Required Argument

path A quoted pathname that can be either Windows or POSIX.

Optional Arguments

`"Absolute"|"Relative"` Specifies whether the returned pathname is in absolute or relative terms. The default value is `Absolute`.

`"POSIX"|"Windows"` Specifies whether the returned pathname is in Windows or POSIX style. The default is `POSIX`.

`Base(path)` Specifies the quoted base pathname, useful if `Relative` is specified. The default is the default directory. See "Set Default Directory(path)".

`Search` Searches through the specified directories for the specified string. In the example below, JMP searches $SAMPLE_DATA/ and $SAMPLE_DATA/Time Series/ for Air.jmp. If Air.jmp is in both directories, the first instance is returned. Without the search option, `Convert File Path()` uses the default directory instead.

Example

```
Set File Search Path(
   {Convert File Path( "$SAMPLE_DATA/" ),
   Convert File Path( "$SAMPLE_DATA/Time Series/" )}
);
Convert File Path( "Air.jmp", Search );
   Get File Search Path() = {"/C:/Program Files/SAS/JMPPRO/16/Samples/Data/",
      "/C:/Program Files/SAS/JMPPRO/17/Samples/Data/Time Series/"};
   Convert File Path("Air.jmp", search) = "/C:/Program
      Files/SAS/JMPPRO/17/Samples/Data/Time Series/Air.jmp";
```

Copy Directory(*from path*, *to path*, <Recursive(*Boolean*)>)

Description

Copies files from one directory to another, optionally copying subdirectories. The directory name is created in the *to path* and should not be part of the *to path* argument.

Returns

Returns 1 if the directory is copied; otherwise, returns 0.

Required Arguments

from path Specifies the directory containing the files to copy to the new directory. *from path* is a quoted string.

to path Specifies the path where the new directory should be created and to which the files are copied. *to path* is a quoted string.

Optional Argument

Recursive(*Boolean*)> Specifies whether to copy the *from path* subdirectory structure to the *to path*.

Notes

Copy Directory(*from path*, *to path*, *Boolean*) is deprecated.

Copy File(*from path*, *to path*)

Description

Copies one file to a new file using the same or a different name.

Returns

Returns 1 if the file is copied; otherwise, returns 0.

Arguments

from path Specifies the path and file name to copy to the new file. *from path* is a quoted string.

to path Specifies the path and file name for the new file. *from path* is a quoted string.

Note

Copy File() does not replace a file if it already exists.

Create Database Connection(*dataSourceName*|"Connect Dialog",<DriverPrompt(*Boolean*)>);

Description

Creates a connection to the specified database or prompts the user to provide database log in information.

Returns

A handle to the database connection.

Required Argument

dataSourceName The quoted server connection string that contains information such as the data source name and user name.

Optional Arguments

"Connect Dialog" Opens the Select Data Source window, from which the user selects the database.

Driver Prompt(*Boolean*) Enables the ODBC driver to prompt for the connection information if necessary.

Examples

Specify the data source name, user name, and password:

```
Create Database Connection( "dsn=Books;UID=johnsmith;password=Christmas" );
```

Request that the ODBC driver prompt the user to enter connection information, because the quoted connection string does not specify the password:

```
Create Database Connection( "dsn=Books;UID=johnsmith", Driver Prompt( 1 ) );
```

Enable the user to select the data source, specify "Connect Dialog":

```
Create Database Connection( "Connect Dialog" );
```

Create Directory(*path*)

Description

Creates a new directory at the specified path location.

Returns

Returns 1 if the directory is created; otherwise, returns 0.

Arguments

path Specifies the quoted path where the new directory should be located.

`Creation Date(path)`

Description

Returns the creation date for the specified file or directory.

Returns

Creation date.

Arguments

path Specifies the quoted directory or path and file name for the query.

`Delete Directory(path, <Allow Undo(Boolean)>)`

Description

Deletes the specified directory and its contents and any subdirectories.

Returns

Returns 1 if the directory is deleted; otherwise, returns 0.

Arguments

path Specifies the path and directory for deletion.

`Allow Undo(Boolean)` Allows undo operations, for example, moving to the Recycle Bin or Trash Can.

`Delete File(path, <Allow Undo(Boolean)>)`

Description

Deletes the specified file.

Returns

Returns 1 if the file is deleted; otherwise, returns 0.

Required Argument

`path` Specifies the quoted path and file name for deletion.

Optional Argument

`Allow Undo(Boolean)` Allows undo operations, for example, moving to the Recycle Bin or Trash Can.

`Directory Exists(path)`

Description

Verifies the specified directory exists.

Returns

Returns 1 if the directory exists; otherwise returns 0.

Arguments

path Specifies the quoted path and directory for verification.

`File Exists(path)`

Description

Verifies the specified file name exists at the specified path.

Returns

Returns 1 if the file exists; otherwise returns 0.

Arguments

path Specifies the quoted path and file name for verification.

`File Size(path)`

Description

Determines the size of the file within the specified path.

Arguments

path Specifies the quoted path and file name.

Example

```
File Size( "$SAMPLE_DATA/Big Class.jmp" );
```

13142

`Files In Directory(path, <Recursive(Boolean)>)`

Description

Returns a list of filenames in the *path* given.

Returns

List of filenames. If `Recursive(Boolean)` is not specified, directory names are included in the list.

Required Argument

path A valid quoted pathname.

Optional Argument

`Recursive(Boolean)` A keyword that causes all folders in the path (and all folders that they contain, and so on) to be searched for files.

Notes

`Files In Directory(path, "Recursive"|Boolean)` is deprecated.

Find All(<Project(*title*|*index*|*display box*|*window*)>, *type*, <"Invisible"|"Private">)

Description

Finds all open files of the specified type.

Required Argument

type A named argument that defines the type of resources that you want to close. The allowable types are: `Data Tables`, `Reports`, and `Journals`.

Optional Argument

`Project(`*title*`|index|`*display box*`|`*window*`)` Closes the specified project window.

`"Invisible"` Closes all invisible data tables.

`"Private"` Closes all private data tables.

Example

The following example finds all open data tables:

```
exdt1 = Open( "$SAMPLE_DATA/Big Class.jmp" );
exdt2 = Open( "$SAMPLE_DATA/Animals.jmp" );
windows = Find All( Data Tables );
For( i = 1, i <= N Items( windows ), i++,
  Write( Char( windows[i] << Get Window Title ) || "\!N" )
);
  Big Class
  Animals
```

Get Default Directory()

Description

Retrieves the user's default directory. This path is used for subsequent relative paths.

If the default directory was set using `Set Default Directory()`, JMP returns the specified path as long as `Get Default Directory()` and `Set Default Directory()` are in the same script.

See "Set Default Directory(path)".

Returns

The absolute pathname as a quoted string.

Arguments

none

Notes

`Get Default Directory()` also gets the path of an active saved scripting window.

Get Excel Worksheets(*absolute path*)

Description

Retrieves a list of worksheets that are in the specified Microsoft Excel workbook. If no worksheets are found, an empty list is returned. The path is a quoted string.

Notes

The function supports .xlsx and Excel 1997 or later workbooks.

Get File Search Path()

Description

Retrieves the current list of directories to search for opening files.

This list is configured using the `Set File Search Path()` function. See "Set File Search Path(path|{list of paths})".

Returns

A list of pathnames as quoted strings.

Get Path Variable(*name*)

Description

Retrieves the value of name, a path variable.

Returns

The absolute pathname as a quoted string.

Argument

`name` A quoted string that contains a path variable. (Examples: SAMPLE_DATA and SAMPLE_SCRIPTS)

Google Sheet Export(*dt*, *email*, *spreadsheet URL|spreadsheet ID|new spreadsheet name*, *sheet name*)

Description

Exports a data table to a Google sheet.

Returns

"1" if the export is successful.

Arguments

`dt` The data table.

`email` The quoted Google email address. @gmail.com is unnecessary.

`spreadsheet URL|ID` The quoted spreadsheet's URL or ID (the string that precedes "spreadsheets/d/").

`new spreadsheet name` The quoted name of the new spreadsheet that you are creating.

sheet name The quoted name of the sheet (or tab) within the spreadsheet.

Notes

- JMP features such as formulas and List Check column properties are not supported in Google Sheets.
- If the spreadsheet is empty, look in the JMP log for error messages. On Windows, select **View > Log**. On macOS, select **Window > Log**.

See Also

See *Using JMP* for details about security, country restrictions, and more.

Google Sheet Import(*email*, *spreadsheet URL*|*spreadsheet ID*, <*sheet name*|{*sheet name*, *sheet name*}, *Google Sheet Settings*>)

Description

Imports sheets from a Google Sheet.

Returns

A data table (or the first data table imported if several sheets are imported at once).

Required Arguments

email The quoted Google email address. @gmail.com is unnecessary.

spreadsheet URL|*ID* The quoted spreadsheet's URL or ID (which precedes "spreadsheets/d/").

Optional Arguments

sheet name The quoted name of the sheet or sheets that you want to import.

Google Sheet Settings The settings that describe how the data is imported.

See Also

See *Using JMP* for details about security, country restrictions, and more.

Is Directory(*path*)

Description

Returns 1 if the quoted path argument is a directory and 0 otherwise.

Is Directory Writable(*path*)

Description

Returns 1 if the directory specified in the quoted path is writable and 0 otherwise.

Is File(*path*)

Description

Returns 1 if the quoted path is a file and 0 otherwise.

Is File Writable(*path*)

Description

Returns 1 if the file specified in the quoted path is writable and 0 otherwise.

JSON To Data Table(*JSON string*, (<Private(*Boolean*)>|<Invisible(*Boolean*)>), <Guess(Stack(*Boolean*)|"Tall"|"Wide"|"Huge"|"Pandas")>)

Description

Converts a quoted JSON string to a data table.

Returns

A data table reference. The parsing of an empty value, "" string, missing value, or any other invalid value returns an empty data table.

Required Argument

JSON string The quoted JSON string.

Optional Arguments

Private(Boolean) Hides the data table completely. Specify this argument if the user doesn't need to interact with the data table.

Invisible(Boolean) Hides the data table from view but shows it in the JMP Home Window.

Guess(Stack(*Boolean*)|"Tall"|"Wide"|"Huge"|"Pandas") Stack applies to nodes that repeat within a parent node that is creating rows. By default, extra values are stored in a single table cell separated by commas. If the value is 1, repeating values are stacked in extra rows. Be careful stacking data. It can cause non-obvious data errors.

"Tall" imports the data in a tall data table. Select this option when the JSON file contains many rows. This option is the default setting.

"Wide" imports the data in a wide data table. Select this option when the JSON file contains many columns.

"Huge" imports the data in a tall and wide data table.

"Pandas" imports data in the pandas format.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt = JSON to Data Table(
  "[ { \!"name\!": \!"KATIE\!", \!"age\!": 12, \!"sex\!": \!"F\!",
  \!"height\!": 59, \!"weight\!": 95 }, { \!"name\!": \!"LOUISE\!",
  \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!": 61, \!"weight\!": 123 }, {
```

```
    \!"name\!": \!"JANE\!", \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!":
    55, \!"weight\!": 74 } ]"
);
```

Notes

Stacking the data can cause non-obvious data errors. Values that are supposed to be in the same row might not be. Here's an example of stacking data:

```
d =
"\[
{
        "toys": [{
                "Wheels": 2,
                "Color": "Red"
        },
        {
                "Wheels": 4,
                "Size": "Large"
        }]
}
]\";

JSON to Data Table( d, Guess( Stack( 1 ) ) );
```

Figure 2.1 Example of Stacking Data that Shows Errors

	Wheels	Color	Size
1	2	Red	Large
2	4		

The first toy should be red and have 2 wheels. The second toy should be large and have 4 wheels.

JSON To List(*JSON string*)

Description

Converts quoted JSON string to a JSL list that represents the structure specified by the JSON data. The parsing of an empty value, "" string, missing value, or any other invalid value returns {}.

Example

```
l = JSON To List(
  "[ { \!"name\!": \!"KATIE\!", \!"age\!": 12, \!"sex\!": \!"F\!",
  \!"height\!": 59, \!"weight\!": 95 }, { \!"name\!": \!"LOUISE\!",
  \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!": 61, \!"weight\!": 123 }, {
```

```
    \!"name\!": \!"JANE\!", \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!":
    55, \!"weight\!": 74 } ]"
);
Show( l );
```

JSON Literal(*JSON string*)

Description

Returns a valid JSON Boolean or null constant based on the specification of the parameter. The parsing of an empty value, "" string, missing value, or any other invalid value returns `Empty()`.

Last Modification Date(*path*)

Description

Returns the last modification date of the path.

Returns

Last modification date.

Arguments

path The quoted directory or file name.

Load Text File(*path*, <Charset(character set), Force("Throw"|"Alert"|"Silent")>)>, <Line Separator(character)>, <"XMLParse">|<"SASODSXML">|<"JSON">|<BLOB(<arguments>)>)>)

Description

Reads the text file at *path* into a JSL variable.

Returns

A quoted string.

Required Argument

path A quoted pathname that points to a text file. The path can be a URL.

Optional Arguments

`Charset("charater set")` Specifies the character set. Valid character sets are `"Best Guess"`, `"utf-8"`, `"utf-16"`, `"us-ascii"`,`" windows-1252"`, `"x-max-roman"`, `"x-mac-japanese"`, `"shift-jis"`, `"euc-jp"`, `"utf-16be"`, and `"gb2312"`.

`Force("Throw"|"Alert"|"Silent")` Specifies what happens if the character set cannot be detected.

`Line Separator(`*character*`)` Specifies the quoted end-of-line character. For example, `"\!n"` specifies a line feed character. `"\!t"` specifies a tab.

`"XMLParse"` Converts an XML file into JSL.

`"SASODSXML"` Parses the text file as SAS ODS default XML.

"JSON" Converts JSON into an expression tree.

BLOB(<*arguments*>) Returns data from the file as a blob rather than a quoted string. The following optional arguments are for reading parts of the file:

- ReadOffsetFromBegin(*n*) specifies the zero-based offset to begin reading from the beginning of the file.
- ReadOffsetFromEnd(*n*) specifies the zero-based offset to begin reading from the end of the file.
- ReadLength(*n*) specifies the number of bytes to read from the file, either from the beginning of the file or from one of the offset values.
- Base64Compressed(*Boolean*) specifies how the blob is converted to a printable representation. 0, the default and recommended setting, uses JMP's ASCII~HEX representation. 1 means the blob is compressed and converted to base 64 when printed.

Move Directory(*from path*, *to path*)

Description

Moves a directory and its contents (including subdirectories) from the quoted path to another quoted path.

Returns

Returns 1 if the directory is moved; otherwise returns 0.

Arguments

from path Specifies the path and directory for relocation.

to path Specifies the destination path and directory.

Move File(*from path*, *to path*)

Description

Moves a file from the quoted path to another quoted path with the same or different file name.

Returns

Returns 1 if the file is moved. Otherwise, it returns 0.

Arguments

from path Specifies the quoted path and file name for relocation.

to path Specifies the quoted destination path and file name.

Notes

On Windows, when you move a file to a folder that does not exist, Windows creates the folder and returns 1. On macOS, the folder is not created, and an error is returned.

Open(*path*, <*arguments*>)

Description

Opens the data table or other JMP file or object created from a file named by the *path*. If no path is specified, the Open window appears. Also opens JSON and HDF5 files. See the examples in the JMP Scripting Index in the Help menu for more information about which arguments apply to specific file types.

Required Argument

path The quoted path to the file that you want to open.

Optional Arguments

Add to Recent Files(*Boolean*) Determines whether the file is added to the Recent Files list in the Home Window.

Charset(*character set*) The available character set options for importing text files are "Best Guess", "utf-8", "utf-16", "us-ascii," windows-1252", "x-max-roman", "x-mac-japanese", "shift-jis", "euc-jp", "utf-16be", and "gb2312".

Column Names Start(*n*)|Column Names are on Line(*n*) Specifies the line number that column names start in the imported text file. If the text file uses returns between cells, column names could be on multiple lines.

Columns(*colName*=*colType*(*colWidth*),...) Specifies the columns by name in the text file to import into a data table where:

- colName: Specifies the column name used in the imported text file.
- colType("Character"|"Numeric"): Specifies whether the specified column contains character or numeric data.
- colWidth(*n*): Specifies the integer width of the specified column.

Columns(<*arguments*>) For ESRI shapefiles (.shp), this argument and its settings indicate the following:

- Shape=numeric(*n*): Specifies the column number in the imported ESRI shapefile that contains the shape number.
- Part=numeric(*n*): Specifies the column number in the imported ESRI shapefile that contains the part number.
- X=numeric(*n*): Specifies the column number in the imported ESRI shapefile that contains the decimal degree for the longitude (range of ±180°).
- Y=numeric(*n*): Specifies the column number in the imported ESRI shapefile that contains the decimal degree for the latitude (range of ±90°).

"Column Names Only" Returns a list of column names.

Compress Allow List Check(*Boolean*) Specifies that JMP can compress data table created from the imported text file.

Compress Character Columns(*Boolean*) Specifies that JMP should compress data table columns that contain character data from the imported text file.

`Compress Numeric Columns(Boolean)` Specifies that JMP should compress data table columns that contain numeric data from the imported text file.

`Concatenate Worksheets(Boolean)` Specifies that JMP should combine the imported Excel worksheets into one data table.

`Create Concatenation Column(Boolean)` Specifies that JMP should combine columns from an imported Excel file into one column.

`Data Starts(n)|Data Starts on Line(n)` Specifies the line number where data starts in the imported text file.

`Debug JSL(Boolean)` Opens the specified JSL script in the Debugger instead of opening it.

`End Of Field("Tab"|"Space"|"Comma"|"Semicolon"|"Other"|"None")` Specifies the quoted character used to delimit the end of a field in the imported text file. To specify multiple characters, separate each character designation by a comma. If you use `"Other"`, designate the delimiter with `EOF Other()` argument.

`End Of Line("CRLF"|"CR"|"LF"|"Semicolon"|"Other")` Specifies the quoted character used to delimit the end of a line in the imported text file. To specify multiple characters, separate each character designation by a comma. If you use `"Other"`, designate the delimiter with `EOL Other()` argument.

`EOF Other(char)` If the imported text file uses an end-of-field character other than the one specified by `End of Field()`, this argument specifies the character used.

`EOL Other(char)` If the imported text file uses an end of line character other than the one specified by `End of Line()`, this argument specifies the character used.

`"Excel Wizard"` Opens Microsoft Excel worksheets in the Excel Import Wizard. If you omit this argument, the worksheets open directly as a data table.

`"text"|"journal"|"sas"|"script"|"png"|"jmp"` An optional quoted string that specifies the type of file that you are opening. This can be useful if your file does not have a file extension, the file extension of the file does not match the contents of the file, or you want to import a JSL BLOB. If you do not specify this string, the file opens in the default program for the file extension.

Note: The path argument should be used for a zip archive. The extension (.zip) is not required. See "Zip Archives" for the messages that you can send to a zip archive. The basic functionality is to get a list of files in the zip archive, to read a file in the zip archive into either a quoted string or a blob, and to write files into the zip archive. Note that reading a zip archive temporarily puts the contents into memory. Reading very large zip archives can cause errors.

`"Force Refresh"` Closes the specified JMP (.jrn, .jsl, .jrp, or .jmpappsource) file without saving and tries to reopen the file from disk. This argument deletes any changes made since the last time the file was opened.

`HTML Table(n, <ColumnNames(n)>, <DataStarts(n)>)` To import a table from an HTML web page, use the URL as the filepath. The optional *n* argument specifies which

table number, *n*, on the web page to open. If you omit the value, only the first table on the page is imported. The optional `ColumnNames(n)` specifies the row that contains column names. The optional `DataStarts(n)` specifies the row on which the data begins.

Tip: If the table you are importing contains images, they are first imported as text. To load the images in your JMP data table, run the automatically generated table script named Load Pictures. A new expression column containing the images is created. See *Using JMP* for more information about expression columns.

`Ignore Columns(col1, ...)` Specifies the quoted column names in the JMP data table or other JMP file that should not be included in the data table.

`"Invisible"` Opens the file as invisible. This quoted keyword applies to these files: data table, JMP file, external, text, Excel, SAS, ESRI shapefile, and HTML. The data table appears only in the JMP Home Window and the Window menu.

`Labels(Boolean)` Specifies the imported text file contains labels or column headers as the first line in the file. The default value is `True`.

`Lines to Read(n)` Specifies the number of lines in the text file to import. JMP starts counting lines after column names are read.

`Number of Columns(n)` Specifies the number of columns contained in the imported text file.

`Run JSL(Boolean)` Runs the specified JSL file instead of opening it. Include a Boolean argument or an expression that contains a Boolean value. If the script begins with `//!`, which automatically runs the script, include the Boolean value `(0)` to open the script instead.

`Password(password)` Specifies the quoted password for a password-protected SAS file to avoid entering it manually. The password is not encrypted. (Password-protected Microsoft Excel files cannot be imported.)

`"PDF Wizard"` Opens a PDF file in the PDF Import Wizard, where you can control how data is imported.

`"Private"` Opens the table as invisible and without showing it in the JMP Home Window or Window menu. For example, you might create a private data table to hold temporary data that the user does not need to see. This quoted keyword applies to the following files: data table, JMP file, external, text, Excel, SAS, ESRI shapefile, or HTML. Creating a private data table speeds the process of getting to the data; it does not save the computer from allocating the memory necessary to hold the data table data.

`Quarantine Action("Allow Scripts"|"Block Scripts"|"Do Not Open"|"Show Dialog")` Specifies whether scripts run when you open downloaded data tables. Also provides an option to display a window that prompts users to examine or open the data table. If they click Examine, the scripts are disabled.

`Scan Whole File(Boolean)` Specifies how long JMP scans the text file to determine data types for the columns. This is a Boolean value. The default value is true; the entire

file is scanned until the data type is determined. To import large files, consider setting the value to false, which scans the file for five seconds.

`Select Columns(col1, ...)` Specifies the quoted column names in the JMP data table or other JMP file that should be included in the data table.

`"Strip Quotes"|Strip Enclosing Quotes(Boolean)` If the fields in the text file are quoted, setting this to `True` removes the quotes, and setting it to `False` does not remove the quotes. The default value is `True`.

`Table Contains Column Headers(Boolean)` Indicates the imported text file contains labels or column headers as the first line in the file. The default value is `True`.

`"Text Wizard"` Opens the text file in the text import window, where you can select import options. Otherwise, the Text Data Files options in the JMP preferences apply, and the text file is automatically imported as a data table.

`Treat Empty Columns as Numeric(Boolean)` Specifies that JMP should import text file columns of missing data as numeric rather than character. Possible missing value indicators are a period, a Unicode dot, NaN, or a blank quoted string. The default value is `False.`

`Use Apostrophe as Quotation Mark(Boolean)` Declares apostrophes as quotation marks in importing text files. This option is not recommended unless your data comes from a nonstandard source that places apostrophes around data fields rather than quotation marks. The default value is `False`.

`Use Labels for Var Names(Boolean)` For SAS data sets, this option specifies to use SAS labels as JMP columns names. The default value is `False`.

`Use for All Sheets(Boolean)` Indicates that JMP should use the `Worksheets` settings for all worksheets in the Excel file to be opened as a data table.

`Worksheet Settings(Boolean, <options>)` specifies options for importing an Excel file into a JMP data table. Available options are:

- `Has Column Headers(Boolean)` specifies that the Excel file has column headers in the first row.
- `Number of Rows in Headers(n)` specifies the number of rows in the Excel file used as column headers.
- `Headers Start on Row(n)` specifies the row number in the Excel file where the column headers begin. Default is row 1.
- `Data Starts on Row(n)` specifies the row number in the Excel file where the data begins.
- `Data Starts on Column(n)` specifies the column number in the Excel file where the data begins.
- `Data Ends on Row(n)` specifies the row number in the Excel file where data ends.
- `Data Ends on Column(n)` specifies the column number in the Excel file where the data ends.

– `Replicated Spanned Rows(`*`Boolean`*`)` specifies that the Excel file contains spanned columns should be imported into JMP as spanned columns.

– `Suppress Hidden Rows(`*`Boolean`*`)` indicates that JMP should not import rows hidden in the Excel file.

– `Suppress Hidden Columns(`*`Boolean`*`)` indicates that JMP should not import columns hidden in the Excel file.

– `Treat as Hierarchy(`*`Boolean`*`)` indicates that JMP should treat multiple column headers (merged cells) as hierarchies when importing an Excel file. If `True`, the Excel file opens with the merged columns stacked (Tables > Stacked).

`Worksheets("Sheet Name"|{"Sheet Name", "Sheet Name",...}|`*`n`*`)` Opens the specified Excel file worksheet by name, all worksheets in a list of names, or the indexed number of the worksheet. If the worksheets are not specified, all worksheets in the spreadsheet open as separate data tables.

Note: You can import .xls worksheets from a web site by specifying worksheets arguments. .xlsx worksheets cannot be imported from a web site using `Open()`.

`"XML Wizard"` Opens XML files in the XML Import Wizard. If you omit this argument, the XML file opens as a data table.

```
Year Rule | Two Digit Year Rule
  ("1900-1999"|"1910-2009"|"1920-2019"|"1930-2029"|"1940-2039"|
  "1950-2049"|"1960-2059"|"1970-2069"|"1980-2079"|"1990-2089"|"2000-2099")
```

Indicates the year format used in the imported text file. For example, if the earliest date is 1979, use "1970-2069".

Open Database(dataSourceName|"Connect Dialog", "SELECT ..."|"SQLFILE..."|tableName, <"Invisible"|"Private">, <outputTableName>)

Description

Opens the database indicated by *`dataSourceName`* (or opened by the user) with the *`SELECT`* , *`SQLFILE`*, or *`tableName`* arguments.

Returns

A data table named *`outputTableName`*.

Required Argument

`dataSourceName`|`"Connect Dialog"` dataSourceName specifies the name of the data source. `"Connect Dialog"` shows a window from which the user selects the data source.

Optional Arguments

`"Invisible"` Creates an invisible data table that hides the table from view but lists it in the JMP Home Window and Window menu.

`"Private"` Hides the data table completely. Creating a private data table speeds the process of getting to the data; it does not save the computer from allocating the memory necessary to hold the data table data.

`outputTableName` The name of the data table in JMP.

Example

```
Open Database(
   "DSN=dBASE Files;DBQ=C:\Program Files\SAS\JMPPRO\17\Samples\Import Data\;",
   // SQL statement
   "SELECT HEIGHT, WEIGHT FROM Bigclass", // selected columns
   "hw" // name of the output data table
);
```

Parse JSON(JSON string)

Description

Converts JSON string into an associative array or list based on the structure of the JSON data. Convert the result to a list if the parsed JSON object contains more than one member. The parsing of an empty value, "" quoted string, missing value, or any other invalid value returns `Empty()`.

Example

The following example converts JSON into a list:

```
j = Parse JSON(
   "[ { \!"name\!": \!"KATIE\!", \!"age\!": 12, \!"sex\!": \!"F\!",
   \!"height\!": 59, \!"weight\!": 95 }, { \!"name\!": \!"LOUISE\!",
   \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!": 61, \!"weight\!": 123 }, {
   \!"name\!": \!"JANE\!", \!"age\!": 12, \!"sex\!": \!"F\!", \!"height\!":
   55, \!"weight\!": 74 } ]"
);
Show( j );
   j = {["age" => 12, "height" => 59, "name" => "KATIE", "sex" => "F", "weight"
      => 95], ["age" => 12, "height" => 61, "name" => "LOUISE", "sex" => "F",
      "weight" => 123], ["age" => 12, "height" => 55, "name" => "JANE", "sex"
      => "F", "weight" => 74]};
```

Pick Directory(<"Prompt">, <*path*>, <Show Files(*Boolean*)>)

Description

Prompts the user for a directory, returning the directory path as a quoted string.

Returns

The path for the directory that the user selects.

Optional Arguments

`"Prompt"` The quoted string appears at the top of the Browse window (Windows) or the Finder window (macOS).

`path` The quoted string specifies the initial directory that appears in the Pick Directory window.

`Show Files(Boolean)` 1 shows files in the Pick Directory window. 0 hide the files. The default is 0.

`Pick File(<"Prompt">, <initial directory>, <{filter list}>, <first filter>, <Save Flag(Boolean)>, <default file>), <"Multiple">)`

Description

Prompts the user to select one or more files in the Open window.

Returns

The path of the file that the user selects.

Optional Arguments

`"Prompt"` A quoted string. If provided, that string appears at the top of the Open window.

`initial directory` A quoted string that is a valid filepath to a folder. If provided, it specifies where the Open window begins. If not provided, or if it's an empty string, the JMP Default Directory is used.

`filter list` A list of quoted strings that define the filetypes to show in the Open window. See the following example for syntax.

`first filter` An integer that specifies which of the filters in the filter list to use initially. If you use an integer that is too large or small for the list (for example, 4 for a list of 3), the first filter in the list is used.

`Save Flag(Boolean)` A Boolean that specifies whether the Open window or Save window is used. 0 lets the user select a file to open in JMP. 1 lets the user save a new, empty file of the selected type in the selected folder. The default value is 0.

`default file` The quoted name of the file that appears in the window by default.

`"Multiple"` The quoted name that lets the user select multiple files if the `Save Flag` value is 0.

Notes

Although all arguments are optional, they are also positional. For example, you cannot specify a filter list without also specifying the caption and the initial directory.

The buffer size in the computer's physical memory affects the number of files the user can open.

Example

The following script assigns *Select JMP File* as the window title; shows the JMP Samples/Data directory; shows *JMP Files* and *All Files* in the File name list and selects *JMP Files*; displays the Open window; and shows the sample data file name Hollywood Movies.jmp.

```
Pick File(
   "Select JMP File",
   "$SAMPLE_DATA",
```

```
    {"JMP Files|jmp;jsl;jrn", "All Files|*"},
    1,
    0,
    "Hollywood Movies.jmp"
);
```

Rename Directory(*old path name*, *new directory name*)

Description

Renames a directory without moving or copying it.

Returns

Returns 1 if the directory is renamed; otherwise, returns 0.

Arguments

old path name A quoted string that specifies the path and old directory name.

new directory name A quoted string that specifies the new directory name.

Notes

When you specify the *new directory Name*, include only the directory name, not the entire path.

Rename File(*old path name*, *new file name*)

Description

Renames a file without moving or copying it.

Returns

Returns 1 if the file is renamed; otherwise, returns 0.

Arguments

old path name Specifies the path and old file name.

new file name Specifies the new file name.

Notes

When you specify the *new name*, include only the file name, not the entire path.

Save Text File(*path*, *text*|BLOB, <Mode("Replace"|"Append")>)

Description

Returns the path name of the created file. *text* is quoted string.

Set Default Directory(*path*)

Description

Sets the default directory, which is used for resolving relative paths.

See Also

"Get Default Directory()"

Set File Search Path(*path*|{*list of paths*})

Description

Sets the current list of directories to search for opening files. Using {"."} as the path configures JMP to use the current directory.

Example

```
Set File Search Path( {"C:\JMP\13\source", "C:\Program
  Files\SAS\JMPPRO\17\Samples"} );
```

See Also

"Get File Search Path()".

Set Path Variable(*name*, <*path*>)

Description

Sets the path stored in the variable.

Required Argument

name The quoted name of the variable.

Optional Argument

path The path.

TripleS Import(*path*, <*arguments*>)

Description

Imports the specified Triple-S Survey (SSS) file. The SSS format consists of a pair of files: .xml or .sss, and a .csv, .dat, or .asc file. Both sets of files must have the same root name and be in the same folder.

Required Argument

path A quoted string that contains the full path to the .xml or .sss file.

Optional Arguments

"Invisible" A quoted keyword that hides the table from view. The data table appears only in the JMP Home Window and the Window menu. Hidden data tables remain in memory until they are explicitly closed, reducing the amount of memory that is available to JMP. To explicitly close the hidden data table, call Close(*dt*), where *dt* is the data table reference returned by TripleS Import.

Use Labels for Imported Column Names(*Boolean*) Converts the label names to column headings. The default value is 1.

Example

```
dt = TripleS Import( "C:\Data\airlines.sss", "Invisible", Use Labels for
  Imported Column Names( 0 ) );
```

Financial Functions

Double Declining Balance(*cost*, *salvage*, *life*, *period*, <*factor*>)

Description

Returns the depreciation of an asset for a specified period of time. The function uses the double-declining balance method or some other depreciation factor.

Required Arguments

cost The initial cost.

salvage The value at the end of the depreciation.

life The number of periods in the depreciation cycle.

period The length of the period, in the same units as *life*.

Optional Argument

factor The rate at which the balance declines. The default value is 2.

Notes

This function is equivalent to the Excel function DDB.

Future Value(*rate*, *nper*, *pmt*, <*pv*>, <*Type(Boolean)*>)

Description

Returns the future value of an investment that is based on periodic, constant payments and a constant interest rate.

Required Arguments

rate The interest rate.

nper The number of periods.

pmt The constant payment.

Optional Arguments

pv The present value. The default value is 0.

Type(*Boolean*) 0 specifies end-of-period payments. 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function FV.

`Interest Payment(rate, per, nper, pv, <fv>, <Type(Boolean)>)`

Description

Returns the interest payment for a given period for an investment that is based on periodic, constant payments and a constant interest rate.

Required Arguments

rate The interest rate.

per The period for which you want the interest.

nper The total number of periods.

pv The present value.

Optional Arguments

fv The future value. The default value is 0.

Type(Boolean) 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function IPMT.

`Interest Rate(nper, pmt, pv, <fv>, <Type(Boolean)>, <guess>)`

Description

Returns the interest rate per period of an annuity.

Required Arguments

nper The total number of periods.

pmt The constant payment.

pv The present value.

Optional Arguments

fv The future value. The default value is 0.

Type(Boolean) 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

guess Your estimate of the rate The default value is 0.1 (10%).

Notes

This function is equivalent to the Excel function RATE.

`Internal Rate of Return(values, <guess>)`

`Internal Rate of Return(guess, value1, value2, ...)`

Description

Returns the internal rate of return for a series of cash flows in the *values* argument.

Arguments

values A one-dimensional matrix of values. If the second form of the function is used, list each value separately.

guess What you think is near the result. The default value is 0.1 (10%). This argument is required if you specify values. It is required if you specify individual values.

Notes

This function is equivalent to the Excel function IRR.

Modified Internal Rate of Return(*values*, *finance rate*, *reinvest rate*)

Modified Internal Rate of Return(*finance rate*, *reinvest rate*, *value1*, *value2*, ...)

Description

Returns the modified internal rate of return for a series of periodic cash flows. The cost of investment and the interest received on reinvested cash is included.

Arguments

values A one-dimensional matrix of values. If the second form of the function is used, list each value separately.

finance rate The interest rate that you pay on the money in the cash flows.

reinvest rate The interest rate that you receive on the cash flows when you reinvest them.

Notes

This function is equivalent to the Excel function MIRR.

Net Present Value(*rate*, *values*)

Net Present Value(*rate*, *value1*, *value2*, ...)

Description

Returns the net present value of an investment by using a discount rate and a series of future payments (negative values) and income (positive values).

Arguments

rate The discount rate.

values A one-dimensional matrix of values. If the second form of the function is used, list each value separately.

Notes

This function is equivalent to the Excel function NPV.

`Number of Periods(rate, pmt, pv, <fv>, <type(0|1)>)`

Description

Returns the number of periods for an investment that is based on periodic, constant payments and a constant interest rate.

Required Arguments

rate The interest rate.

pmt The constant payment.

pv The present value.

Optional Arguments

fv The future value. The default value is 0.

type(0|1) 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function NPER.

`Payment(rate, nper, pv, <fv>, <type(0|1)>)`

Description

Returns the payment for a loan that is based on constant payments and a constant interest rate.

Required Arguments

rate The interest rate.

nper The total number of periods.

pv The present value.

Optional Arguments

fv The future value. The default value is 0.

type(0|1) 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function PMT.

`Present Value(rate, nper, pmt, <fv>, <type(0|1)>)`

Description

Returns the present value of an investment.

Arguments

rate The interest rate per period.

nper The total number of periods.

pmt The constant payment.

fv The future value. The default value is 0.

type(0|1) 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function PV.

Principal Payment(*rate*, *per*, *nper*, *pv*, <*fv*>, <*type(0|1)*>)

Description

Returns the payment on the principal for a given period for an investment that is based on periodic, constant payments and a constant interest rate.

Required Arguments

rate The interest rate per period.

per The period for which you want the interest.

nper The total number of periods.

pv The present value.

Optional Arguments

fv The future value. The default value is 0.

type(0|1) An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Notes

This function is equivalent to the Excel function PPMT.

Straight Line Depreciation(*cost*, *salvage*, *life*)

Description

Returns the straight-line depreciation of an asset for one period.

Arguments

cost The initial cost of the asset.

salvage The value at the end of the depreciation.

life The number of periods in the depreciation cycle.

Notes

This function is equivalent to the Excel function SLN.

Sum Of Years Digits Depreciation(*cost*, *salvage*, *life*, *per*)

Description

Returns the sum-of-years' digits depreciation of an asset for a specified period.

Arguments

cost The initial cost of the asset.

salvage The value at the end of the depreciation.

life The number of periods in the depreciation cycle.

per The length of the period, in the same units as life.

Notes

This function is equivalent to the Excel function SYD.

Graphics Functions

Add Color Theme({*name*, <*flags*>, {*color*}, <{*position*}>)

Description

Creates a custom color theme that you can apply to components such as markers, data table rows, and treemaps. Add the color theme to the JMP Preferences by including `Add Color Theme(...)` inside `Preferences()`.

Returns

Null.

Required Arguments

`name` The quoted name of the color theme.

`color` Lists of RGB values. These values define the blocks in categorical color themes and the gradients in continuous color themes. Each list of RGB values corresponds to a slider in the preferences Color Themes window.

`position` An optional list of numbers between 0 and 1 with one position per color. Each position corresponds to a slider in the preferences Color Themes window. If you omit the position, the sliders are evenly spaced.

Optional Argument

`flags` A flag for the Continuous or Categorical color theme list and category of color.

```
Continuous, <Continuous>, Sequential
Continuous, <Continuous>, Diverging
Continuous, <Continuous>, Chromatic
Categorical, <Continuous>, Sequential
Categorical, <Continuous>, Diverging
Categorical, Qualitative
Categorical, <Continuous>, Chromatic
```

With the default JMP color themes, Sequential colors transition from left to right or right to left. Diverging colors are lighter in the middle. Chromatic colors consist of blocks or gradients of bright color. All categories except for Qualitative can be both continuous and categorical.

If you omit the flag, the color is shown in the Continuous, Sequential and Categorical, Sequential categories.

Examples

The following example creates a continuous color theme named Blue to Purple. The color is in the Diverging category. RGB values are defined in the four lists.

```
Add Color Theme(
   {"Blue to Purple", {"Continuous", "Diverging"}, {{0, 0, 255},
   {57, 108, 244}, "white", {128, 0, 100}}} );
```

Notes

Any style except for Qualitative can be Continuous and Categorical at the same time. For example, the Cool to Warm Diverging theme is in the Continuous and Categorical theme lists. In JMP, select **Preferences > Graphs** to see examples.

To delete a color theme, use `Remove Color Theme()`.

See Also

"Remove Color Theme(name|{name, <flags>, {color, ...}, <{position, ...}>}>)"

Arc(*x1*, *y1*, *x2*, *y2*, *startangle*, *endangle*)

Description

Inscribes an arc in the rectangle described by the arguments.

Returns

Null.

Arguments

x1, *y1* The point at the top left of the rectangle

x2, *y2* The point at the bottom right of the rectangle

startangle, *endangle* The starting and ending angle in degrees, where 0 degrees is 12 o'clock and the arc or slice is drawn clockwise from *startangle* to *endangle*.

Arrow(<*pixellength*>, {*x1*, *y1*}, {*x2*, *y2*})

Description

Draws an arrow from the first point to the second point. The optional first argument specifies the length of the arrow's head lines (in pixels).

Returns

Null.

Required Arguments

{*x1*, *y1*}, {*x2*, *y2*} Two lists of two numbers that each specify a point in the graph.

Optional Argument

pixellength The length of the arrowhead in pixels.

Notes

The two points can also be enclosed in square brackets: `Arrow(<pixellength>, [x1, x2], [y1, y2])`.

`Back Color(name)`

Description

Sets the color used for filling the graph's background.

Returns

Null.

Argument

`name` A quoted color name or a color index (such as `"red"` or `"3"` for the color red).

`Char To Path(path)`

Description

Converts a path specification from a quoted string to a matrix.

Returns

A matrix.

Arguments

`path` A quoted string that contains the path specification.

`Circle({x, y}, radius|PixelRadius(n), <...>, <"Fill">)`

Description

Draws a circle centered at {x, y} with the specified radius.

Returns

Null.

Required Arguments

`{x, y}` A number that describes a point in the graph

`radius` The length of the circle's radius in relation to the vertical axis. If the vertical axis is resized, the circle is also resized.

`PixelRadius(n)` The length of the circle's radius in pixels. If the vertical axis is resized, the circle is *not* resized.

Optional Argument

`"Fill"` A positional argument indicating that all circles defined in the function are filled with the current fill color. If `"Fill"` is omitted, the circle is empty.

Notes

The center point and the radius can be placed in any order. You can also add additional center point and radius arguments and draw more than one circle in one statement. One

point and several radii results in a bull's-eye. Adding another point still draws all previous circles, and then adds an additional circle with the last radius specified. This means that this code:

```
Graph Box(circle({20, 30}, 5, {50, 50}, 15))
```

results in three circles, not two. First, a circle with radius 5 is drawn at 20, 30. Second, a circle with radius 5 is drawn at 50, 50. Third, a circle with radius 15 is drawn at 50, 50.

Color To HLS(*color*)

Description

Converts the *color* argument (including any JMP color) to a list of HLS values.

Returns

A list of the hue, lightness, and saturation components of *color*. The values range between 0 and 1.

Argument

color A number from the JMP color index.

Example

The output from ColorToHLS() can either be assigned to a single list variable or to a list of three scalar variables:

```
hls = Color To HLS( 8 );
{h, l, s} = Color To HLS( 8 );
Show( hls, h, l, s );
  hls = {0.778005464480874, 0.509803921568627, 0.976};
  h = 0.778005464480874;
  l = 0.509803921568627;
  s = 0.976;
```

Color To RGB(*color*)

Description

Converts the *color* argument (including any JMP color) to a list of RGB values.

Returns

A list of the red, green, and blue components of *color*. The values range between 0 and 1.

Argument

color A number from the JMP color index.

Example

The output from ColorToRGB() can either be assigned to a single list variable or to a list of three scalar variables:

```
rgb = Color To RGB( 8 );
{r, g, b} = Color To RGB( 8 );
Show( rgb, r, g, b );
  rgb = {0.670588235294118, 0.0313725490196078, 0.988235294117647};
```

```
r = 0.670588235294118;
g = 0.0313725490196078;
b = 0.988235294117647;
```

Contour(*xVector*, *yVector*, *zGridMatrix*, *zContour*, <*messages*>)

Description

Draws contours given a grid of values.

Returns

None.

Required Arguments

xVector The *n* values that describe *zGridMatrix*.

yVector The *m* values that describe *zGridMatrix*.

zGridMatrix An *nxm* matrix of values on some surface.

zContour The values for the contour lines.

Optional Messages

<<zColor(color, option) The colors to use for the contour lines.

<<"Fill"|"Fill Between"|"Fill Below"|"Fill Above" Specifies where the contour fill appears.

<<Transparency(vector) Specifies the transparency level of the fill in a vector.

Contour Function(*zExpr*, *xName*, *yName*, (*z*|*zMatrix*), < <<xGrid(*min*, *max*, *incr*)>, < <<yGrid(*min*, *max*, *incr*)>, < <<zColor(*color*, *option*)>, < <<*zLabeled*>, < <<"Filled">, < <<"FillBetween">, < <<"Ternary">, < <<Transparency(*number*|*matrix*))

Description

Draws sets of contour lines of the expression, a function of the two symbols. The *z* argument can be a single value or an index or matrix of values.

Returns

None.

Arguments

zExpr Any expression (for example, Sine(y)+Cosine(x)).

xName, yName The values to use in the expression.

z|zMatrix A z-value or a matrix of z-values.

Optional Messages

<<xGrid, <<yGrid Defines a box, beyond which the contour lines are not drawn.

<<zColor Defines the color in which to draw the contour lines. The argument can be either a scalar or a matrix but must evaluate to numeric.

<<zLabeled Labels the contours.

<<"Filled" Fills the contour levels using the current fill color.

<<"FillBetween" Fills only between adjacent contours using the current fill color. For nz contours specified, this option fills nz-1 regions for the intervals between the nz values. Using this option is recommended over using the <<Filled option.

<<"Ternary" Clips lines to be within the ternary coordinate system inside ternary plots.

<<Transparency(*number|matrix*) Sets the transparency level of the fill. A vector of numbers between 0 and 1 are sequenced through and cycled for the z contours. This option should be used only in conjunction with the <<FillBetween option.

Drag Line(*xMatrix*, *yMatrix*, <*dragScript*>, <*mouseUpScript*>)

Description

Draws line segments between draggable vertices at the coordinates given by the matrix arguments.

Returns

None.

Required Arguments

xMatrix A matrix of *x*-coordinates.

yMatrix A matrix of *y*-coordinates.

Optional Arguments

dragScript Any valid JSL script. The script is run at drag.

mouseUpScript Any valid JSL script. The script is run at mouseup.

Drag Marker(*xMatrix*, *yMatrix*, <*dragScript*>, <*mouseUpScript*>)

Description

Draws draggable markers at the coordinates given by the matrix arguments.

Returns

None.

Arguments

xMatrix A matrix of *x*-coordinates.

yMatrix A matrix of *y*-coordinates.

dragScript Any valid JSL script. The script is run at drag.

mouseUpScript Any valid JSL script. The script is run at mouseup.

Drag Polygon(*xMatrix*, *yMatrix*, <*dragScript*>, <*mouseupScript*>)

Description

Draws a filled polygon with draggable vertices at the coordinates given by the matrix arguments.

Returns

None.

Required Arguments

xMatrix A matrix of *x*-coordinates.

yMatrix A matrix of *y*-coordinates.

Optional Arguments

dragScript Any valid JSL script. The script is run at drag.

mouseUpScript Any valid JSL script. The script is run at mouseup.

`Drag Rect(xMatrix, yMatrix, <dragScript>, <mouseupScript>)`

Description

Draws a filled rectangle with draggable vertices at the first two coordinates given by the matrix arguments.

Returns

None.

Required Arguments

xMatrix A matrix of two *x*-coordinates.

yMatrix A matrix of two *y*-coordinates.

Optional Arguments

dragScript Any valid JSL script. The script is run at drag.

mouseUpScript Any valid JSL script. The script is run at mouseup.

Notes

xMatrix and *yMatrix* should each contain exactly two values. The resulting coordinate pairs should follow the rules for drawing a `Rect()`. The first point (given by the first value in *xMatrix* and the first value in *yMatrix*) must describe the top, left point in the rectangle. The second point (given by the second value in *xMatrix* and the second value in *yMatrix*) must describe the bottom, right point in the rectangle.

`Drag Text(xMatrix, yMatrix, string, <dragScript>, <mouseUpScript>)`

Description

Draws the *string* (or all the items if a list is specified) at the coordinates given by the matrix arguments.

Returns

None.

Arguments

xMatrix A matrix of *x*-coordinates.

yMatrix A matrix of *y*-coordinates.

string A quoted string to be drawn in the graph.

dragScript Any valid JSL script. The script is run at drag.

mouseUpScript Any valid JSL script. The script is run at mouseup.

Fill Color(*name|color|rgbList*)

Description

Sets the color used for filling solid areas.

Returns

None.

Arguments

name|color|rgbList The quoted color name, index, or list of RGB values.

Fill Pattern(*name|mask|image*)

Description

Sets the pattern for filled areas.

Arguments

name|mask|image The quoted color name, matrix of values between 0 and 1, or image path.

Get Color Theme Detail(*name*)

Description

Returns a script for the specified color theme.

Example

The following example returns the script for the JMP default color theme:

```
Get Color Theme Details( "JMP Default" );
  {"JMP Default", 9221, {{213, 72, 87}, {57, 177, 67}, {64, 111, 223}...}}
```

Get Color Theme Names(<*kind*>)

Description

Returns a list of all color theme names or color themes of the specified kind. The kinds include "Continuous", "Categorical", "Sequential", "Diverging", "Qualitative", or "Chromatic".

Example

The following example returns all color themes:

```
Get Color Theme Names();
  {"Green to Black to Red", "Green to White to Red", "White to Black"...}
```

The following example returns the diverging color themes:

```
Get Color Theme Names( "diverging" );
  {"Green to Black to Red", "Green to White to Red", "Blue to Gray to Red"...}
```

Gradient Function(*zExpr*, *xName*, *yName*, [*zLow*, *zHigh*], zColor([*colorLow*, *colorHigh*]), < <<xGrid(*min*, *max*, *incr*)>, < <<yGrid(*min*, *max*, *incr*)> < <<Transparency(*alpha*|*vector*))

Description

Fills a set of rectangles on a grid according to a color determined by the expression value as it crosses a range corresponding to a range of colors.

Required Arguments

zExpr Any expression (for example, Sine(y)+Cosine(x)).

xName, *yName* The values to use in the expression.

zLow, *zHigh* A matrix of low and high values (2 to 10).

zColor([colorLow, colorHigh) The two colors that are blended together (4 is green, 6 is orange).

Optional Messages

<<*xGrid*, <<*yGrid* Specifies a box, beyond which the gradients are not drawn.

<<*zColor* The color in which to draw the contour lines. The argument can be either a scalar or a matrix but must evaluate to numeric.

<<Transparency(*number*|*matrix*) The transparency level of the fill. A vector of numbers between 0 and 1 are sequenced through and cycled for the z contours.

Example

```
Gradient Function(Log(a * a + b * b),
a, b, [2 10],
zColor([4, 6]));
```

H Line(*x1*, *x2*, *y*)

H Line(*y*)

Description

Draws a horizontal line at *y* across the graph. If you specify start and end points on the *x*-axis (*x1* and *x2*), the line is drawn horizontally at *y* from *x1* to *x2*. You can also draw multiple lines by using a matrix of values in the *y* argument.

H Size()

Description

Returns the horizontal size of the graphics frame in pixels.

Handle(*xPos*, *yPos*, *dragScript*, *mouseUpScript*)

Description

Places draggable marker at coordinates given by *xPos*, *yPos*. The first script is executed at drag and the second at mouseup.

Heat Color(*x*, < <<*theme*>)

Heat Color(*x*)

Description

Returns the JMP color that corresponds to *n* in the color "*theme*".

Returns

An integer that is a JMP color.

Required Argument

x A value between 0 and 1.

Optional Message

theme A quoted color theme that is supported by Cell Plot. The default value is "Blue to Gray to Red".

HLS Color(*h*, *l*, *s*)

HLS Color({*h*, *l*, *s*})

Description

Converts hue, lightness, and saturation values into a JMP color number.

Returns

An integer that is a JMP color number.

Arguments

Hue, lightness, and saturation, or a list containing the three HLS values. All values should be between 0 and 1.

In Path(*x*, *y*, (*pathMatrix*|*pathText*)

Description

Determines it the point described by *x* and *y* falls in *path*.

Returns

True (1) if the point (x, y) is in the given path, False(0) otherwise.

Arguments

x and *y* The coordinates of a point.

pathMatrix|*pathText* Either a matrix or a quoted string describing a path.

`In Polygon(x, y, xMatrix, yMatrix)`

`In Polygon(x, y, xyPolygon)`

Description

Returns 1 or 0, indicating whether the point (*x*, *y*) is inside the polygon that is defined by the *xMatrix* and *yMatrix* vector arguments.

xMatrix, *yMatrix* can also be combined into a two-column matrix (*xyPolygon*), allowing you to use three arguments instead of four. Also, *x* and *y* can be conformable vectors, and then a vector of 0s and 1s are returned based on whether each (*x*, *y*) pair is inside the polygon.

`Level Color(i)`

`Level Color(i, n)`

`Level Color(i, n, <theme>)`

`Level Color(i, <theme>)`

Description

Assigns a JMP color to categorical data in a graphic.

Returns

An integer that is a JMP color.

Required Arguments

i An integer that is greater than or equal to 1 and less than or equal to the number of categories specified by n.

n The number of categories.

Optional Argument

theme A quoted color theme from the Value Color list of the Column Properties window. If not specified, the JMP Default color theme is applied.

Notes

When the second argument is a quoted character string and not *n*, then the second argument determines the color theme.

`Line({x1, y1}, {x2, y2}, ...), <<ValueSpace(Boolean)`

`Line([x1, x2, ...], [y1, y2, ...]), <<ValueSpace(Boolean)`

Description

Draws a line between points.

Arguments

`{x1, y1}, {x2, y2}|[x1, x2, ...], [y1, y2, ...]` Can be any number of lists of two points, separated by commas or a matrix of *x* values and a matrix of *y* values.

<<ValueSpace(*Boolean*) Draws lines that follow the projection when the line represents a movement of the underlying data, such as a bubble trail in a bubble plot. The Boolean value can be a constant or an expression.

Line Style(*name*|*number*)

Description

Sets the line style used to draw the graph.

Argument

n Can be either a quoted style name or the style's number:

- 0 or "Solid"
- 1 or "Dotted"
- 2 or "Dashed"
- 3 or "DashDot"
- 4 or "DashDotDot"

Marker(<*rowState*>, {*x1*, *y1*}, {*x2*, *y2*}, ...)

Marker(<*rowState*>, [*x1*, *x2*, ...], [*y1*, *y2*, ...})

Description

Draws one or more markers at the points described either by lists or matrices. The optional *rowState* argument sets the type of marker.

Marker Size(*n*)

Description

Sets the size used for markers.

Mousetrap(*dragScript*, *mouseUpscript*)

Description

Captures click coordinates to update graph properties. The first script is executed at drag and the second script at mouseup.

New Heat Image(*matrix*, <*colorTheme*>)

Description

Creates a heat map image based on a matrix and color theme or gradient.

Arguments

matrix A matrix of values. The function creates a heat map that has the same dimensions as the input matrix.

colorTheme A color theme or set of gradient properties that define the mapping of values to colors. The example below shows the use of a custom gradient.

Example

```
width = 256;
height = 256;
wstride = 16;
hstride = 16;
b = J( hstride, wstride, Random Normal() );
a = Transform Each( {z, {row, col}},
   J( height, width, 0 ),
   b[Floor( (row - 1) / hstride ) + 1,
   Floor( (col - 1) / wstride ) + 1]
);
E1 = New Heat Image(
   a,
   gradient(
      {Color Theme( "Blue To Gray To Orange" ),
      Scale Type( "Standard Deviation" ),
      Show Missing Color( "On" ),
      Reverse Gradient( 1 )}
   )
);
New Window( "test", Lineup Box( E1 ) );
```

Normal Contour(*prob*, *meanMatrix*, *stdMatrix*, *corrMatrix*, <*colorsMatrix*>, <*fill=x*>)

Description

Draws normal probability contours for *k* populations and two variables.

Required Arguments

prob A scalar or matrix of probabilities.

meanMatrix A matrix of means of size *k* by 2.

stdMatrix A matrix of standard deviations of size *k* by 2.

corrMatrix A matrix of correlations of size *k* by 1.

Optional Arguments

colorsMatrix Specifies the color(s) for the k contour(s). The colors must be specified as JSL colors (either JSL color integer values or return values of JSL Color functions such as RGB Color or HLS Color).

fill=x Specifies the amount of transparency for the contour fill color.

Oval(*x1*, *y1*, *x2*, *y2*, <*Fill(Boolean)*>)

Oval({*x1*, *y1*}, {*x2*, *y2*}, <*Fill(Boolean)*>)

Description

Draws an oval inside the rectangle whose diagonal has the coordinates (*x1, y1*) and (*x2, y2*). *Fill* is Boolean. If *Fill* is 0, the oval is empty. If *Fill* is nonzero, the oval is filled with the current fill color. The default value for *Fill* is 0.

Path((*pathMatrix*|*pathText*), <*Fill(Boolean)*>)

Description

Draws a stroke along the given path. If a fill is specified, the interior of the path is filled with the current fill color.

Required Arguments

pathMatrix An Nx3 matrix. If you don't specify the path matrix, specify the path text.

pathText A quoted string that contains SVG code.

Optional Arguments

Fill(Boolean) Specifies whether a line is drawn (0) or the path is filled (1). The default value is 0.

Notes

A path matrix has three columns, for x and y, and a flag. The flag value for each point can be 0 for control, 1 for move, 2 for line segment, 3 for cubic Bézier segment, and any negative value to close the path.

Path To Char(*path*)

Description

Converts a path specification from a matrix to a quoted string.

Returns

A quoted string.

Argument

path An Nx3 path matrix.

Notes

A path matrix has three columns, for x and y, and a flag. The flag value for each point can be 0 for control, 1 for move, 2 for line segment, 3 for cubic Bézier segment, and any negative value to close the path.

Pen Color(*n*)

Description

Sets the color used for the pen.

`Pen Size(n)`

Description

Sets the thickness of the pen in pixels.

`Pick Color(<window title>, <name|index|RGBlist>)`

Description

Creates a color picker, which enables the user to select a color to apply to graphs. The operating system color picker lets users select a predefined color or create their own color. You can also specify a default color in your script. If you omit the default color, Black is selected.

Returns

The color that the user selected in the operating system's color picker.

Arguments

window title The quoted title of the color picker window.

name The name of the default JMP color.

index The number of the default JMP color.

RGBlist The RGB values of the default color.

`Pie(left, top, right, bottom, startAngle, endAngle)`

Description

Draws a filled pie slice. The two points describe a rectangle, within which is a virtual oval. Only the slice described by the start and end angles is drawn.

`Pixel Line To(h, v)`

Description

Draws a one-pixel-wide line from the current pixel location to the location given in pixel coordinates. Set the current pixel location using the `Pixel Origin()` and `Pixel Move To()` functions.

`Pixel Move To(h, v)`

Description

Moves the current pixel location to a new location given in pixel coordinates.

`Pixel Origin(x, y)`

Description

Sets the origin, in graph coordinates, for subsequent `Pixel Line To()` or `Pixel Move To()` functions.

`Polygon({x1, y1}, {x2, y2}, ...)`

`Polygon(xMatrix, yMatrix)`

Description

Draws a filled polygon defined by the listed points.

`Polygon Area({x1, y1}, {x2, y2}, ...)`

`Polygon Area(xMatrix, yMatrix)`

Description

Calculates the area of the specified polygon.

Examples

```
area = Polygon Area( {0, 0}, {0, 10}, {10, 10}, {10, 0} );
area = Polygon Area( [10 20 30], [10 30 20] );
```

`Polygon Centroid({x1, y1}, {x2, y2}, ...)`

`Polygon Centroid(xMatrix, yMatrix)`

Description

Calculates the centroid of the specified polygon.

Examples

```
{cx, cy} = Polygon Centroid( {0, 0}, {0, 10}, {10, 10}, {10, 0} );
centroid = Polygon Centroid( [10 20 30], [10 30 20] );
```

`Pixel Path(h, v, (path matrix|path text), <fill=Boolean)>, <scale=n>, <orient={n, n})`

Description

Draws a stroke along the given pixel-based path if the fill is 0, or paints the interior of the path if the fill is not 0.

Required Arguments

h, *v* Specifies the horizontal and vertical position.

path matrix Contains three columns for *h*, *v*, and flags for each point in the path. The flag values are 0 for control, 1 for move, 2 for line segment, 3 for cubic Bézier segment, and are negative if the point also closes the path. Each flag (one for each point) can be 0, 1, 2 or 3, (or negative, if it closes the shape).

If you don't specify the path matrix, specify the path text.

path text Supports SVG syntax. The path is scaled (`scale`) and translated about its origin according to the optional parameters (`fill` and `orient`), with the orientation specified in the axis space.

Optional Arguments

`fill(`*Boolean*`)` 0 fills the shape. 1 draws an empty shape.

`orient` Rotates the object. For example, an orientation of `{Sqrt( 2 ), Sqrt( 2 )}` draws the shape on a 45-degree angle.

`scale` Sizes the object. For example, a scale of 1.0 draws the shape as it is defined; a scale of 2.0 draws it twice as big; and a scale of 0.5 draws it half as big.

`Pixel Text(<properties>, {h, v}, text, ...)`

Description

Moves to the {*h*, *v*} pixel position and draws text that the quoted *text* argument specifies.

Required Arguments

h The horizontal pixel position.

v The vertical pixel position.

`text` The quoted string to display at the start and end of the pixels.

Optional Arguments

`center justified` Center justifies the text.

`right justified` Right justifies the text.

`erased` Omits pixels from the edges of the rectangle of text. The text is drawn over the graph content in the background color, erasing the graph within the rectangle to make the text more readable.

`boxed` Displays a box around the text.

`counterclockwise` Rotates the text counterclockwise.

`clockwise` Rotates the text clockwise.

`Rect(`*left*`, `*top*`, `*right*`, `*bottom*`, <Fill(`*Boolean*`)>)`

`Rect({`*left*`, `*top*`}, {`*right*`, `*bottom*`}, <Fill(`*Boolean*`)>)`

Description

Draws a rectangle whose diagonal has the coordinates (*left*, *top*) and (*right*, *bottom*). *Fill* is Boolean. If *Fill* is 0, the rectangle is empty. If *Fill* is nonzero, the rectangle is filled with the current fill color. The default value for *Fill* is 0.

Remove Color Theme(name|{name, <*flags*>, {*color*, ...}, <{*position*, ...}>}>)

Description

Removes a custom color theme from the global list, either by name or by the full color theme object.

Required Arguments

name The quoted name of the color theme.

color The RGB values for the color.

Optional Arguments

flags A number that represents metadata such as whether the theme is continuous or categorical. Run `Get Color Theme Details("name")` on the color theme and use the flag that is returned.

position A number between 0 and 1. There is one number per color that indicates where on the gradient that color is, where 0 is the beginning and 1 is the end.

Example

```
Remove Color Theme( {"Yellow Blue", 0, {{255, 255, 0}, {0, 0, 255}}, {0.0,
  1.0}} );
```

RGB Color(*r*, *g*, b)

RGB Color({*r*, *g*, *b*})

Description

Converts red, green, and blue values into a JMP color number.

Returns

An integer that is a JMP color number.

Arguments

Red, green, and blue, or a list containing the three RGB values. All values should be between 0 and 1.

Text(<*properties*>, ({*x*, *y*}|{*left*, *bottom*, *right*, *top*}), *text*)

Description

Draws the quoted string *text* at the given point, either the x and y axes or the left, bottom, right, and top axes.

Properties can be any of several named arguments: `Center Justified`, `Right Justified`, `Erased`, `Boxed`, `Counterclockwise`, `Position`, and named arguments. The position, named arguments, and quoted strings can be added in any order. The position and named arguments apply to all the strings.

`Text Color(n)`

Description

Sets the color for text strings.

`Text Font(fontName, <size>, <bold italic underline strikeout>, <angle>)`

Description

Sets the font for text strings. Use without arguments to get the current font properties. Angle is in degrees clockwise.

Quote the font properties. To apply multiple properties, specify them together, as in `"bold italic"`.

`Text Size(n)`

Description

Sets the font size in points for text strings.

`Transparency(alpha)`

Description

Sets the transparency of the current drawing, with *alpha* between 0 and 1 where 0 is clear (no drawing) and 1 is completely opaque (the default).

Notes

Not all operating systems support transparency.

`V Line(x, <y1, y2>)`

Description

Draws a vertical line at *x* across the graph. If you specify start and end points on the *y*-axis (*y1* and *y2*), the line is drawn vertically at *x* from *y1* to *y2*. You can also draw multiple lines by using a matrix of values in the *x* argument.

`V Size()`

Description

Returns the vertical size of the graphics frame in pixels

`X Function(zExpr, yName, <Min(min), Max(max), Fill(Boolean), Inc(upper bound))`

Description

Draws a plot of the function as the *yName* is varied over the *y*-axis of the graph.

X Origin()

Description

Returns the *x*-value for the left edge of the graphics frame.

X Range()

Description

Returns the distance from the left to right edge of the display box. For example, `X Origin() + X Range()` is the right edge.

X Scale(*xMin*, *xMax*)

Description

Sets the range for the horizontal scale. The default value for *xMin* is 0. The default value for *xMax* is 100.

XY Function(*x*(*t*), *y*(*t*), *t*, Min(*min*), Max(*max*), (Inc(*bound*)|Steps(*min*))

Description

Combines an expression of *x(t)* and *y(t)* to draw an x-y curve for the specified range of parameter *t*. Each time *t* it is varied between Min and Max, the *x* and *y* expressions are evaluated using the current value of *t*.

Note: Either *Inc()* or *Steps()* is needed if the default granularity misses details.

Y Function(*yExpr*, *xName*, <Min(*min*), Max(*max*), Fill(*Boolean*), Inc(*bound*)>)

Description

Draws a plot of the function as the symbol is varied over the *x*-axis of the graph.

Y Origin()

Description

Returns the *y*-value for the bottom edge of the graphics frame.

Y Range()

Description

Returns the distance from the bottom to top edges of a display box. For example, `Y Origin() + Y Range()` is the top edge.

Y Scale(*yMin*, *yMax*)

Description

Sets the range for the vertical scale. If you do not specify a scale, it defaults to 0, 100.

HTTP Functions

Decode 64 Blob(*string*)

Description

Decodes the quoted string using Base-64 encoding.

Encode 64 Blob(*string*)

Description

Encodes the quoted string using Base-64 encoding.

List Functions

As List(*matrix*)

Description

Converts a matrix into a list. Multi-column matrices are converted to a list of row lists.

Returns

A list.

Argument

matrix Any matrix.

Concat Items({*string1*, *string2*, ...}, <*delimiter*>})

Description

Converts a list of quoted string expressions into one string, with each item separated by a delimiter. The delimiter is a blank if unspecified.

Returns

The concatenated quoted string.

Arguments

string Any quoted string.

delimiter An optional quoted string that is placed between each item. The delimiter can be more than one character long.

Example

```
str1 = "one";
str2 = "two";
str3 = "three";

comb = Concat Items({str1, str2, str3});
   "one two three"
comb = Concat Items({str1, str2, str3}, " : ");
   "one : two : three"
del = ",";
comb = Concat Items({str1, str2, str3}, del);
   "one,two,three"
```

`Eval List({`*list*`})`

Description

Evaluates expressions inside *list*.

Returns

A list of the evaluated expressions.

Arguments

list A list of valid JSL expressions.

`Insert(`*source*`, `*item*`, <`*position*`>)`

`Insert(`*source*`, `*key*`, `*value*`)`

Description

Inserts a new *item* into the *source* at the given *position*. If *position* is not given, *item* is added to the end.

For an associative array: Adds the *key* into the *source* associative array and assigns *value* to it. If the *key* exists in *source* already, its value is overwritten with the new *value*.

Required Arguments

source A quoted string, list, vector, expression, or associative array.

item|*key* Any value to be placed within *source*. For an associative array, *key* might or might not be present in *source*.

value A value to assign to the *key*.

Optional Arguments

position Optional numeric value that specifies the position in *source* to place the *item* into.

`Insert Into(source, item, <position>)`

`Insert Into(source, key, value)`

Description

Inserts a new item into the *source* at the given position in place. The *source* must be an L-value.

Arguments

source A variable that contains a quoted string, list, vector, display box, expression, or associative array.

item|key Any value to be placed within source. For an associative array, *key* might be present in *source*.

position Optional numeric value that specifies the position in *source* to place the *item* into.

value A value to assign to the *key*.

`Is List(x)`

Description

Returns 1 if the evaluated argument is a list, or 0 otherwise.

`Items(string, <delimiter>, <Include Boundary Delimiters(Boolean)>)`

Description

Returns a list of (possibly empty) quoted sub-strings separated by exactly one of any of the characters specified in the *delimiter* argument.

Arguments

`string` The quoted string that is evaluated.

`Delimiter` (Optional) The character used as a boundary. If *delimiter* is absent, an ASCII space is used. If *delimiter* is the empty quoted string, each character is treated as a separate word.

`Include Boundary Delimiters(Boolean)` (Optional) Returns the empty strings between the boundary and the delimiter.

Example

```
Items( "http://www.jmp.com", ":/." );
  {"http", "", "", "www", "jmp", "com"}
Items(",toy,", ",");
  {"toy"}
Items(",toy,", ",", Include Boundary Delimiters( 1 ));
  {"", "toy", ""}
  /* There is no text between the boundary (the beginning of the quoted string)
     and the comma delimiter, so you get an empty string. The same priniciple
     applies to the delimiter at the end of the string. */
```

```
List(a, b, c, ...)
{a, b, c, ...}
```

Description

Constructs a list from a set of items.

```
N Items(source)
```

Description

Determines the number of elements in the *source* specified.

Returns

For a list or display box, the number of items in the list or display box is returned. For an associative array, the number of keys is returned. For a matrix, the number of elements in the matrix is returned. For a namespace, the number of functions and variables in the namespace is returned. For a class object, the number of methods, functions, and variables is returned.

Arguments

`source` A list, associative array, matrix, display box, or namespace.

```
Remove(source, position, <n>)
Remove(source, {items})
Remove(source, key)
```

Description

Deletes the *n* item(s), starting from the indicated *position*. If *n* is omitted, the item at *position* is deleted. If *position* and *n* are omitted, the item at the end is removed. For an associative array: Deletes the *key* and its value.

Returns

A copy of the *source* with the items deleted.

Arguments

`source` A quoted string, list, vector, expression, or associative array.

`position or key` An integer (or list of integers) that points to a specific item (or items) in the list or expression.

`n` (Optional) An integer that specifies how many items to remove.

`Remove From(source, position, <n>)`

`Remove From(source, key)`

Description

Deletes the *n* item(s) in place, starting from the indicated *position*. If *n* is omitted, the item at *position* is deleted. If *position* and *n* are omitted, the item at the end is removed. For an associative array: Deletes the *key* and its value. The *source* must be an L-value.

Returns

The original *source* with the items deleted.

Arguments

`source` A quoted string, list, vector, expression, display box, or associative array.

`position or key` An integer (or list of integers) that points to a specific item (or items) in the list or expression.

`n` (Optional) An integer that specifies how many items to remove.

`Reverse(source)`

Description

Reverse order of elements or terms in the *source*.

Argument

`source` A quoted string, list, matrix, or expression.

`Reverse Into(source)`

Description

Reverses the order of elements or terms in *source* in place.

Argument

`source` A quoted string, list, matrix, display box, or expression.

`Shift(source, <n>)`

Description

Shifts an item or *n* items from the front to the back of the *source*.

Arguments

`source` A quoted string, list, matrix, or expression.

`n` (Optional) An integer that specifies the number of items to shift. Positive values shift items from the beginning of the *source* to the end. Negative values shift items from the end of the *source* to the beginning. The default value is 1.

`Shift Into(source, <n>)`

Description

Shifts items in place.

Arguments

`source` A quoted string, list, matrix, display box, or expression.

`n` (Optional) An integer that specifies the number of items to shift. Positive values shift items from the beginning of the *`source`* to the end. Negative values shift items from the end of the *`source`* to the beginning. The default value is 1.

`Sort List({list}|expr)`

Description

Sort the elements or terms of *`list`* or *`expr`*.

`Sort List Into({list}|expr)`

Description

Sort the elements or terms of *`list`* or *`expr`* in place.

`Substitute(`*`string`*`, "substring", "replacementString", ...)`

`Substitute({list}, listItem, replacementItem, ...)`

`Substitute(Expr(sourceExpr), Expr(findExpr), Expr(replacementExpr), ...)`

Description

This is a search and replace function. It searches for a specific portion (second argument) of the source (first argument), and replaces it (third argument).

If a quoted string, finds all matches to *`substring`* in the source *`string`*, and replaces them with the *`replacementString`*.

If a list, finds all matches to *`listItem`* in the source *`list`*, and replaces them with the *`replacementItem`*.

If an expression, finds all matches to the *`findExpr`* in the *`sourceExpr`*, and replaces them with the *`replacementExpr`*. Note that all expressions must be enclosed within an `Expr()` function.

Arguments

`string, list, sourceExpr` A quoted string, list, matrix, or expression in which to perform the substitution.

`substring, listItem, findExpr` A quoted string, list item, or expression to be found in the source string, list, or expression.

`replacementString, replacementItem, replacementExpr` A quoted string, list item, or expression to replace the found string, list item, or expression.

`Substitute Into(`*`string`*`, substring, replacementString, ...)`

`Substitute Into(list, listItem, replacementItem, ...)`

`Substitute Into(Expr(sourceExpr), Expr(findExpr), Expr(replacementExpr), ...)`

Description

This is a search and replace function, identical to `Substitute()` except in place. It searches for a specific portion (second argument) of the source (first argument), and replaces it (third argument). The first argument must be an L-value.

If a quoted string, finds all matches to *`substring`* in the source *`string`*, and replaces them with the *`replacementString`*.

If a list, finds all matches to *`listItem`* in the source *`list`*, and replaces them with the *`replacementItem`*.

If an expression, finds all matches to the *`findExpr`* in the *`sourceExpr`*, and replaces them with the *`replacementExpr`*. Note that all expressions must be enclosed within an `Expr()` function.

Arguments

`string, list, sourceExpr` A quoted string, list, matrix, or expression in which to perform the substitution.

`substring, listItem, findExpr` A quoted string, list item, or expression to be found in the source string, list, or expression.

`replacementString, replacementItem, replacementExpr` A quoted string, list item, or expression to replace the found string, list item, or expression.

`Words(`*`string`*`, <delimiter>)`

Description

Extracts the words from the quoted *`string`* according to the delimiters given. The default delimiter is ASCII whitespace. If you include a second argument, any and all characters in that argument are considered delimiters. If *`delim`* is an empty string, each character is treated as a separate word.

Examples

```
Words( "the quick brown fox" );
   {"the","quick","brown","fox"}
Words( "Doe, Jane P.",", ." );
   {"Doe","Jane","P"}
```

MATLAB Integration Functions

JMP provides the following interfaces to access MATLAB. The basic execution model is to first initialize the MATLAB connection, perform the required MATLAB operations, and then terminate the MATLAB connection. In most cases, these functions return 0 if the MATLAB operation was successful or an error code if it was not. If the MATLAB operation is not successful, a message is written to the Log window. The single exception to this is `MATLAB Get( )`, which returns a value.

MATLAB JSL Function Interfaces

`MATLAB Connect( <named arguments> )`

Description

Initializes the MATLAB integration interfaces and returns an active MATLAB integration interface connection as a scriptable object.

Returns

MATLAB scriptable object.

Named Arguments

`Echo(Boolean)` Sends the MATLAB source lines to the JMP log. The default value is true.

`MATLAB Control( <named arguments> )`

Description

Sends control operations to signal MATLAB with external events such as source line echoing.

Returns

None.

Arguments

None.

Named Arguments

`Echo(Boolean)` Global. Echo MATLAB source lines to the JMP log.

`Visible(Boolean)` Global. Determine whether to show or hide the active MATLAB workspace.

MATLAB Execute({ list of inputs }, { list of outputs }, mCode, <named arguments>)

Description

Submits the MATLAB code to the active global MATLAB connection given a list of inputs. Upon completion, retrieves a list of outputs.

Returns

0 if successful, otherwise nonzero.

Arguments

`{ list of inputs }` Positional, name list. List of JMP variable names to send to MATLAB as inputs.

`{ list of outputs }` Positional, name list. List of JMP variable names to retrieve from MATLAB as outputs.

`mCode` Positional, quoted string. The MATLAB code to submit.

Named Arguments

`Expand(Boolean)` Perform an Eval Insert on the MATLAB code prior to submission.

`Echo(Boolean)` Echo MATLAB source lines to the JMP log. Default is true.

Example

The following example sends the JMP variables x and y to MATLAB, executes the MATLAB statement *z = x * y*, and then gets the MATLAB variable z and returns it to JMP.

```
MATLAB Init();
x = [1 2 3];
y = [4 5 6];
MATLAB Execute( {x, y}, {z}, "z = x * y;" );
Show( z );
```

MATLAB Get(name)

Description

Gets named variable from MATLAB to JMP.

Returns

Value of named variable.

Arguments

`name` Positional. The name of a JMP variable to be sent to MATLAB.

Example

Suppose that a matrix named `qbx` and a structure named `df` are present in your MATLAB connection.

```
// get the MATLAB variable qbx and placed it into a JMP variable qbx
qbx = MATLAB Get( qbx );
```

```
/* get the MATLAB variable df and placed it into a JMP data table
referenced by df */
df = MATLAB Get( df );
```

Table 2.1 shows what JMP data types can be exchanged with MATLAB using the MATLAB Get( ) function. Getting lists from MATLAB recursively examines each element of the list and sends each base MATLAB data type. Nested lists are supported.

Table 2.1 JMP and MATLAB Equivalent Data Types for MATLAB Get( )

MATLAB Data Type	JMP Data Type
Double	Numeric
Logical	Numeric (0 | 1)
String	String
Integer	Numeric
Date/Time	Numeric
Structure	Data Table
Matrix	Numeric Matrix
Numeric Vector	Numeric Matrix
String Vector	List of Strings
Graph	Picture object

MATLAB Get Graphics(format)

Description

Get the last graphic object written to the MATLAB graph display window in a specific graphic format. The graphic object can be returned in several graphic formats.

Returns

JMP Picture object.

Argument

format Positional. The format the MATLAB graph display window contents are to be converted to. Valid formats are png, bmp, jpeg, jpg, tiff, tif, and gif.

MATLAB Get Version

Description

Returns the version number of MATLAB being used with the JMP MATLAB interfaces.

MATLAB Init(<named arguments>)

Description

Initializes the MATLAB integration interfaces.

Returns

Return code.

Named Arguments

`Echo(Boolean)` Sends MATLAB source lines to the JMP log. This option is global. The default value is true.

MATLAB Is Connected()

Description

Determines whether a MATLAB connection is active.

Returns

1 if connected, otherwise 0.

MATLAB JMP Name To MATLAB Name(name)

Description

Maps a JMP variable name to its corresponding MATLAB variable name using MATLAB variable name naming rules.

Returns

A quoted string, mapped MATLAB variable name.

Argument

`name` Positional. The name of a JMP variable to be sent to MATLAB.

MATLAB Send(name, <named arguments>)

Description

Sends the named variable from JMP to MATLAB.

Returns

0 if successful, otherwise nonzero.

Arguments

`name` Positional. The name of a JMP variable to be sent to MATLAB.

Named Arguments

The following optional arguments apply to data tables only:

`Selected(Boolean)` Send selected rows from the referenced data table to MATLAB.

`Excluded(Boolean)` Send only excluded rows from the referenced data table to MATLAB.

`Labeled(Boolean)` Send only labeled rows from the referenced data table to MATLAB.

`Hidden(Boolean)` Send only hidden rows from the referenced data table to MATLAB.

`Colored(Boolean)` Send only colored rows from the referenced data table to MATLAB.

`Markered(Boolean)` Send only markered rows from the referenced data table to MATLAB.

`Row States(Boolean, <named arguments>)` Send row states from referenced data table to MATLAB by adding an additional data column named "RowState". Create multiple selections by adding together individual settings. The row state consists of individual settings with the following values:

- Selected = 1
- Excluded = 2
- Labeled = 4
- Hidden = 8
- Colored = 16
- Markered = 32

The following optional, named Row States arguments are supported:

`Colors(Boolean)` Send row colors. Adds additional data column named "RowStateColor".

`Markers(Boolean)` Send row markers. Adds additional data column named "RowStateMarker".

Example

```
// create a matrix, assign it to X, and send the matrix to MATLAB
X = [1 2 3];
ml = MATLAB Send( X );

/* open a data table, assign a reference to it to dt, and send the
data table along with its current row states to MATLAB */
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
ml = MATLAB Send( dt, Row States( 1 ) );
```

Table 2.2 shows what JMP data types can be exchanged with MATLAB using the `MATLAB Send( )` function. Sending lists to MATLAB recursively examines each element of the list and sends each base JMP data type. Nested lists are supported.

Table 2.2 JMP and MATLAB Equivalent Data Types for `MATLAB Send( )`

MATLAB Data Type	JMP Data Type
Double	Numeric
String	String
Double Matrix	Matrix
Structure	Data Table

Example

```
MATLAB Init( );
X = 1;
MATLAB Send( X );
S = "Report Title";
MATLAB Send( S );
M = [1 2 3, 4 5 6, 7 8 9];
MATLAB Send( M );
MATLAB Submit( "
X
S
M
" );
MATLAB Term( );
```

MATLAB Send File(filename, <MATLAB Name(name)>)

Description

Sends a data file to MATLAB.

Arguments

`filename` Specifies a quoted string that identifies the pathname to the file to be sent to MATLAB.

`MATLAB Name` Enables you to change the name of the file sent to MATLAB.

MATLAB Submit File(pathname, <named arguments>)

Description

Submits statements to MATLAB using a file pointed to by pathname.

Returns

0 if successful, otherwise nonzero.

Arguments

`Pathname` Positional, quoted string. Pathname to file containing MATLAB source lines to be executed.

Named Arguments

`Expand(Boolean)` Perform an Eval Insert on the MATLAB code prior to submission.

`Echo(Boolean)` Echo MATLAB source lines to the JMP log. Default is true.

`MATLAB Submit( mCode, <named arguments> )`

Description

Submits the MATLAB code to the active global MATLAB connection.

Returns

0 if successful, otherwise nonzero.

Arguments

`mCode` Positional, quoted string. The MATLAB code to submit.

Named Arguments

`Expand(Boolean)` Perform an Eval Insert on the MATLAB code prior to submission.

`Echo(Boolean)` Echo MATLAB source lines to the JMP log. Default is true.

Example

The following example creates two vectors of random points and plots them as x and y variables.

```
MATLAB Init();
mc = MATLAB Submit("/[
   x = rand(5);
   fprintf('%f/n', x);
   y = rand(5);
   fprintf('%f/n', x);
   z = plot(x, y);
]/" );
```

`MATLAB Term();`

Description

Terminates the currently active MATLAB integration interface.

Returns

1 if an active MATLAB connection exists, otherwise returns 0.

Arguments

None.

Matrix Functions

All(A, ...)

Returns

1 if all matrix arguments are nonzero; 0 otherwise.

Any(A, ...)

Returns

1 if one or more elements of one or more matrices are nonzero; 0 otherwise.

B Spline Coef(x, Internal Knot Grid, <degree=3>, <Knot End Points=min(x)||max(x)>)

Description

Finds the matrix of B-spline coefficients for the data in the *x* argument.

Returns

The matrix of B-spline basis coefficients. This matrix can be used as a design matrix in a linear model. The first column of the matrix contains an intercept term.

Arguments

`x` A row or column vector that contains the data.

`Internal Knot Grid` Either a single number that designates the number of desired knot points based on percentiles of *x* or a vector of values that designate the internal knot points. The number of internal knots must be greater than zero and less than or equal to the number of unique elements in *x* minus two.

`degree` A number that indicates the degree of the B-splines. Defaults to 3.

`Knot End Points` A 2x1 matrix that designates the lower and upper knot points. If this argument is not specified, the default lower and upper knot points are the minimum and maximum values of *x*, respectively.

Notes

This function is used in column formulas created by the Functional Data Explorer platform.

CDF(Y)

Description

Returns values of the empirical cumulative probability distribution function for *Y*, which can be a vector or a list. Cumulative probability is the proportion of data values less than or equal to the value of *QuantVec*.

Syntax

```
{QuantVec, CumProbVec} = CDF(YVec)
```

`Chol Update(L, V, C)`

Description

If L is the Cholesky root of an *n*x*n* matrix A, then after calling `cholUpdate` L is replaced with the Cholesky root of A+V*C*V' where C is an *m*x*m* symmetric matrix and V is an n*m matrix.

`Cholesky(A)`

Description

Finds the lower Cholesky root (L) of a positive semi-definitive matrix, L*L' = A.

Returns

L (the Cholesky root).

Arguments

`A` a symmetric matrix.

`Correlation(matrix, < <<"Pairwise">, < <<"Shrink">, < <<Freq(vector)>, < <<Weight(vector)>)`

Description

Calculates the correlation matrix of the data in the *matrix* argument.

Returns

The correlation matrix for the specified matrix.

Argument

`matrix` A matrix that contains the data. If the data has *m* rows and *n* columns, the result is an m-by-m matrix.

`"Pairwise"` Uses the pairwise method for missing values rather than the row-wise method.

`"Shrink"` Performs the Schafer-Strimmer shrinkage estimate.

`<<Freq(vector)` A vector that specifies frequencies for the rows of the matrix argument.

`<<Weight(vector)` A vector that specifies weights for the rows of the matrix argument.

Notes

By default, rows are discarded if they contain missing values. If the "Pairwise" option is specified, all pairs of nonmissing values are used in the correlation matrix calculation.

This function uses multithreading if available, so it is recommended for large problems with many rows.

When a column is constant, the correlations for it are 0, and the diagonal element is also 0.

`Covariance(matrix, < <<"Pairwise">, < <<"Shrink">, < <<Freq(vector)>, < <<Weight(vector)>)`

Description

Calculates the covariance matrix of the data in the *matrix* argument.

Returns

The covariance matrix for the specified matrix.

Argument

`matrix` A matrix that contains the data. If the data has *m* rows and *n* columns, the result is an m-by-n matrix.

`"Pairwise"` Uses the pairwise method for missing values rather than the row-wise method.

`"Shrink"` Performs the Schafer-Strimmer shrinkage estimate.

`<<Freq(vector)` A vector that specifies frequencies for the rows of the matrix argument.

`<<Weight(vector)` A vector that specifies weights for the rows of the matrix argument.

Notes

By default, rows are discarded if they contain missing values. If the "Pairwise" option is specified, all pairs of nonmissing values are used in the covariance matrix calculation.

This function uses multithreading if available, so it is recommended for large problems with many rows.

`Design(vector, < levelsList | <<levels, <<ElseMissing >)`

Description

Creates a design matrix that contains a column of 1s and 0s for each unique value of a *vector* of values.

Returns

A design matrix with a column of 1s and 0s for each unique value of the argument or a list that contains the design matrix and a list of levels.

Argument

`vector` A vector.

`levelsList` An optional list or matrix argument that specifies the levels in the returned matrix.

`<<levels` An optional argument that changes the return value to a list that contains the design matrix and a list of the levels.

`<<ElseMissing` An optional argument that changes the handling of values in the *vector* argument that do not appear in the *levelsList* argument. If this argument is specified, missing values are placed in the design matrix. Otherwise, 0s are placed in the design matrix.

Notes

Missing values in the *levelsList* argument are not ignored. For example:

```
Show( Design ( ., [. 0 1] ),
Design( 0, [. 0 1] ),
Design( 1, [. 0 1] ),
Design( [0 0 1 . 1], [. 0 1] ),
Design( {0, 0, 1, ., 1}, [. 0 1] ) );
   Design(., [. 0 1]) = [1 0 0];
   Design(0, [. 0 1]) = [0 1 0];
   Design(1, [. 0 1]) = [0 0 1];
   Design([0 0 1 . 1], [. 0 1]) =
   [  0 1 0,
      0 1 0,
      0 0 1,
      1 0 0,
      0 0 1];
   Design({0, 0, 1, ., 1}, [. 0 1]) =
   [  0 1 0,
      0 1 0,
      0 0 1,
      1 0 0,
      0 0 1];
```

Design Last(vector, < levelsList, <<ElseMissing >)

Description

Creates a design matrix that contains a column of 1s and 0s for all but the last of the unique values of the argument. The last level is coded as a row of 0s.

Returns

A full-rank design matrix or a list that contains the design matrix and a list of levels.

Arguments

`vector` A vector.

`levelsList` An optional list or matrix argument that specifies the levels in the returned matrix. If this argument is specified, the last level in this list or matrix is treated as the last level in the design matrix. Otherwise, the last level is defined as the largest value in the *vector* argument.

`<<ElseMissing` An optional argument that changes the handling of values in the *vector* argument that do not appear in the *levelsList* argument. If this argument is specified, missing values are placed in the design matrix. Otherwise, 0s are placed in the design matrix.

`Design Nom(vector, < levelsList | <<levels, <<ElseMissing >)`

`DesignF(vector, < levelsList | <<levels, <<ElseMissing >)`

Description

Creates a design matrix that contains a column of 1s and 0s for all but the last of the unique values of the argument. The last level is coded as a row of -1s.

Returns

A full-rank design matrix or a list that contains the design matrix and a list of levels.

Argument

`vector` A vector.

`levelsList` An optional list or matrix argument that specifies the levels in the returned matrix. If this argument is specified, the last level in this list or matrix is treated as the last level in the design matrix. Otherwise, the last level is defined as the largest value in the *`vector`* argument.

`<<levels` An optional argument that changes the return value to a list that contains the design matrix and a list of levels.

`<<ElseMissing` An optional argument that changes the handling of values in the *`vector`* argument that do not appear in the *`levelsList`* argument. If this argument is specified, missing values are placed in the design matrix. Otherwise, 0s are placed in the design matrix.

Notes

Missing values in the *`levelsList`* argument are not ignored. For example:

```
Show( Design Nom( ., [. 0 1] ),
Design Nom( 0, [. 0 1] ),
Design Nom( 1, [. 0 1] ),
Design Nom( [0 0 1 . 1], [. 0 1] ),
Design Nom( {0, 0, 1, ., 1}, [. 0 1] ) );
   Design Nom(., [. 0 1]) = [1 0];
   Design Nom(0, [. 0 1]) = [0 1];
   Design Nom(1, [. 0 1]) = [-1 -1];
   Design Nom([0 0 1 . 1], [. 0 1]) = [0 1, 0 1, -1 -1, 1 0, -1 -1];
   Design Nom({0, 0, 1, ., 1}, [. 0 1]) = [0 1, 0 1, -1 -1, 1 0, -1 -1];
```

`Design Ord(vector, < levelsList | <<levels, <<ElseMissing >)`

Description

Creates a design matrix that contains a column for all but the last of the unique values of the argument. The first level is coded as a row of 0s. Each subsequent (nth) level in the *`levelsList`* argument is coded as a row of (n-1) 1s and the rest 0s.

Returns

A full-rank design matrix or a list that contains the design matrix and a list of levels.

Argument

`vector` A vector.

`levelsList` An optional list or matrix argument that specifies the levels in the returned matrix.

`<<levels` An optional argument that changes the return value to a list that contains the design matrix and a list of levels.

`<<ElseMissing` An optional argument that changes the handling of values in the *`vector`* argument that do not appear in the *`levelsList`* argument. If this argument is specified, missing values are placed in the design matrix. Otherwise, 0s are placed in the design matrix.

Notes

Missing values in the *`levelsList`* argument are not ignored. For example:

```
Show( Design Ord( ., [. 0 1] ),
Design Ord( 0, [. 0 1] ),
Design Ord( 1, [. 0 1] ),
Design Ord( [0 0 1 . 1], [. 0 1] ),
Design Ord( {0, 0, 1, ., 1}, [. 0 1] ) );
   Design Ord(., [. 0 1]) = [0 0];
   Design Ord(0, [. 0 1]) = [1 0];
   Design Ord(1, [. 0 1]) = [1 1];
   Design Ord([0 0 1 . 1], [. 0 1]) = [1 0, 1 0, 1 1, 0 0, 1 1];
   Design Ord({0, 0, 1, ., 1}, [. 0 1]) = [1 0, 1 0, 1 1, 0 0, 1 1];
```

Det(A)

Description

Determinant of a square matrix.

Returns

The determinant.

Argument

`A` A square matrix.

Diag(A, <B>)

Description

Creates a diagonal matrix from a square matrix or a vector. If two matrices are provided, concatenates the matrices diagonally.

Returns

The matrix.

Argument

`A` a matrix or a vector.

`Direct Product(A, B)`

Description

Direct (Kronecker) product of square matrices or scalars A[i,j]*B.

Returns

The product.

Arguments

`A, B` Square matrices or scalars.

`Distance(x1, x2, <scales>, <powers>)`

Description

Produces a matrix of distances between rows of x1 and rows of x2.

Returns

A matrix.

Arguments

`x1, x2` Two matrices.

`scales` Optional argument to customize the scaling of the matrix.

`powers` Optional argument to customize the powers of the matrix.

`E Div(A, B)`

`A:/B`

Returns

A matrix that contains the element-by-element division of two matrices.

Arguments

`A, B` Two matrices. Both matrix elements must have the same dimensions.

`E Max(A, B)`

Returns

A matrix that contains the element-by-element maximum values of two or more matrices or scalar arguments.

Arguments

`A, B` Two or more matrices or scalars. All matrix elements must have the same dimensions.

E Min(A, B)

Returns

A matrix that contains the element-by-element minimum values of two or more matrices or scalar arguments.

Arguments

A, B Two or more matrices or scalars. All matrix elements must have the same dimensions.

E Mult(A, B)

A:*B

Description

Element-by-element multiplication of two matrices.

Returns

A matrix that contains the element-by-element multiplication of two or more matrices or scalar arguments.

Arguments

A, B Two or more matrices or scalars. All matrix elements must have the same dimensions.

Eigen(A)

Description

Eigenvalue decomposition.

Returns

A list {M, E} such that E * Diag(M) * E = A'.

Argument

A A symmetric matrix.

Estimate Bartlett Factor Score(dataRow, ManMeans, LatMeans, S, A)

Description

Estimates factor scores from a structural equation model (SEM) using Bartlett's method.

Returns

A row vector of estimated factor scores based on the structural equation model.

Arguments

dataRow A row vector of data values.

ManMeans A vector of model-implied manifest variable means.

LatMeans A vector of model-implied latent variable means.

S The symmetric S matrix from a structural equation model.

A The rectangular A matrix from a structural equation model.

Estimate Factor Score(dataRow, Covariance, ManMeans, LatMeans)

Description

Estimates factor scores from a structural equation model (SEM).

Returns

A row vector of estimated factor scores based on the structural equation model.

Arguments

dataRow A row vector of data values.

Covariance A model-implied variance-covariance matrix.

ManMeans A vector of model-implied manifest variable means.

LatMeans A vector of model-implied latent variable means.

Note

This function is used in the Save Factor Scores option in an Structural Equation Model report.

Fourier Basis Coef(x, Number Pairs, <Period=max(x)-min(x)+1>)

Description

Finds the matrix of Fourier basis coefficients for the data in the *x* argument.

Returns

The matrix of Fourier basis coefficients. This can be used as a design matrix in a linear model. The first column of the matrix contains an intercept term. The remaining columns contain pairs of basis coefficients, where pair *i* is defined as the sin() and cos() of *i* * (2 * π / Period) * x.

Arguments

x A row or column vector that contains the data.

Number Pairs The number of sin() and cos() pairs for the Fourier basis.

Period The period for trigonometric functions that make up the Fourier basis.

Notes

This function is used in column formulas created by the Functional Data Explorer platform.

G Inverse(A)

Description

Generalized (Moore-Penrose) matrix inverse.

H Direct Product(A, B)

Description

Horizontal direct product of two square matrices of the same dimension or scalars.

Hough Line Transform(matrix, <NAngle(number)>, <NRadius(number)>)

Description

Takes a matrix of intensities and transforms it in a way that is useful for finding streaks in the matrix. Produces a matrix containing the Hough Line Transform with angles as columns and radiuses as rows.

Argument

`matrix` A matrix that can be derived from the intensities of an image, but is more likely from a semiconductor wafer that may have defects across in a streak due to planarization machines.

`NAngle(number)` Enter the number of the angle to obtain a different sized transform. The default is 180 degrees.

`NRadius(number)` Enter the number of the radius to obtain a different sized transform. The default is sqrt(NRow*nRow+nCol*nCol).

Identity(n)

Description

Creates an *n*-by-*n* identity matrix with ones on the diagonal and zeros elsewhere.

Returns

The matrix.

Argument

`n` An integer.

Index(i, j, <increment>)

i::j

Description

Creates a column matrix whose values range from *i* to *j*.

Returns

The matrix.

Arguments

`i, j` Integers that define the range: *i* is the beginning of the range, *j* is the end.

`increment` Optional argument to change the default increment, which is +1.

`Inv()`

See "Inverse(A)".

`Inv Update(A, X, 1|-1)`

Description

Efficiently update an $\mathbf{X}'\mathbf{X}$ matrix.

Arguments

`A` The matrix to be updated.

`X` One or more rows to be added to or deleted from the matrix *A*.

`1|-1` The third argument controls whether the row or rows defined in the second argument, *X*, are added to or deleted from the matrix *A*. `1` means to add the row or rows and `-1` means to delete the row or rows.

`Inverse(A)`

`Inv(A)`

Description

Returns the matrix inverse. The matrix must be square non-singular.

`Is Matrix(x)`

Description

Returns 1 if the evaluated argument is a matrix, or 0 otherwise.

`J(nrows, <ncols>, <value>)`

Description

Creates a matrix of identical values.

Returns

The matrix.

Arguments

`nrows` Number of rows in matrix. If `ncols` is not specified, `nrows` is also used as `ncols`.

`ncols` Number of columns in matrix.

`value` The value used to populate the matrix. If `value` is not specified, 1 is used.

`KDTable(matrix)`

Description

Returns a table to efficiently look up near neighbors.

Returns

A KDTable object.

Argument

`matrix` A matrix of k-dimensional points. The number of dimensions or points is not limited. Each column in the matrix represents a dimension to the data, and each row represents a data point.

Messages

`<<Distance between rows(row1, row2)` Returns the distance between two the two specified rows in the KDTable. The distance applies to removed and inserted rows as well.

`<<K nearest rows(stop, <position>)` Returns a matrix. Position is a point that is described as a row vector for the coordinate of a row, or as the number of a row. If *`position`* is not specified, returns the *n* nearest rows and distances to all rows. If *`position`* is specified, returns the *n* nearest rows and distances to either a point or a row. *`Stop`* is either *n* or {*n*, *limit*}. The limit parameter limits the number of rows that will be found. It can be specified one of two ways: a number, like 5, means return the 5 nearest rows. A list, like {5,10}, means return up to 5 nearest rows, stopping when the distance of 10 is exceeded. In the second case, the last row may have a distance greater than 10. Since the command continues until it finds the closest row beyond the stop radius, this point is also returned. This can be especially useful if there are no rows within the radius.

`<<Remove rows(number | vector)` Remove either the row specified by *`number`* or the rows specified by *`vector`*. Returns the number of rows that were removed. Rows that were already removed are ignored.

`<<Insert rows(number | vector)` Re-insert either the row specified by *`number`* or the rows specified by *`vector`*. Returns the number of rows that were inserted. Rows that were already inserted are ignored.

Notes

When rows are removed or inserted, the row indices do not change. You can remove and re-insert only rows that are in the KDTable object. If you need different rows, construct a new KDTable.

`Least Squares Solve(y, X, < <<noIntercept, <<weights(OptionalWeightVector), <<method("Sweep"|"GInv")>)`

Description

Computes least squares regression estimates for the assumed model y = X * beta + error.

Returns

A list that contains the matrix `Beta=Inverse(X'X)X'y` and the estimated variance matrix of Beta.

Optional Named Arguments

<<noIntercept Specifies a no-intercept model.

<<weights(optional weight vector) Specifies a vector of weights to perform weighted least squares.

<<method("Sweep" | "GInv") Specifies the method for solving the normal equations. The default Sweep method is more computationally efficient, but you can also specify the generalized inverse ("GInv") method, which is more numerically stable.

`Linear Regression(y, X, < <<noIntercept, <<printToLog, <<weight(OptionalWeightVector), <<freq(OptionalFreqVector)>)`

Description

Fits a linear regression for the assumed model y = X * beta + error.

Returns

A list that contains a vector of the estimates, a vector of the standard error, and a list of diagnostics. The list of diagnostics contains vectors of the *t* statistics and *p*-values for the estimates, as well as the R-Square and adjusted R-Square values for the regression fit.

Optional Named Arguments

<<noIntercept Excludes the intercept.

<<printToLog Prints a summary of the fit to the log.

<<weight(vector) Specifies a vector of weights to perform weighted least squares.

<<freq(vector) Specifies a vector of frequencies for each row of y and X.

Example

```
n = 10;
x = J( n, 1, Random Normal() );
y = 1 + x * 3 + J( n, 1, Random Normal() );
{Estimates, Std_Error, Diagnostics} = Linear Regression( y, x, <<printToLog );
As Table( y || x );
Bivariate( Y( :Col1 ), X( :Col2 ), Fit Line( 1 ) );
```

`Loc(A)`

`Loc(A, item)`

Description

Returns a matrix of subscript positions where *A* is nonzero and nonmissing. For the two-argument function, Loc returns a matrix of positions where *item* is found within *A*. If the first argument is a list, the second argument is required.

Argument

A a matrix or a list

item the item to be found within the matrix or list *A*

Loc Max(A)

Description

Returns the position of the maximum element in a matrix.

Returns

An integer that is the specified position.

Argument

A a matrix

Loc Min(A)

Description

Returns the position of the minimum element in a matrix.

Returns

An integer that is the specified position.

Argument

A a matrix

Loc NonMissing(matrix, ..., {list}, ...)

Description

Returns indices of nonmissing rows in matrices or lists. In lists, the function can also return indices of nonempty characters.

Returns

The new matrix or list.

Loc Sorted(A, B)

Description

Returns a column vector of subscript positions where the values of *A* have values less than or equal to the values in *B* based on a binary search. *A* must be a matrix sorted in ascending order without missing values.

Returns

The new matrix, which has the same dimensions as *B*. If a value in *B* is less than the first value in *A*, the returned subscript position for that value is 1.

Argument

A, B matrices

`Matrix({{x11, ..., x1m}, {x21, ..., 2m}, {...}, {xn1, ..., xnm}})`

`Matrix({x1, ..., xn})`

`Matrix(n, m)`

Description

Constructs an *n*-by-*m* matrix. The following specification methods are available:

- If you specify a list of *n* lists that each contain *m* row values, the matrix is formed by vertically concatenating the evaluated lists. The list items must evaluate to numeric values or row vectors, and the dimensions of the items must be conformable.
- If you specify a single list of *n* items, the return value is an *n*-by-1 column vector. The items of the evaluated list must evaluate to numeric values.
- If you specify two integer arguments, the return value is a matrix of zeros that contains *n* rows and *m* columns.

Examples

```
Matrix({{1, 2, 3}, {4, [5 6]}, {7, 8, 9}});
   [1 2 3, 4 5 6, 7 8 9]
Matrix({{[1 2 3], 4, 5, 6, 7, 8, 9}});
   [1 2 3 4 5 6 7 8 9]
Matrix({2,3+7});
   [2, 10]
Matrix(2,3);
   [0 0 0, 0 0 0]
```

`Matrix Mult(A, B)`

`C=A*B, ...`

Description

Matrix multiplication.

Arguments

`A, B, ...` Two or more matrices, which must be conformable (all matrices after the first one listed must have the same number of rows as the number of columns in the first matrix).

Notes

`Matrix Mult()` allows only two arguments, while using the * operator enables you to multiply several matrices.

`Matrix Rank(A)`

Description

Returns the rank of the matrix `A`.

Mode({list} or matrix)

Description

Selects the most frequent item from a numeric or character list or a numeric matrix. In the event of a tie, the lower value is selected. If multiple arguments are specified, a combination of numeric values and quoted character strings is acceptable.

Arguments

Specify either a list or a matrix.

Multivariate Normal Impute(yVec, meanYvec, symCovMat, colMin, colMax)

Description

Imputes missing values in yVec based on the mean and covariance.

Arguments

yVec The vector of responses.

meanYvec The vector of response means.

symCovMat A symmetric matrix containing the response covariances. If the covariance matrix is not specified, then JMP imputes with means.

colMin A vector of column minimums. Provides lower bounds for the imputations.

colMax A vector of column maximums. Provides upper bounds for the imputations.

NChooseK Matrix(n, k)

Description

Returns a matrix of n things taken k at a time (n select k).

N Col(x)

N Cols(x)

Description

Returns the number of columns in either a data table or a matrix.

Argument

x Can be a data table or a matrix.

Ortho(A, <Centered(0)>, <Scaled(1)>)

Description

Orthonormalizes the columns of matrix *A* using the Gram Schmidt method. Centered(0) makes the columns to sum to zero. Scaled(1) makes them unit length.

`Ortho Poly(vector, order)`

Description

Returns orthogonal polynomials for a *vector* of indices representing spacings up to the *order* given.

`P Spline Coef(x, Internal Knot Grid, <degree=3>)`

Description

Finds the matrix of penalized basis spline (P-spline) coefficients for the data in the *x* argument.

Returns

The matrix of P-spline basis coefficients, which is the truncated power basis of the specified *degree*. The truncated power basis of degree p with knots k_1 through k_K is defined as follows:

$$1, x, x^2, \ldots, x^p, (x-k_1)_+^p, \ldots, (x-k_K)_+^p$$

where $(x - k_1)_+$ is the positive part of $(x - k_1)$ and is set to zero for negative values of $(x - k_1)$.

Arguments

`x` A row or column vector that contains the data.

`Internal Knot Grid` Either a single number that designates the number of desired knot points based on percentiles of *x* or a vector of values that designate the internal knot points.

`degree` A number that indicates the degree of the P-splines. Defaults to 3.

Notes

This function is used in column formulas created by the Functional Data Explorer platform.

`Parallel Assign({thread_local_var = global_var, ...}, matrix[a, b] = expression using a and b)`

Description

Uses multiple threads to assign values to the matrix. Enables you to take advantage of multiple cores on a computer. The function has two arguments.

- The first argument is a list of assignment statements that copies global variables into each thread's local variable list.
- The second argument is an assignment expression with a left-hand-side that is a matrix with one or two prototype indexes and a right hand side that can be any JSL expression using those indexes and the local variables from the list (and in JMP global: variables).

Example

The following example provides read access to the global namespace.

```
a = 42;
x = [1 2 3, 4 5 6, 7 8 9];
Show( Parallel Assign( {}, x[i, j] = global:a ) );
Show( x );
```

```
Parallel Assign({}, x[i,j] = global:a) = 1;
x =
[  42 42 42,
   42 42 42,
   42 42 42];
```

Print Matrix(M, <named arguments>)

Description

Returns a quoted string that contains a well-formatted matrix. You can use the function, for example, to print the matrix to the log.

Argument

`M` A matrix.

Optional Named Arguments

`<<ignore locale(Boolean)` Set to false (0) to use the decimal separator for your locale. Set to true (1) to always use a period (`.`) as a separator. The default value is false (0).

`<<decimal digits(n)` An integer that specifies the number of digits after the decimal separator to print.

`<<style("style name")` Use one of three available styles: Parseable is a reformatted JSL matrix expression. Latex is formatted for LaTeX. If you specify Other, you must define the following three arguments.

`<<separate("character")` Define the separator for concatenated entries.

`<<line begin("character")` Define the beginning line character.

`<<line end("character")` Define the ending line character.

QR(A)

Description

Returns the QR decomposition of *A*. Typical usage is `{Q, R} = QR(A)`.

Random SVD(X, <nSingularValues=min(nRow, nCol)>, <nOver=10>, <nIter=2>)

Description

Computes the singular value decomposition (SVD) of matrix X using the randomized singular value decomposition. This decomposition can be faster than a traditional SVD, and it typically achieves an accurate approximation. It is particularly fast for large matrices when only a few singular values are desired. The method randomly projects the

data into a low-dimensional space, performing the SVD in the low-dimensional space, and then computing an estimate of the SVD for the original data. The dimension of the computational space is equal to the number of singular values plus the number of oversampled columns.

Returns

Returns a list {**U**, **M**, **V**} such that **U***diag(**M**)***V**′ is equal to **X**.

Arguments

`X` A matrix.

`nSingularValues` The number of singular values to return. By default, this value is the minimum of the number of rows and number of columns in the matrix X.

`nOver` The number of oversampled columns.

`nIter` The number of power iterations.

`Rank Index(vector)`

`Rank(vector)`

Description

Returns a vector of indices that, used as a subscript to the original *`vector`*, sorts the vector by rank. Excludes missing values. Lists of numbers or quoted strings are supported in addition to matrices.

`Ranking(vector)`

Description

Returns a vector of ranks of the values of *`vector`*, low to high as 1 to *`n`*, ties arbitrary. Lists of numbers or quoted strings are supported in addition to matrices.

`Ranking Tie(vector)`

Description

Returns a vector of ranks of the values of *`vector`*, but ranks for ties are averaged. Lists of numbers or quoted strings are supported in addition to matrices.

`Scoring Impute(rowWithMissing, VMat, colMeanVec, colStdDevVec)`

Description

Provides streaming functionality for the Automated Data Imputation (ADI) algorithm.

Returns

Returns the row vector with the missing values imputed using the standard least squares estimation.

Arguments

`rowwithMissing` A row vector that contains missing values.

`VMat` A loading matrix that is produced by the ADI algorithm.

`colMeanVec` A vector of the column means ignoring missing cells.

`colStdDevVec` a vector of the column standard deviations ignoring missing cells.

`Shape(A, nrow, <ncol>, < <<bycol>)`

Description

Reshapes the matrix *A* across rows to the specified dimensions. Each value from the matrix *A* is placed into the reshaped matrix. By default, the values are placed row-by-row.

Returns

The reshaped matrix.

Arguments

`A` A matrix.

`nrow` The number of rows that the new matrix should have.

`ncol` (Optional) The number of columns the new matrix should have.

`<<bycol` (Optional) Specifies that the values be placed into the reshaped matrix column-by-column, instead of row-by-row.

Notes

If *ncol* is not specified, the number of columns is whatever is necessary to fit all of the original values of the matrix into the reshaped matrix.

If a missing value is specified for *nrow*, the number of rows is whatever is necessary to fit all of the original values of the matrix into the reshaped matrix.

If the new matrix is smaller than the original matrix, the extra values are discarded.

If the new matrix is larger than the original matrix, the values are repeated to fill the new matrix.

Examples

```
a = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });
   [  1 2 3,
      4 5 6,
      7 8 9]

Shape(a, 2);
   [  1 2 3 4 5,
      6 7 8 9 1]

Shape(a, 2, 2);
   [  1 2,
      3 4]

Shape(a, 4, 4);
```

```
[  1 2 3 4,
   5 6 7 8,
   9 1 2 3,
   4 5 6 7]

Shape(a, 4, 4, <<bycol);
[  1 5 9 4,
   2 6 1 5,
   3 7 2 6,
   4 8 3 7]
```

Solve(A, b)

Description

Solves a linear system. In other words, `x=inverse(A)*b`.

Sort Ascending(source)

Description

Returns a copy of a list or matrix *source* with the items in ascending order.

Sort Descending(source)

Description

Returns a copy of a list or matrix *source* with the items in descending order.

Sparse SVD(X, <nSingularValues=min(nRow, nCol)>, <tolerance=1e-10>)

Description

Computes the singular value decomposition (SVD) of matrix X using the implicitly restarted, partially reorthogonalized Lanczos method for sparse matrices.

Returns

Returns a list {**U**, **M**, **V**} such that **U***diag(**M**)***V**′ is equal to **X**.

Arguments

`X` A matrix.

`nSingularValues` The number of singular values to return. By default, this value is the minimum of the number of rows and number of columns in the matrix X.

`tolerance` The tolerance for the convergence criterion.

Spline Coef(x, y, lambda)

Description

Returns a five column matrix of the form `knots||a||b||c||d` where `knots` is the unique values in *x*.

x is a vector of regressor variables, y is the vector of response variables, and lambda is the smoothing argument. Larger values for lambda result in smoother splines.

Spline Eval(x, coef)

Description

Evaluates the spline predictions using the coef matrix in the same form as returned by SplineCoef(), in other words, knots||a||b||c||d. The x argument can be a scalar or a matrix of values to predict. The number of columns of coef can be any number greater than 1 and each is used for the next higher power. The powers of x are centered at the knot values. For example, the calculation for coef of knots||a||b||c||d is j is such that knots[j] is the largest knot smaller than x.

xx = x-knots[j] is the centered *x* value:

```
result = a[j] + xx * (b[j] + xx * (c[j] + xx * d[j]))
```

The following line is equivalent:

```
result = a[j] + b[j] * xx + c[j] * xx ^ 2 + d[j] * xx ^ 3
```

Spline Smooth(x, y, lambda)

Description

Returns the smoothed predicted values from a spline fit.

x is a vector of regressor variables, y is the vector of response variables, and lambda is the smoothing argument. Larger values for lambda result in smoother splines.

SVD(A)

Description

Singular value decomposition.

Sweep(A, <indices>)

Description

Sweeps, or inverts a matrix a partition at a time.

Trace(A)

Description

The trace, or the sum of the diagonal elements of a square matrix.

Transpose(A)

Description

Transposes the rows and columns of the matrix *A*.

Returns

The transposed matrix.

Arguments

A A matrix.

Equivalent Expression

A‘

V Concat(A, B, ...)

Description

Vertical concatenation of two or more matrices.

Returns

The new matrix.

Arguments

Two or more matrices.

V Concat To(A, B, ...)

Description

Vertical concatenation in place. This is an assignment operator.

Returns

The new matrix.

Arguments

Two or more matrices.

V Max(matrix)

Description

Returns a row vector containing the maximum of each column of *matrix*.

V Mean(matrix)

Description

Returns a row vector containing the mean of each column of *matrix*.

`V Median(matrix)`

Description

Returns a row vector containing the median of each column of *matrix*.

`V Min(matrix)`

Description

Returns a row vector containing the minimum of each column of *matrix*.

`V Quantile(matrix, p)`

Description

Returns a row vector containing the p^{th} quantile of each column of *matrix*.

`V Standardize(matrix)`

Description

Returns a matrix column-standardized to mean = 0 and standard deviation = 1.

`V Std(matrix)`

Description

Returns a row vector containing the standard deviations of each column of *matrix*.

`V Sum(matrix)`

Description

Returns a row vector containing the sum of each column of *matrix*.

`Varimax(matrix, <norm=1>)`

Description

Performs a varimax rotation.

Returns

A list that contains the rotated matrix and the orthogonal rotation matrix.

Arguments

`matrix` A matrix to be rotated.

`norm` Specify 1 to perform a normalized rotation, and specify 0 to perform a non-normalized rotation. The default value is 1.

Vec Diag(A)

Description

Creates a vector from the diagonals of a square matrix *A*.

Returns

The new matrix.

Arguments

A A square matrix.

Notes

Using a matrix that is not square results in an error.

Vec Quadratic(symmetric matrix, rectangular matrix)

Description

Constructs an *n*-by-*m* matrix. Used in calculation of hat values.

Returns

The new matrix.

Arguments

Two matrices. The first must be symmetric.

Equivalent Expression

```
Vec Diag(X*Sym*X')
```

VPTree(matrix)

Description

Returns a table that is used for efficiently looking up nearest neighbors. The vantage-point tree algorithm is particularly useful for wide data problems.

Returns

A VPTree object.

Argument

matrix A matrix of *k*-dimensional points. The number of dimensions or points is not limited. Each column in the matrix represents a dimension to the data, and each row represents a data point.

Wavelet Basis Coef(x, grid, coeff, <wavelet = "Harr" | "Biorthogonal" | "Coiflet" | "Daubechies" | "Symlet">, <param = 0>)

Description

Finds the predictions for vector x for the specified Wavelet model.

Returns

A vector of predictions.

Arguments

`x` A row or column vector that contains the data.

`grid` A vector specifying the grid of the data.

`coeff` A vector of wavelet coefficients.

`wavelet` The name of the wavelet model.

`param` An optional wavelet model parameter.

Notes

This function is used in column formulas created by the Functional Data Explorer platform.

Numeric Functions

Abs(n)

Description

Calculates the absolute value of *n*.

Returns

Returns a positive number of the same magnitude as the value of *n*.

Argument

`n` Any number.

Ceiling(n)

Description

If *n* is not an integer, rounds *n* to the next highest integer.

Returns

Returns the smallest integer greater than or equal to *n*.

Argument

`n` Any number.

Derivative(expr, {name, ...}, ...)

Description

Calculates the derivative of the *expr* expression with respect to *name*.

Returns

Returns the derivative.

Arguments

`expr` Any expression. Indirect arguments (for example, `Name Expr`, `Expr`, `Eval`) are supported.

`name` Can be a single variable or a list of variables.

Notes

Adding an additional variable (`Derivative(expr, name, name2)`) takes the second derivative.

`Floor(n)`

Description

If *n* is not an integer, rounds *n* to the next lowest integer.

Returns

Returns the largest integer less than or equal to *n*.

Argument

`n` Any number.

Examples

```
Floor( 2.7 );
  2
Floor( -.5 );
  -1
```

`Integrate(expr, varname, lowLimit, upLimit, <<Tolerance(1e-10), <<StoreInfo({list}), <<StartingValue(val))`

Description

Integrates an expression with respect to a scalar value, using the adaptive quadrature method from Gander and Gautschi (2000).

Arguments

`expr` an expression that defines the integrand.

`varname` the name of the variable of integration. If this variable contains a value, that value specifies a starting value that is used as a typical value to improve the accuracy of the integral.

`lowLimit` specifies the lower limit of integration. To specify negative infinity as the lower limit of integration, set this to missing.

`upLimit` specifies the upper limit of integration. To specify positive infinity as the upper limit of integration, set this to missing.

`StoreInfo` saves diagnostics of the numerical integration routine to the argument of StoreInfo().

StartingValue specifies a starting value that is used as a typical value to improve the accuracy of the integral.

Invert Expr(expr, name)

Description

Attempts to unfold *expr* around *name*.

Mod()

See "Modulo(number, divisor)"

Modulo(number, divisor)

Mod(number, divisor)

Description

Returns the remainder when *number* is divided by *divisor*.

Examples

```
Modulo( 6, 5 );
  1
```

Normal Integrate(muVector, sigmaMatrix, expr, x, nStrata, nSim)

Description

Returns the result of radial-spherical integration for smooth functions of multivariate, normally distributed variables.

Arguments

muVector A vector.

sigmaMatrix A matrix.

expr An expression in terms of the variable *x*.

x The variable used in the expression *expr*.

nStrata Number of strata.

nSim Number of simulations.

Num Deriv(f(x,...), <parnum=1>)

Description

Returns the numerical derivative of the *f(x,...)* function with respect to one of its arguments. You can specify that argument as the second argument in the *Num Deriv* function. If no second argument is specified, the derivative is taken with respect to the function's first argument. The derivative is evaluated using numeric values specified in the *f(x,...)* function expression.

Notes

The `Num Deriv()` function might appear not to produce the correct results as seen here:

```
x = 3;
n = Num Deriv( 3 * x ^ 2 );
    // 9.00000000001455
```

The preceding usage is not correct. The function was designed to be used in the Nonlinear platform to differentiate functions for which it does not know the analytic derivatives. The proper usage takes the following form:

```
x = 3;
f = Function( {x}, 3 * x ^ 2 );
n = Num Deriv( f( x ), 1 );
   // 18.000029999854
```

`Num Deriv2(f(x,...))`

Description

Returns the numerical second derivative of the *f(x,...)* function with respect to *x*. The derivative is evaluated using numeric values specified in the *f(x,...)* function expression.

`Round(n, places)`

Description

Rounds *n* to number of decimal *places* given.

`Simplify Expr(expr(expression))`

`Simplify Expr(nameExpr(global))`

Description

Algebraically simplifies an expression

Optimization Functions

`Constrained Maximize(expr, {x1(low1, up1), x2(low2, up2), ...}, messages)`

Description

Finds the values for the *x* arguments, specified as a list, that maximize the *expr* expression with optional linear constraints. You must either specify lower and upper bounds in parentheses for each argument or with the optional `Set Variable Limit()` message. The *x* arguments can be scalar values or vectors.

In the following messages, A is a matrix of coefficients. `x = [x1, x2, ...]` is the vector of arguments. `b` is a vector that forms the right side of the expression.

Messages

`<<Less than EQ({A, b})` Sets the constraint to less than or equal to the specified values `(A*x <= b)`.

`<<Greater Than EQ({A, b})` Sets the constraint to greater than or equal to the specified values `(A*x >= b)`.

`<<Equal To({A, b})` Sets the constraint as equal to the specified values `(A*x = b)`.

`<<Starting Values([x1Start, x2Start, ...])` Specifies a starting point.

`<<Max Iter(int)` An integer that specifies the maximum number of iterations to be performed.

`<<Tolerance(p)` *p* sets the tolerance for the convergence criterion. The default tolerance is 10^{-5}.

`<<Show Details("true")` Returns a list with the final values for (objective value, number of iterations, gradient, and Hessian). Shows the step-by-step results of the optimizer in the log.

`<<SetVariableLimit({low,high})` Specifies vectors for the lower and upper limits for the optimization variables.

`Constrained Minimize(expr, {x1(low1, up1), x2(low2, up2), ...}, messages)`

Description

Finds the values for the *x* arguments, specified as a list, that minimize the *expr* expression with optional linear constraints. You must either specify lower and upper bounds in parentheses for each argument or with the optional `Set Variable Limit()` message. The *x* arguments can be scalar values or vectors.

In the following messages, A is a matrix of coefficients. `x = [x1, x2, ...]` is the vector of arguments. `b` is a vector that forms the right side of the expression.

Messages

`<<Less than EQ({A, b})` Sets the constraint to less than or equal to the specified values `(A*x <= b)`.

`<<Greater Than EQ({A, b})` Sets the constraint to greater than or equal to the specified values `(A*x >= b)`.

`<<Equal To({A, b})` Sets the constraint as equal to the specified values `(A*x = b)`.

`<<Starting Values([x1Start, x2Start, ...])` Specifies a starting point.

`<<Max Iter(int)` An integer that specifies the maximum number of iterations to be performed.

`<<Tolerance(p)` *p* sets the tolerance for the convergence criterion. The default tolerance is 10^{-5}.

<<Show Details("true") Returns a list with the final values for (objective value, number of iterations, gradient, and Hessian). Shows the step-by-step results of the optimizer in the log.

<<SetVariableLimit({low,high}) Specifies vectors for the lower and upper limits for the optimization variables.

Desirability(yVector, desireVector, y)

Description

Fits a function to go through the three points, suitable for defining the desirability of a set of response variables (*y*s). *yVector* and *desireVector* are matrices with three values, corresponding to the three points defining the desirability function. The actual function depends on whether the desire values are in the shape of a larger-is-better, smaller-is-better, target, or antitarget.

Returns

The desirability function.

Arguments

yVector Three input values.

desireVector the corresponding three desirability values.

y the value of which to calculate the desirability.

LPSolve(A, b, c, L, U, neq, nle, nge, <slackVars(Boolean)>)

Description

Returns a list containing the decision variables (and slack variables if applicable) in the first list item and the optimal objective function value (if one exists) in the second list item.

Arguments

A A matrix of constraint coefficients.

b A matrix that is a column of right hand side values of the constraints.

c A vector of cost coefficients of the objective function.

L, U Matrices of lower and upper bounds for the variables.

neq The number of equality constraints.

nle The number of less than or equal inequalities.

nge The number of greater than or equal inequalities.

slackVars(Boolean) (Optional) Determines whether the slack variables are returned in addition to the decision variables. The default value is 0.

Notes

The constraints must be listed as equalities first, less than or equal inequalities next, and greater than or equal inequalities last.

Maximize(expr, {x1(low1, up1), x2(low2, up2), ...}, messages)

Description

Finds the values for the *x* arguments, specified as a list, that maximize the expression *expr*. You can specify lower and upper bounds in parentheses for each argument. Additional arguments for the function enable you to set the maximum number of iterations, tolerance for convergence, and view more details about the optimization. The Newton-Raphson method is used when an analytical derivative is found for the Hessian. Otherwise, the Symmetric-Rank One method (SR1), a quasi-Newton method, is used.

Messages

<<Max Iter(int) An integer that specifies the maximum number of iterations to be performed. The default maximum number of iterations is 250.

<<Tolerance(p) *p* sets the tolerance for the convergence criterion. The default tolerance is 10^{-8}.

<<Details("both" | "displaySteps" | "returnDetails") Specifies what output is returned. If "displaySteps" is specified, step-by-step results of the optimization appear in the Log window. If "returnDetails" is specified, the function returns a list that contains the final values for the objective value, number of iterations, gradient, and Hessian. Specify "both" to get the return value and the results in the Log.

<<Gradient(exprList) Specifies a list of expressions that define the analytical gradient that is used for the optimization. Each expression in the list represents a derivative of the expression *expr*.

<<Hessian(exprList) Specifies a list of expressions that define the analytical Hessian that is used for the optimization. Each expression in the list represents the upper triangular portion of the Hessian matrix in row-major order.

<<Method(NR | SR1) Specifies either the Newton-Raphson (NR) method or the Symmetric-Rank One (SR1) method for the optimization method.

<<UseNumericDeriv("true") Specifies that the optimization use a numeric approximation.

Minimize(expr, {x1(low1, up1), x2(low2, up2), ...}, messages)

Description

Finds the values for the *x* arguments, specified as a list, that minimize the expression *expr*. You can specify lower and upper bounds in parentheses for each argument. Additional arguments for the function enable you to set the maximum number of iterations, tolerance for convergence, and view more details about the optimization. The Newton-Raphson method is used when an analytical derivative is found for the Hessian. Otherwise, the Symmetric-Rank One method (SR1), a quasi-Newton method, is used.

Messages

<<Max Iter(int) An integer that specifies the maximum number of iterations to be performed. The default maximum number of iterations is 250.

`<<Tolerance(p)` *p* sets the tolerance for the convergence criterion. The default tolerance is 10^{-8}.

`<<Details("both" | "displaySteps" | "returnDetails")` Specifies what output is returned. If "displaySteps" is specified, step-by-step results of the optimization appear in the Log window. If "returnDetails" is specified, the function returns a list that contains the final values for the objective value, number of iterations, gradient, and Hessian. Specify "both" to get the return value and the results in the Log.

`<<Gradient(exprList)` Specifies a list of expressions that define the analytical gradient that is used for the optimization. Each expression in the list represents a derivative of the expression *expr*.

`<<Hessian(exprList)` Specifies a list of expressions that define the analytical Hessian that is used for the optimization. Each expression in the list represents the upper triangular portion of the Hessian matrix in row-major order.

`<<Method(NR | SR1)` Specifies either the Newton-Raphson (NR) method or the Symmetric-Rank One (SR1) method for the optimization method.

`<<UseNumericDeriv("true")` Specifies that the optimization use a numeric approximation.

Probability Functions

Beta Density(x, alpha, beta, <theta=0>, <sigma=1>)

Description

Returns the probability density function (pdf) evaluated at *x* of the beta distribution. The pdf is parameterized as follows:

$$f(x) = \frac{1}{B(\alpha, \beta)\sigma^{\alpha+\beta-1}}(x-\theta)^{\alpha-1}(\theta+\sigma-x)^{\beta-1}$$

where $B(\cdot)$ is the Beta function.

Arguments

`x` A quantile at which the pdf is evaluated. *x* must be between *theta* and *theta* + *sigma*.

`alpha, beta` Shape parameters α and β, which must both be greater than 0.

`theta` Optional threshold parameter θ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Notes

The beta distribution is useful for modeling the probabilistic behavior of random variables that are constrained to fall in the interval [0, 1], such as proportions.

Beta Distribution(x, alpha, beta, <theta=0>, <sigma=1>)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the beta distribution. The cdf uses the same parameterization as the Beta Density() function.

Arguments

x A quantile at which the cdf is evaluated. *x* must be between *theta* and *theta* + *sigma*.

alpha, beta Shape parameters α and β, which must both be greater than 0.

theta Optional threshold parameter θ. The default is 0.

sigma Optional scale parameter σ, which must be greater than 0. The default is 1.

Beta Quantile(p, alpha, beta, <theta=0>, <sigma=1>)

Description

Returns the p^{th} quantile from a beta distribution with shape arguments *alpha* and *beta*. The quantile function does not have a closed form equation.

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

alpha, beta Shape parameters α and β, which must both be greater than 0.

theta Optional threshold parameter θ. The default is 0.

sigma Optional scale parameter σ, which must be greater than 0. The default is 1.

Cauchy Density(q, <center=0>, <scale=1>)

Description

Returns the probability density function (pdf) evaluated at *q* of a Cauchy distribution. The pdf is parameterized as follows:

$$f(q) = \frac{1}{\pi\sigma}\frac{1}{1+\left(\frac{q-\mu}{\sigma}\right)^2}$$

Arguments

q A quantile at which the pdf is evaluated.

center Optional location parameter μ. The default is 0.

scale Optional scale parameter, σ, which must be greater than 0. The default is 1.

Cauchy Distribution(q, <center=0>, <scale=1>)

Description

Returns the cumulative distribution function (cdf) probability that a Cauchy distributed random variable is less than *q*. The cdf is parameterized as follows:

$$F(q) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x-\mu}{\sigma}\right)$$

Arguments

`q` A quantile at which the cdf is evaluated.

`center` Optional location parameter μ. The default is 0.

`scale` Optional scale parameter, σ, which must be greater than 0. The default is 1.

`Cauchy Quantile(p, <center=0>, <scale=1>)`

Description

Returns the p^{th} quantile from a Cauchy distribution. The p^{th} quantile is the value for which the probability is p that a random value would be less than or equal to p. The quantile function is parameterized as follows:

$$F^{-1}(p) = \sigma\tan\left[\pi\left(p + \frac{1}{2}\right)\right] + \mu$$

Arguments

`p` The probability of the quantile desired. p must be between 0 and 1.

`center` Optional location parameter μ. The default is 0.

`scale` Optional scale parameter σ, which must be greater than 0. The default is 1.

`ChiSquare Density(q, df, <nc=0>)`

Description

Returns the probability density function (pdf) evaluated at q of the chi-square distribution. The pdf is parameterized as follows:

$$f(q) = \exp(-\lambda/2)\sum_{r=0}^{\infty}\frac{(\lambda/2)^r}{r!}f_{n+2r}(q)$$

where $f_{n+2r}(q)$ is the density of a central chi-square distribution with $n+2r$ degrees of freedom.

Arguments

`q` A quantile at which the pdf is evaluated. q must be greater than or equal to 0.

`df` The degrees of freedom n, which must be greater than 0.

`nc` Optional noncentrality parameter λ, which must be nonnegative. The default is 0.

ChiSquare Distribution(q, df, <nc=0>)

Description

Returns cumulative distribution function at quantile *x* for chi-square with *df* degrees of freedom centered at *nc*. The cdf is parameterized as follows:

$$F(q) = \exp(-\lambda/2) \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} F_{n+2r}(q) \ ,$$

where $F_{n+2r}(q)$ is the cumulative distribution of a central chi-square distribution with $n+2r$ degrees of freedom.

Arguments

q A quantile at which the cdf is evaluated. *q* must be greater than or equal to 0.

df The degrees of freedom *n*, must be greater than 0.

nc The optional noncentrality parameter λ, must be nonnegative. The default is 0.

ChiSquare Log CDistribution(x, df, <nc=0>)

Description

Returns the log of (1 - value), where value is the cumulative distribution function evaluated at *x* of the chi-square distribution with *df* degrees of freedom and noncentrality parameter *nc*.

ChiSquare Log Density(x, df, <nc=0>)

Description

Returns the log of the value of the probability density function evaluated at *x* of the chi-square distribution with *df* degrees of freedom and noncentrality parameter *nc*.

ChiSquare Log Distribution(x, df, <nc=0>)

Description

Returns the log of the value of the cumulative distribution function evaluated at quantile *x* of the chi-square distribution with *df* degrees of freedom and noncentrality parameter *nc*.

ChiSquare Noncentrality(x, df, prob)

Description

Returns the chi-square distribution noncentrality parameter *nc* that satisfies the following:

prob = ChiSquare Distribution(*x, df, nc*)

Arguments

x A quantile at which the cdf is evaluated.

`df` The degrees of freedom *n*, which must be greater than 0.

`prob` The probability of the quantile desired; *prob* must be between 0 and 1.

`ChiSquare Quantile(p, df, <nc=0>)`

Description

Returns the p^{th} quantile from a chi-square distribution with *df* degrees of freedom, centered at *nc*. The quantile function does not have a closed form equation.

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`df` The degrees of freedom *n*, which must be greater than 0.

`nc` Optional noncentrality parameter λ, which must be nonnegative. The default is 0.

`Dunnett P Value(q, nTrt, dfe, <lambdaVec=.>)`

Description

Returns the *p*-value from Dunnett's multiple comparisons test.

Arguments

`q` A number that is the test statistic.

`nTrt` The number of treatments being compared to the control treatment.

`dfe` The error degrees of freedom.

`lambdaVec` A vector of parameters. If *lambdaVec* is missing (`.`), each of the parameters is set to 1/Sqrt(2).

`Dunnett Quantile(1-alpha, nTrt, dfe, <lambdaVec=.>)`

Description

Returns the quantile used in Dunnett's multiple comparisons test.

Arguments

`1-alpha` A number that is the confidence level.

`nTrt` The number of treatments being compared to the control treatment.

`dfe` The error degrees of freedom.

`lambdaVec` A vector of parameters. If *lambdaVec* is missing (`.`), each of the parameters is set to 1/Sqrt(2).

`Exp Density(x, <theta=1>)`

`Exponential Density(x, <theta=1>)`

Description

Returns the probability density function (pdf) evaluated at *x* of the exponential distribution. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\theta}\exp(-x/\theta)$$

Arguments

`x` A quantile at which the pdf is evaluated. *x* must be greater than or equal to 0.

`theta` Optional scale parameter θ, which must be greater than 0. The default is 1.

`Exp Distribution(x, <theta=1>)`

`Exponential Distribution(x, <theta=1>)`

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the exponential distribution. The cdf is parameterized as follows:

$$F(x) = 1 - \exp(-x/\theta)$$

Arguments

`x` A quantile at which the cdf is evaluated. *x* must be greater than or equal to 0.

`theta` Optional scale parameter θ, which must be greater than 0. The default is 1.

`Exp Quantile(p, <theta=1>)`

`Exp Quantile(p, <theta=1>)`

Description

Returns the p^{th} quantile from an exponential distribution with scale parameter *theta*. The quantile function is parameterized as follows:

$$F^{-1}(p) = -\theta\log(1-p)$$

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`theta` Optional scale parameter θ, which must be greater than 0. The default is 1.

`F Density(x, dfnum, dfden, <nc=0>)`

Description

Returns the probability density function (pdf) evaluated at *x* for the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden*, with optional noncentrality parameter *nc*.

$$f(x) = \exp(-\lambda/2) \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{\mathrm{B}\left(\frac{v_2}{2}, \frac{v_1}{2}+r\right) r!} \left(\frac{v_1}{v_2}\right)^{\frac{v_1}{2}+r} \left(1+\frac{v_1}{v_2}x\right)^{-\left(\frac{v_1+v_2}{2}+r\right)} x^{\frac{v_1}{2}-1+r}$$

where B(·) is the Beta function.

Arguments

x A quantile at which the pdf is evaluated. *x* must be greater than 0.

dfnum The degrees of freedom, v_1, of the chi-square distribution in the numerator of the *F*-distribution. *dfnum* must be greater than 0.

dfden The degrees of freedom, v_2, of the chi-square distribution in the denominator of the *F*-distribution. *dfden* must be greater than 0.

nc Optional noncentrality parameter λ, which must be nonnegative. The default is 0.

F Distribution(x, dfnum, dfden, <nc=0>)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* for the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden* and noncentrality parameter *nc*.

F Log CDistribution(x, dfnum, dfden, <nc=0>)

Description

Returns the log of (1 - value), where value is the cumulative distribution function evaluated at *x* of the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden*, with optional noncentrality parameter *nc*.

F Log Density(x, dfnum, dfden, <nc=0>)

Description

Returns the log of the value of the probability density function (pdf) evaluated at *x* for the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden*, with optional noncentrality parameter *nc*.

F Log Distribution(x, dfnum, dfden, <nc=0>)

Description

Returns the log of the value of the cumulative distribution function (cdf) evaluated at *x* for the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden* and noncentrality parameter *nc*.

F Noncentrality(x, dfnum, dfden, prob)

Description

Returns the F distribution noncentrality parameter *nc* that satisfies the following:

prob = F Distribution(*x, dfnum, dfden, nc*)

See Also

"F Distribution(x, dfnum, dfden, <nc=0>)"

F Power(alpha, dfh, dfm, d, n)

Description

Returns the power from a given situation involving an *F* test or a *t* test.

Arguments

`alpha` The significance level of the test. *alpha* must be between 0 and 1.

`dfh` The hypothesis degrees of freedom. *dfh* must be greater than 0.

`dfm` The degrees of freedom in the whole model. *dfm* must be greater than 0.

`d` The squared effect size, defined as Δ^2/σ^2. In this equation, σ^2 is the error variance and Δ^2 is defined as follows:

$\Delta^2 = (\bar{x} - \mu)^2$ for a one-sample *t* test

$\Delta^2 = \frac{(\bar{x}_1 - \bar{x}_2)^2}{4}$ for a two-sample *t* test

$\Delta^2 = \sqrt{\sum_{i=1}^{k} \frac{(\bar{x}_i - \bar{x})^2}{k}}$ for a *k*-sample *F* test

`n` The total number of observations. *n* must be greater than *dfm*.

F Quantile(x, dfnum, dfden, <nc=0>)

Description

Returns the p^{th} quantile from the F distribution with numerator and denominator degrees of freedom *dfnum* and *dfden* and noncentrality parameter *nc*.

F Sample Size(alpha, dfh, dfm, d, power)

Description

Returns the sample size from a given situation involving an *F* test or a *t* test.

Arguments

`alpha` The significance level of the test. *`alpha`* must be between 0 and 1.

`dfh` The hypothesis degrees of freedom. *`dfh`* must be greater than 0.

`dfm` The degrees of freedom in the whole model. *`dfm`* must be greater than 0.

`d` The squared effect size, defined as Δ^2/σ^2. In this equation, σ^2 is the error variance and Δ^2 is defined as follows:

$$\Delta^2 = (\bar{x} - \mu)^2 \text{ for a one-sample } t \text{ test}$$

$$\Delta^2 = \frac{(\bar{x}_1 - \bar{x}_2)^2}{4} \text{ for a two-sample } t \text{ test}$$

$$\Delta^2 = \sqrt{\sum_{i=1}^{k} \frac{(\bar{x}_i - \bar{x})^2}{k}} \text{ for a } k\text{-sample } F \text{ test}$$

`power` The desired power for the test.

Frechet Density(x, mu, sigma)

Description

Returns the probability density function (pdf) evaluated at *`x`* of the Fréchet distribution. The pdf is parameterized as follows:

$$f(x) = \exp\left[-\exp\left(-\frac{\log(x) - \mu}{\sigma}\right)\right]\exp\left(-\frac{\log(x) - \mu}{\sigma}\right)\frac{1}{x\sigma}$$

Arguments

`x` A quantile at which the pdf is evaluated. *`x`* must be greater than 0.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

Frechet Distribution(x, mu, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *`x`* of the Fréchet distribution. The cdf is parameterized as follows:

$$F(x) = \exp\left[-\exp\left(-\frac{\log(x) - \mu}{\sigma}\right)\right]$$

Arguments

`x` A quantile at which the cdf is evaluated. *x* must be greater than 0.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`Frechet Quantile(p, mu, sigma)`

Description

Returns the p^{th} quantile from a Fréchet distribution with location *mu* and scale *sigma*. The quantile function is parameterized as follows:

$$F^{-1}(p) = \exp[-\sigma\log\{-\log(p)\} + \mu]$$

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`Gamma Density(x, <alpha=1>, <scale=1>, <threshold=0>)`

Description

Returns the probability density function (pdf) evaluated at *x* of the Gamma distribution. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}}(x-\theta)^{\alpha-1}\exp(-(x-\theta)/\beta)$$

Arguments

`x` A quantile at which the pdf is evaluated. *x* must be greater than θ.

`alpha` Optional shape parameter α, which must be greater than 0. The default is 1.

`scale` Optional scale parameter β, which must be greater than 0. The default is 1.

`threshold` Optional threshold parameter θ. The default is 0.

`Gamma Distribution(x, <alpha=1>, <scale=1>, <threshold=0>)`

`IGamma(x, <alpha=1, scale=1, threshold=0>)`

Description

Returns the cumulative distribution function (cdf) evaluated at quantile *x* for the gamma distribution with parameters *alpha*, *scale*, and *threshold*.

Gamma Log CDistribution(x, <alpha=1>, <scale=1>, <threshold=0>)

Description

Same as `Log(1 – Gamma Distribution(x, alpha))` except that it has a much greater range.

Gamma Log Density(x, <alpha=1>, <scale=1>, <threshold=0>)

Description

Same as `Log(Gamma Density(x, alpha))` except that it has a much greater range.

Gamma Log Distribution(x, <alpha=1>, <scale=1>, <threshold=0>)

Description

Same as `Log(Gamma Distribution(x, alpha))` except that it has a much greater range.

Gamma Quantile(p, <alpha=1>, <scale=1>, threshold>)

Description

Returns the *p*th quantile from the gamma distribution with the *alpha, scale,* and *threshold* parameters given.

GenGamma Density(x, mu, sigma, lambda)

Description

Returns the probability density function (pdf) evaluated at *x* of an extended generalized gamma probability distribution. The pdf is parameterized as follows:

$$f(x) = \begin{cases} \frac{|\lambda|}{x\sigma}\phi_{\lg}[\lambda\omega + \log(\lambda^{-2}); \quad \lambda^{-2}] & \text{if } \lambda \neq 0 \\ \frac{1}{x\sigma}\phi_{\text{nor}}(\omega) & \text{if } \lambda = 0 \end{cases}$$

where $\omega = [\log(x) - \mu]/\sigma$. Note that the following is the pdf for the standardized log-gamma variable with shape parameter $\kappa > 0$:

$$\phi_{\lg}(z;\kappa) = \frac{1}{\Gamma(\kappa)}\exp[\kappa z - \exp(z)]$$

Note that $\phi_{\text{nor}}(\cdot)$ is the standard normal pdf.

Arguments

x A quantile at which the pdf is evaluated. *x* must be greater than 0.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ.

GenGamma Distribution(x, mu, sigma, lambda)

Description

Returns the cumulative distribution function (cdf) of the extended generalized gamma distribution. The cdf is parameterized as follows:

$$F(x) = \begin{cases} \Phi_{lg}[\lambda\omega + \log(\lambda^{-2}); \quad \lambda^{-2}] & \text{if } \lambda > 0 \\ \Phi_{nor}(\omega) & \text{if } \lambda = 0 \\ 1 - \Phi_{lg}[\lambda\omega + \log(\lambda^{-2}); \quad \lambda^{-2}] & \text{if } \lambda < 0 \end{cases}$$

where $\omega = [\log(x) - \mu]/\sigma$. Note that the following is the cdf for the standardized log-gamma variable with shape parameter $\kappa > 0$:

$$\Phi_{lg}(z;\kappa) = \Gamma_I[\exp(z);\kappa]$$

where $\Gamma_I[\cdot]$ denotes the incomplete gamma function. Note that $\Phi_{nor}(\cdot)$ is the standard normal cdf.

Arguments

`x` A quantile at which the cdf is evaluated. *x* must be greater than 0.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ.

GenGamma Quantile(p, mu, sigma, lambda)

Description

Returns the p^{th} quantile from an extended generalized gamma distribution with parameters *mu*, *sigma*, and *lambda*. The quantile function does not have a closed form equation.

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ.

GLog Density(x, mu, sigma, lambda)

Description

Returns the probability density function (pdf) evaluated at ***x*** of a generalized logarithmic distribution. The pdf is parameterized as follows:

$$f(x) = \phi\left\{\frac{1}{\sigma}\left[\log\left(\frac{x+\sqrt{x^2+\lambda^2}}{2}\right)-\mu\right]\right\}\frac{x+\sqrt{x^2+\lambda^2}}{\sigma(x^2+\lambda^2+x\sqrt{x^2+\lambda^2})}$$

where $\phi(\cdot)$ is the standard normal pdf.

Arguments

`x` A quantile at which the pdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ, which must be greater than 0.

Notes

When the shape parameter is equal to zero, the distribution reduces to a Lognormal(μ, σ).

GLog Distribution(x, mu, sigma, lambda)

Description

Returns the probability that a generalized logarithmically distribution random variable is less than ***x***. The cdf is parameterized as follows:

$$F(x) = \Phi\left\{\frac{1}{\sigma}\left[\log\left(\frac{x+\sqrt{x^2+\lambda^2}}{2}\right)-\mu\right]\right\}$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

`x` A quantile at which the cdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ, which must be greater than 0.

GLog Quantile(p, mu, sigma, lambda)

Description

Returns the p^{th} quantile from a generalized logarithmic distribution.

IGamma()

See "Gamma Distribution(x, <alpha=1>, <scale=1>, <threshold=0>)".

Johnson Sb Density(q, gamma, delta, theta, sigma)

Description

Returns the probability density function (pdf) evaluated at *q* of a Johnson Sb distribution. The pdf is parameterized as follows:

$$f(q) = \phi\left[\gamma + \delta \ln\left(\frac{q-\theta}{\sigma-(q-\theta)}\right)\right]\left(\frac{\delta\sigma}{(q-\theta)(\sigma-(q-\theta))}\right)$$

where $\phi(\cdot)$ is the standard normal pdf.

Arguments

q A quantile at which the pdf is evaluated. *q* must be in the interval *theta* to *theta* + *sigma*.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Johnson Sb Distribution(q, gamma, delta, theta, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *q* of a Johnson Sb distribution. The pdf is parameterized as follows:

$$F(q) = \Phi\left[\gamma + \delta \ln\left(\frac{q-\theta}{\sigma-(q-\theta)}\right)\right]$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

q A quantile at which the cdf is evaluated. *q* must be in the interval *theta* to *theta* + *sigma*.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Johnson Sb Quantile(p, gamma, delta, theta, sigma)

Description

Returns the p^{th} quantile from a Johnson Sb distribution.

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Johnson Sl Density(x, gamma, delta, theta, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of a Johnson Sl distribution. The pdf is parameterized as follows:

$$f(x) = \frac{\delta}{|x-\theta|}\phi\left[\gamma + \delta\ln\left(\frac{x-\theta}{\sigma}\right)\right]$$

where $\phi(\cdot)$ is the standard normal pdf.

Arguments

x A quantile at which the pdf is evaluated. *x* must be greater than *theta* if *sigma* is 1 and less than *theta* if *sigma* is -1.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Parameter σ that indicates if the distribution is skewed positively or negatively. sigma must be equal to either +1 (skewed positively) or -1 (skewed negatively).

Johnson Sl Distribution(q, gamma, delta, theta, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *q* of a Johnson Sl distribution.

$$F(x) = \begin{cases} \Phi\left[\gamma + \delta\ln\left(\frac{x-\theta}{\sigma}\right)\right], & \sigma = 1 \\ 1 - \Phi\left[\gamma + \delta\ln\left(\frac{x-\theta}{\sigma}\right)\right], & \sigma = -1 \end{cases}$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

q A quantile at which the cdf is evaluated. *q* must be greater than *theta* if *sigma* is 1 and less than *theta* if *sigma* is -1.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Parameter σ that defines if the distribution is skewed positively or negatively. Sigma must be equal to either +1 (skewed positively) or -1 (skewed negatively).

Johnson Sl Quantile(p, gamma, delta, theta, sigma)

Description

Returns the p^{th} quantile from a Johnson Sl distribution.

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Parameter σ that defines if the distribution is skewed positively or negatively. Sigma must be equal to either +1 (skewed positively) or -1 (skewed negatively).

Johnson Su Density(x, gamma, delta, theta, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of a Johnson Su distribution. The pdf is parameterized as follows:

$$f(x) = \frac{\delta}{\sigma}\left[1 + \left(\frac{x-\theta}{\sigma}\right)^2\right]^{-1/2} \phi\left[\gamma + \delta \sinh^{-1}\left(\frac{x-\theta}{\sigma}\right)\right]$$

where $\phi(\cdot)$ is the standard normal pdf.

Arguments

x A quantile at which the pdf is evaluated.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Johnson Su Distribution(q, gamma, delta, theta, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *q* of a Johnson Su distribution. The cdf is parameterized as follows:

$$F(x) = \Phi\left[\gamma + \delta \sinh^{-1}\left(\frac{x-\theta}{\sigma}\right)\right]$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

q A quantile at which the cdf is evaluated.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Johnson Su Quantile(p, gamma, delta, theta, sigma)

Description

Returns the p^{th} quantile from a Johnson Su distribution.

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

LEV Density(x, mu, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of the largest extreme value distribution with location *mu* and scale *sigma*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\sigma}\exp\left[-\frac{x-\mu}{\sigma} - \exp\left(-\frac{x-\mu}{\sigma}\right)\right]$$

Arguments

x A quantile at which the pdf is evaluated.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

LEV Distribution(x, mu, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the largest extreme value distribution with location *mu* and scale *sigma*. The cdf is parameterized as follows:

$$F(x) = \exp\left[-\exp\left(-\frac{x-\mu}{\sigma}\right)\right]$$

Arguments

x A quantile at which the cdf is evaluated. *x* must be greater than *sigma*.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

LEV Quantile(p, mu, sigma)

Description

Returns the p^{th} quantile from a largest extreme value distribution with location *mu* and scale *sigma*. The quantile function is parameterized as follows:

$$F^{-1}(p) = -\sigma\log(-\log(p)) + \mu$$

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

LogGenGamma Density(x, mu, sigma, lambda)

Description

Returns the probability density function (pdf) evaluated at *x* of a log generalized gamma probability distribution with parameters *mu*, *sigma*, and *lambda*. The pdf is parameterized as follows:

$$f(x) = \begin{cases} \frac{|\lambda|}{\sigma}\phi_{lg}[\lambda\omega + \log(\lambda^{-2});\lambda^{-2}] & \text{if } \lambda \neq 0 \\ \frac{1}{\sigma}\phi_{nor}(\omega) & \text{if } \lambda = 0 \end{cases}$$

where $\omega = [x - \mu]/\sigma$. Note that the following is the pdf for the log-gamma variable with shape parameter $\kappa > 0$:

$$\phi_{lg}(z;\kappa) = \frac{1}{\Gamma(\kappa)}\exp[\kappa z - \exp(z)]$$

Note that $\phi_{nor}(\cdot)$ is the standard normal pdf.

Arguments

`x` A quantile at which the pdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ.

LogGenGamma Distribution(x, mu, sigma, lambda)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the log generalized gamma distributed random variable (with parameters *mu*, *sigma*, and *lambda*) The cdf is parameterized as follows:

$$F(x) = \begin{cases} \Phi_{lg}[\lambda\omega + \log(\lambda^{-2});\lambda^{-2}] & \text{if } \lambda > 0 \\ \Phi_{nor}(\omega) & \text{if } \lambda = 0 \\ 1 - \Phi_{lg}[\lambda\omega + \log(\lambda^{-2});\ \lambda^{-2}] & \text{if } \lambda < 0 \end{cases}$$

where $\omega = [x - \mu]/\sigma$. Note that the following is the cdf for the log-gamma variable with shape parameter $\kappa > 0$:

$$\Phi_{lg}(z;\kappa) = \Gamma_I[\exp(z);\kappa]$$

where $\Gamma_I[\cdot]$ denotes the incomplete gamma function. Note that $\Phi_{nor}(\cdot)$ is the standard normal cdf.

Arguments

`x` A quantile at which the cdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ.

LogGenGamma Quantile(p, mu, sigma, lambda)

Description

Returns the p^{th} quantile from a log generalized gamma distribution.

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

lambda A shape parameter λ.

Logistic Density(x, mu, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of a logistic distribution with location *mu* and scale *sigma*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\sigma}\frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\left[1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right]^2}$$

Arguments

x A quantile at which the pdf is evaluated.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

Logistic Distribution(x, mu, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the logistic distribution with location *mu* and scale *sigma*. The cdf is parameterized as follows:

$$F(x) = \frac{1}{\left[1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right]}$$

Arguments

x A quantile at which the cdf is evaluated. *x* must be greater than σ.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

Logistic Quantile(p, mu, sigma)

Description

Returns the p^{th} quantile from a logistic distribution with location *mu* and scale *sigma*. The quantile function is parameterized as follows:

$$F^{-1}(p) = -\sigma\log\left(\frac{1}{p} - 1\right) + \mu$$

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`Loglogistic Density(x, mu, sigma)`

Description

Returns the probability density function (pdf) evaluated at *x* of a loglogistic distribution with location *mu* and scale *sigma*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{x\sigma}\frac{\exp\left(\frac{\log(x) - \mu}{\sigma}\right)}{\left[1 + \exp\left(\frac{\log(x) - \mu}{\sigma}\right)\right]^2}$$

Arguments

`x` A quantile at which the pdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`Loglogistic Distribution(x, mu, sigma)`

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the loglogistic distribution with location *mu* and scale *sigma*. The cdf is parameterized as follows:

$$F(x) = \frac{1}{1 + \exp\left(-\frac{\log(x) - \mu}{\sigma}\right)}$$

Arguments

`x` A quantile at which the cdf is evaluated.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

Loglogistic Quantile(p, mu, sigma)

Description

Returns the p^{th} quantile from a loglogistic distribution with location *mu* and scale *sigma*. The quantile function is parameterized as follows:

$$F^{-1}(p) = \exp\left[-\sigma\log\left(\frac{1}{p} - 1\right) + \mu\right]$$

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

Lognormal Density(x, mu, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of a lognormal distribution with location *mu* and scale *sigma*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{x}\phi\left[\frac{\log(x) - \mu}{\sigma}\right]$$

where $\phi(\cdot)$ is the standard normal pdf.

Arguments

x A quantile at which the pdf is evaluated. *x* must be greater than or equal to 0.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

Lognormal Distribution(x, mu, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of a lognormal distribution with location *mu* and scale *sigma*. The cdf is parameterized as follows:

$$F(x) = \Phi\left[\frac{\log(x) - \mu}{\sigma}\right]$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

x A quantile at which the pdf is evaluated. *x* must be greater than or equal to 0.

mu The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

Lognormal Quantile(x, mu, sigma)

Description

Returns the p^{th} quantile of a lognormal distribution with location *mu* and scale *sigma*.

Normal Biv Distribution(x, y, r, <mu1>, <s1>, <mu2>, <s2>)

Description

Computes the probability that an observation (X, Y) is less than or equal to (*x, y*) with correlation coefficient *r* where X is individually normally distributed with mean *mu1* and standard deviation *s1* and Y is individually normally distributed with mean *mu2* and standard deviation *s2*. If *mu1*, *s1*, *mu2*, and *s2* are not given, the function assumes the standard normal bivariate distribution with *mu1*=0, *s1*=1, *mu2*=0, and *s2*=1.

Normal Density(x, <mean=0>, <stddev=1>)

Description

Returns the probability density function (pdf) evaluated at *x* for the normal distribution with *mean* and *stddev*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Arguments

`x` A quantile at which the pdf is evaluated.

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Notes

The normal distribution is bell shaped and symmetrical.

Normal Distribution(x, <mean=0>, <stddev=1>)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* for the normal distribution with *mean* and *stddev*. The cdf is parameterized as follows:

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$

Note that $\Phi(\cdot)$ is the standard normal cdf, defined as follows:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x \exp\left(-\frac{t^2}{2}\right) dt$$

Arguments

x A quantile at which the pdf is evaluated.

mu Optional location parameter μ. The default is 0.

sigma Optional scale parameter σ, which must be greater than 0. The default is 1.

`Normal Log CDistribution(x, <mean=0>, <std dev=1>)`

Description

Returns 1 - log (value) of the distribution function at quantile *x* for the normal distribution.

`Normal Log Density(x, <mean=0>, <stddev=1>)`

Description

Returns the log of the value of the density function at quantile *x* for the normal distribution with *mean* and *stddev*. The default *mean* is 0. The default *stddev* is 1.

`Normal Log Distribution(x, <mean=0>, <std dev=1>)`

Description

Returns the log of the value of the distribution function at quantile *x* for the normal distribution.

`Normal Mixture Density(q, mean, stdev, probability)`

Description

Returns the density at *q* of a normal mixture distribution with group means *mean,* group standard deviations *stdev,* and group probabilities *probability*. The *mean, stdev,* and *probability* arguments are all vectors of the same size.

`Normal Mixture Distribution(q, mean, stdev, probability)`

Description

Returns the probability that a normal mixture distributed variable with group means *mean,* group standard deviations *stdev,* and group probabilities *probability* is less than *q*. The *mean, stdev,* and *probability* arguments are all vectors of the same size.

Normal Mixture Quantile(p, mean, stdev, probability)

Description

Returns the p^{th} quantile, the values for which the probability is *p* that a random value would be lower. The *mean*, *stdev*, and *probability* arguments are all vectors of the same size.

Normal Quantile(p, <mean=0>, <stddev=1>)

Probit(p, <mean=0>, <stddev=1>)

Description

Returns the p^{th} quantile from the normal distribution with *mean* and *stddev*. The default *mean* is 0. the default *stddev* is 1.

Probit()

See "Normal Quantile(p, <mean=0>, <stddev=1>)".

SEV Density(x, mu, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of the smallest extreme distribution with location *mu* and scale *sigma*. The pdf is parameterized as follows:

$$f(x) = \frac{1}{\sigma}\exp\left[\frac{x-\mu}{\sigma} - \exp\left(\frac{x-\mu}{\sigma}\right)\right]$$

Arguments

x A quantile at which the pdf is evaluated.

mu The location parameter μ.

sigma The scale parameter σ, which must be greater than 0.

SEV Distribution(x, mu, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the smallest extreme distribution with location *mu* and scale *sigma*. The cdf is parameterized as follows:

$$F(x) = 1 - \exp\left[-\exp\left(\frac{x-\mu}{\sigma}\right)\right]$$

Arguments

x A quantile at which the cdf is evaluated. *x* must be greater than σ.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

SEV Quantile(p, mu, sigma)

Description

Returns the p^{th} quantile of the smallest extreme distribution with location *mu* and scale *sigma*. The quantile function is parameterized as follows:

$$F^{-1}(p) = \sigma\log[-\log(1-p)] + \mu$$

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

SHASH Density(x, gamma, delta, theta, sigma)

Description

Returns the probability density function (pdf) evaluated at *x* of a sinh-arcsinh (SHASH) distribution. The pdf is parameterized as follows:

$$f(x) = \frac{\delta\cosh(w)}{\sqrt{\sigma^2 + (x-\theta)^2}}\phi[\sinh(w)]$$

where

$\phi(\cdot)$ is the standard normal pdf

$$w = \gamma + \delta\sinh^{-1}\left(\frac{x-\theta}{\sigma}\right)$$

Arguments

`x` A quantile at which the pdf is evaluated.

`gamma` The shape parameter γ.

`delta` The shape parameter δ, which must be greater than 0.

`theta` The location parameter θ.

`sigma` The scale parameter σ, which must be greater than 0.

SHASH Distribution(x, gamma, delta, theta, sigma)

Description

Returns the cumulative distribution function (cdf) evaluated at *x* of the sinh-arcsinh (SHASH) distribution. The cdf is parameterized as follows:

$$F(x) = \Phi\left[\sinh\left(\gamma + \delta \sinh^{-1}\left(\frac{x-\theta}{\sigma}\right)\right)\right]$$

where $\Phi(\cdot)$ is the standard normal cdf.

Arguments

x A quantile at which the cdf is evaluated.

gamma The shape parameter γ.

delta The shape parameter δ, which must be greater than 0.

theta The location parameter θ.

sigma The scale parameter σ, which must be greater than 0.

SHASH Quantile(p, gamma, delta, theta, sigma)

Description

Returns the p^{th} quantile from a sinh-arcsinh (SHASH) distribution (with parameters *gamma*, *delta*, *theta*, and *sigma*).

Arguments

p The probability of the quantile desired. *p* must be between 0 and 1.

gamma The shape parameter γ.

delta The shape parameter δ, which must be greater than 0.

theta The location parameter θ.

sigma The scale parameter σ, which must be greater than 0.

Students t Density()

See "t Density(x, df, <nc=0>)".

Students t Distribution()

See "t Distribution(q, df, <nc=0>)".

Students t Quantile()

See "t Quantile(p, df, <nc=0>)".

t Density(x, df, <nc=0>)

Students t Density(x, df, <nc=0>)

Description

Returns the probability density function (pdf) evaluated at *x* of the Student's *t* distribution with degrees of freedom *df*. The pdf is parameterized as follows:

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\nu\pi}} \left[1 + \frac{x^2}{\nu}\right]^{-\left(\frac{\nu+1}{2}\right)}$$

Arguments

x A quantile at which the pdf is evaluated.

df The degrees of freedom ν, which must be greater than or equal to 1.

nc The optional noncentrality parameter λ, which must be nonnegative. The default is 0.

t Distribution(q, df, <nc=0>)

Students t Distribution(q, df, <nc=0>)

Description

Returns the probability that a Student's *t* distributed random variable is less than *q*. *nc* defaults to 0.

t Log CDistribution(x, df, <nc=0>)

Description

Returns 1 - log (value) of the normal distribution function at quantile *x* for the *t* distribution.

t Log Density(x, df, <nc=0>)

Description

Returns the log of the value of the density function at quantile *x* for the *t* distribution.

t Log Distribution(x, df, <nc=0>)

Description

Returns the log of the value of the distribution function at quantile *x* for the *t* distribution.

`t Noncentrality(x, df, prob)`

Description

Returns the *t* distribution noncentrality parameter *nc* that satisfies the following:

prob = T Distribution(*x, df, nc*)

`t Quantile(p, df, <nc=0>)`

`Students t Quantile(p, df, <nc=0>)`

Description

Returns the p^{th} quantile from the Student's *t* distribution with degrees of freedom *df*. *nc* defaults to 0.

`Tukey HSD P Value(q, n, dfe)`

Description

Returns the *p*-value from Tukey's HSD multiple comparisons test.

Arguments

`q` The test statistic. The test statistic that is specified is Tukey's adjusted critical value, which is the quantile of Tukey's studentized range distribution divided by the square root of 2.

`n` The number of groups in the study.

`dfe` The error degrees of freedom, based on the total study sample.

`Tukey HSD Quantile(1-alpha, n, dfe)`

Description

Returns the quantile used in Tukey's HSD multiple comparisons test. The quantile that is returned is Tukey's adjusted critical value, which is the quantile of Tukey's studentized range distribution divided by the square root of 2.

Arguments

`1-alpha` The confidence level.

`n` The number of groups in the study.

`dfe` The error degrees of freedom, based on the total study sample.

`Weibull Density(x, shape, <scale=1>, <threshold=0>)`

Description

Returns the probability density function (pdf) evaluated at *x* of the Weibull distribution. The pdf is parameterized as follows:

$$f(x) = \frac{\beta}{\alpha}\left(\frac{x-\theta}{\alpha}\right)^{\beta-1}\exp\left[-\left(\frac{x-\theta}{\alpha}\right)^{\beta}\right]$$

Arguments

`x` A quantile the pdf is evaluated at. *x* must be greater than *threshold*.

`shape` Shape parameter β, which must be greater than 0.

`scale` Optional scale parameter α, which must be greater than 0. The default is 1.

`threshold` Optional threshold parameter θ. The default is 0.

Weibull Distribution(x, shape, <scale=1>, <threshold=0>)

Description

Returns the cumulative distribution function (cdf) at *x* of the Weibull distribution. The cdf is parameterized as follows:

$$F(x) = 1 - \exp\left[-\left(\frac{x-\theta}{\alpha}\right)^{\beta}\right]$$

Arguments

`x` A quantile at which the pdf is evaluated. *x* must be greater than *threshold*.

`shape` Shape parameter β, which must be greater than 0.

`scale` Optional scale parameter α, which must be greater than 0. The default is 1.

`threshold` Optional threshold parameter θ. The default is 0.

Weibull Quantile(p, shape, <scale=1>, <threshold=0>)

Description

Returns the p^{th} quantile from the Weibull distribution with the parameters given. The quantile function is calculated as follows:

$$F^{-1}(p) = \alpha[\ln(1-p)]^{\frac{1}{\beta}} + \theta$$

Arguments

`p` The probability of the quantile desired. *p* must be between 0 and 1.

`shape` Shape parameter β, which must be greater than 0.

`scale` Optional scale parameter α, which must be greater than 0. The default is 1.

`threshold` Optional threshold parameter θ. The default is 0.

Programming Functions

As Boolean(x)

Description

Evaluates a JSL expression and returns a JSL Boolean value for use with JSON data.

Example

```
x = 45;
b = As Boolean( x > 2 );
Show( b );
   b = true;
```

As Column(name)

As Column(dt, name)

:name

dt:name

Description

This scoping operator forces *name* to be evaluated as a data table column in the current data table (or the table given by the optional data table reference argument, *dt*) rather than as a global variable.

Arguments

`name` Variable name.

`dt` The data table reference

Notes

`:name` refers to a column name in the current data table. You can also specify which data table to refer to by use `dt:name`.

As Constant(expr)

Description

Evaluates an expression once to create a value that does not change after it is computed.

Returns

The result of the evaluation.

Argument

`expr` Any JSL expression.

Notes

A few platforms that can save prediction columns to a data table use `As Constant()`. The function is wrapped around the part of the formula that is constant across all rows. The argument is evaluated for the first row and then the result is used without re-evaluation for subsequent rows.

`As Global(name)`

`::name`

Description

This scoping operator forces *name* to be evaluated as a global variable rather than as a data table column.

Arguments

`name` Variable name.

`As List(matrix)`

See "As List(matrix)".

`As Name(string)`

Description

Evaluates argument as a quoted string and changes it into a name.

Returns

A name.

`As Namespace(name)`

Description

Accesses the specified namespace. An error is thrown if no such namespace exists.

Returns

The namespace.

Arguments

`name` Unquoted name of a defined namespace.

`As Scoped(namespace, variable)`

`namespace:variable`

Description

Accesses the specified *variable* within the specified *namespace*.

Returns

The value of the variable, or an error the scoped variable is not found.

Arguments

`namespace` The name of a defined namespace.

`variable` A variable defined within *`namespace`*.

`Associative Array({key, value}, ...)`

`Associative Array(keys, values)`

Description

Creates an associative array (also known as a dictionary or hash map).

Returns

An associative array object.

Arguments

Either list of key-value pairs; or a list, matrix, or data table column that contains keys followed by a list, matrix, or data table column, respectively, that contains the corresponding values.

`Class Exists(class)`

Description

Returns a value indicating whether a class definition represented by the class reference is a defined class.

Returns

0 or 1.

Argument

`class` A quoted string representation of the name of a defined class or reference to an instantiated class object.

`Clear Globals(<name>, <name>, ...)`

Description

Clears the values for all global symbols. Symbols in any scope other than global are not affected. If one or more names are specified, only those global symbols are cleared.

Returns

Null.

Optional Arguments

`name` Any global variable name(s).

See Also

"Clear Symbols(<name>, <name>, ...)"

Clear Log()

Description

Empties the log.

Clear Symbols(<name>, <name>, ...)

Description

Clear the values for all symbols in any and all scopes. If one or more names are specified, only those symbols are cleared.

Returns

Null.

Optional Arguments

name Any global variable name(s).

See Also

"Clear Globals(<name>, <name>, ...)"

Close Log()

Description

Closes the log.

Define Class("class name", <Base Class("base class name", <"base class name", ...>),> <Show(All(Boolean)) | Show(<Members(Boolean),> <Methods(Boolean),> <Functions(Boolean)>),> <Assignment Statements>)

Description

Defines a new class object.

Example

```
Define Class(
   "aa",
   _init_ = Method( {} ); x = 1; m1 = Method( {a, b}, a * b )
);
```

Delete Classes(<Force(Boolean)>, < <class>, ...>)

Description

Deletes all currently defined classes.

Optional Arguments

Force(Boolean) Deletes the class or classes even if they are in use.

`class` Specifies the classes to delete. You can specify more than one class. This argument can be a quoted string representation of the name of a defined class or a reference to an instantiated class object.

`Delete Globals(<name>, <name>, ...)`

Description

Deletes all global symbols, except global symbols that are locked. Symbols in any scope other than global are not affected. If one or more names are specified, only those global symbols are cleared.

Optional Arguments

`name` Any global variable name(s).

See Also

"Delete Symbols(<name>, <name>, ...)"

`Delete Namespaces(<Force(Boolean expression)>, < <namespace reference>, ...>)`

`Delete(<Force(Boolean expression)>, < <namespace reference>, ...>)`

Description

Deletes all currently defined namespaces or one or more specific namespaces.

Optional Arguments

`Force(Boolean expression)` Deletes the namespace even if it's in use.

`namespace reference` Specifies the namespaces to delete. You can specify more than one namespace reference.

Notes

- When you delete a namespace that contains locked namespaces, an error appears in the log. Use the `Force()` argument to delete the locked namespaces.
- With no arguments, `Delete Namespaces()` ignores locked namespaces.

`Delete Symbols(<name>, <name>, ...)`

Description

Deletes all symbols in any and all scopes. If one or more names are specified, only those symbols are deleted.

Optional Arguments

`name` Any global variable name(s).

See Also

"Delete Globals(<name>, <name>, ...)"

Eval(expr)

Description

Evaluates `expr`, and then evaluates the result of `expr` (unquoting).

Returns

The result of the evaluation.

Argument

`expr` Any JSL expression.

Eval Insert(*string*, <startDel>, <endDel>, < <<Use Locale(1) >)

Description

Allows for multiple substitutions.

Returns

The result.

Arguments

`string` A quoted string with embedded expressions.

`startDel` Optional starting delimiter. The default value is ^.

`endDel` optional ending delimited. The default value is the starting delimiter.

`Use Locale(1)` Optional argument that preserves locale-specific numeric formatting.

Eval Insert Into(*string*, <startDel>, <endDel>)

Description

Allows for multiple substitutions in place. The same operation as in `Eval Insert` is performed, and the result is placed into the quoted *`string`*.

Returns

The result.

Arguments

`string` A quoted string variable that contains a string with embedded expressions.

`startDel` Optional starting delimiter. The default value is ^.

`endDel` optional ending delimited. The default value is the starting delimiter.

Eval List

See "Eval List({list})".

`Exit(<NoSave>)`

`Quit(<NoSave>)`

Description

Exits JMP.

Returns

Void.

Arguments

`NoSave` Optional, named command; exits JMP without prompting to save any open files. This command is *not* case-sensitive, and spaces are optional.

`First(expr, <expr>, ...)`

Description

Evaluates all expressions provided as arguments.

Returns

Only the result of the first evaluated expression.

Arguments

`expr` Any valid JSL expression.

`Function({arguments}, <{local variables}>, <Return(<expr>)>, script)`

Description

Stores the body *`script`* with *`arguments`* as local variables.

Returns

The function as defined. If the `Return()` argument is specified, the expression is returned.

When called later, it returns the result of the *`script`* given the specified *`arguments`*.

Arguments

`{arguments}` A list of arguments to pass into the function. You can specify some arguments as optional or required.

`{local variables}` A list of variables that are local to the function. You can declare local variables in three ways:

```
{var1, var2}
{var1=0, var1="a string"}
{Default Local}
```

The last option declares that all unscoped variables used in the function are local to the function.

`Return(expr)` This optional argument returns an expression from an user defined function. If a null expression is used, a period, ".", is returned.

`script` Any valid JSL script.

Get Class Names(< <class>, ...>)

Description

Gets a set of names to all classes or the set of specific class references.

Arguments

`class` A quoted string representation of the name of a defined class or a reference to an instantiated class object.

Returns

A list of class names as determined by the arguments to the function.

Get Classes(< <class>, ...>)

Description

Gets a set of references to all classes or the set of specific class references.

Arguments

`class` A quoted string representation of the name of a defined class or a reference to an instantiated class object.

Returns

A list of class references as determined by the arguments to the function.

Get Environment Variable("variable")

Description

Retrieves the value of an operating system environment variable.

Returns

A quoted string that contains the value of the specified environment variable. If the specified variable is not found, an empty string is returned.

Arguments

`"variable"` A quoted string that contains the name of an environment variable.

Notes

On macOS, environment variable names are case-sensitive. On Windows, the names are case-insensitive.

Get Locale Setting()

Gets a local setting such as a decimal setting.

Get Log(<n>)

Description

Returns a list of lines from the log.

Returns

A list of quoted strings. Each string contains one line from the log.

Argument

n Optional, integer. If no argument is specified, all the lines are returned. If a positive number is specified, the first *n* lines are returned. If a negative number is specified, the last *n* lines are returned. If *n*=0, no lines are returned (an empty list). If the log is empty, an empty list is returned.

Get Namespace Names(< <namespace reference>,...>)

Description

Returns a list of the names of all currently defined namespaces.

Example

```
nsaa = New Namespace(
   "aa",
   {
      x = 1
   }
);
nsbb = New Namespace(
   "bb",
   {
      y = 1
   }
);
lns = Get Namespace Names();
Show( lns );
nsaa << Delete;
nsbb << Delete;
```

Get Namespaces(< <namespace reference>,...>)

Description

Returns a list of currently defined namespaces.

Example

```
nsaa = New Namespace(
   "aa",
   {
      x = 1
   }
);
nsbb = New Namespace(
   "bb",
   {
```

```
        y = 1
    }
);
lns = Get Namespaces();
```

Include("pathname", <named arguments>)

Description

Opens the script file identified by the quoted string *pathname*, parses the script in it, and executes it.

Returns

Whatever the included script returns. If you use the <<Parse Only option, Include returns the contents of the script.

Named Arguments

<<Parse Only Parses the script but does not execute the script.

<<New Context Causes the included script to be run its own unique namespace. When the parent and included scripts use the global namespace, include <<Names Default to Here along with <<New Context.

<<Allow Include File Recursion Lets the included script include itself.

Notes

If a trailing space is included in the path name, the space is ignored on Windows. On macOS, the script fails.

Include File List()

Description

Returns a list of files that are included at the point of execution.

Is Class(class)

Description

Returns a value that indicates whether the class reference is a class object.

Argument

A class reference to an instantiated class object.

Returns

Returns a zero or a 1.

Is Log Open()

Description

Returns result if log window is open.

Length

See "Length(string)".

List

See "List(a, b, c, ...)".

Local({name=value, ...}, script)

Description

Resolves *names* to local expressions.

Local Here(expression)

Description

Creates a local Here namespace block. Use this function to prevent name collisions when multiple scripts are executed from the same root namespace (for example, when a script executes two button scripts that have the same variables). The argument can be any valid JSL expression.

Lock Namespaces(<*string*>,|< {*string*}, ...>)

Description

Locks all variables or specified named variables in this namespace and prevents variables from being added, changed, or removed.

Example

```
ns = New Namespace(
   "aaa"
);
ns << Lock Namespaces;
Try( ns << Delete Namespaces, Show( exception_msg ) );
Delete Namespaces();
Try( Delete Namespaces( "aaa" ), Show( exception_msg ) );
```

Lock Globals(name1, name2, ...)

Description

Locks one or more global variables to prevent it or them from being changed.

Lock Symbols(<name>, <name>, ...)

Description

Locks the specified symbols, which prevents them from being modified or cleared. If no symbols are provided, all global symbols are locked. If no symbols are provided and the script has the *Names Default To Here* mode turned on, then all local symbols are locked.

Log Capture(expr)

Description

Evaluates the expr, captures the output that would normally be sent to the log, and instead returns it.

Returns

A quoted string that contains the log output.

Argument

Any valid JSL expression.

Notes

No output appears in the log.

Method({arg1 = val1, ...}, script)

Description

Creates a method within a class. Note that methods use local scoping for all variables that are not explicitly scoped, with the exception of class member variables.

Arguments

`{ arg1 = val1, … }` The set of expected arguments and optional initialization expressions to be passed to the method when called.

`script` Any valid JSL script.

N Items

See "N Items(source)".

Names Default To Here(Boolean)

Description

Determines where unresolved names are stored, either as a global or local (if *Boolean* is 0) or in the Here scope (if *Boolean* is 1).

`Namespace(name)`

Description

Returns a reference to the named namespace (*name*).

Argument

`Name` A namespace name quoted string or a reference to a namespace.

`Namespace Exists(name)`

Description

Returns 1 if a namespace with the specified *name* exists; otherwise, returns 0.

`New Namespace(<"name">, <{expr, ...}>)`

Description

Creates a new namespace with the specified name. If a name is not provided, an anonymous name is provided.

Returns

A reference to the namespace.

Arguments

`name` An optional, quoted string that contains the name of the new namespace.

`{list of expressions}` An optional list of expressions within the namespace.

`Open Log(<Boolean>)`

Description

Opens the log. Include the Boolean argument to make the window active, even if it is already open.

```
New Object("class name"(constructor arguments))
New Object(class name(constructor arguments))
New Object(class reference(constructor arguments))
```

Description

Creates an instance object of a class.

Arguments

`"class name"` Name of the class to be instantiated.

`class name` Unquoted name of the class to be instantiated.

`class reference` Reference to an existing class object that will be used to instantiate a new object of the same class.

`constructor arguments` Set of arguments to be passed to the _init_ constructor.

Example

```
Define Class(
  "complex",
  real = 0; imag = 0;
  _init_ = Method( {a, b}, real = a; imag = b; );
  Add = Method( {y}, complex( real + y:real, imag + y:imag ) );
  Sub = Method( {y}, complex( real - y:real, imag - y:imag ) );
  Mul = Method( {y},
    complex( real * y:real - imag * y:imag, imag * y:real + real * y:imag )
  );
  Div = Method( {y},
    t = complex( 0, 0 );
    mag2 = y:Magsq();
    t:real = real * y:real + imag * y:imag;
    t:imag = imag * y:real + real * y:imag;
    t:real = t:real / mag2;
    t:imag = t:imag / mag2;
    t;
  );
  Magsq = Method( {}, real * real + imag * imag );
  Mag = Method( {}, Sqrt( real * real + imag * imag ) );
  To String = Method( {}, Char( real ) || " + " || Char( imag ) || "i" )
);
cl = New Object( complex( 1, 2 ) );
```

Parameter({name=value, ...}, model expression)

Description

Defines formula parameters for models for the Nonlinear platform.

Parse(*string*)

Description

Converts a quoted character string into a JSL expression.

Print(expr, expr, ...)

Description

Prints the values of the specified *expressions* to the log.

Quit()

See “Exit(<NoSave>)”.

`Recurse(function)`

Description

Makes a recursive call of the defining *function*.

`Save Log(pathname)`

Description

Writes the contents of the log to the specified file location.

`Send(obj, message)`

`obj << message`

Description

Sends a *message* to a platform *object*.

`Set Environment Variable( "variable", <"value">)`

Description

Sets the environment variable to the value specified. If the "value" argument is missing or is an empty quoted string, then the environment variable is deleted from the JMP process environment variable table.

`Show(expr, expr, ...)`

Description

Prints the name and value of each *expression* to the log.

`Show Classes(< <class>,...>)`

Description

Shows the contents of user-defined classes in the log. You can specify more than one class. If you do not specify an argument, all user-defined classes are shown in the log.

Example

```
Define Class(
   "aa",
   _init_ = Method( {} ); x = 1; m1 = Method( {a, b}, a * b )
);
Define Class(
   "bb",
   _init_ = Method( {} ); y = 1; m2 = Method( {a, b}, a / b )
);
Show Classes();
   // Class aa
```

```
_init_ = Method( {} );
m1 = Method( {a, b}, a * b );
x = 1;

// Class bb

_init_ = Method( {} );
m2 = Method( {a, b}, a / b );
y = 1;
```

`Show Globals()`

Description

Shows the values for all global symbols. Symbols in any scope other than global are not shown.

See Also

"Show Symbols()"

`Show Namespaces(< <namespace reference>,...>)`

Description

Shows the contents of all user-defined namespaces, both named and anonymous. You can specify zero or more namespaces.

`Show Symbols()`

Description

Shows the values for all symbols in any and all scopes.

See Also

"Show Globals()"

`Sort List`

See "Sort List({list}|expr)".

`Sort List Into`

See "Sort List Into({list}|expr)".

`Throw("text")`

Description

Returns a `Throw`. If you include *`text`*, throwing stores *`text`* in a global *`exception_msg`*. If *`text`* begins with "!" and is inside a `Try()` expression, throwing creates an error message

about where the exception was caught. "!" stops the script even if the Throw() is caught by the second argument of Try().

`Try(expr1, expr2)`

Description

Evaluates *expr1*. If the evaluation returns a Throw, execution stops, and nothing is returned. *expr2* is evaluated next to return the result.

Examples

```
Try( Sqrt( "s" ), "invalid" );
   "invalid"

Try( Sqrt( "s" ), exception_msg );
   {"Cannot convert argument to a number [or matrix]"(1, 2, "Sqrt",
      Sqrt/*###*/("s"))}
```

Notes

Expr2 can be a quoted character string or the global exception message (exception_msg) that contains more information about the error returned.

`Type(x)`

Description

Returns a quoted string that names the type of object *x* is. The list of possible types is: Unknown, List, DisplayBox, Picture, Column, TableVar, Table, Empty, Pattern, Date, Integer, Number, String, Name, Matrix, RowState, Expression, Associative Array, BLOB.

`Unlock Symbols(name1, name2, ...)`

`Unlock Globals(name1, name2, ...)`

Description

Unlocks the specified symbols that were locked with a Lock Symbols() or Lock Globals() command.

`Wait(n)`

Description

Pauses *n* seconds before continuing the script. The default setting is 3 seconds. Specifying Wait(0) enables one cycle of message processing. For example, you can use this function to allow a button press in the UI. The shortest duration that actually allows JMP to pause is n = 0.01. The longest duration you can specify without prompting a JMP dialog is n = 60*60*4.

Notes

You can use `Wait(n)` if you want something to stay on the screen long enough to see it, if you need a platform to finish launching before scripting it, or if you need to press buttons in the UI while the script runs.

`Watch(all | name1, ...)`

Description

Shows variables from global, here, and local namespaces and their values in a window. If "all" is provided as the argument, all globals are placed into the window.

Notes

- New globals are not added to the window list.
- Watching associative arrays that have been modified using messages is not supported.

`Wild()`

Description

Only used with `Extract Expr()` for expression matching to denote a wildcard position that matches any expression.

`Wild List()`

Description

Only used with `Extract Expr()` for expression matching to denote a series of wildcard arguments that match any expression.

`Write("text")`

Description

Prints *`text`* to the log without surrounding quotation marks.

Python Integration Functions

`Python Connect(<Echo(Boolean),> <Path(path),> <Use Python Version(`*`string`*`),> <Python System Path(list)>)`

Description

Initializes the Python integration interfaces and returns an active Python integration interface connection as a scriptable object.

Returns

A Python scriptable object.

Optional Named Arguments

`Echo(Boolean)` Global argument. Prints the Python source lines to the JMP log. The default value is true.

`Path` Specifies the path to the Python DLL or shared library.

`Use Python Version(`*`string`*`)` Specifies which version of Python should be used for JMP-to-Python processing.

`Python System Path` Specifies a JSL list of paths that define a Python sys path set on macOS.

Python Control(<named arguments>)

Description

Sends control operations to signal Python with external events, such as source line echoing.

Returns

Returns 0 if the call succeeded and 1 if an error occurred.

Optional Named Arguments

`Interactive(Boolean)` Enables interactive mode in the Python matplotlib package. Determines whether the graphics window is released or closed when graphics rendering is complete.

`Echo(Boolean)` Global argument. Prints the Python source lines to the JMP log. The default value is true.

Python Disconnect

Description

Terminates the Python interfaces.

Python Execute({list of inputs}, {list of outputs}, Python_Code, named_arguments)

Description

Submits Python code to the active global Python integration interface connection given a list of inputs. On completion, returns a list of outputs.

Returns

Returns 0 if successful and 1otherwise.

Positional Arguments

`{list of inputs}` A list of JMP variable names to be sent to Python as inputs.

`{list of outputs}` A list of JMP variable names to be retrieved from Python as outputs.

`Python_Code` The Python code to submit.

Named Arguments

See "Python Submit(Python_Code, <named_arguments>)".

Example

This example initiates the Python connection, sends a character variable, a numeric variable, and a set of matrices to Python. Python is then instructed to perform a set of matrix operations on the sent matrices. The `Python Execute()` function then get the set of matrices created by the matrix operations and gets the values of the character and numeric variables that was originally sent. Upon completion of the data retrieval, the Python connection is closed.

```
Python Init();
a = "abcdef";
d = 3.141;
v = [9 8 7, 6 5 4, 3 2 1];
m = [1 2 3, 4 5 6, 7 8 9];
ml = Python Execute(
   {v, m, a, d},
   {x1, x2, y1, y2, z1, z2, a, d},
   "\[
import numpy as np
x1 = np.multiply(v, m) # matrix product
print('x1=', x1)
x2 = np.divide(v, m) # matrix division
print('x2=', x2)
y1 = np.dot(v, m) # dot product of v and m
print('y1=', y1)
y2 = np.dot(m, v) # dot product of m and v
print('y2=', y2)
z1 = np.inner(v, m) # inner product of v and m
print('z1=', z1)
z2 = np.inner(m, v) # inner product of m and v
print('z2=', z2)
]\"
);
Show( v, m, ml, x1, x2, y1, y2, z1, z2, a, d );
Python Term();
   x1= [[  9.  16.  21.]
    [ 24.  25.  24.]
    [ 21.  16.   9.]]
   x2= [[ 9.          4.          2.33333333]
    [ 1.5         1.          0.66666667]
    [ 0.42857143  0.25        0.11111111]]
   ...
```

Python Get(name)

Description

Gets a named variable from Python to JMP.

Returns

Returns the value of the named variable.

Argument

name The name of the Python variable to be sent to JMP. The argument can represent any of the following Python data types: numeric, quoted string, matrix, list, or data frame.

Example

```
Python Init(); // initiate the Python connection

qbx = "The right stuff";

// send the qbx variable and sample data table "Animals.jmp" to Python
Python Send( qbx );

dt = Open( "$SAMPLE_DATA/Animals.jmp" );
Python Send( dt );
Close( dt, nosave );

// get the Python variable qbx and place it into a JMP variable qbx
qbx = Python Get( qbx );

/* get the Python variable dt and place it into a JMP data table
referenced by df */
df = Python Get( dt );

Python Term();

Show( qbx );
df << New Data View;
Wait( 10 );
Close( df, nosave );
Python Term();
   qbx = "The right stuff";
   0
```

Python Get Graphics(format)

Description

Gets the last graphics object written to the Python graph display window in the specified graphics format. The graphics object can be returned in several different graphic formats.

Returns

Returns a JMP picture object.

Argument

`format` The format that the Python graph display window contents are to be converted to. Valid formats are `png`, `bmp`, `jpeg`, `jpg`, `tiff`, `tif`, and `gif`.

`Python Get Version`

Description

Returns the version number of Python being used with the JMP Python interfaces.

`Python Init(<Echo(Boolean),> <Path(path),> <Use Python Version(`*`string`*`),> <Python System Path({list})>`

Description

Initializes the Python integration interfaces.

Returns

Returns 0 if operation is successful and 1 if not successful.

Optional Named Arguments

`Echo(Boolean)` Global argument. Prints the Python source lines to the JMP log. The default value is true.

`Path` Specifies the path to the Python DLL or shared library.

`Use Python Version(`*`string`*`)` Specifies which version of Python should be used for JMP-to-Python processing.

`Python System Path` Specifies a JSL list of paths that define a Python sys path set on macOS.

`Python Is Connected`

Description

Determines whether a Python integration interface connection is currently connected to Python.

Returns

Returns 1 if connected and 0 otherwise.

`Python JMP Name to Python Name(name)`

Description

Maps a JMP variable name to its corresponding Python variable name using Python variable name naming rules.

Returns

A quoted string, the mapped Python name.

Argument

`name` The name of the JMP variable to be sent to Python.

`Python Send(name)`

Description

Sends a named variable from JMP to Python.

Returns

Returns 0 if successful.

Argument

`name` The name of the JMP variable to be sent to Python.

`Python Send File(filename, <, Python Name(name)>)`

Description

Sends a data file to Python. The filename argument is a quoted string that specifies a pathname to the file to be sent to Python.

`Python Submit(Python_Code, <named_arguments>)`

Description

Submits Python code to the active global Python integration interface connection.

Returns

Returns 0 if successful and non-zero otherwise.

Named Arguments

`Python_Code` The Python code to submit. Statements can be a quoted string value or a list of string values.

`Expand(Boolean)` (Optional) Performs an `Eval Insert()` on the Python code before submission.

`Echo(Boolean)` (Optional) Prints the Python source lines to the JMP log.

Example

```
Python Init(); // initiate the Python connection
commands =
"
friends = ['john', 'pat', 'gary', 'michael']
print(friends)
for i, name in enumerate(friends):
    print( \!"iteration {iteration} is {name}\!".format(iteration=i,
   name=name))
";
```

```
Python Submit( commands );
Python Term();
   ['john', 'pat', 'gary', 'michael']
   iteration 0 is john
   iteration 1 is pat
   iteration 2 is gary
   iteration 3 is michael

   0
```

Python Submit File(path)

Description

Submits statements to Python using the file specified in the path name.

Argument

`path` The path to the file that contains the Python source lines to be executed.

Python Term

Description

Terminates the currently active Python integration interface.

Returns

Returns 0 if successful and 1 otherwise.

R Integration Functions

R Connect(<named_arguments>)

Description

Returns the current R connection object. If there is no connection to R, it intializes the R integration interfaces and returns an active R integration interface connection as a scriptable object.

Returns

R scriptable object.

Arguments

`Echo(Boolean)` (Optional) Sends all source lines to the JMP log. This option is global. The default value is `true`.

R Control(Interrupt|Async(Boolean)|Echo(Boolean))

Description

Changes the control options for R.

R Execute({ list of inputs }, { list of outputs }, "rCode", <named_arguments>)

Description

Submit the specified R code to the active global R connection given a list of inputs. On completion, the outputs are returned into the specified list.

Returns

0 if successful; nonzero otherwise.

Arguments

`{ list of inputs }` A list of JMP variable names to be sent to R as inputs.

`{ list of outputs }` A list of JMP variable names to contain the outputs returned from R.

`rCode` A quoted string that contains the R code to submit.

`Expand(Boolean)` An optional, Boolean, named argument. Performs an `Eval Insert()` on the R code before submitting to R.

`Echo(Boolean)` An optional, Boolean, named argument. Sends all source lines to the JMP log. This option is global. The default value is `true`.

Example

Send the JMP variables x and y to R, execute the R statement z <- x * y, and then get the R variable z and return it to JMP.

```
x = [1 2 3];
y = [4 5 6];
rc = R Execute( {x, y}, {z}, "z <- x * y" );
```

R Get(variable_name)

Description

Gets the named variable from R to JMP.

Returns

The value of the named variable.

Argument

`name` Required. The name of an R variable whose value to return to JMP.

Example

Assume that a matrix named qbx and a data frame named df are present in your R connection.

```
// get the R variable qbx and placed it into a JMP variable qbx
qbx = R Get( qbx );

// get the R variable df and placed it into a JMP data table referenced by df
df = R Get( df );
```

R Get Graphics("format")

Description

Gets the last graphics object written to the R graph display window in the specified format.

Returns

A JMP picture object.

Argument

`format` Required. Specifies the graphics format to be used. Valid formats are `png`, `bmp`, `jpeg`, `jpg`, `tiff`, `tif`, and `gif`.

R Get Version

Description

Returns the version number of R being used with JMP R interfaces.

R Init(named_arguments)

Description

Initializes the R session.

Returns

0 if the initialization is successful; any nonzero value otherwise.

Argument

`Echo(Boolean)` (Optional) Sends all source lines to the JMP log. This option is global. The default value is `true`.

R Is Connected()

Description

Determines whether a connection to R exists.

Returns

1 if connected; 0 otherwise.

Arguments

None.

R JMP Name to R Name(name)

Description

Maps the specified JMP variable name to the corresponding R variable name using R naming rules. Requires an active connection to R.

Argument

`name` The name of a JMP variable to be sent to R.

Returns

A quoted string that contains the R name.

`R Send( name, <R Name( name )>)`

Description

Sends named variables from JMP to R.

Returns

0 if the send is successful; any nonzero value otherwise.

Arguments

`name` required. The name of a JMP variable to be sent to R.

`R Name(name)` (Optional) You can give the variable that you send to R a different name. For example

```
R Send(Here:x, R Name("localx"))
```

For data tables only:

`Selected(Boolean)` optional, named, Boolean. Send only selected rows from the referenced data table to R.

`Excluded(Boolean)` optional, named, Boolean. Send only excluded rows from the referenced data table to R.

`Labeled(Boolean)` optional, named, Boolean. Send only labeled rows from the referenced data table to R.

`Hidden(Boolean)` optional, named, Boolean. Send only hidden rows from the referenced data table to R.

`Colored(Boolean)` optional, named, Boolean. Send only colored rows from the referenced data table to R.

`Markered(Boolean)` optional, named, Boolean. Send only markered rows from the referenced data table to R.

`Row States(Boolean, <named arguments>)` optional, named. Includes a Boolean argument and optional named arguments. Send row state information from the referenced data table to R by adding an additional data column named "RowState". Multiple row states are created by adding together individual settings. Here are the individual values:

- Selected = 1
- Excluded = 2
- Hidden = 4
- Labeled = 8
- Colored = 16

– Markered = 32

Here are the named arguments for the `Row States()` argument:

`Colors(Boolean)` optional, named, Boolean. Sends row colors. Adds additional data column named "RowStateColor".

`Markers(Boolean)` optional, named, Boolean. Sends row markers. Adds additional data column named "RowStateMarker".

Examples

Create a matrix, assign it to X, and send the matrix to R:

```
X = [1 2 3];
rc = R Send( X );
```

Open a data table, assign a reference to it (*dt*), and send the data table, along with its current row states, to R:

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
rc = R Send( dt, Row States(1) );
```

R Send File("pathname", <R Name("name")>)

Description

Sends the specified data file from JMP to R.

Returns

0 if the send is successful; any nonzero value otherwise.

Arguments

`pathname` required. A quoted string that contains a pathname for a file.

`R Name(name)` (Optional) You can give the data file that you send to R a different name.

R Submit("rCode", <named_arguments>)

Description

Submits the specified R code to the active global R connection.

Returns

0 if successful; nonzero otherwise.

Arguments

`rCode` A required, quoted string that contains the R code to submit.

`Expand(Boolean)` An optional, Boolean, named argument. Performs an `Eval Insert()` on the R code before submitting to R.

`Echo(Boolean)` An optional, Boolean, named argument. Sends all source lines to the JMP log. This option is global. The default value is `true`.

`Async(Boolean)` An optional, Boolean, named argument. If set to true (1), the submit can be canceled either by pressing the ESCAPE key, or by using this message to an R connection: `rconn<<Control( Interrupt( 1 ) )`. False (0) is the default value.

Example

```
rc = R Submit("\[
  x <- rnorm(5)
  print(x)
  y <- rnorm(5)
  print(y)
  z = plot(x, y)
]\" );
```

R Submit File("pathname")

Description

Submits statements to R using a file pointed in the specified *pathname*.

Returns

0 if successful; nonzero otherwise.

Argument

Pathname A quoted string that contains the pathname to the file that contains the R code to be executed.

R Term()

Description

Terminates the currently active R integration interface.

Returns

Returns 0 if the termination is successful and -1 otherwise.

Arguments

None

Random Functions

Col Shuffle(<By var,...>)

Description

Creates a random ordering of the row numbers of the current data table when used in a column formula.

Note: This function is generally used in a column formula.

Returns

A random integer between 1 and the number of rows in the current data table.

Argument

`By var` (Optional) A By variable enables you to randomly order the rows within the groups of the By variable values.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << New Column( "Shuffle", Numeric, Continuous, Set Formula( Col Shuffle() ) );
```

This example creates a column formula that shuffles the order of the row numbers (1 to 40) each time the formula is evaluated. Each number appears only once.

Make Validation Formula(rates, <<Stratification Columns(cols), <<Grouped Columns(cols), <<Cutpoint Column(col))

Description

Generates a validation column. This function is primarily used by the Make Validation Column utility.

Arguments

`rates` Vector of three rates that specify the training, validation, and test rates, respectively.

`<<Stratification Columns` Assigns one or more stratification columns.

`<<Grouped Columns` Assigns one or more grouping columns.

`<<Cutpoint Column` Assigns a numeric cutpoint column.

Predictive and Specialized Modeling

Random Beta(alpha, beta, <theta=0>, <sigma=1>)

Description

Returns a random number from a beta distribution with two shape parameters, *alpha* and *beta*, and optional parameters *theta* and *sigma*.

Arguments

`alpha, beta` Shape parameters α and β, which must both be greater than 0.

`theta` Optional threshold parameter θ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Beta Binomial(n, p, <delta=0>)

Description

Returns a random number from a beta binomial distribution for *n* trials with probability *p* and overdispersion parameter *delta*.

Arguments

n The number of trials, which must be greater than or equal to 2. If the specified *n* is not an integer, the non-integer part is truncated.

p The probability of success for each trial, which must be between 0 and 1.

delta The overdispersion parameter δ, which must be between Maximum[-*p*/(*n*-*p*-1), -(1-*p*)/(*n*-2+*p*)] and 1. The default is 0.

`Random Binomial(n, p)`

Description

Returns a random number from a binomial distribution with *n* trials and probability *p* of the event of interest occurring.

Arguments

p The probability of success for each trial, which must be between 0 and 1.

n The number of trials.

`Random Category(probA, resultA, probB, resultB, <..., ...,> resultElse)`

Description

Returns one of the specified result expressions at random, chosen from pairs of probability and result expressions. A random uniform number is generated and compared to the *prob* arguments to determine which *result* argument is returned.

Arguments

probA Numeric value between 0 and 1 that represents the probability of the corresponding result expression being returned.

resultA Expression that corresponds to *probA*.

resultElse Expression that is returned if no previous result expression has been returned.

`Random Cauchy()`

Description

Returns a random number from a Cauchy distribution with a median of zero.

`Random ChiSquare(df, <nc=0>)`

Description

Returns a random number from a chi-square distribution with given *df* (degrees of freedom) and optional noncentrality parameter.

Arguments

df The degrees of freedom *n*, which must be greater than 0.

nc Optional noncentrality parameter λ, which must be nonnegative. The default is 0.

Random Exp()

Description

Returns a random number from an exponential distribution with scale parameter equal to 1. Equivalent to the negative log of `Random Uniform`.

Random F(dfnum, dfden, <noncentral=0>)

Description

Returns a random number from an F distribution with a given *dfnum*, *dfden*, and optional noncentrality parameter.

Arguments

`dfnum` The degrees of freedom, v_1, of the chi-square distribution in the numerator of the *F*-distribution. *dfnum* must be greater than 0.

`dfden` The degrees of freedom, v_2, of the chi-square distribution in the denominator of the *F*-distribution. *dfden* must be greater than 0.

`noncentral` Optional noncentrality parameter λ, which must be nonnegative. The default is 0.

Random Frechet(<mu=0>, <sigma=1>)

Description

Returns a random number from a Fréchet distribution with the location *mu* and scale *sigma*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Gamma(alpha, <scale=1>)

Description

Returns a random numbers from a gamma distribution for given *alpha* and optional *scale*.

Arguments

`alpha` The shape parameter α, which must be greater than 0.

`scale` Optional scale parameter β, which must be greater than 0. The default is 1.

Random Gamma Poisson(lambda, <sigma=1>)

Description

Returns a random number from a gamma Poisson distribution with parameters *lambda* and *sigma*.

Arguments

`lambda` The shape parameter λ, which much be greater than 0.

`sigma` Optional overdispersion parameter σ, which must be greater than or equal to 1. The default is 1. When the overdispersion parameter is 1, the distribution reduces to a Poisson(λ) distribution.

`Random GenGamma(<mu=0>, <sigma=1>, <lambda=0>)`

Description

Returns a random number from an extended generalized gamma distribution with parameters *mu*, *sigma*, and *lambda*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

`lambda` Optional shape parameter λ. The default is 0.

`Random Geometric(p)`

Description

Returns a random number from the geometric distribution with probability *p* that a specific event occurs at any one trial.

`Random GLog(mu, sigma, lambda)`

Description

Returns a random number from a generalized logarithmic distribution with parameters *mu*, *sigma*, and *lambda*.

Arguments

`mu` The location parameter μ.

`sigma` The scale parameter σ, which must be greater than 0.

`lambda` A shape parameter λ, which must be greater than 0.

`Random Index(n, k)`

Description

Returns a *k* by 1 matrix of random integers between 1 and *n* with no duplicates.

`Random Integer(n)`

`Random Integer(k, n)`

Description

Returns a random integer from 1 to *n* or from *k* to *n*.

Random Johnson Sb(gamma, delta, theta, sigma)

Description

Returns a random number from a Johnson Sb distribution with parameters *gamma*, *delta*, *theta*, and *sigma*.

Arguments

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Random Johnson Sl(gamma, delta, theta, <sigma=1>)

Description

Returns a random number from a Johnson Sl distribution with parameters *gamma*, *delta*, *theta*, and optional *sigma*.

Arguments

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Optional parameter σ that indicates if the distribution is skewed positively or negatively. sigma must be equal to either +1 (skewed positively) or -1 (skewed negatively). The default is +1.

Random Johnson Su(gamma, delta, theta, sigma)

Description

Returns a random number from a Johnson Su distribution with parameters *gamma*, *delta*, *theta*, and *sigma*.

Arguments

gamma Shape parameter γ.

delta Shape parameter δ, which must be greater than 0.

theta Location parameter θ.

sigma Scale parameter σ, which must be greater than 0.

Random LEV(<mu=0>, <sigma=1>)

Description

Returns a random number from an LEV distribution with the location *mu* and scale *sigma*.

Arguments

mu Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random LogGenGamma(<mu=0>, <sigma=1>, <lambda=0>)

Description

Returns a random number from a log generalized gamma distribution with parameters *mu*, *sigma*, and *lambda*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

`lambda` Optional shape parameter λ. The default is 0.

Random Logistic(<mu=0>, <sigma=1>)

Description

Returns a random number from a logistic distribution with location *mu* and scale *sigma*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Loglogistic(<mu=0>, <sigma=1>)

Description

Returns a random number from a loglogistic distribution with location *mu* and scale *sigma*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Lognormal(<mu=0>, <sigma=1>)

Description

Returns a random number from a lognormal distribution with location *mu* and scale *sigma*.

Arguments

`mu` Optional location parameter μ. The default is 0.

`sigma` Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Multivariate Normal(mean, covar, <nrows=1>)

Description

Returns a random vector from a multivariate normal distribution with mean vector *mean* and covariance matrix *covar*. To generate multiple vectors, specify an integer greater than 1 for the *nrows* argument. When *nrows* is greater than 1, the return value is a matrix. The number of columns in the random vector or matrix is equal to the number of rows in the *covar* argument.

Arguments

mean Mean vector for the multivariate normal distribution.

covar Covariance matrix for the multivariate normal distribution. This matrix must be a symmetric square matrix that contains the same number of columns as the mean vector.

nrows Optional argument that specifies the number of random vectors returned. The default number of rows is 1.

Random Negative Binomial(n, p)

Description

Returns a random number from a negative binomial distribution for *n* successes with probability of success *p*.

Random Normal(<mu=0>, <sigma=1>)

Description

Returns a random number from a normal distribution with mean *mu* and standard deviation *sigma*.

Arguments

mu Optional location parameter μ. The default is 0.

sigma Optional scale parameter σ, which must be greater than 0. The default is 1.

Random Normal Mixture(meanvec, sdvec, probabvec)

Description

Returns a random number from a normal mixture distribution with the specified arguments.

Arguments

meanvec A vector that contains group means.

sdvec A vector that contains the group standard deviations.

probvec A vector that contains the group probabilities.

Random Poisson(lambda)

Description

Returns a random number from a Poisson distribution with shape parameter *lambda*.

Arguments

lambda The shape parameter λ, which much be greater than 0.

Random Reset(seed)

Description

Restarts the random number sequences with *seed*. To ensure reproducible results, specify a positive integer less than or equal to 4,294,967,295 for the random seed.

Notes

You can use the **Random Reset** function to set a random seed that enables you to reproduce a set of random numbers. The random number algorithms in JMP are sometimes refined, and therefore the results might not be consistent across different versions of JMP.

Random Seed State(<seed state>)

Description

Retrieves or restores the random seed state to or from a BLOB object.

Random SEV(<mu=0>, <sigma=1>)

Description

Returns a random number from an SEV distribution with the specified location *mu* and scale *sigma*.

Arguments

mu Optional location parameter μ. The default is 0.

sigma Optional scale parameter σ, which must be greater than 0. The default is 1.

Random SHASH(gamma, delta, theta, sigma)

Description

Returns a random number from a sinh-arcsinh (SHASH) distribution with parameters *gamma*, *delta*, *theta*, and *sigma*.

Arguments

gamma The shape parameter γ.

delta The shape parameter δ, which must be greater than 0.

theta The location parameter θ.

sigma The scale parameter σ, which must be greater than 0.

Random Shuffle(matrix)

Description

Returns the matrix with the elements shuffled into a random order.

Random t(df, <noncentral=0>)

Description

Returns a random number from a t distribution with the specified *df* (degrees of freedom). The noncentrality argument may be negative or positive. The default value of *noncentral* is 0.

Random Triangular(min, mode, max)

Random Triangular(mode, max)

Random Triangular(mode)

Description

Generates a random number from a triangular distribution between 0 and 1 with the *mode* that you specify. The triangular distribution is typically used for populations that have a small number of data.

Arguments

`min` Specifies the lower limit of the triangular distribution. The default value is 0.

`mode` Specifies the mode of the triangular distribution.

`max` Species the upper limit of the triangular distribution. The default value is 1.

Notes

If you specify only the mode, the minimum value is 0, and the maximum value is 1. If you specify the mode and maximum value, the minimum value is 0 by default.

Random Uniform()

Random Uniform(x)

Random Uniform(min, max)

Description

Generates a random number from a uniform distribution between 0 and 1. `Random Uniform(x)` generates a number between 0 and x. `Random Uniform (min, max)` generates a number between *min* and *max*. The result is an approximately even distribution.

Random Weibull(shape, <scale=1>)

Description

Returns a random number from a Weibull distribution with parameters *shape* and optional *scale*.

Arguments

`shape` Shape parameter β, which must be greater than 0.

`scale` Optional scale parameter α, which must be greater than 0. The default is 1.

Resample Freq(<rate=1, <column>>)

Description

Generates a frequency count for sampling with replacement. If no arguments are specified, the function generates a 100% resample.

Note: This function is generally used in a column formula.

Arguments

`rate` (Optional) Specifies the rate of resampling. The default value is 1. A negative value specifies that fractional frequencies are allowed.

`column` (Optional) If you specify *column*, you must also specify *rate*. The sample size is calculated by the rate multiplied by the sum of the specified column. If rate is negative, then the sample size is the negative of the rate multiplied by the sum of the specified column. If you do not specify a column, the generated frequencies sum to the number of rows.

Example

To ensure that the numbers in the frequency column match each time you run the script, use `As Constant()`. `As Constant()` evaluates an expression to create a constant value that does not change after it has been computed.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dc = dt << New Column( "column",
   Formula(
      As Constant(
         Random Reset( 123 );
         0;
      ) + Resample Freq()
   )
);
dc << Eval Formula;
```

Notes

– A typical use of this function generates a column with many 1s, some 0s, some 2s, and so forth, corresponding to which rows were randomly assigned any of *n* randomly selected rows.

– A typical use of this with an *existing* frequency column produces a new frequency column whose values are similar to the old frequency column (have the same expected value); however, the values vary somewhat due to random selection at the rates corresponding to the old frequency column.

Row Functions

As Table(matrix, <matrix 2, ...>, < <<invisible >, < <<private >, < <<Column Names({list}) >)

Description

Creates a new data tables from the *matrix*.

Returns

The new data table.

Argument

matrix Any matrix.

<<invisible Creates an invisible data table that hides the table from view but lists it in the JMP Home Window and Window menu.

<<private Hides the table completely. Creating a private data table speeds the process of getting to the data; it does not save the computer from allocating the memory necessary to hold the data table data.

<<Column Names(list) The list specified column names for the data. The argument is a list of quoted column names.

Col Stored Value(<dt>, col, <row>)

Description

Returns the data values stored in the column and disregards values assigned through column properties (such as Missing Value Codes).

Arguments

dt Optional reference to a data table. If this value is not supplied, the current data table is used.

col Name of the column.

row (Optional) Row name or number. If this value is not specified, the current row is used.

Example

Suppose that the Missing Value Codes column property is assigned to the x1 column to treat "999" as a missing value. Another column includes a formula that calculates the

mean. To use the value "999" instead of a missing value to calculate the mean, use `Col Stored Value()` in the formula:

```
Mean( Col Stored Value( :x1 ), :x2, :x3 )
```

`Column(<dt>, "name", "formatted")`

`Column(<dt>, n)`

Description

Gets a reference to the data table column.

Arguments

`dt` Optional reference to a data table. If this is not supplied, the current data table is used.

`name` A quoted string that is the name of the column.

`formatted` A quoted string that returns the formatted string of the cell value.

`n` The column number.

`Column Name(n)`

Description

Determines the name of the column specified by number.

Returns

The name of the n^{th} column as an expression (not a quoted string).

Argument

`n` The number of a column.

`Count(from, to, step, times)`

Description

Used for column formulas. Creates row by row the values beginning with the *from* value and ending with the *to* value. The number of *steps* specifies the number of values in the list between and including the *from* and *to* values. Each value determined by the first three arguments of the count function occurs consecutively the number of *times* that you specify. When the *to* value is reached, `count` starts over at the *from* value. If the *from* and *to* arguments are data table column names, `count` takes the values from the first row only. Values in subsequent rows are ignored.

Returns

The last value.

Arguments

`from` Number, column reference, or expression. `Count` starts counting with this value.

`to` Number, column reference, or expression. `Count` stops counting with this value.

`step` Number or expression. Specifies the number of steps to use to count between *from* and *to*, inclusive.

`times` Number or expression. Specifies the number of times each value is repeated before the next step.

Examples

/* the rows in the column named *colname* are filled with the series 0, 3, 6, 0,

```
... until all rows are filled */
For Each Row(:colname[row()]=count(0, 6, 3, 1))

/* the rows in the column named colname are filled with the series 0, 0, 3,
   3, 6, 6, 0, ... until all rows are filled */
For Each Row(:colname[row()]=count(0, 6, 3, 2))
```

Notes

`Count()` is dependent on `Row()`, and is therefore mainly useful in column formulas.

Current Data Table(<dt>)

Description

Without an argument, gets the current (topmost) data table. With an argument, sets the current data table.

Returns

Reference to the current data table.

Argument

`dt` Optional name of or reference to a data table.

Notes

Private tables cannot be made current with `Current Data Table()`.

Data Table(n)

Data Table("name")

Get Data Table(<project(title|index|box|window),> name|index)

Description

Gets reference to the *n*th open data table or the table with the given *name* in a global variable.

Returns

Reference to the specified data table.

Argument

`n` Number of a data table.

`name` Quoted string, name of a data table.

Dif(col, n)

Description

Calculates the difference of the value of the column *col* in the current row and the value *n* rows previous to the current row.

Returns

The difference.

Arguments

`col` A column name (for example, `:age`).

`n` A number.

Dim(<dt|matrix>)

Description

Returns a row vector with the dimensions of the current data table, a specified data table, or a matrix. The dimensions are the number of rows and the number of columns and are listed in that order.

Arguments

`dt` A data table.

`matrix` A matrix.

Notes

If no argument is specified, the dimensions of the current data table are returned.

Get Data Table List(<Project(title|index|box|window>)

Description

Returns a list of all open data tables.

Notes

Use `Project(0)` to specify no project when running the expression in a project.

Lag(col, n)

Description

Returns for each row the value of the column *n* rows previous.

N Row(dt); NRow(matrix)

N Rows(dt); NRows(matrix)

Description

Returns the number of rows in the data table given by *dt* or in the *matrix.*

N Table()

Description

Returns the number of open data tables. Private tables are not included.

New Column("name", <"data type">, <"modeling type">, <Width(n)>, Format("format", width, precision), <Formula()>, <Set Values>, <Like(column reference)>, <actions>)

Description

Adds a new column named *"name"* after the last column in *dt*. Unless otherwise specified, columns are numeric, continuous, and 12 characters wide.

Returns

A column reference.

Notes

Can also be used as a message: `dt<<New Column`.

The `Like()` argument copies the data type, modeling type, format, formula, and other properties from the reference column into the new column.

See Also

"dt<<New Column(`name`, `<data type>`, `<modeling type>`, <Format(`format`, `width`)>, <Formula()>, <Set Values(`{..., ..., }`)>, <Set Property`(properties)`>)"

New Table("name", <visibility("invisible" | "private" | "visible")>, <actions>)

Description

Creates a new data table with the specified *name*.

Arguments

`name` A quoted string that contains the name of the new table.

`visibility` Optional quoted keyword. `invisible` hides the data table from view but lists it in the JMP Home Window and Window menu. `private` hides the table completely. `visible` shows the data table. "`visible`" is the default value.

Note: Creating a private data table speeds the process of getting to the data; it does not save the computer from allocating the memory necessary to hold the data table data.

`actions` Optional argument that can define the new table.

`Row()`

`Row() = y`

Description

Returns or sets the current row number. No argument is expected.

`Sequence(from, to, <step size>, <repeat times>)`

Description

Produces an arithmetic sequence of numbers across the rows in a data table. The *step size* and *repeat times* arguments are optional, and the default value for both is 1.

`Subscribe to Data Table List(<subscriber name|"">, <OnOpen(function)|OnClose(function)|OnRename(function)>)`

Description

Subscribes to the data table list. You will be notified when a new data table has been added, closed, or renamed.

`Subscript(a, b, c)`

`list[i]`

`matrix[b, c]`

Description

Subscripts for lists extract the *i*th item from the *list*, or the *b*th row and the *c*th column from a *matrix*.

`Suppress Formula Eval(Boolean)`

Description

Turns off automatic calculation of formulas for all data tables.

`Unsubscribe to Data Table List(<subscriber name>, <"OnOpen"|"OnClose"|"All">)`

Description

Removes a subscription to the data table list that has been added through `Subscribe to Data Table List()`.

Row State Functions

As Row State(i)

Description

Converts *i* into a row state value.

Returns

A row state from the *i* given.

Argument

i an integer

Color Of(rowstate)

Description

Returns or sets the color index.

Returns

The color index of *rowstate*.

Argument

rowstate a row state argument

Example

Set the color of the fifth row to red.

```
Color Of( Rowstate( 5) ) = 3
```

Color State(i)

Description

Returns a row state with the color index of *i*.

Returns

A row state.

Argument

i index for a JMP color

Combine States(rowstate, rowstate, ...)

Description

Generates a row state combination from two or more row state arguments.

Returns

A single numeric representation of the combined row states.

Arguments

`rowstate` Two or more row states.

`Excluded(rowstate)`

Description

Returns or sets an excluded index.

Returns

The excluded attribute, 0 or 1.

Argument

`rowstate` One or more row states.

`Excluded State(num)`

Description

Returns a row state for exclusion from the *num* given.

`Hidden(rowstate)`

Description

Returns or sets the hidden index.

`Hidden State(num)`

Description

Returns a row state for hiding from the *num* given.

`Hue State(num)`

Description

Returns a hue state from the *num* given.

`Labeled(rowstate)`

Description

Returns or sets the labeled index.

`Labeled State(num)`

Description

Returns a labeled state from the *num* given.

Marker Of(rowstate)

Description

Returns or sets the marker index of a row state.

Marker State(num)

Description

Returns a marker state from the *num* given.

Row State(<dt,> <n>)

Description

Returns the row state changed from the initial condition of the active row or the *n*th row.

Arguments

`dt` Optional positional argument: a reference to a data table. If this argument is not in the form of an assignment, then it is considered a data table expression.

`n` The row number.

Example

The following example creates the data table references and then returns the row state of row 1 in Big Class.jmp:

```
dt1 = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt2 = Open( "$SAMPLE_DATA/San Francisco Crime.jmp" );
Row State( dt1, 1 );
```

Selected(rowstate)

Description

Returns or sets the selected index.

Selected State(num)

Description

Returns a selected state from the *num* given.

Shade State(num)

Description

The Shade State function assigns 5 shade levels to a color or hue.

SAS Integration Functions

As C Expr(x)

Description

Returns a C programming language representation of the expression.

Returns

A quoted string.

As JavaScript Expr(x)

Description

Returns a JavaScript representation of the expression.

Returns

A quoted string.

As JSON Expr(x)

Description

Returns a JSON (JavaScript Object Notation) representation of the expression.

Returns

A quoted string.

As Python Expr(x)

Description

Returns a Python representation of the expression.

Returns

A quoted string.

As SAS Expr(x)

Description

Converts an expression to a version that is more suitable for SAS DATA step. The code must be wrapped in a PROC DS2 call. Use `Expr(...)` for literal expressions. Use `NameExpr(name)` for expressions stored in a variable. Otherwise, the expression returns the expression to convert.

Returns

A quoted string.

Current Metadata Connection()

Description

Returns the active SAS metadata server connection, if any, as a scriptable object.

Current SAS Connection()

Description

Gets the active global SAS server connection, if any, as a scriptable object.

Get SAS Version Preference()

Description

Returns the SAS version selected in the SAS Integration page of the Preferences as a quoted string.

JMP6 SAS Compatibility Mode(Boolean)

Description

Setting this to 1 (true) causes SAS operators to operate in a mode compatible with JMP 6 capabilities.

Meta Connect(<"machine", port>, <"authDomain">, <"username">, <"password">, <named arguments>)

Meta Connect(<Profile("profile name")>, <Password("password")>, <named arguments>)

Meta Connect(<Environment("environment name")>, <named arguments>)

Description

Connects to a SAS Metadata Server. If no arguments are specified, an empty connection window appears. If some arguments are specified, a window partially filled in with the argument values appears. If all arguments are specified, the connection is made and no window appears.

Returns

1 if connection is successful, 0 if not.

Arguments

machine (Optional) A quoted string that contains the DNS name of the machine.

port Required if machine is specified. A quoted string or integer that contains the port on which the metadata server listens.

authDomain (Optional) A quoted string that contains the authentication domain for the credentials supplied. Not necessary unless *username* and *password* are included.

username (Optional) A quoted string that contains the user name for the connection.

`password` (Optional) A quoted string that contains the password for the connection.

Optional Named Arguments

`Profile("profile_name")` A quoted string that contains the name of the metadata server connection profile from which connection information should be retrieved.

`Environment("environment_name")` A quoted string that contains the name of the WIP environment from which connection information should be retrieved.

`Password("password")` A quoted string that contains the password for the specified profile name.

`CheckPreferenceOnly(0|1)` If specified, `Meta Connect` returns the status of the **I want to connect to a SAS Metadata Server** option in the SAS Integration page of JMP Preferences. If that box is checked, **Meta Connect** returns 1; if not, 0.

`Repository(`*`string`*`)` Takes a quoted string that contains the name of the repository to which to connect.

`ProfileLookup(0|1)` If *`machine`* and *`port`* are specified rather than a profile name, and `ProfileLookup` is specified, an attempt is made to find a metadata server connection profile with a machine name and port matching those provided. If one is found, other connection information (such as authentication domain, user name, and password) is obtained from that profile.

`Prompt(Always|Never|IfNeeded)` Takes one of the keywords `Always` (always prompt before attempting to connect), `Never` (never prompt, just fail), or `IfNeeded` (the default; prompt if connection with the given arguments fails).

`SASVersion("<version number>" <,Strict>)` Attempts to change the SAS version preference to the specified value before making the metadata server connection. If the SAS version is already locked to a different version than the one specified, the SASVersion argument will fail. By default, if the SAS version cannot be set, JMP will try the metadata server connection. However, if you include `Strict` as the second argument, the inability to change the SAS version will be treated as an error, and JSL processing will stop. If you do not include `Strict`, the SAS version argument is treated as a hint and will set the version preference if it can. JMP will still try to connect if the version cannot be set. The order you put these arguments in can make a difference. The attempt to change the SAS Version is made immediately when that argument is encountered. That can affect the validity of other arguments, particularly for `MetaConnect`. The only valid value for `SASVersion` is "9.4". Note: Using the `SASVersion` argument has the same effect as changing the SAS Server Version on the SAS Integration Preferences page.

Notes

- If no arguments are included and if no profile is saved, the Connect to SAS Metadata Server window appears.
- If you connect to a physical workspace server, there is no metadata server involved, so metadata security is never applied. You must connect to a SAS Metadata Server and

then connect to a logical workspace server. Then metadata security is enforced on the metadata-defined libraries you access.

`Meta Create Profile("profile", <named arguments>)`

Description

Creates a metadata server connection profile and adds it to the current user's set of saved metadata server connection profiles.

Returns

1 if *profile* was successfully created, otherwise 0.

Arguments

`profile` A quoted string that contains the name of the created profile. If a profile by the given name already exists, `MetaCreateProfile` fails unless *Replace* is specified.

Optional Named Arguments

`HostName("name")` A quoted string that contains the name of the host computer running the SAS Metadata Server that this profile will connect to.

`Port(n)` The port number (*n*) that the SAS Metadata Server is listening for connections on.

`AuthenticationDomain("domain") | AuthDomain("domain")` A quoted string that sets the authentication domain to use for the connection.

`Description("desc") | Desc("desc")` A quoted string that sets a description for this profile.

`Password("password")` A quoted string that contains the password to store in this profile.

`Replace(0|1)` If *name* matches a profile that already exists, `Replace` must be specified for the existing profile to be replaced by the one provided. The default value is False (0).

`UserName("username")` A quoted string that contains the user name that this profile uses to connect to the SAS Metadata Server.

`UseSingleSignOn(0|1)` If specified, this profile attempts to use Single Sign-On (currently also known as Integrated Windows Authentication) to connect to the SAS Metadata Server. This option is valid only for connecting to SAS 9.4 Metadata Servers. If `UseSingleSignOn` is True(1), *UserName* and *Password* cannot be specified. The default value is False (0).

`Meta Delete Profile("name")`

Description

Deletes the named metadata server connection profile from the current user's set of saved metadata server connection profiles

Returns

1 if profile was successfully deleted, otherwise 0.

Argument

`name` A quoted string that contains the name of the profile to delete.

`Meta Disconnect()`

Description

Disconnect the current SAS Metadata Server connection, if any.

Returns

Void.

`Meta Get Environments()`

Description

Returns a list of the SAS Environments that are defined in the SAS Environments definition file, which is configured in the SAS Preferences.

`Meta Get Repositories()`

Description

Gets a list of the repositories available on the current SAS Metadata Server connection.

Returns

A list of repository names as quoted strings.

`Meta Get Servers()`

Description

Get a list of the SAS Servers that are registered in the SAS Metadata Repository to which the session is currently connected.

Returns

A list of server names as quoted strings.

`Meta Get Stored Process("path")`

Description

Get a stored process object from the currently connected SAS Metadata Repository.

Returns

Stored Process scriptable object.

Arguments

`path` Quoted string that is the path to the stored process in metadata, starting at the BIP Tree.

Meta Is Connected()

Description

Determines whether a current connection to a SAS Metadata Server exists.

Returns

1 if a connection exists; 0 otherwise.

Arguments

None.

Meta Set Repository("repositoryName")

Description

Set the SAS Metadata Repository to use for metadata searches.

Returns

1 if setting the repository was successful, 0 otherwise.

Arguments

repositoryName A quoted string that contains the name of the repository to make current.

SAS Assign Lib Refs("libref", "path", <"engine">, <"engine options">)

Description

Assign a SAS libref on the active global SAS server connection.

Returns

1 if successful, 0 otherwise.

Arguments

libref A quoted string that contains a library reference (8-character maximum) to assign.

path A quoted string that contains the full path on the SAS server to the library being assigned.

engine Optional, quoted string that contains the engine for the SAS server to use when accessing members of this library.

engine options Optional, quoted string that contains the options needed for the engine being used.

SAS Connect(<"machine_name">, <"port">, <named_arguments>)

Description

Connect to a local, remote, or logical SAS server.

Returns

SAS Server scriptable object.

Arguments

`machine_name` (Optional) A quoted string that can contain a physical machine name or the name of a metadata-defined (logical) server. In the first case, the port must be provided. In the second case, a port must *not* be provided. If neither *`name`* nor *`port`* are included, and JMP is running on Windows, a connection to SAS on the local machine (via COM) is attempted, and all named arguments are ignored.

`port` (Optional) A quoted string or integer. If *`name`* is a physical machine name, this is the port on that machine to connect to. If *`name`* is a metadata-defined (logical) server, *`port`* must *not* be included.

Optional Named Arguments

`UserName("name")` A quoted string that contains the user name for the connection.

`Password("password")` A quoted string that contains the password for the connection.

`ReplaceGlobalConnection(0|1)` A Boolean. The default value is `True`. If `True`, and a successful SAS server connection is made, this connection replaces the active SAS connection that becomes the target of other global SAS JSL function calls. If `False`, the global SAS connection is not changed, and the returned SASServer scriptable object should be used to send messages to this server connection.

`ShowDialog(0|1)` A Boolean. The default value is `False`. If `True`, other arguments (except `ReplaceGlobalConnection`) are ignored and the SAS Server Connection window appears. This provides the JSL programmer a way to open the SAS Connect window.

`Prompt(Always|Never|IfNeeded)` A keyword. `Always` means always prompt before attempting to connect. `Never` means never prompt even if the connection attempt fails (just fail and send an error message to the log), and `IfNeeded` (the default value) means prompt if the attempt to connect with the given arguments fails (or is not possible with the information given).

`ConnectLibraries(0|1)` A Boolean. Defaults to the SAS Integration Preference setting governing whether to automatically connect metadata-defined libraries when connecting to a SAS server. If true, all metadata-defined libraries are connected at SAS server connection time, which can be time-consuming. If false, metadata-defined libraries are not connected. To connect specific libraries later, use the `SAS Connect Libref` global function or `Connect Libref` message to a SAS server object.

`SASVersion("<version number>" <,Strict>)` Attempts to change the SAS version preference to the specified value before making the metadata server connection. If the SAS version is already locked to a different version than the one specified, the SASVersion argument will fail. By default, if the SAS version cannot be set, the metadata server connection will still be tried. However, if you include `Strict` as the second argument, the inability to change the SAS version will be treated as an error and JSL processing will stop. If you do not include `Strict`, the SAS version argument is treated as a hint and will set the version preference if it can, but if it cannot it will still try to connect. The order you put these arguments in can make a difference. The attempt to

change the SAS Version is made immediately when that argument is encountered. That can affect the validity of other arguments, particularly for MetaConnect. The only valid value for SASVersion is "9.4". Note: Using the SASVersion argument has the same effect as changing the SAS Server Version on the SAS Integration Preferences page.

Example

```
// prompt for login credentials
Meta Connect( "dev.company.com", 28561 );

sas = SAS Connect( "SASApp" );

// dump some libraries and data sets to the JMP log
Show( sas << Get Librefs() );
Show( sas << Get Data Sets( "Chocolate Enterprises 2017" ));

sas << Import Data( "Chocolate Enterprises 2017", "Products" );

/* The preceding lines produce the following text in the JMP log and import
   the data set: */
sas << Get Librefs:{"BOOKS", "CHOCOENT", "EGSAMP", "TEST", "TST92", "MAPS",
   "SASDATA", "SASHELP", "SASUSER", "SGFDATA", "TEMPDATA", "V6LIB", "WORK"}
sas << Get Data Sets("Chocolate Enterprises 2008"):{"CHOC_DATA",
   "CHOC_SURVEY", "CUSTOMERS", "ORDER_DETAIL", "PRODUCTS", "SALES_ANALYSIS",
   "SALES_SUMMARY"}
```

Get Librefs returns the short library names, not the longer logical names. However, you can use either one, and metadata security will still be applied for metadata-defined libraries.

SAS Connect Lib Refs(libref)

Description

Connects a SAS libref on the active SAS server connection.

Returns

1 if successful and 0 otherwise.

SAS Deassign Lib Refs("libref")

Description

De-assign a SAS libref on the active global SAS server connection.

Returns

1 if successful; 0 otherwise.

Arguments

libref A quoted string that contains the library reference to de-assign.

SAS Disconnect()

Description

Disconnect the active global SAS connection, if any.

Returns

1 if a SAS connection exists and was successfully disconnected, 0 otherwise.

Arguments

None.

SAS Export Data(dt, "library", "dataset", <named_arguments>)

Description

Exports a JMP data table to a SAS data set in a library on the active global SAS server connection.

Returns

`1` if the data table was exported successfully; `0` otherwise.

Arguments

`dt` data table or a reference to a data table.

`"library"` the library to which to export the data table.

`"dataset"` the name of the new SAS data set.

Optional Named Arguments

`Columns(list)|Columns(col1, col2, ...)` A list of columns or a comma-separated list of columns.

`Password("password")` A quoted string that contains the password to serve as the READ, WRITE, and ALTER password for the exported SAS data set. If the exported data set is replacing an existing data set with an ALTER password, this password is used as the ALTER password for overwriting the data set. If *`Password`* is specified, values for *`ReadPassword`*, *`WritePassword`*, and *`AlterPassword`* are ignored.

`ReadPassword("password")` A quoted string that contains the password to serve as the READ password for the exported SAS data set.

`WritePassword("password")` A quoted string that contains the password to serve as the WRITE password for the exported SAS data set.

`AlterPassword("password")` A quoted string that contains the password to serve as the ALTER password for the exported SAS data set. If the exported data set is replacing an existing data set with an ALTER password, this password is used as the ALTER password for overwriting the data set.

`PreserveSASColumnNames(0|1)` A Boolean. If true *and* the JMP data table originally came from SAS, the original SAS column names are used in the exported SAS data set. The default value is `False`.

`PreserveSASFormats(0|1)` A Boolean. If true *and* the JMP data table originally came from SAS, the original SAS formats and informats are applied to the columns in the exported SAS data set. The default value is `True`.

`ReplaceExisting(0|1)` A Boolean. If true, an existing SAS data set with the specified name in the specified library is replaced by the exported SAS data set. If false, a data set with the specified name already exists in the specified library; the export is stopped. The default value is false.

`SaveJMPMetadata(0|1)` Includes SAS 9.4 Extended Attributed to store JMP metadata (such as table script and column properties). Default is 0 (disabled).

`HonorExcludedRows(0|1)` A Boolean. If true, any rows in the JMP data table that are marked as excluded are not exported. The default value is false.

Notes

Information about the export is sent to the log.

SAS Get Data Sets("libref")

Description

Returns a list of the data sets defined in a SAS library.

Returns

A list of quoted strings.

Arguments

`libref` A quoted string that contains the SAS libref or friendly library name associated with the library for which the list of defined SAS data sets is returned.

SAS Get File("source", "dest", "encoding")

Description

Get a file from the active global SAS server connection. JMP creates a FILENAME statement (with an encoding, if specified) and uses it to read the file on the SAS server.

Returns

1 if successful, 0 otherwise.

Arguments

`source` A quoted string that contains the full path of file on the server to be downloaded to the client machine.

`dest` A quoted string that contains the full path on the client machine for where to put the copy of the file downloaded from the server.

`encoding` A quoted string that contains the encoding used in the file (for example, `"utf-8"`). The server must support the specified encoding.

SAS Get File Names("fileref")

Description

Get a list of filenames found in the given fileref on the active global SAS server connection.

Returns

A list of quoted strings.

Arguments

`fileref` A quoted string that contains the name of the fileref from which to retrieve filenames.

SAS Get File Names In Path("path")

Description

Get a list of filenames found in the given path on the active global SAS server connection.

Returns

A list of quoted strings.

Arguments

`path` A quoted string that contains the directory path on the server from which to retrieve filenames.

SAS Get File Refs()

Description

Get a list of the currently defined SAS filerefs on the active global SAS server connection.

Returns

List of two lists. The first list is a list of quoted strings of fileref names. The second is a corresponding list of strings of physical names.

SAS Get Lib Refs(<named arguments>)

Description

Get a list of the currently defined SAS librefs on the current global SAS server connection.

Returns

A list of quoted strings.

Named Arguments

`Friendly Names(0|1)` Optional, Boolean. If `True`, then for any libraries that have friendly names (metadata-defined libraries), the friendly name is returned rather than the 8-character libref.

SAS Get Log()

Description

Retrieve the SAS Log from the active global SAS server connection.

Returns

A quoted string.

SAS Get Output()

Description

Retrieve the listing output from the last submission of SAS code to the current global SAS server connection.

Returns

A quoted string.

SAS Get Results()

Description

Retrieve the results of the previous SAS Submit as a scriptable object, which allows significant flexibility in what to do with the results.

Returns

A SAS Results Scriptable object.

SAS Get Var Names(*string*, <"dataset">, <password("password")>)

Description

Retrieves the variable names contained in the specified data set on the current global SAS server connection.

Returns

A list of quoted strings.

Arguments

`string` A quoted string that contains one of the following:

- The name of the SAS Library containing the SAS data set to be imported. In that case, the *`dataset`* name argument is required.
- The full member name of the SAS data set to be imported, in the form "libname.membername".
- The SAS Folders tree path to a logical SAS data table to be imported. This option requires a connection to a SAS 9.4 Metadata Server.

`dataset` (Optional) A quoted string that contains the name of the data set from which to retrieve variable names.

`password("password")` (Optional) A quoted string that contains the read password for the data set. If this is not provided and the data set has a read password, the user is prompted to enter it.

`SAS Import Data(`*`string`*`, <"dataset">, <named arguments>)`

Description

Import a SAS data set from the active global SAS server connection into a JMP table.

Returns

JMP Data Table object.

Arguments

`string` A quoted string that contains one of the following:

- The name of the SAS Library containing the SAS data set to be imported. In that case, the *"dataset"* name argument is required. The name can be a friendly metadata library name or a SAS 8-character library name.
- The full member name of the SAS data set to be imported, in the form "libname.membername".
- The SAS Folders tree path to a logical SAS data table to be imported. This option requires a connection to a SAS 9.4 Metadata Server.

`dataset` (Optional) A quoted string that contains the name of the data set.

Optional Named Arguments

`Columns("list")|Columns(col1, col2, ...)` A quoted string list or multiple strings that contain the names of columns to include in the import.

`ConvertCustomFormats(0|1)` The default value is True (1). If True and custom formats are found in the SAS data set being imported, an attempt is made to convert the SAS custom formats to JMP value labels for those columns.

`Invisible(0|1)` The default value is False (0). If true, the JMP data table is hidden from view. The data table appears only in the JMP Home Window and the Window menu. Hidden data tables remain in memory until they are explicitly closed, reducing the amount of memory that is available to JMP. To explicitly close the hidden data table, call `Close(dt)`, where *dt* is the data table reference returned by SASImportData.

`Where("filter")` A quoted string that contains the filter to use when importing data, as in `Where("salary<50000")`.

`Password("password")` A quoted string that contains the read password for the data set. If this is not provided and the data set has a read password, the user is prompted to enter it.

`UseLabelsForVarNames(0|1)` If `True`, the labels from the SAS data set become the column names in the resulting JMP table. If `False`, the variable names from the SAS data set become the column names in the JMP table. The default value is `False`.

`RestoreJMPMetadata(0|1)` Includes SAS 9.4 Extended Attributed to store JMP metadata. Default is 0 (disabled).

`Sample(named arguments)` optional, named. Allows a random sample of the SAS data set to be imported into JMP. If both Where and Sample are specified, the WHERE clause is used to filter the SAS data set first, and then a random sample of the resulting rows is taken based on the arguments supplied to Sample. Note that Sample uses PROC SURVEYSELECT on the SAS server, which is available only if the SAS/STAT package is licensed and installed on that server. The documentation for PROC SURVEYSELECT might be helpful in understanding how sampling is performed. By default (if no arguments are supplied), a 5% simple random sample is taken. Here are the available arguments (all optional):

- `Simple | Unrestricted`: If `Simple` is specified, sampling is performed without replacement. If `Unrestricted` is specified, sampling is performed with replacement. These two options are mutually exclusive and only one can be specified.
- `SampleSize(int) | N(int)`: Total number of rows for the sample, or number of rows per strata level for stratified sampling
- `SampleRate(number) | Rate(number) | Percent(number)`: Specifies the sampling rate. For stratified sampling, the rate is applied to each strata level. Note that the supplied value is assumed to be a percentage, so `SampleRate(3.5)` means a 3.5% sampling rate.
- `Strata({col1, col2, ...}) | Strata(col1, col2, ...)`: Perform stratified random sampling using the column names supplied as Strata variables.
- `NMin(int)`: Minimum number of rows (either overall or per strata level for stratified sampling) to return. Only applies to rate-based sampling.
- `NMax(int)`: Maximum number of rows (either overall or per strata level for stratified sampling) to return. Only applies to rate-based sampling.
- `Seed(int)`: Number to use as the seed for sampling. Useful for replicating the same sample. By default, the seed is a random number based on time of day. See PROC SURVEYSELECT documentation.
- `OutputHits(0|1)`: Boolean; the default value is false. When doing Unrestricted sampling, if the same row of the input data set is selected more than once, by default that row still appears only once in the resulting JMP data table, with the NumberHits column indicating the number of times that the row was selected. Setting OutputHits to true causes an input row that is selected multiple times to appear multiple times in the resulting JMP data table.
- `SelectAll(0|1)`: Boolean, the default value is true. If `SelectAll` is true, PROC SURVEYSELECT selects all stratum rows whenever the stratum sample size exceeds the total number of rows in the stratum. If `SelectAll` is false and PROC SURVEYSELECT finds a case where the stratum sample size exceeds the total number

of rows in a given stratum, an error results and sampling fails. `SelectAll` only applies to Simple random sampling.

`SQLTableVariable(0|1)` If `True`, an SQL table variable is created in the resulting JMP table that shows the SQL that was submitted to SAS to obtain the data. If `False`, the SQL table variable is not created. The default value is `True`. If an SQL table variable is created and the data set required a read password, the password is masked.

SAS Import Query("sqlquery", <named arguments>)

Description

Execute the requested SQL query on the current global SAS server connection, importing the results into a JMP data table.

Returns

JMP Data Table object.

Arguments

`sqlquery` A quoted string that contains the SQL query to perform and from which to import the result.

Optional Named Arguments

`ConvertCustomFormats(0|1)` The default value is true. If true and custom formats are found in the SAS data set being imported, an attempt is made to convert the SAS custom formats to JMP value labels for those columns.

`Invisible(0|1)` The default value is false. If true, the JMP data table is hidden from view. The data table appears only in the JMP Home Window and the Window menu. Hidden data tables remain in memory until they are explicitly closed, reducing the amount of memory that is available to JMP. To explicitly close the hidden data table, call `Close(dt)`, where *`dt`* is the data table reference returned by `SAS Import Query`.

`UseLabelsForVarNames(0|1)` The default value is true. If `True`, the labels from the SAS data set become the column names in the resulting JMP table. If `False`, the variable names from the SAS data set become the column names in the JMP table.

`RestoreJMPMetadata(0|1)` Includes SAS 9.4 Extended Attributes to store JMP metadata. Default is 0 (disabled).

`SQLTableVariable(0|1)` The default value is true. If `True`, an SQL table variable is created in the resulting JMP table that shows the SQL that was submitted to SAS to obtain the data. If `False`, the SQL table variable is not created. If an SQL table variable is created and the data set required a read password, the password is masked.

SAS Is Connected()

Description

Discovers whether there is an active global SAS server connection.

Returns

1 if an active global SAS connection exists, 0 otherwise.

`SAS Is Local Server Available()`

Description

Returns True if a local SAS Server is available; otherwise, returns False.

`SAS Load Text File("path")`

Description

Download the file specified in path from the active global SAS server connection and retrieve its contents as a quoted string.

Returns

A quoted string.

Arguments

"path" A quoted string that contains the full path on the server of the file to download and retrieve the contents as a string.

`SAS Name("name")`

`SAS Name({list of names})`

Description

Converts JMP variable names to SAS variable names by changing special characters and blanks to underscores and various other transformations to produce a valid SAS name.

Returns

A quoted string that contains one or more valid SAS names, separated by spaces.

Argument

"name" A quoted string that represents a JMP variable name; or a list of quoted JMP variable names.

`SAS Open For Var Names("path")`

Description

Opens a SAS data set only to obtain the names of its variables, returning those names as a list of quoted strings.

Returns

A list of variable names in the file.

Argument

path A quoted string that is a pathname of a SAS data set.

SAS Send File("source", "dest", "encoding")

Description

Send a file from the client machine to the active global SAS server connection. JMP creates a FILENAME statement (with an encoding, if specified) and uses it to save the file on the SAS server.

Returns

1 if successful, 0 otherwise.

Arguments

`source` A quoted string that contains the full path of the file on the client machine to be uploaded to the server.

`dest` A quoted string that contains the full path on the server that receives the file uploaded from the client machine.

`encoding` A quoted string that contains the encoding used in the file (for example, "utf-8"). The server must support the specified encoding.

SAS Submit("sasCode", <named arguments>)

Description

Submit some SAS code to the active global SAS server connection.

Returns

1 if successful, 0 otherwise.

Arguments

`sasCode` A quoted string that contains the SAS code to submit.

Optional Named Arguments

`Async(0|1)` A Boolean. If True (1), the submit occurs asynchronously (in the background). Use the `Get Submit Status()` message on the SAS Server Scriptable Object to determine the status of the submit. The default value is False (0).

`ConvertCustomFormats(0|1)` A Boolean. When SAS data sets generated by submitted SAS code are imported into JMP after the submit completes (see `Open Output Datasets`), the value of `ConvertCustomFormats` determines whether an attempt is made to convert any custom formats found on columns in the SAS data to JMP value labels. The default value is True (1).

`DeclareMacros(var1, var2, ...)` JSL variable names. Provides a simple way to pass the values of JSL variables to SAS as macro variables. Each JSL variable specified should evaluate to a quoted string or numeric value. Fully qualified JSL variables names, only the variable name is sent to SAS. For example, `namespace:variable_name` becomes `variable_name` in SAS.

`GetSASLog(<Boolean|OnError>, <JMPLog|Window>)` A Boolean. If no arguments are supplied, the SAS Log is retrieved and displayed in the location indicated in SAS Integration Preferences. The first argument to `GetSASLog` can be either a Boolean value

or the keyword OnError. If a Boolean value is supplied, true means display the SAS Log, and false means not to display it. OnError instructs JMP to only show the SAS Log if an error occurred in the submit. The second argument to GetSASLog tells JMP where to display the SAS Log. If JMPLog is specified, the SAS Log is appended to the JMP Log. If Window is specified, the SAS Log is opened in a separate window.

`GraphicsDevice(`*`string`*`) or GDevice(`*`string`*`)` A quoted string that specifies a value for the GDEVICE SAS option to be used for graphics generated by the submitted SAS code. The value must be a valid SAS graphics device. The default value is determined in Preferences.

`Interactive(0|1)` JMP includes the `QUIT` statement in the generated wrapper code. Interactive PROCs work even if JMP is generating the ODS wrapper. On every SUBMIT, specify the argument that is part of an interactive sequence. Otherwise, QUIT will be generated in both the prologue-generated and epilogue-generated code.

`NoOutputWindow` A Boolean. If `True`, the SAS Output window containing the listing output from the submission does not appear. The default value is `False`.

`ODS(0|1)` If true, additional SAS code is submitted causing ODS results to be generated for the submitted SAS code. The default value is determined in Preferences.

`ODSFormat(`*`string`*`)` A quoted string that determines the format of generated ODS results. Valid values are “HTML”, “RTF”, and “PDF”. The default value is determined in Preferences.

`ODSGraphics(0|1)` If true, ODS statistical graphics are generated for the submitted SAS code. Setting `ODSGraphics` to true causes ODS to also be set to true. The default value is determined in Preferences.

`ODSStyle(`*`string`*`)` A quoted string that specifies the ODS Style to use when generating ODS results. *`String`* must be a valid SAS Style. The default value is determined in Preferences.

`ODSStyleSheet(path)` A quoted string that specifies a local CSS style sheet to use when formatting generated ODS results. *`Path`* must be a path to a CSS file valid for the client machine (the machine running JMP). The default value is determined in Preferences.

`OnSubmitComplete(script)` A quoted string that specifies a JSL script that should be run when the submit completes. This is especially useful for asynchronous submits. If script is the name of a defined JSL function, that function is executed, with the SAS Server scriptable object passed as the first argument.

`OpenODSResults(0|1)` If true, ODS results that are generated by the submitted SAS code (due to ODS being true) are automatically opened after the submit completes. The default value is True (1).

`OpenOutputDatasets(<All|None|dataset1, dataset2, ...>)` JMP detects when submitted SAS code creates new SAS data sets. `OpenOutputDatasets` (which can be abbreviated `OutData`) determines what, if anything, is done with those data sets with the SAS Submit completes. If `All` is specified, all data sets generated by the SAS code are imported into JMP when the SAS Submit completes. If `None` is specified, none of the

generated data sets are imported. If there are specific data sets known to be generated by the submitted SAS code that you want to be imported into JMP when the SAS submit completes, you can alternative provide their names, and only the requested data sets are imported. The default value is determined in Preferences.

`Title(`*`string`*`)` A quoted string that specifies the window title to use for the window that displays ODS output from the submit.

`SAS Submit File("filename", <named arguments>)`

Description

Submit a SAS code file to the active global SAS server connection.

Returns

1 if successful; 0 otherwise.

Arguments

`filename` A quoted string that contains the name of file containing SAS code to submit.

Named Arguments

Same as for SAS Submit.

SQL Functions

Note: Database table names that contain the characters `$# -+/%()&|;?` are not supported.

`As SQL Expr(x, <style>)`

Description

Converts an expression to code that you can use in an SQL Select statement. Use `Expr(...)` for literal expressions. Use `NameExpr(name)` for expressions stored in a variable. Otherwise, the expression returns the expression to convert.

Returns

A quoted string that contains the expression converted to valid SQL syntax for use in an SQL Select statement.

```
New SQL Query(Connection
("ODBC:connection_string")|("SAS:connection_string"),
Select(Column("column", "t1")), From(Table("table", <Schema("schema")>,
```

```
<Alias("t1")>)), <Options(JMP 12 Compatible(1)|JMP 13 Compatible(1)|Run on
Open(1)))>
```

```
New SQL
Query(Connection("ODBC:connection_string;")|("SAS:connection_string;"),
Custom("SELECT col1, col2, col3 FROM table;")), <Options(JMP 12
Compatible(1)|JMP 13 Compatible(1)|Run on Open(1))>
```

Creates an SQL Query object for the specified connection, columns, data table, or for the custom SQL query.

Returns

A data table that contains the queried data. The data table includes the quoted SQL query string and table scripts for modifying and updating the query.

Arguments

`Connection` The quoted string for an ODBC or SAS connection.

`Select` The column that you want to select and its alias.

`From` The table that is queried and the optional schema and column alias.

`Custom` An SQL statement that selects columns from the specified table.

`Version` The minimum JMP version required to open the query. If this condition is not met, a message regarding compatibility is written to the log, and the query does not open.

`Options` Boolean. `JMP 12 Compatible` is included in generated scripts when you select the Query Builder preference to create a JMP 12 compatible option or select the corresponding Query Builder red triangle menu option. The option enables JMP 12 users to run a JMP 13 query that might contain compatibility issues. Include `Run on Open(1)` to run the query when opened rather than opening the query in edit mode.

Example

```
New SQL Query(
  Connection(
     "ODBC Connection String..."
  ),
  QueryName( "g6_Movies" ),
  Select( Column( "ItemNo", "t1" ), Column( "LengthMins", "t1" ), Column(
  "Genre", "t1" ) ),
  From( Table( "g6_Movies", Schema( "SQBTest" ), Alias( "t1" ) ) )
) << Run Background( On Run Complete( dt = queryResult ) );

Show( dt );
```

Notes

- Query Builder creates a symbol called `queryResult` in the context of an `On Run Complete()` script. This is a reference to the data table imported by the query. `queryResult` enables you to assign a global variable to the table for later use.
- `New SQL Query()` always closes the ODBC connection after performing the query.

- After you run `New SQL Query()`, the database connection that you make appears in the Database > Open Table window as an available connection. But the connection you see in the Database > Open Table window is due to JMP keeping track of how to connect to that database; no ODBC connection is left open.
- There is not a command to close an ODBC connection established with `New SQL Query()` because `New SQL Query()` always closes the connection after performing the query.
- After you run `New SQL Query()`, the database connection appears in the Database > Open Table window as an available connection. JMP is keeping track of how to connect to that database; the ODBC connection was not left open.

```
Query(<<dt1|Table(dt1, alias1)>, ..., <dtN, aliasN)>>, <private |
invisible>, <scalar>, sqlStatement )
```

Description

Performs a SQL query on selected data tables.

Returns

The result of the query, either a data table or a single value.

Arguments

`dt1, dtN` (Optional) A variable that has been assigned to the data table.

`Table` (Optional) Passes a reference to the data table.

`alias1, aliasN` Specifies the alias of the database table.

`private` (Optional) Avoids showing the resulting data table. Using a private data table speeds the process of getting to the data; it does not save the computer from allocating the memory necessary to hold the data table data.

`invisible` (Optional) Hides the resulting data table from view. The data table appears only in the JMP Home Window and the Window menu. Hidden data tables remain in memory until they are explicitly closed, reducing the amount of memory that is available to JMP. To explicitly close the hidden data table, call `Close(dt)`, where *`dt`* is the data table reference.

`scalar` (Optional) Indicates that the query returns a single value.

`sqlStatement` Required. The SQL statement, most likely a SELECT statement. The statement must be the last argument.

Example

The following example selects all data for students who are older than 14 years of age.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
result = Query( Table( dt, "t1" ), "SELECT * FROM t1 WHERE age > 14;" );
```

See Also

"SQL Functions Available for JMP Queries"

Statistical Functions

Arc Finder(X(col), Y(col), Group(lot, wafer))

Description

Finds arcs in the point data and creates a new column that identifies the arcs.

Example

```
dt = Open( "$SAMPLE_DATA/Wafer Stacked.jmp" );
Arc Finder(
  Group( :Lot, :Wafer ),
  X( :X_Die ),
  Y( :Y_Die ),
  Min Distance( 12 ), // minimum distance among 3 points to seed an arc
  Min Radius( 15 ), // minimum radius of the acceptable arc
  Max Radius( 2000 ), // maximum radius of acceptable arc
  Max Radius Error( 2 ), // how close a point needs to be added
  Min Arc Points( 5 ), // how many points to define an arc
  Number of Searches( 500 ), // how many random probes of data
  Max Number Arcs( 3 ) // number of arcs searched for
);
dt << Color or Mark by Column( :Arc Number );
dt << Graph Builder(
  Size( 1539, 921 ),
  Variables( X( :X_Die ), Y( :Y_Die ), Wrap( :Lot_Wafer Label ), Color( :Arc
  Number ) ),
  Elements( Points( X, Y, Legend( 6 ) ) )
);
```

Notes

- The function is scaled for data that have a range of 30 to 50 units.
- The function is suitable only for data that are subset to the interesting defect points.
- It is not suitable when the density of points is high.

ARIMA Forecast(column, length, model, estimates, from, to)

Description

Determines the forecasted values for the specified rows of the specified column using the specified model and estimates.

Returns

A vector of forecasted values for *column* within the range defined by *from* and *to*.

Arguments

`column` A data table column.

`length` Number of rows within the column to use.

`model` Messages for Time Series model options.

`estimates` A list of named values that matches the messages sent to `ARIMA Forecast()`. If you perform an ARIMA Forecast and save the script, the estimates are part of the script.

`from, to` Define the range of values. Typically, *`from`* is between 1 and *`to`*, inclusive. If *`from`* is less than or equal to 0, and if *`from`* is less than or equal to *`to`*, the results include filtered predictions.

`Best Partition(xindices, yindices, <<Ordered, <<Continuous Y, <<Continuous X)`

Description

Experimental function to determine the optimal grouping.

Returns

A list.

Arguments

`xindices, yindices` Same-dimension matrices.

`Col Cumulative Sum(name, <By var, ...>)`

`Cumulative Sum(name)`

Description

Returns the cumulative sum for the current row. `Col Cumulative Sum` supports By columns, which do not need to be sorted.

Arguments

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

`Col Maximum(name, <By var, ...>)`

`Col Max(name)`

Description

Calculates the maximum value across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The maximum value that appears in the column.

Arguments

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Mean(name, <By var, ...>)

Description

Calculates the mean across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The mean of the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Median(name, <By var, ...>)

Description

Calculates the median across all rows of the specified column. The ordering is cached internally to speed up multiple evaluations.

Returns

The median of the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

`Col Minimum(name, <By var, ...>)`

`Col Min(name)`

Description

Calculates the minimum value across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The minimum value that appears in the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

`Col Mode(name, <By var, ...>)`

Description

Calculates the mode across all rows of the specified column. The ordering is cached internally to speed up multiple evaluations.

Returns

The mode of the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Moving Average(name, options, <By var, ...>)

Moving Average(name, options)

Description

Returns the moving average over a given interval based at the current row. `Col Moving Average` supports By columns.

Arguments

`name` A column name.

`Weighting(1|0|n)` Required positional argument. Determines how the values are weighted. 1 indicates uniform weighting. 0 indicates incremental weighting (a ramp or triangle). Any other number is the parameter for an exponential moving average (EWMA or EMA).

`Before(1|0|n)` Positional argument. Controls the size of the range (or window) by including the specified number of items before the current item in the average (in addition to the current item). The default value, -1, means all of the preceding items.

`After(1|0|n)` Positional argument. Controls the size of the range (or window) by including the specified number of items after the current item in the average (in addition to the current item). The default value, 0, means no following items.

`Partial Window is Missing` Boolean positional argument. Controls how missing values are treated. By default, missing values are ignored. 0 computes the average of partial windows.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Examples

```
// equal weighting of a five-item lagging range
Col Moving Average( x, 1, 4 );

// ramp weighting of all preceding items
Col Moving Average( x, 0 );

// triangle weighting of a five-item centered range
Col Moving Average( x, 0, 2, 2 );

// exponential weighting of all preceding items
Col Moving Average( x, 0.25 );
```

Col N Missing(name, <By var, ...>)

Description

Calculates the number of missing values across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The number of missing values in the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Number(name, <By var, ...>)

Description

Calculates the number of nonmissing values across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The number of nonmissing values in the column.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Quantile(name, p, <ByVar>)

Description

Calculates the specified quantile *p* across all rows of the specified column. The result is internally cached to speed up multiple evaluations.

Returns

The value of the quantile.

Argument

`name` A column name.

`p` A specified quantile *p* between 0 and 1.

ByVar (Optional) A By group.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
Col Quantile( :height, .5 );
   63
```

63 is the 50th percentile, or the median, of all rows in the height column.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

`Col Rank(column, <ByVar, ...>, <<tie("average"|"arbitrary"|"row"|"minimum"))`

Description

Ranks each row's value, from 1 for the lowest value to the number of columns for the highest value. Ties are broken arbitrarily.

Arguments

`column` The column to be ranked.

`ByVar` (Optional) A By variable to compute statistics across groups of rows.

`<<tie` Determines how the tie is broken. A tie occurs when the values being ranked are the same. For the data `[33 55 77 55]`, `33` has rank 1 and 77 has rank 4, and the question is how to assign ranking for the `55`s. `average` reports the average of the possible rankings, 2.5, for both 55s. `arbitrary` matches JMP 12 behavior by assigning the possible rankings in an unspecified order, which could be 2 and 3 or 3 and 2. `row` assigns the ranks in the order that they originally appear. (The first 55 would be 2 and the second 55 would be 3.)

`minimum` gives both values the lowest possible rank, 2.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

`Col Simple Exponential Smoothing(column, alpha, <ByVar> )`

Description

Returns the simple exponential smoothing prediction for the current row using smoothing weight *alpha*.

Arguments

`column` The column of time series observations.

`alpha` The smoothing weight.

`ByVar` (Optional) A By variable to compute predictions across groups of rows. By variables do not need to be presorted.

Notes

The predicted value for row *t* is given by the following:

```
Predicted[t] = alpha * Observed[t-1] + (1-alpha) * Predicted[t-1]
```

By definition, `Predicted[1] = Observed[1]`.

Col Standardize(name,<By var, ...>)

Description

Calculates the column mean divided by the standard deviation across all rows of the specified column.

Returns

The standardized mean.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. If a By variable is specified, the values are standardized against the mean and standard deviation of their corresponding By variable group.

Notes

Standardizing centers the variable by its sample standard deviation. Thus, the following commands are equivalent:

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << New Column( "stdht", Formula( Col Standardize( height ) ) );
dt << New Column( "stdht2",
   Formula( (height - Col Mean( height )) / Col Std Dev( height ) )
);
```

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Std Dev(name,<By var, ...>)

Description

Calculates the standard deviation across rows in a column. The result is internally cached to speed up multiple evaluations.

Returns

The standard deviation.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Col Sum(name,<By var, ...>)

Description

Calculating all missing values (`Col Sum(.,.)`) returns missing. Calculates the sum across rows in a column. The result is internally cached to speed up multiple evaluations.

Returns

The sum.

Argument

`name` A column name.

`By var` (Optional) A By variable to compute statistics across groups of rows. Use the By variable in a column formula or in a `For Each Row()` function.

Notes

If a data value is assigned by a column property (such as Missing Value Codes), use `Col Stored Value()` to base the calculation on the value stored in the column instead.

See Also

"Col Stored Value(<dt>, col, <row>)"

Fit Censored(Distribution("name"), YLow(vector) | Y(Vector), <YHigh(vector)>, <Weight(vector)>, <X(matrix)>, <Z(matrix)>,

`<HoldParm(vector)>, <Use random sample to compute initial values(percent)>, <Use first N observations to compute initial values(nobs)>)`

Description

Fits a distribution using censored data.

Returns

A list that contains parameter estimates, the covariance matrix, the log-likelihood, the AICc, the BIC, and a convergence message. See *Fitting Linear Models*.

Arguments

`Distribution("name")` The quoted name of the distribution to fit.

`YLow(vector) | Y(Vector)` If you do not have censoring, then use `Y` and an array of your data, and do not specify `YHigh`. If you do have censoring, then specify `YLow` and `YHigh` as the lower and upper censoring values, respectively.

Optional Arguments

`YHigh(vector)` A vector that contains the upper censoring values. Specify this only if you have censoring and also specify YLow.

`Weight(vector)` A vector that contains the weight values.

`X(matrix)` The regression design matrix for location.

`Z(matrix)` The regression design matrix for scale.

`HoldParm(vector)` An array of specified parameters. The parameters should be nonmissing where they are to be held fixed, and missing where the are to be estimated. This is primarily used to test hypotheses that certain parameters are zero or some other specific value.

`Use random sample to compute initial values(percent)` A percent of the observations to be used in the computation of the initial values. Specify this if the data vector is large.

`Use first N observations to compute initial values(nobs)` A number of observations at the start of the data vector to be used in the computation of the initial values. Specify this if the data vector is large.

`Fit Circle(Xvec, Yvec)`

Description

Fits a circle that best goes through three or more points using a least squares approach. If only three points are specified, a direct solution can be found, and the sum of squared errors is zero.

Returns

A list that contains the X and Y coordinates of the center point of the circle, the length of the radius, and the sum of squared errors.

Arguments

`Xvec` Vector of X coordinates of three or more points.

Yvec Vector of Y coordinates of three or more points.

Syntax

```
{Xcenter, yCenter, radius, SSE} = Fit Circle(Xvec, Yvec)
```

Hier Clust(x)

Description

Returns the clustering history for a hierarchical clustering using Ward's method (without standardizing data).

Argument

x A data matrix.

IRT Ability(Q1, <Q2, Q3, ... Qn,> parmMatrix)

Description

Returns scores for the latent variable in an item response theory model with n binary items and a matrix of known parameters. The parameter matrix should contain as many rows as there are parameters in the model and as many columns as there are items in the analysis.

Arguments

Q1, Q2, ..., Qn A set of n binary items.

parmMatrix A matrix of parameters from an item response theory model.

KDE(vector, <named arguments>)

Description

Returns a kernel density estimator with automatic bandwidth selection.

Argument

vector A vector.

Optional Named Arguments

<<weights Must be a vector of the same length as vector, and can contain any nonnegative real numbers. Weights represents frequencies, counts, or similar concepts.

<<bandwidth(n) A nonnegative real number. Enter a value of 0 to use the bandwidth selection argument.

<<bandwidth scale(n) A positive real number.

<<bandwidth selection(n) *n* must be 0, 1, 2, or 3, corresponding to Sheather and Jones, Normal Reference, Silverman rule of thumb, or Oversmoother, respectively.

<<kernel(n) *n* must be 0, 1, 2, 3, or 4, corresponding to Gaussian, Epanechnikov, Biweight, Triangular, or Rectangular, respectively.

`LenthPSE(x)`

Description

Returns Lenth's pseudo-standard error of the values within a vector.

Argument

x A vector.

`Max()`

See "Maximum(var1, var2, ...)".

`Maximum(var1, var2, ...)`

`Max(var1, var2, ...)`

Description

Returns the maximum value of the arguments or of the values within a single matrix or list argument. If multiple arguments are specified, they must be all numeric values or all quoted strings.

`Mean(var1, var2, ...)`

Description

Returns the arithmetic mean of the arguments or of the values within a single matrix or list argument.

`Median(var1, var2, ...)`

Description

Returns the median of the arguments or of the values within a single matrix or list argument.

`Min()`

See "Minimum(var1, var2, ...)".

`Minimum(var1, var2, ...)`

`Min(var1, var2, ...)`

Description

Returns the minimum value of the arguments or of the values within a single matrix argument. If multiple arguments are specified, they must be either all numeric values or all quoted strings.

`N Missing(expression)`

Description

Rowwise number of missing values in variables specified.

`Number(var1, var2, ...)`

Description

Rowwise number of nonmissing values in variables specified.

`Product(i=initialValue, limitValue, bodyExpr)`

Description

Multiplies the results of *bodyExpr* over all *i* until the *limitValue* and returns a single product.

`Quantile(p, arguments)`

Description

Returns the p^{th} quantile of the arguments. The first argument can be a scalar or a matrix of values between 0 and 1. The remaining arguments can also be specified as values within a single matrix or list argument.

`Range(var1, var2, ...)`

Description

Returns the minimum and maximum values of the arguments. The result is returned as a two-element row vector that contains the minimum and the maximum.

`Robust PCA(X, <Lambda(2/sqrt(max(nrow, ncol)))>, <tolerance=1e-10>, <maxit(75)>, <Center(1)>, <Scale(1)>)`

Description

Performs a sequence of singular value decompositions and thresholding steps to decompose the data matrix into a low-rank matrix and a sparse matrix of residuals.

Returns

`A` The low-rank matrix estimation.

`E` The sparse matrix of residuals.

`S` A vector of singular values.

Arguments

`X` A data matrix.

`Lambda` Specifies a value greater than 0 that determines the sparsity of the matrix of residuals. For larger values of Lambda, the matrix of residuals is more sparse.

`tolerance` The convergence criterion.

`maxit` The maximum number of SVD iterations.

`Center` Centers the data prior to performing the SVD iterations.

`Scale` Scales the data prior to performing the SVD iterations

`Std Dev(var1, var2, ...)`

Description

Rowwise standard deviation of the variables specified.

`Sum(var1, var2, ...)`

Description

Rowwise sum of the variables specified. Calculating all missing values (`Sum(.,.)`)returns missing.

`SSQ(x1, ...)`

Description

Returns the sum of squares of all elements. Takes numbers, matrices, or lists as arguments and returns a scalar number. Skips missing values.

`Summarize(<dt>, <by>, <count>, <sum>, <mean>, <min>, <max>, <stddev>, <corr>, <quantile>, <first>)`

Description

Gathers summary statistics for a data table and stores them in global variables.

Returns

None.

Arguments

`dt` Optional positional argument: a reference to a data table. If this argument is not in the form of an assignment, then it is considered a data table expression.

All other arguments are optional and can be included in any order. Typically, each argument is assigned to a variable so you can display or manipulate the values further.

`name=By(col | list | Eval)` Using a BY variable changes the output from single values for each statistic to a list of values for each group in the BY variable.

`Summarize YByX(X(<x columns>, Y (<y columns>), Group(<grouping columns>), Freq(<freq column>), Weight(<weight column>))`

Description

Calculates all Fit Y by X combinations on large-scale data sets.

Returns

A data table of *p*-values and logworth values for each Y and X combination.

Arguments

`X(col)` The factor columns used in the fit model.

`Y(col)` The response columns used in the fit model.

`Group(gcol)` The group of columns used in the fit model.

`Freq(col)` The frequency (for each row) column used in the fit model.

`Weight(col)` The importance (or influence) column used in the fit model.

Notes

Performs the same function as the Response Screening platform.

`Summation(init, limitvalue, body)`

Description

Summation sums the results of the *body* statement(s) over all *i* to return a single value.

`Tolerance Limit(1-alpha, p, n)`

Description

Constructs a *1-alpha* confidence interval to contain proportion *p* of the means with sample size *n*.

Transcendental Functions

`Arrhenius(n)`

Description

Converts the temperature *n* to the value of explanatory variable in Arrhenius model.

Returns

11604.5181215503/(n+273.15)

Argument

`n` Temperature in Celsius.

Notes

This is frequently used as a transformation.

`Arrhenius Inv(n)`

Description

The inverse of the Arrhenius function. Converts the value *n* to the temperature in Celsius.

Returns

(11604.5181215503/n)-273.15

Argument

n The value of the converted explanatory variable in Arrhenius model.

Notes

This is frequently used as a transformation.

`Beta(a, b)`

Description

$$\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Returns

Returns the beta function.

Arguments

a, b numbers

`Cytometry Logicle(x, T, W, M, A)`

Description

Computes a cytometry logicle transformation. For more information about the logicle transformation, see Moore and Parks (2012).

`Cytometry Logicle Inverse(y, T, W, M, A)`

Description

Computes the inverse cytometry logicle transformation. For more information about the logicle transformation, see Moore and Parks (2012).

`Digamma(n)`

Description

The derivative of the log of the gamma function (LGamma).

Returns

The digamma function evaluated at *n*.

Argument

n A number

Exp(a)

Description

Raises e to the power *a*.

Returns

e^a.

Argument

a A number

Equivalent Expression

e()^a

ExpM1(x)

Description

Returns a more accurate calculation of *Exp(x)-1* when *x* is very small.

Factorial(n)

Description

Multiplies all numbers 1 through *n*, inclusive

Returns

The product.

Arguments

n Any integer

Notes

One and only one argument must be specified.

FFT({list}, <named arguments>)

Description

Conducts a Fast Fourier Transformation (FFT) on a list of matrices.

Returns

The function takes one matrix, or a list of matrices for complex numbers. The returned value is a list of two matrices with the same dimensions as the first argument.

Argument

List A list of one or two matrices. If one is provided, it is considered to be the real part. If two are provided, the first is the real part and the second is the imaginary part. Both matrices must have the same dimensions, and both must have more than one row.

Optional Named Arguments

<<inverse(Boolean) If true (1), an inverse FFT is conducted.

<<multivariate(Boolean) If true (1), a multivariate FFT is conducted. If false(0), a spatial FFT is conducted.

<<scale(number) Multiplies the return values by the specified *number*.

Fit Transform To Normal(Distribution("name"), Y(vector), <Freq(vector))

Description

Fits a transformation to normality for a vector of data. This includes the Johnson Sl, Johnson Sb, Johnson Su, and GLog distributions.

Returns

A list that contains parameter estimates, the covariance matrix, the log-likelihood, AICc, a convergence message, and the transformed values. See *Fitting Linear Models*.

Gamma(t, <limit>)

Description

The gamma function of *x*, or for each row if *x* is a column:

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx$$

Returns

The gamma.

Arguments

t a number or a column

limit optional limit. The default is infinity.

Notes

Gamma(t, limit) is the same integral as Gamma(t) but with the limit of integration that is defined instead of infinity.

LGamma(t)

Description

Returns the log gamma function for *t*, which is the natural log of gamma.

Ln(n)

Description

Returns the natural logarithm (base *e* logarithm) of *n*.

`Log(n, <base>)`

Description

Returns the natural logarithm (base *e* logarithm) of *n*. An optional second argument lets you specify a different base. For example, `Log(n,3)` for the base 3 logarithm of *n*. The Log argument can be any numeric expression. The expression `Log(e())` evaluates as 1, and `Log(32,2)` is 5.

`Log10(n)`

Description

Returns the common (base 10) logarithm of *n*.

`Log1P(n)`

Description

Same as Log(1 + *x*), except that it is more accurate when *x* is very small.

`Logist(x)`

Description

Returns 1/(1+Exp(-x)), which converts a number in the domain -∞...+∞ into range 0...1. The function is useful in logistic regression.

`Logist Percent(p)`

Description

Similar to the `Logist()` function but with the result scaled from 0 to 100.

`Logit(p)`

Description

Returns `log(p/(1-p))`.

`Logit Percent(p)`

Description

Similar to the `Logit()` function with the argument 0 to 100 rather than 0 to 1.

N Choose K(n, k)

Description

This function returns the number of n things taken k at a time ("n choose k") and is computed in the standard way using factorials, as $n!/(k!(n-k)!)$. For example, `NChooseK(5,2)` evaluates as 10.

Notes

This is implemented internally in JMP using lGamma functions. The result is not always an integer.

Power(a, <b>)

a^b

Description

Raises *a* to the power of *b*.

Returns

The product of *a* multiplied by itself *b* times.

Arguments

a Can be a variable, number, or matrix.

b (Optional) Can be a variable or a number.

Notes

For `Power()`, the second argument (*b*) is optional, and the default value is 2. `Power(a)` returns a^2.

Root(n, <r>)

Description

Returns the *r*th root of *n*, where *r* defaults to 2 for square root.

SbInv(z, gamma, delta, theta, sigma)

Description

Returns a transformation of a standard normal variable to a double bounded Johnson variable.

SbTrans(x, gamma, delta, theta, sigma)

Description

Returns a transformation of a double bounded Johnson variable to a standard normal variable.

`Scheffe Cubic(x1, x2)`

Description

Returns `x1*x2*(x1-x2)`. This function supports notation for cubic mixture models.

`SHASHInv(z, gamma, delta, theta, sigma)`

Description

Returns a transformation of a standard normal variable to a sinh-arcsinh (SHASH) variable. The transformation is calculated as σ`*sinh((arcsinh(z)-`γ`)/`δ`)+`θ.

`SHASHTrans(x, gamma, delta, theta, sigma)`

Description

Returns a transformation of a sinh-arcsinh (SHASH) variable to a standard normal variable. The transformation is calculated as `sinh(`γ`+`δ`*arcsinh((x-`θ`)/`σ`))`.

`SlInv(z, gamma, delta, theta, sigma)`

Description

Returns a transformation of a standard normal variable to a Johnson Sl variable.

`SlTrans(x, gamma, delta, theta, sigma)`

Description

Returns a transformation of a Johnson Sl variable to a standard normal variable.

`Sqrt(n)`

Description

Returns the square root of *n*.

`Squash(expr)`

Description

An efficient computation of the function 1/ [1 + exp(*expr*)].

`Squish(expr)`

Description

Equivalent to `Squash(-expr)`, or $1/(1+e^{-\text{expr}})$.

`SuInv(z, gamma, delta, theta, sigma)`

Description

Returns a transformation of a standard normal variable to an unbounded Johnson variable.

`SuTrans(x, gamma, delta, theta, sigma)`

Description

Returns a transformation of an unbounded Johnson variable to a standard normal variable.

`Trigamma()`

Description

Returns the trigamma function evaluated at *n*. The trigamma function is the derivative of the digamma function.

Trigonometric Functions

JMP's trigonometric functions expect all angle arguments in radians.

`ArcCosH(x)`

Description

Inverse hyperbolic cosine.

Returns

The inverse hyperbolic cosine of *x*.

Argument

x Any number, numeric variable, or numeric expression.

`ArcCosine(x)`

`ArCos(x)`

Description

Inverse cosine.

Returns

The inverse cosine of *x*, an angle in radians.

Argument

x Any number, numeric variable, or numeric expression.

`ArcSine(x)`

`ArSin(x)`

Description

Inverse sine.

Returns

The inverse sine of *x*, an angle in radians.

Argument

`x` Any number, numeric variable, or numeric expression.

`ArcSinH(x)`

Description

Inverse hyperbolic sine.

Returns

The inverse hyperbolic sine of *x*.

Argument

`x` Any number, numeric variable, or numeric expression.

`ArcTangent(x1, <x2=1>)`

`ArcTan(x1 <x2=1>)`

`ATan(x1 <x2=1>)`

Description

Inverse tangent.

Returns

The inverse trigonometric tangent of *x1/x2*, where the result is in the range -Pi()/2, Pi()/2.

Argument

`x1` Any number, numeric variable, or numeric expression.

`x2=1` Specifies *atan2*.

`ArcTanH(x)`

Description

Inverse hyperbolic tangent.

Returns

The inverse hyperbolic tangent of *x*.

Argument

`x` Any number, numeric variable, or numeric expression.

`CosH(x)`

Description

Hyperbolic cosine.

Returns

The hyperbolic cosine of *x*.

Argument

x Any number, numeric variable, or numeric expression.

`Cosine(x)`

`Cos(x)`

Description

Cosine.

Returns

The cosine of *x*.

Argument

x Any number, numeric variable, or numeric expression. The angle in radians.

`Sine(expr)`

`Sin(expr)`

Description

Returns the sine.

`SinH(expr)`

Description

Returns the hyperbolic sine.

`Tangent(expr)`

`Tan(expr)`

Description

Returns the tangent of an argument given in radians.

`TanH(expr)`

Description

Returns the hyperbolic tangent of its argument.

Utility Functions

Add(a, b, ...)

a+b+...

Description

Adds the values of the listed arguments. No arguments are changed.

Returns

The sum.

Arguments

For Add(), a comma-separated list of variables, numbers, or matrices.

For a+b, any number of variables, numbers, or matrices.

Notes

- Any number of arguments is permitted. If no argument is specified, Add() returns 0.
- Add() returns missing if any arguments are missing. To ignore missing values, use Sum().

See Also

the *Scripting Guide*

See "Sum(var1, var2, ...)"

Beep()

Description

Produces an alert sound.

Returns

Null.

BLOB MD5(blob)

Description

Converts the *blob* argument into a 16-byte blob.

Notes

The 16-byte blob is the MD5 checksum, or the hash, of the source blob.

BLOB Peek(blob, offset, length)

Description

Creates a new blob from a subrange of bytes of the *blob* argument.

Returns

A blob object.

Arguments

`blob` a binary large object.

`offset` An integer that specifies how many bytes into the blob to begin construction. The first byte is at offset 0, the second byte at offset 1.

`length` An integer that specifies how many bytes to copy into the new blob, starting at the offset.

Build Information()

Description

Returns the build date and time, whether it's a release or debug build, and the product name in a quoted comma-delimited string.

Caption({*h*, *v*}, "text", <Delayed(seconds)>, <Font(font)>, <Font Size(size)>, <Text Color("color")>, <Back Color("color")>, <Spoken(Boolean)

Description

Displays a caption window at the location described by *{h, v}* that displays *text*. The caption can be delayed before being displayed by *seconds*, or can be spoken. You can also specify the font type, size, and color and background color.

Returns

Null.

Arguments

`{h, v}` a list with two values. *h* is the horizontal displacement from the top left corner of the monitor in pixels. *v* is the vertical displacement from the top left corner in pixels.

`text` A quoted string or a reference to a string that is to be displayed in the caption.

`Delayed(seconds)` *seconds* is optional delay before displaying the caption. Setting this option causes this caption and all subsequent captions to be delayed by the specified number of seconds.

`Font(font)` Specify the font type.

`Font Size(size)` Specify the font size.

`Text Color("color")` Specify the color of text.

`Back Color("color")` Specify the background color.

`Spoken(Boolean)` Causes *text* to be spoken as well as displayed. The current setting (on or off) remains in effect until switched by another `Caption` statement that includes a `Spoken` setting.

Datafeed()

See "Open Datafeed()".

Debug Break()

When the JSL Debugger is open, this function stops a JSL script from executing at that point in the script. This function is useful for tracking in the debugger under user-specified conditions. If the JSL Debugger is not running, this function does not execute.

Decode64 BLOB(*string*)

Description

Decodes a printable quoted string of base 64 text into a blob.

Returns

A blob.

Arguments

`string` A quoted base-64 encoded string.

Example

```
Decode64 BLOB( "dGhlIHF1aWNrIGJyb3duIGZveA==" );
  Char To BLOB( "the quick brown fox", "ascii~hex" )
```

Decode64 Double(*string*)

Description

Creates a floating point number from a quoted base-64 encoded string.

Returns

A floating point number.

Arguments

`string` A quoted base-64 encoded string.

Disable JMP Live URL(url)

Description

Specifies the URLs that users cannot publish to. This preference must be added to jmpStartAdmin.jsl.

See the Common Tasks chapter in the *Scripting Guide* for details about the location of jmpStartAdmin.jsl.

If a URL appears in both the `Disable JMP Server()` and `Enable JMP Server()` lists, publishing to the URL is disabled.

Example

```
Disable JMP Live URL( "https://public.jmp.com" )
```

`Divide(a, b)`

`Divide(x)`

`a/b`

Description

Divides *a* by *b*. If only one argument is given (`divide(x)`), divides 1 by *x*.

Returns

The quotient of *a*/*b*; or the reciprocal of *x* (1/*x*) if only one argument is provided.

Arguments

`a, b, x` Can be a variable, number, or matrix.

Notes

If both arguments are matrices, it does matrix division.

`Empty()`

Description

Used in the formula editor for making empty boxes.

Returns

Missing.

Arguments

None.

`Enable JMP Live URL(url)`

Description

Specifies the URLs that users can publish to. This preference must be added to jmpStartAdmin.jsl.

See the Common Tasks chapter in the *Scripting Guide* for details about the location of jmpStartAdmin.jsl.

If a URL appears in both the `Enable JMP Server()` and `Disable JMP Server()` lists, publishing to the URL is disabled.

Example

```
Enable JMP Live URL( "https://public.jmp.com" )
```

`Encode64 BLOB(x)`

Description

Encodes a blob into a printable quoted string of base 64 text.

Returns

A quoted base-64 encoded string.

Example

```
Encode64 BLOB( Char To BLOB( "the quick brown fox" ) );
  "dGhlIHF1aWNrIGJyb3duIGZveA=="
```

Encode64 Double(n)

Description

Creates a quoted base-64 encoded string from a floating point number.

Returns

A quoted base-64 encoded string.

Arguments

`n` A floating point number.

Faure Quasi Random Sequence(nDim, nRow)

Description

Generates a sequence of spacefilling quasi random numbers using the Faure sequence.

Get Addin("id")

Description

Retrieves a registered add-in by *id*.

Returns

A scriptable object for the add-in. Returns empty if no add-in with the specified ID was found.

Argument

`"id"` The ID of an installed add-in.

Get Addins()

Returns

A list of all registered add-ins.

Get Addr Info("address", <port>)

Description

Converts a name to its numeric address.

Returns

A list of quoted strings. The first element is the command (`Get Addr Info`). The second is the results (for example, "ok" if the command was successful). The third is a list of strings

of information. Included in that information is the address that corresponds to the name that was supplied.

Arguments

`address` A quoted string that specifies the name (for example, `"www.sas.com"`).

`port` The port of the address.

Get Clipboard()

Description

Returns text from the computer's clipboard. If the content is not text, the result is null.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Copy Table Script( "Distribution" );
s = Get Clipboard();
nw = New Window( "Script", Script Box( s ) );
```

Get Name Info("address", <port>)

Description

Converts a numeric address to its name.

Returns

A list of quoted strings. The first element is the command (`GetNameInfo`). The second is the results (for example, "ok" if the command was successful). The third is a list of strings of information. Included in that information is the port name that corresponds to the address that was supplied.

Arguments

`address` A quoted string that specifies the numeric address (for example, `"149.173.5.120"`).

`port` The port of the address.

Get Platform Preferences(<platform <(option, ...)> ... >)

Get Platform Preference(<platform <(option, ...)> ... >)

Description

Returns the preferences for the specified platforms.

Returns

A list of platform preferences.

Argument

`platform` (Optional) Specifies the platform name. If not specified, all platform preferences are returned. You can specify one or more preferences for a platform.

option (Optional) Specifies the preference value. If not specified, all platform preference values are returned.

Notes

Table 2.3 describes the syntax for getting platform preferences.

Table 2.3 Get Platform Preferences() Syntax

Syntax	Description
`Get Platform Preferences( )`	Returns the current option values for all platform preferences.
`Get Platform Preferences( Platform )`	Returns the current option values for the specified platform preferences.
`Get Platform Preferences( Platform( Option ) )`	Returns the current option values for the specified platform.
`Get Platform Preferences( <<Changed )`	Returns the current option values that have changed for all platforms.
`Get Platform Preferences( Platform( <<Changed ) )`	Returns the current option values that have changed for all platform preferences.
`Get Platform Preferences( Platform( Option ( <<Changed ) ) )`	Returns the current option values that have changed for the specified platform.

Examples

Suppose that the user modified several platform preferences through the JMP Platforms window or a script.

```
Platform Preferences(
  Distribution( Set Bin Width( 2 ), Horizontal Layout( 1 ) ),
  Model Dialog( Keep Dialog Open( 1 ) ),
  Graph Builder( Legend Position( "Bottom" ) )
);
```

To return all of the modified platform preferences, use `Get Platform Preferences( <<Changed )`:

```
Get Platform Preferences( <<Changed );
```

```
Platform Preferences(
   Distribution( Horizontal Layout( 1 ), Set Bin Width( 2, <<On ) ),
   Graph Builder( Legend Position( "Bottom", <<On ) ),
   Model Dialog( Keep dialog open( 1 ) )
);
```

`Get Preferences(<preference_name>)`

`Get Preference(<preference_name>)`

Description

Returns the settings for the specified preferences.

Returns

A list of preference settings.

Argument

`preference_name` (Optional) If no preference is specified, all preferences are returned. Otherwise, the settings for the specified preference are returned.

Notes

The preferences for the following areas are not accessible in JSL: Text Data Files, Windows Specific, Mac OS Settings, Fonts, Communications, Script Editor, and JMP Updates. For more information about getting platform preferences, see "Get Platform Preferences(<platform <(option, ...)> ... >)".

`Glue(expr1, expr2, ...)`

`expr1; expr2`

Description

Evaluates each argument in turn.

Returns

The result of the last argument evaluated.

Arguments

One or more valid JSL expressions.

Notes

A semicolon placed between expressions is the more commonly used form for `Glue()`.

`Gzip Compress(blob)`

Description

Compresses a blog of data into a gzip blob.

Example

```
Gzip Compress(
   Char To BLOB(
```

```
        "random data does not usually compress well and may get larger"
    )
);
    Char To BLOB(

        "~1F~8B~08~00~00~00~00~00~00~0A~0D~CA~C1~0D~00~21~08~04~C0V~B6~B5~CDA~FC
        ~80~5C~00c~EC^~E7=~C9)~E1~106~21~A1~85~19~8DU~8Bf~07_~F8~9FZ~85~ADfx~13~
        CE~83~A1~0Dc~0E~CD~0B~94*~16~1E=~00~00~00",
        "ascii~hex"
```

Gzip Uncompress(blob)

Description

Uncompresses a blob of gzip data into a blob.

Example

```
Gzip Uncompress( /*typically this data might come from Gzip Compress() but
    might also come from a .gz file using Load Text File() with the blob
    option*/
    Char To BLOB(

    "~1F~8B~08~00~00~00~00~00~00~0A~0D~CA~C1~0D~00~21~08~04~C0V~B6~B5~CDA~FC~8
    0~5C~00c~EC^~E7=~C9)~E1~106~21~A1~85~19~8DU~8Bf~07_~F8~9FZ~85~ADfx~13~CE~8
    3~A1~0Dc~0E~CD~0B~94*~16~1E=~00~00~00",
        "ascii~hex"
    )
);
    Char To BLOB(
        "random data does not usually compress well and may get larger",
        "ascii~hex"
    )
```

Host Is("argument")

Description

Determines whether the host environment is the specified OS.

Returns

True (1) if the current host environment matches the argument, false (0) otherwise.

Argument

`Argument Windows"` or "`Mac`" tests for the specified operating system.

Is Alt Key()

Description

Returns 1 if the Alt key is being pressed, or 0 otherwise.

Notes

On a macOS, `Is Alt Key()` tests for the Option key.

`Is Command Key()`

Description

Returns 1 if the Command key is being pressed, or 0 otherwise.

`Is Context Key()`

Description

Returns 1 if the Context key is being pressed, or 0 otherwise.

`Is Control Key()`

Description

Returns 1 if the Control key is being pressed, or 0 otherwise.

Notes

On a macOS, `Is Control Key()` tests for the Command key.

`Is Option Key()`

Description

Returns 1 if the Option key is being pressed, or 0 otherwise.

`Is Shift Key()`

Description

Returns 1 if the Shift key is being pressed, or 0 otherwise.

`JMP Product Name()`

Description

Returns either `"Standard"`, `"Pro"`, or `"Student"`, depending on which version of JMP is licensed.

`JMP Version()`

Description

Returns the version number of JMP that you are running.

Returns

release.revision<.fix>

Arguments

none

```
Load DLL("path" <,AutoDeclare(Boolean | Quiet | Verbose)>)
Load DLL("path" <, Quiet | Verbose>)
```

Description

Loads the DLL in the specified *path*.

Arguments

`path` A pathname that specifies where to load the DLL.

`AutoDeclare(Boolean | Quiet | Verbose)` Optional argument. `AutoDeclare(1)` and `AutoDeclare(Verbose)` write verbose messages to the log. `AutoDeclare(Quiet)` turns off log window messages. If you omit this option, verbose messages are written to the log.

`Quiet | Verbose` Optional argument. When you use `Declare Function()`, this option turns off log window messaging (`Quiet`) or turns on log window messaging (`Verbose`).

See Also

Once a DLL is loaded, you send the DLL object messages to interact with it. See "Dynamic Link Library (DLL) Messages" for more information about these messages.

```
Mail("address"|"addresses", "subject", "message", <"attachment filepath" |
{"attachment 1 filepath", "attachment 2 filepath", ...}>)
```

Description

(Windows) Sends e-mail (using MAPI) to the *address* with the specified *subject* and *message* texts. Sends one or more attachments specified by the optional *attachment* argument. The attachment argument can evaluate to a quoted string or list of strings.

(macOS) Creates an e-mail in the user's Mail application. The user must click **Send** in the e-mail. In Microsoft Outlook, you must manually add attachments to the e-mail.

Examples

To send an email with multiple attachments on Windows:

```
Mail(
   "yourname@company.com",
   "New data and script",
   "Today's updated data table and script are attached.",
   {"$DOCUMENTS/wd.jsl", "$DOCUMENTS/survey.jmp"}
);
```

or:

```
list = {"$DOCUMENTS/wd.jsl", "$DOCUMENTS/survey.jmp"};
Mail(
   "yourname@company.com",
```

```
    "New data and script",
    "Today's updated data table and script are attached.",
    list
);
```

To send an email to multiple recipients:

```
Mail(
    {"hername@company.com", "hisname@company.com"},
    "Database updates",
    "Today's sales database contains the numbers from last month."
);
```

Notes

On macOS, `Mail()` works on Yosemite and later operating systems.

`Main Menu(`*`string`*`, <`*`string`*`>)`

Description

Execute the command found on JMP's menu named by the quoted *`string`*.

Arguments

`string` The internal path name as shown in the menu editor for items. For example, "NEW" is the internal name for the New subcommand in the File menu.

`string` (Optional) The name of the window to send the command to.

Examples

`Main Menu()` accepts either a full path or a partial path. If a partial name is used, and there are other menu items with the same name, the first menu item found is executed. JMP searches the top-level menu (File, Tables, DOE, and so on) first for the partial name and then searches inside each of those menus in order.

```
Main Menu( "File:New:Data Table" ); // full path
Main Menu( "Data Table" ); // partial path
```

`Minus(a)`

`-a`

Description

Reverses the sign of *a*.

Returns

-a if *a* is positive (`a=3; -a=-3; Minus(a)=-3`).

a if *a* is negative (`a=-3; -a=3; Minus(a)=3`).

0 if *a* is 0 (`a=0; -a=0; Minus(a)=0`).

Missing if *a* is missing.

Argument

a Can be variable or a number. A variable must contain a number or a matrix.

Multiple File Import(arguments)

Description

Imports one or more files into a data table. You can create this JSL by selecting Save Script to Script Window from the Multiple File Import window.

Returns

Creates a Multiple File Import Object. The object accepts messages to set a folder, filter files, and specify import options.

Arguments

<<Set Folder Specifies the folder that contains the files you want to import.

<<Set Name Filter (Optional) Specifies the file name or extension of the files. The name filter uses * to represent zero-or-more characters and ? to represent exactly one character. * and ? also match a period. The default setting is *.*, or all files. Because you can specify files using a semicolon-delimited list of filters, file names that include semicolons or | must be specified using ? or *.

<<Set Name Enable(Boolean) Enables the name filter. The setting is off by default.

<<Set Size Filter (Optional) Filters the file list by file size. Specify the sizes by kB (kilobytes, or 1000 bytes) in a list. The default values are based on the size range of the files in the file list.

<<Set Size Enable(Boolean) (Optional) Enables the size filter. The default setting is off.

<<Set Date Filter (Optional) Filters the file list by date and time. Specify the date and time in a list in seconds. The default values are based on the date and time range of the files in the file list.

<<Set Date Enable(Boolean) (Optional) Enables the date filter. The default setting is off.

<<Set Add File Name Column(Boolean) (Optional) Includes a column that contains the imported file name. The default setting is off.

<<Set Add File Size Column(Boolean) (Optional) Includes a column that contains the size of the imported file. The default setting is off.

<<Set Add File Date(Boolean) (Optional) Includes a column that contains the time and date stamp of the imported file. The default setting is off.

<<Set Import Mode(Row Per File|Row Per Line|CSV Data) (Optional) Specifies the format of the file that is imported: whole file on one row, one line on one row, and CSV. CSV Data is the default setting.

`<<Set Charset(Best Guess|utf-8|utf-16|us~ascii|windows-1252|x-mac-roman|x-mac-japanese|shift-jis|euc-jp|utf-16be|gb2312)` (Optional) The character set in the imported file. The character set specified in the General preferences (Open Text File Charset) is set by default.

`<<Set Stack Mode(Stack Similar|TablePerFile)` (Optional) Specifies how the files are combined. Stack Similar is the default setting. (When JMP detects that the files have the same columns, the files are concatenated into a single data table.)

`<<Set CSV Has Headers(Boolean)` (Optional) Specifies whether the CSV file contains a header row. The setting is on by default.

`<<Set CSV Allow Numeric(Boolean)` (Optional) Sets the data type to numeric. The setting is on by default.

`<<Set CSV First Header Line(`*n*`)` (Optional) Specifies the header row number. 1 is the default setting.

`<<Set CSV Number of Header Lines(`*n*`)` (Optional) Specifies the number of header rows. 1 is the default setting.

`<<Set CSV First Data Line(n)` (Optional) Specifies the first line that contains data. 2 is the default setting.

`<<Set CSV EOF Comma(Boolean)` (Optional) Specifies a comma delimiter. The setting is on by default.

`<<Set CSV EOF Tab(Boolean)` (Optional) Specifies a tab delimiter.

`<<Set CSV EOF Space(Boolean)` (Optional) Specifies a space delimiter.

`<<Set CSV EOF Spaces(Boolean)` (Optional) Specifies spaces as the delimiter.

`<<Set CSV EOF Other("")` (Optional) Specifies a custom delimiter.

`<<Set CSV EOF CRLF(Boolean)` (Optional) Specifies carriage return and line feed end-of-line characters. The setting is on by default.

`<<Set CSV EOF CR(Boolean)` (Optional) Specifies a carriage return end-of-line character. The setting is on by default.

`<<Set CSV EOF LF(Boolean)` (Optional) Specifies a line feed end-of-line character.

`<<Set CSV Semicolon(Boolean)` (Optional) Specifies a semicolon end-of-line character. The setting is off by default.

`<<Set CSV EOL Other("")` (Optional) Specifies a custom end-of-line character.

`<<Set CSV Quote("")` (Optional) Specifies the character used as a quote. The default setting is `\!"`, a double quotation mark.

`<<Set CSV Escape("")` (Optional) Specifies the escape sequence such as a backlash instead of doubling the quotation mark.

`<<Import Data` Imports the data.

Example

```
mfi = Multiple File Import(
  <<Set Folder( "$SAMPLE_IMPORT_DATA" ),
  <<Set Name Filter( "UN*.csv" ), // import files with this name
```

```
    <<Set Name Enable( 1 ) // display the file name in a column
)
<<Import Data();
```

`Multiply(a, b, ...)`

`a*b*...`

Description

Multiplies all values. No arguments are changed.

Returns

The product.

Arguments

Any number of variables, numbers, or matrices.

Notes

Any number of arguments is permitted. If no argument is specified, `Multiply()` returns 1.

`Name(`*`string`*`)`

Description

A name is something to call an item.

- If the name begins with an alphabetic character or underscore, and continues with alphanumeric characters, whitespace, Unicode mathematical symbols and certain punctuation (apostrophes, percentage signs, periods, backslashes, and underscores, then you can use the name directly in scripts.
- You can use names that do not follow these rules with the `Name()` keyword.

Examples

(`"name"n`) is preferred. `Name( `*`string`*` )` is deprecated.

```
"my-var-name"n = "hello";
Length( "my-var-name"n )
   5
```

`New OAuth2 Token(user(yourgoogleaccount@gmail.com), client ID(`*`string`*`), client secret(`*`string`*`), refresh token(`*`string`*`), token URL(`*`string`*`))`

Description

Creates an OAuth2 token for securely accessing data across different web APIs.

Arguments (Required)

`user` The user name, email, or personal identifier for the account being accessed.

`client ID` The public identifier that acts as an API key.

`client secret` The private identifier that corresponds to Client ID.

`refresh token` A token used to get access tokens.

`token URL` The URL that access tokens are received from. Unique to every service and accessible through their API or OAuth page.

Arguments (Authorization Code Grant)

`redirect URL( "https://app.getpostman.com/oauth2/callback" )` The URL that an access code is sent back to. Unless your company or the service provides one, we recommended that you create a free Postman account and use this redirect.

`client secret( "1aB893cdDeFf2D" )` The private identifier that corresponds to Client ID.

`request auth( ... )` Extra parameters indicating that you'll use an Authorization Code flow. Requires `Auth URL()`. Some services require scope. You can add custom fields with Fields.

`scope( "spreadsheets email docs" )` A space-separated list of scopes. Unique to every service, and accessible through their API or OAuth page. Only usable in `Request Auth()`.

`auth URL( "https://www.example.com/oauth2/v1/authorize" )` The URL for requesting authorization. Unique to every service, and accessible through their API or OAuth page. Only usable in `Request Auth()`.

Arguments (Implicit Grant)

`redirect URL( "https://app.getpostman.com/oauth2/callback" )` The URL that an access code is sent back to. Unless your company or the service provides one, it's recommended to create a free Postman account and use this redirect.

Arguments (Resource Owner)

`password( "wordpass123" )` The password that corresponds to the username.

`client secret( "1aB893cdDeFf2D" )` The private identifier that corresponds to Client ID.

Arguments (Custom Data)

`fields( fields )` Custom fields that are equivalent to HTTP Request's Form(fields(fields)). Can be specified both in `New OAuth2 Token()` and in `Request Auth()`. Only necessary if the service requires information that is not defined in the OAuth 2.0 standard.

`headers( headers )` Custom headers that are equivalent to `HTTP Request's Headers( headers )`. Can only be specified in `New OAuth2 Token()`. Only necessary if the service requires information not defined in the OAuth 2.0 standard.

Example

```
token = New OAuth 2 Token (
  User( "yourgoogleaccount@gmail.com" ),
  Refresh Token( "1a2b3c4e5F" ),
  Token URL( "https://www.example.com/oauth2/token" ),
  Client ID( "12ab" ),
  Client Secret( "3456dEfG" )
```

```
);
```

See Also

See your API documentation for more information about how to get values such as the client secret and token URL.

`Open Datafeed()`

`Datafeed()`

Description

Creates a Datafeed object and window.

Returns

A reference to the Datafeed object.

Arguments

No arguments are required. You usually set up the basic operation of the data feed within the `Open Datafeed()` command, however.

`Open Help("Help"|"Statistics Index"|"Scripting Index", ...)`

Description

Opens the specified help window.

`Parse XML(`*`string`*`, On Element("tagname", Start Tag(expr), End Tag(expr), Text))`

Description

Parses an XML expression using the `On Element` expressions for the specified XML tags.

Example

```
XMLData = "
<Book name='Foods'>All you want to know
       <Chapter num='1'>Fruit
             <kind>Apple</kind>
             <kind>Cherry</kind>
             <ps>I love dessert!</ps>
       </Chapter>
       <Chapter num='2'>Bread
             <kind>Wheat</kind>
             <kind>Corn</kind>
             <ps>I love sandwiches!</ps>
       </Chapter>
       <Chapter num='3'>Veggy
             <kind>Squash</kind>
             <kind>Cabbage</kind>
             <ps>I love anything else!</ps>
```

```
        </Chapter>
and more.
</Book> ";

// variables are initialized so text can be concatenated
title = "";
subtitle = "";
chap = "";
chapnum = "";
ps = "";

Parse XML( XMLData,
      On Element( "Book",
// capture the name attribute during the start of the Book
            Start Tag( title = XML Attr( "name" ) ),
/* this book has split the subtitle and needs to join the text;
the joined text will be used by endTag.
Text(...) supplies the JSL.*/
            Text( subtitle = subtitle || " -- " || Trim( XML Text() ) ),
/* after endTag processes the variables, set them back
to their initial state, just in case there is a second book
to process in the same XML. */
           endTag( Write( "\!n", title, " ", subtitle ); title = ""; subtitle
  = ""; )
      ),
      On Element( "Chapter",
// capture the chapter number during the start of the Chapter
            Start Tag( chapnum = XML Attr( "num" ) ),
/* the chapter text is joined together, newlines
and extra space is trimmed, and a single space is used to
separate the separated texts. The <kind> tag is ignored by
this ParseXML specification. The <kind> text is processed
by this Text(...) because it wasn't consumed by any other
On Element. */
            Text( chap = chap || Trim( XML Text() ) || " " ),
/* after endTag processes the variables, set them back to
their initial state, because there is another chapter
that needs to start with a clean slate.*/
           endTag( Write( "\!n", chapnum, " ", chap, " ps: ", ps ); chapnum =
  ""; chap = ""; ps = ""; )
      ),
      On Element( "ps", End Tag( ps = XML Text() ) )
);
  1 Fruit Apple Cherry  ps: I love dessert!
  2 Bread Wheat Corn  ps: I love sandwiches!
  3 Veggy Squash Cabbage  ps: I love anything else!
  Foods  -- All you want to know -- and more.
```

```
Platform Preferences(platform(option(value)), ...)
Platform Preference(platform(option(value)), ...)
Set Platform Preferences(platform(option(value)), ...)
Set Platform Preference(platform(option(value)), ...)
```

Description

Sets and resets values for platform options and turns the options on and off.

Arguments

`platform` Specifies the platform of the preference.

`option` Specifies the preference name.

`value` Specifies the preference value.

Notes

Table 2.4 describes the syntax for setting platform preferences.

Table 2.4 Platform Preferences() Syntax

Syntax	Description

`Platform Preferences( <<Default )`
`Platform Preferences( <<Factory Default )`
`Platform Preferences( Default )`
	Resets all platform preferences to their default values.

`Platform Preferences( Platform( <<Default ) )`
`Platform Preferences( Platform( <<Factory Default ) )`
`Platform Preferences( Platform( Default ) )`
	Resets the specified platform preferences to their default values.

`Platform Preferences( Platform( option ( <<Default ) ) )` `Platform Preferences( Platform( option ( <<Factory Default ) ) )`	Resets the specified platform option to its default value.
`Platform Preferences( Platform( option( value, <<On ) ) )`	Sets the value of the specified platform option and turns it on.
`Platform Preferences( Platform( option( value, <<Off ) ) )`	Sets the value of the specified platform option and turns it off.

Example

The following expression selects (or turns on) Set Bin Width in the Distribution platform preferences and sets the value to 2:

```
Platform Preferences( Distribution( Set Bin Width( 2 ) ) );
```

The following expression changes the Set Bin Width value and turns the option off:

```
Platform Preferences( Distribution( Set Bin Width( 2, <<Off ) ) );
```

The following expression resets the default Set Bin Width value and deselects the preference:

```
Platform Preferences( Distribution( Set Bin Width( <<Default ) ) );
```

```
Polytope Uniform Random(samples, A, b, L, U, neq, nle, nge, <nwarm=200>,
<nstride=25>)
```

Description

Generates random uniform points over a convex polytope.

Arguments

`Samples` The number of random points to be generated.

`A` The constraint coefficient matrix.

`B` The right hand side values of constraints.

`L, U` The lower and upper bounds for the variables.

`neq` The number of equality constraints.

`nle` The number of less than or equal inequalities.

`nge` The number of greater than or equal inequalities.

`nwarm` (Optional) The number of warm-up repetitions before points are written to the output matrix.

`nstride` (Optional) The number of repetitions between each point that is written to the output matrix.

Notes

The constraints must be listed as equalities first, less than or equal inequalities next, and greater than or equal inequalities last.

```
Preferences(pref1(value1), ...)
Preference(pref1(value1), ...)
Pref(pref1(value1), ...)
Prefs(pref1(value1), ...)
Set Preferences(pref1(value1), ...)
Set Preference(pref1(value1), ...)
```

Description

Sets preferences for JMP.

Arguments

`Add Files Opened by Scripts to the Recent Files List(Boolean)` Determines whether a file that is opened by a script is added to the Home Window's Recent Files list.

`Analysis Destination(window)` Specifies where to route new analyses.

`Annotation Font("font", size, "style")` Font choice for annotations in reports.

`Axis Font("font", size, "style")` Font choice for axis labels.

`Axis Title Font("font", size, "style")` Font choice for axis titles.

`Background Color( {R, G, B} | <color> )` Sets the background color for windows.

`Calculator Boxing(Boolean)` Turns on boxing to show hierarchy of expressions.

`Conditional Formatting Rules` Creates rules for conditionally formatting text in reports. See "Examples".

`Data Table Font("font", size, "style")` Font choice for data tables.

`Data Table Title on Output(Boolean)` Titles reports with name of data table.

`Date Title on Output(Boolean)` Titles reports with current date.

`Evaluate OnOpen Scripts("always"|"never"|"prompt")` Determines whether an `On Open` table script is run after the user opens the data table. By default, the user is prompted. Their choice is remembered each time they open the data table in the current JMP session. Scripts that execute other programs are never run.

`Excel Has Labels(Boolean)` When on, forces JMP to interpret the first row of data as column headings.

`Excel Selection(Boolean)` When on, the user is prompted for which non-blank Excel worksheets should be imported from an Excel workbook.

`File Location Settings(<Directory Type>("<path>"<,"initial directory">))` Valid directory types are:

`Data Files Directory` Sets the default location for data files.

`Help Files Directory` Sets the default location for help files.

`Installation Directory` By default, this location is set to the JMP installation folder on Windows:

"C:\Program Files\SAS\JMP\17" or "C:\Program Files\SAS\JMPPro\17"

`License File Path` Sets the default location for JMP license file.

`Preferences File Directory` Sets the default location for the preferences settings file.

`Save As Directory` Sets the default location for Save As file operations.

`Foreground Color(color)` Sets the foreground color for windows.

`Formula Font("font", size, "style")` Font choice for the formula editor.

`Graph Background Color(color)` Sets the color for the background area inside the graph frame.

`Graph Marker Size(size)` Default size for drawing markers.

`Heading Font("font", size, "style")` Font choice for table column headings in reports.

`Initial JMP Starter Window(Boolean)` Specifies whether the JMP Starter window is shown at launch.

`Initial Splash Window(Boolean)` Enables you to show or suppress the initial splash screen.

`Maximum JMP Call Depth(size)` Sets the default for the maximum call depth (or stack size) for JMP in which JSL built-in functions, user-defined functions, or `Recurse()` function calls can be made. By default, the maximum call depth is set to 256KB.

Each thread that JMP creates has a 2MB stack by default. Increasing the maximum call depth can cause a physical runtime stack overflow, so incrementally increase this preference in small amounts until you find the best value that works for your JSL script.

`Marker Font("font", size, "style")` Font choice for markers used in plots.

`Monospaced Font("font", size, "style")` Font choice for monospaced text.

`ODBC Suppress Internal Quoting(Boolean)` Prevents internal quoting in SQL statements that contain table and variable names with mixed case and spaces.

`Outline Connecting Lines(Boolean)` Draws lines between titles for same-level outline nodes.

`Print Settings(option(value), ... )` Changes print options on the Page Setup window:

`Margins( <n>, <n>, <n>, <n>)` sets the left, top, right, and bottom margins. Margins are in inches.

`Margins(<n>)` sets all margins to the same value in inches.

`Orientation("portrait" | "landscape")` changes the page's print orientation.

`Headers(<"char">, <"char">, <"char">)` specifies text that appears in the left, middle, and right header.

`Headers(<"char">)` specifies the only text in the header.

`Footers(<"char">, <"char">, <"char">)` specifies text that appears in the left, middle, and right footer.

`Footers(<"char">)` specifies the only text in the footer.

`Scale(<n>)` decreases or increases the percentage at which the content prints.

`Show Explanations(Boolean)` Some analyses have optional text that explains the output.

`Show Menu Tips(Boolean)` Turns menu tips on or off.

`Show Status Bar(Boolean)` Turns display of the status bar on or off.

`Small Font("font", size, "style")` Font choice for small text.

`Text Font("font", size, "style")` Font choice for general text in reports.

`Thin Postscript Lines(Boolean)` macOS only. Specifies that line widths drawn to a Postscript printer be narrower than otherwise.

`Title Font("font", size, "style")` Font choice for titles. Arguments are name of font (for example, `"Times"`), size in points, and style (`"bold"`, `"plain"`, `"underline"`, `"italic"`).

`Use Triple-S Labels as Headings(Boolean)` When on, this argument forces JMP to interpret label names as column headings. Example: Pref(Name("Use Triple-S Labels as Headings")(0)); turns off the preference.

Examples

The following expressions reset all preferences to their default values.

```
Preferences( "Default" );
Preferences( "Factory Default" );
```

The following script creates conditions for formatting text in reports.

```
Preferences(
    Conditional Formatting Rules(
        RuleSet(
           RuleName( "Warning" ),

           // if the value is not equal to 0, format the text as 80% gray
           NotEqualTo( Value( 0 ), Format( TextAlpha( 0.8 ) ) )
         )
    )
);
```

Notes

The preferences for the following areas are not accessible in JSL: Text Data Files, Windows Specific, Mac OS Settings, Fonts, Communications, Script Editor, and JMP Updates.

See Also

"Platform Preferences(platform(option(value)), ...)"`Register Addin("unique_id", "home_folder", <named_arguments>)`

Description

Register a JMP Add-In and load the add-in if it registers successfully.

Returns

If successful, returns a scriptable object representing the registered add-in. If unsuccessful, returns Empty.

Arguments

`unique_id` A quoted string that contains the unique identifier for the add-in. The string can contain up to 64 characters. The string must begin with a letter and contain only letters, numbers, periods, and underscores. Reverse-DNS names are recommended to increase the likelihood of uniqueness.

`home_folder` A quoted string that contains the filepath for the folder containing the add-in files. The filepath must conform to the valid pathname requirement for the host operating system.

`DisplayName( "name" )` An optional, quoted string that contains a name that can be displayed in the JMP user interface wherever add-in names are displayed, instead of the unique ID.

`JMPVersion("version")` An optional string that contains a specific version of JMP. The default value is "All", which enables the add-in to be loaded and run in any version of JMP that supports add-ins. "Current" restricts the use of the add-in to only the current version. Any quoted version number (for example, "7" or "9") restricts the add-in to a single specific version of JMP.

`LoadsAtStartup(Boolean)` An optional Boolean. The default value is True (1), which causes the add-in to be loaded when JMP is started. If the value is False (0), the add-in is not loaded automatically.

`LoadNow(Boolean)` Loads the add-in immediately.

Notes

If a file named addin.def is found in the specified home folder, values from that file are used for any optional arguments that are not included in the `Register Addin()` function.

Example

In the following example, the first argument is the unique identifier. The second argument identifies where the add-in is installed. The third argument is the name that appears where add-in names are displayed (for example, the **View > Add-Ins** menu on Windows).

```
Register Addin("com.company.lee.dan.MyAddIn","$DOCUMENTS/myaddin",
  displayname( "Calculator Addin" ));
```

The second argument becomes the `$ADDIN_HOME` path variable definition. When you refer to the add-in scripts, be sure to include a trailing slash after the path variable.

```
Include("$ADDIN_HOME(com.jmp.jperk.texttocols)/texttocols.jsl");
```

Revert Menu()

Description

Resets your JMP menus to factory defaults.

Run Program(Executable("path/filename.exe"), Options({"/a", "/b", "..."}), Read Function(expression), Write Function(expression), Parameter(expression))

Description

Runs the external program specified by the `Executable` argument, with the command line arguments specified by the `Options` argument.

Results

Returns either a quoted string, a blob, or a `Run Program` object as controlled by the `Read Function` argument.

Arguments

`Executable` The path to the executable. On macOS, type the full path to the executable.

`Options` Command line arguments for the executable.

`Read Function` If `Read Function( "text" )` is specified, a quoted string is returned. If `Read Function( "blob" )` is specified, a blob is returned. The script waits until the external program closes its stdout. `Run Program` then returns all data that the external program has written to its stdout as a quoted string or a blob.

If `Read Function` is not specified, a `Run Program` object is returned.

`Write Function` Optional argument that accepts a function as its value; it does not accept `"text"` or `"blob"`.

`Parameter` Optional argument to read and write the expression in `Read Function`.

Notes:

- Use global variables when `Run Program()` is inside a function.
- The `Run Program` object, which is returned if `Read Function` is not specified, accepts the following messages to read data from the external program's stdout:
  - `<<Read`: reads any available data as a quoted string. If no data is available, an empty string is returned.
  - `<<Can Read`: returns `true` if there is data available to be read.
  - `<<Is ReadEOF`: returns `true` when the external program has completed and all its data has been read.

  You can use these messages to poll for data and process the data as it is produced by the external program.
- A `Run Program` object accepts the following messages to write data to the external program's `stdin`:
  - `<<Write( "text" )`: sends data to the external program's stdin.
  - `<<Can Write`: returns `true` if the external program will accept data immediately; otherwise, calling `<<Write` causes your script to block.
  - `<<WriteEOF`: signals to the external program that you are done sending data to it.
- Instead of sending messages to the returned `Run Program` object, you can specify the `Read Function` argument as an inline function. `RP` is the `Run Program` object.

```
RP = Run Program(
  Executable( ... ),
  Read Function(
    Function( {RP},
      <your code here>
```

```
            RP << Read
        )
    )
);
```

The `Parameter(optParm)` argument is optional in `Read Function`. If specified, the functions defined for `Read Function` and `Write Function` can receive a second argument, which is the value of `optParm`.

Examples

The following script is an example of the `Write Function` argument. `RP` is the `Run Program` object. In this context, it accepts the `<<Write` and `<<WriteEOF` messages.

```
RP = Run Program(
   Executable( ... ),
   Write Function(
      Function( {RP},
         <your code here>
         RP << Write( "Program finished." )
      )
   )
);
```

The following script shows an example of `Parameter(optParm)` argument:

```
RP = Run Program(
   Executable( ... ),
   Parameter( x ),
   Read Function( Function( {RP, optParm},... ) )
);
```

Within the `Read Function`, `optParm` contains the value of *x*. Do not attempt to access the `optParm` argument in your function if you have not specified a `Parameter` argument.

Schedule(n, script)

Description

Queues an event to run the *script* after *n* seconds.

Set Clipboard(*string*)

Description

Evaluates the quoted *string* argument looking for a character result, and then places the string on the clipboard.

Example

```
Set Clipboard( "copy me" );
```

`SetJVMOption( Version("<version number>") )`

Description

Sets the Java Runtime Environment (JRE) version that you want JMP to use (rather than the version installed with JMP). This script must be run before JMP connects to the JRE.

Argument

`version` (Windows only) In the Windows registry, there are two requirements for the JavaSoft/Java Runtime Environment key: the key must include a quoted string called "RuntimeLib" that points to a valid jvm.dll. And the Java Runtime Environment key must include a key named after the quoted JVM version number.

`Set Platform Preference()`

`Set Platform Preferences()`

See "Platform Preferences(platform(option(value)), ...)".

`Set Preference()`

`Set Preferences()`

See "Preferences(pref1(value1), ...)".

`Set Toolbar Visibility( "toolbar name" | default | all, window type | all, "true" | "false" )`

Description

On Windows, shows or hides a toolbar based on the window type or for all windows.

Arguments

`toolbar name | default | all` The internal name of the toolbar (see the **View > Toolbars** list in JMP), the default toolbar for the specified window type, or all toolbars. Include quotes around `"toolbar name"`.

`window type | all` Data table, script, report, journal, or all windows.

`true | false` A quoted string that shows or hides the toolbar.

```
Shortest Edit Script( A, B )
Shortest Edit Script( strings( A, B, <matrix( 0|1 )>, <limit( number )> ) )
Shortest Edit Script( lines( A, B, <matrix( 0|1 )>, <limit( number )>,
<separators( characters )>, <ignore( characters )|ignore white space( )> ) )
Shortest Edit Script( sequences( nA, nB, Function( {iA, iB}, adata[iA] ==
bdata[ib] ) ) )
```

Description

Compares two quoted strings, lines, or sequences.

Returns

Returns a list or a matrix of edit commands. The simplest form returns a list. quoted `strings` and `lines` return a matrix (if set to 1) or a list. `sequences` returns a matrix.

There are three possible commands: common data in both quoted strings, delete data from the first string, and keep data from the second string.

Optional Arguments

`matrix` Indicates whether the returned value is a matrix.

`limit` Stops the evaluation when the edit list exceeds the specified number of inserted or deleted items. Two quoted random strings have a lot of common characters in a lot of distinct sections. The function runs for a long time trying to find a best match. `limit` stops the function sooner.

Optional Lines Arguments

`matrix` Indicates whether the returned value is a matrix.

`limit` Stops the evaluation when the edit list exceeds the specified number of inserted or deleted items. Two quoted random strings have a lot of common characters in a lot of distinct sections. The function runs for a long time trying to find a best match. `limit` stops the function sooner.

`separators` A character that separates words.

`ignore` Ignores the specified spaces or characters in a line.

`ignore white space` Ignores white space in a line.

Optional Sequences Argument

`Function` A user-defined function.

Examples

The following example compares two quoted strings with three common sequences of characters between them.

```
Shortest Edit Script( "abcdef", "abdezgh" );
  {{"Common", "ab"}, {"Remove", "c"}, {"Common", "de"}, {"Insert", "zgh"},
    {"Remove", "f"}}
```

The following example examines each line in quoted string aa and bb:

```
aa = "this is
```

```
a test of
shortest
edit script
lines with several words";

bb = "this is
a test 2 of
shortest
edit ?., script
lineswithseveral words";

Shortest Edit Script( lines( aa, bb, separators( "\!n" ),

  // quote and newline separators
  ignore( "?., " ) ) ); // ignore these characters and spaces
  {{"Common", "this is // lines in aa and bb contain "this is"
  "}, {"Remove", "a test of // only on line 2 of aa
  "}, {"Insert", "a test 2 of // only on line 2 of bb
  "},
  {"Common", "shortest
  edit script
  lines with several words"}}
  // lines in aa and bb contain "shortest", "edit script", and "lines with
     several words"
```

Show Addin Builder Dialog()

Description

Opens a window in which you can make custom add-ins.

Show Addins Dialog()

Description

Opens the Add-In Status window (**View > Add-Ins**).

Arguments

None.

Show Commands()

Description

Lists scriptable objects and operators. Arguments are All, DisplayBoxes, Scriptables, Scriptable Objects, StatTerms, Translations.

`Show Preferences(<"all">)`

Description

Shows current preferences. If no argument is specified, preferences that have been changed are shown. If `"all"` is given as the argument, all preferences are shown.

`Show Properties(object)`

Description

Shows the messages that the given *`object`* can interpret, along with some basic syntax information.

`Sobol Quasi Random Sequence(nDim, nRow)`

Description

Generates a sequence of space-filling quasi random numbers using the Sobol sequence in up to 4000 dimensions.

`Socket(<STREAM | DGRAM>)`

Description

Creates a socket.

Returns

The socket that was created.

Arguments

`STREAM | DGRAM` Optional argument to specify whether the socket is a stream or datagram socket. If no argument is supplied, a stream socket is created.

`Speak(text, <wait(Boolean)>)`

Description

Calls system's speech facilities to read aloud the text. If Wait is turned on, script execution pauses until speaking is done.

`Status Msg("message")`

Description

Writes the quoted message string to the status bar.

Subtract(a, b)

a-b-...

Description

Subtracts the values of the listed arguments, left to right. No arguments are changed.

Returns

The difference.

Arguments

Two or more variables, numbers, or matrices.

Notes

Two or more arguments are permitted.

Unregister Addin("unique_id")

Description

Unregisters (removes) a previously registered add-in.

Argument

`unique_id` A quoted string that contains the unique identifier for the add-in to unregister.

Web(*string*, <JMP Window>)

Description

Opens the URL stored in the quoted *string* in the default web browser.

The `http://` prefix in the URL is optional.

Examples

```
url = "www.jmp.com"; // open the URL in the default web browser
Web( url );

Web( "www.jmp.com" ); // open the URL in the default web browser

Web( "www.jmp.com", JMP Window ); // open the URL in the JMP browser window
```

XML Attr("attr name")

Description

Extracts the quoted string value of an XML argument in the context of evaluating a `Parse XML` command.

XML Decode("xml")

Description

Decodes symbols in XML to ordinary text. For example, `&` becomes &, and `<` becomes <.

Argument

`xml` A quoted string containing XML.

XML Encode("text")

Description

Prepares text for embedding in XML. For example, & becomes `&`, and < becomes `<`.

Argument

`xml` A quoted string containing plain text.

XML Text()

Description

Extracts the quoted string text of the body of an XML tag in the context of evaluating a `Parse XML` command.

Chapter 3

JSL Messages

Summary of Messages for Objects and Display Boxes

This topic provides abbreviated descriptions for many of JMP's general object messages. For complete information about object messages, see the JMP Scripting Index. In JMP, select **Help > Scripting Index**.

For information about platform messages, see *Scripting Guide*.

Contents

Alpha Shape Messages . . . 374
Associative Array Messages . . . 374
Class Messages . . . 375
Data Table Messages . . . 377
General Data Table Messages . . . 377
Column Messages . . . 407
Row Messages . . . 414
Data Filter Messages . . . 416
Data Feed Messages (Windows Only) . . . 419
Display Box Messages . . . 422
All Display Box Messages . . . 422
Axis Box Messages . . . 432
Border Box Messages . . . 437
Data Browser Box Message . . . 438
Data Filter Source Box Message . . . 438
Frame Box Messages . . . 438
Display 3D Box Message . . . 440
Excerpt Box Message . . . 441
Filter Col Selector Message . . . 441
Global Box Message . . . 441
Hier Box Message . . . 441
Matrix Box Messages . . . 441
Nom Axis Box Messages . . . 442
Number Col Box Messages . . . 443
Number Col Edit Box Messages . . . 446
Number Edit Box Message . . . 446
Outline Box Messages . . . 447
Panel Box Messages . . . 448
Plot Col Box Messages . . . 448
Slider Box and Range Slider Box Messages . . . 449
String Col Box Messages . . . 460
Tab Box Messages . . . 451
Table Box Messages . . . 452
Text Box Messages . . . 455
Tree Node and Tree Box Messages . . . 456
Triangulation Messages . . . 458
Window Messages . . . 460
Dynamic Link Library (DLL) Messages . . . 462
Image Messages . . . 464

Interactive HTML Messages 477
Web Report Messages 477
JMP Application Messages 468
JMP App Messages 469
JMP App Module Messages 470
JMP App Module Instance Messages 471
MATLAB Messages 471
Namespace Messages 475
Platform Messages 477
Bubble Plot Messages 482
DOE Messages 483
Partition Messages 483
Response Screening Messages 484
Tabulate Messages 484
Python Integration Messages 485
R Integration Messages 488
SAS Integration Messages 492
Metadata Server Object Messages 492
SAS Server Object Messages 493
Stored Processes Messages 502
SAS Results Messages 508
Schedule Messages 510
Segment Message 511
Sockets Messages 512
SQL Messages 515
Messages for Other Objects 519
Zip Archives 519
Journals 519

Alpha Shape Messages

For the following JSL messages, `ashape` stands for an alpha shape or a reference to one.

`ashape <<Get Alpha`

Returns the current alpha value.

`ashape <<Set Alpha(`*`alpha`*`)`

Sets the current *`alpha`* value and recomputes the triangulation.

`ashape <<Get Tri Alpha`

Returns the alpha values for each triangle.

Associative Array Messages

For the following JSL messages, `map` stands for an associative array or a reference to one.

`map<<First`

Returns the first key within *`map`*, or `Empty()` if *`map`* has no keys. Note that keys are returned in lexicographical order.

`map<<Get Contents`

Returns a list of all key-value pairs within *`map`*.

`map<<Get Keys`

Returns a list of all the keys within *`map`*.

`map<<Get Default Value()`

Returns the implicit value of all absent keys, or `Empty()` if none has been set.

`map<<Get Value(`*`key`*`)`

Returns the value for the *`key`* within *`map`*.

map<<Get Values(*<{keyList}>*)

If no argument is provided, a list of all values within *map* is returned.

If a list of keys is provided, a list of the values corresponding to only those keys is returned.

map<<Insert(*key*, *value*)

Inserts the *key* into *map* and assigns *value* to it. If *key* already exists in *map*, its value is replaced by the new *value* given. This message is equivalent to the function Insert Into.

map<<Next(*key*)

Returns the key following the given *key* within the *map*, or Empty() if *map* has no keys. Note that keys are returned in lexicographical order.

map<<Remove(*key*)

Removes the *key* and *value* from *map*. This message is equivalent to the function Remove From.

map<<Set Default Value(*v*)

Sets the implicit value of all absent keys. Any key added without a value is assigned this value by default.

Class Messages

obj<<Clone

Returns a reference to a new class object that is a copy of the *obj* class object.

obj<<Contains(*quoted string*)

Returns 1 if the *obj* class object contains the specified quoted string expression, and 0 otherwise.

obj<<Delete Class

Deletes the *obj* class object.

obj<<Equal(*classref*)

Returns 1 if the *classref* class object is equal to the *obj* class object, and 0 otherwise.

obj<<First

Returns the quoted string representation of the name of the first member (item) in the *obj* class object. The members (items) in the class object are sorted in alphabetical order.

obj<<Get Contents

Returns a list of members (items) in the *obj* class object. Each element in the list is a two-item list that contains a key and an associated value.

obj<<Get Keys

Returns a list of keys within the *obj* class. Each key is a quoted string representation of the name of a member (item) in the *obj* class object.

obj<<Get Name

Returns a quoted string representation of the name of the *obj* class object.

obj<<Get Value(*key quoted string*)

Returns the value of the specified member (item) within the *obj* class object. The quoted *key quoted string* argument specifies the key to the member (item).

obj<<Get Values

Returns a list of values of the members (items) in the *obj* class object. Each element in the list is the expression that represents the value of each member (item) in the class.

obj<<Insert(*quoted string*, *value*)

Inserts a member (item) into the *obj* class object. The quoted *string* argument is the name of the member (item), and the *value* argument is the expression value of the member (item).

obj<<Lock Class(<*quoted string*|*{quoted stringList}*>)

Locks the *obj* class object, or locks specific members (items) within the *obj* class object. When a class object is locked, members (items) cannot be added, changed, or removed. The quoted *string* or quoted *stringlist* arguments specify a member (item) to lock. You can also specify a list of quoted strings to lock multiple members (items).

obj<<N Items

Returns the number of members (items) in the *obj* class object.

obj<<Next(*quoted string*)

Returns the quoted *string* representation of the name of the member (item) in the *obj* class object that follows the member (item) specified by the quoted *string*. The members (items) in the class object are sorted in alphabetical order.

obj<<Remove(<*string*|*{stringList}*>)

Removes the member (item) specified by the quoted *string* or quoted *stringlist* from the *obj* class. You can remove multiple members (items) using a list of quoted strings.

obj<<Show Contents

Shows the contents of the *obj* class object in the log window.

obj<<Unlock Class(<*quoted string*|*{stringList}*>)

Unlocks the *obj* class object, or unlocks specific members (items) within the *obj* class object. When a class object is unlocked, members (items) can be added, changed, or removed. The quoted *string* or *quoted stringlist* specify a member (item) to unlock. You can also specify a list of quoted strings to unlock multiple members (items).

Data Table Messages

This section contains JSL messages for JMP data tables.

General Data Table Messages

dt<<Add Column Properties(*property argument*, ...)

Adds the specified properties (such as Value Order and Missing Value Codes) to the selected column.

dt<<Add Multiple Columns(*column prefix*, *n*, <"Before First"|"After Last"|After(*column*)>, "Character"|"Numeric"|"Row State", <Field Width(*n*)>)

Description

Adds *n* columns to *dt* at the position indicated.

Required Arguments

column prefix The prefix to add to the new columns names.

n The number of columns to add.

Character A new character column.

Numeric A new numeric column.

Row State A new row state column.

Optional Arguments

Before First Adds the columns before the first column.

After Last Adds the columns after the last column.

After(*column*) Adds the columns after the specified column.

Field Width(*n*) Specifies the width of the columns.

Notes

If you omit arguments, or the arguments are incorrectly specified, the Add Multiple Columns window appears.

---

dt<<Add Rows(<*n*>, <"At Start"|"At End"|After(*row number*)>|{*column name=value pairs*})

Description

Adds rows at the start, at the end, or after a specified row in the data table. The message can also add rows based on the specified column name and value pairs. Those rows are added to the end of the data table.

Notes

If you omit arguments, or the arguments are incorrectly specified, the Add Rows window appears.

---

dt<<Add Scripts to Table(*script*, ...)

dt<<Add Properties to Table(*script*, ...)

Adds the specified scripts to the data table.

---

dt<<Anonymize(<Columns*(column list(s))*>, <Output Table Name(*name*)>);

Removes unique identifiers from data, some column properties, and table scripts. Applies to a data table or the specified list of columns. The new data table has the name that is specified by the quoted *name* argument.

---

dt<<Begin Data Update

Holds off display updating to allow for quick updating of data table cells. Use End Data Update in conjunction with this command to turn display updating back on.

Notes

`Begin Data Update` does not affect the data refresh due to some other table manipulations. For example, when you delete or add columns, the data table is updated and then the data update begins.

`dt<<Clear Column Selection`

Deselects all selected columns.

`dt<<Clear Edit Lock(<"Modify Cells">, <"Add Rows">, <"Add Columns">, <"Delete Rows">, <"Delete Columns">)`

Description

Allows the specified data table operation again.

Notes

If no arguments are specified, all locks are cleared.

`dt<<Clear Row States`

Cancels any row states in effect.

`dt<<Clear Select`

Turns off the current selection.

`dt<<Clone Formula Column(`*`column`*`, `*`n`*`, Substitute Column Reference(`*`column1`*`, `*`{list}`*`))`

Creates *n* new formula columns, substituting references to *column1* with columns from the *list* into the formula from the original *column*.

`dt<<Close Data Grid(`*`Boolean`*`)`

If true, closes the data table grid.

`dt<<Close Side Panels(`*`Boolean`*`)`

If true, closes the side panel in a data table.

`dt<<Color or Mark by Column(`*`column`*`, <`*`named arguments`*`>)`

`dt<<Color by Column(`*`column`*`, <`*`named arguments`*`>)`

`dt<<Marker By Column(`*`column`*`, <`*`named arguments`*`> );`

Description

Assigns colors or markers according to the values of a data table column. If no optional arguments are provided, colors are assigned according to the default color theme.

Required Argument

`column` The column to color or mark.

Optional Named Arguments

`Color(`*`n`*`)` Uses the specified JMP color.

`Add Marker(`*`Boolean`*`)` Shows or hides the marker in the data table.

`Color Theme(`*`color theme`*`)` Uses the specified quoted *`color theme`*.

`Marker Theme(`*`marker theme`*`)` Uses the specified quoted *`marker theme:`* `"Standard"`, `"Hollow"`, `"Paired"`, `"Classic"`, or `"Alphanumeric"`.

`Continuous Scale|Continuous Scale(`*`Boolean`*`)` Assigns colors in a chromatic sequential fashion based on the values in the highlighted column.

`Reverse Scale|Reverse` Reverses the color scheme in use.

`Excluded Rows(`*`Boolean`*`)` If true, applies the row states to excluded columns.

`"Make Window with Legend"` Creates a separate window with a legend.

`dt<<Color Rows by Row State`

Colors the rows in the data table grid using the color assignments by row states. Send the message again to turn off the row colors.

`dt<<Combine Columns(Delimiter("`*`delim`*`"), Columns(`*`column1`*`, `*`column2`*`, etc.), Column Name(`*`quoted string`*`))`

Combines several columns into a single column. Each source columns' values are separated by the delimiter specified as the quoted *`delim`* argument.

Examples

```
dt = Open( "$SAMPLE_DATA/Consumer Preferences.jmp" );
dt << Combine Columns(
   Delimiter( "," ),
   Columns(
      :Brush After Waking Up,
      :Brush After Meal,
      :Brush Before Sleep,
      :Brush Another Time
   ),
   Column Name( "When to Brush" )
```

```
);
```

`dt<<Compress File When Saved(`*`Boolean`*`)`

Compresses the file when the data table is saved.

`dt<<Compress Selected Columns(`*`{column1, ...}`*`)`

Compresses the listed columns into the most compact form that is possible. Columns with character data are compressed to 1 byte if there are fewer than 255 levels. Columns with numeric data are compressed to 1 byte if the integers are between -127 and 127.

`dt<<Concatenate(`*`dt2`*`|Data Table`*`(name)`*`|Multiple Data Table(`*`name`*`) arguments, (<"Private"|"Invisible">), <Output Table Name(`*`name`*`)>|"Append to First Table">, <"Keep Formulas">, <"Create Source Column">)`

Description

Creates a new table (*name*) from the rows of *dt* and *dt2*. By default, Concatenate creates a new data table and appends the rows of each data table that is specified.

Returns

A reference to the concatenated data table.

Required Arguments

`dt2|Data Table(`*`name`*`)|Multiple Data Table(`*`name`*`)` A data table reference or the names of the data table or data tables that you would like to combine.

Optional Arguments

`"Private"` A quoted keyword that opens the data table without displaying it in a data table window.

`"Invisible"` A quoted keyword that hides the data table. Use this argument to keep the data table hidden but use it in a subsequent expression. The data table is displayed in the Home Window's Window List and the Window > Unhide list.

`Output Table name(`*`name`*`)` The name of the final data table. If you do not enter a name, JMP names the data table Untitled # (for example, Untitled 1).

`"Append to First Table"` Appends rows to the first data table reference or data table name in the first argument. This option is an alternative to creating a new data table.

`"Keep Formulas"` Includes formulas in the final data table.

`"Create Source Column"` Adds a column called Source Table to the new data table.

Notes

`"Private"` and `"Invisible"` only apply if *not* using `"Append to First Table"`.

dt<<Copy Column Properties

Copies all of the column properties for the selected columns into a list of separate lists of properties. Optionally, you can specify a list of source columns instead of preselecting them in the data table.

Example

```
dt = Open( "$SAMPLE_DATA/Tiretread.jmp" );
dt << Select Columns( :MODULUS, :ELONG );
dt << Copy Column Properties;
New Window( "Script", Script Box( "//Try paste here" ) );
```

or

```
dt = Open( "$SAMPLE_DATA/Tiretread.jmp" );
dt << Copy Column Properties( {:MODULUS, :ELONG} );
New Window( "Script", Script Box( "//Try paste here" ) );
```

dt<<Copy Selected Properties

Description

Copies the selected table properties to the clipboard.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Select Properties( {"Distribution", "Oneway"} );
proplist = dt << Copy Selected Properties();
New Window( "Script", Script Box( "//Try pasting here" ) );
```

dt<<Copy Table Script("No Data")

Copies the script to recreate the data table onto the clipboard so that it can be pasted somewhere else. Add the "No Data" argument to omit the data.

dt<<Copy Table Scripts

dt<<Copy Selected Properties

Description

Copies the selected scripts to the clipboard.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Select Properties( {"Distribution", "Oneway"} );
proplist = dt << Copy Table Scripts();
New Window( "Script", Script Box( "//Try pasting here" ) );
```

dt<<Data Filter(<Location(*x*, *y*), <"Close Outline">, <"Local">, <Inverse(*Boolean*)>, <Show Columns Selector(*Boolean*)>, <Title(*quoted*

string)>, <Save and Restore Current Row States(*Boolean*)>, <Conditional(*Boolean*)>, <Auto Clear(*Boolean*)>, <Group By AND(*Boolean*)>, <Show Histograms and Bars(*Boolean*)>, <Count Excluded Rows(*Boolean*)>, <Mode(...)>, <Add Filter((*cols*(...), <Where(...)>, <Display(...)>, <Select Missing(*cols*)>, <Order By Count(*cols*)>)>, <Favorites(...)>, <Animation(...)>)

Constructs a data filter. If no arguments are specified, the Add Filter Columns window appears.

Optional Arguments

Location(*x*, *y*) Moves the data filter window to the specified location. *x* and *y* are measured in pixels. 0,0 is the top left of the monitor.

"Close Outline" Closes the data filter outline.

"Local" Enables the filter to be embedded in reports to filter one or more platforms without affecting other reports.

Inverse(*Boolean*) Selects all but the specified rows for all filters.

Show Columns Selector(*Boolean*) If true, a column list is shown that adds a new column to the filter.

Title(*quoted string*) The title that is displayed on the outline.

Save and Restore Current Row States(*Boolean*) Restores your current row states when the Data Filter window is closed.

Conditional(*Boolean*) Limits the categories displayed for the selected filter column.

Auto Clear(*Boolean*) If you have more than one nominal or ordinal column selected in the Data Filter, this option clears any other selections before making a new selection.

Group By AND(*Boolean*) Enables you to create a filter group, specify OR, and add one or ore filters to create second filter group. If you specify Grouped By And, the behavior is reversed and grouped by AND instead.

Show Histograms and Bars(*Boolean*) Shows or hides the histogram and bars in the data filter.

Count Excluded Rows(*Boolean*) Shows or hides the number of excluded rows.

Mode The three modes of filtering: Select(*Boolean*) shows or hides the selected rows in the data table in a highlighted state; Show(*Boolean*) shows or includes the unselected rows and shows the Hide icon; Include(*Boolean*) shows or includes the unselected rows and shows the Exclude icon.

The global data filter default is Select(), Show(0), and Include(0), The local data filter default is Show(1), Include(1), Select() is not a valid option.

Add Filter Creates the data filter. Arguments include Columns(), Where(), Display(), Select Missing(*cols*), and Order By Count(*cols*). Columns() takes one or more column names separated by commas. You can add one or more Where clauses to define the filter.

Where Defines a Where clause by which the data is filtered.

`Display(column, size, display type)` Sets how the specified categorical column levels are displayed in the filter. The arguments are Blocks Display, List Display, Single Category Display, Check Box Display, Radio Box Display. In categorical columns, you can include the `Find(Set Text(quoted string))` argument to include and initialize the search field. `Display` can also be included for a continuous column and can contain a `size` argument.

`Select Missing Cols(cols)` Selects missing values in continuous columns.

`Order by Count(cols)` For a categorical column, this option sorts the values in decreasing order by count.

`Favorites` Saves the current data filter criteria as a favorite.

`Animation` Cycles through the sorted values of the specified column, selecting and deselecting rows. Optional arguments include `Animate Column(col)`, `Animate Rate(number)`, and `"Forward"|"Backward"|"Bounce"`highlights values from first to last. `"Backward"` highlights values from last to first. `"Bounce"` highlights forward and then backward repeatedly.

`dt<<Data View(<named arguments>)`

Description

Duplicates the data table in a new window. If you specify one of the following quoted arguments, the new data table includes only the corresponding rows.

Returns

A reference to the data view.

Optional Named Arguments

`Excluded` The new data table includes only the rows that are marked as excluded in the original data table.

`Labeled|Labelled` The new data table includes only the rows that are marked as labeled in the original data table.

`Hidden` The new data table includes only the rows that are marked as hidden in the original data table.

`Selected` The new data table includes only the rows that are selected in the original data table.

`dt<<Delete Columns(column1, column2, ...)`

`dt<<Delete Column`

Description

Deletes one or more columns from the data table *dt*. Specify which column or columns to delete. Without an argument, deletes the selected columns, if any.

`dt<<Delete Rows(<n>)`

`dt<<Delete Rows({n, o, p, ...})`

`dt<<Delete Rows({n::q})`

`dt<<Delete Rows([n, o, p])`

`dt<<Delete Row(preceding arguments)`

Description

Deletes the currently selected rows or rows specified. Returns the number of rows that were deleted.

`dt<<Delete Scripts(table script name|{table script1, table script2, ...})`

Description

Deletes the specified data table script or scripts with the quoted name or names, or deletes a list of data table scripts.

Notes

In JMP versions prior to 14, use `Delete Table Property` to delete a table script.

`dt<<Delete Table Property(name|{property1, property2, ...})`

Deletes a table property (for example, a script or variable) with the quoted *name*.

`dt<<Delete Table Variable(name)`

Deletes a table variable with the quoted *name*.

`dt<<Disable Undo(Boolean)`

If true, disables undo operations in the data table.

`dt<<End Data Update`

Resumes display updating after a `Begin Data Update` message. These commands are used for quick updates of the data table when many changes have to be made. Speed is gained by turning off display updating.

`dt<<Exclude`

`dt<<Unexclude`

Toggles selected rows in *dt* from excluded to unexcluded or *vice versa*.

dt<<Get All Columns As Matrix

Returns the values from all columns of *dt* in a matrix. Character columns are numbered according to the levels, starting at 1.

dt<<Get As Matrix(*<{list of columns by name}>|<{list of columns by number}>, <column range>*)

Returns values from the numeric columns of *dt* in a matrix. The default output is all numeric columns.

Examples

```
dt1 = Open( "$SAMPLE_DATA/Big Class.jmp" );
cols = dt1 << Get As Matrix(); // returns all numeric columns
Show( cols );
   cols =
   [ 12 59 95,
      12 61 123,
      12 55 74,...]
colnums = dt1 << Get As Matrix( {4, 5} ); // returns columns four and five
Show( colnums );
   colnums = [ 59 95, 61 123, 55 74, 66 145, 52 64, 60 84, 61 128, ...]

dt2 = Open( "$SAMPLE_DATA/Probe.jmp" );
colrange = dt2 << Get As Matrix( 10::22); // returns columns 10 through 22
Show( colrange );
   colrange =
   [ -0.08818069845438 0.711340010166168 1.85904002189636 0.396923005580902
      4.50656986236572 7.86504983901978 1.53891003131866 -2.76178002357483
      0.0711032971739769 5.75577020645142 -3.62023997306824 -0.971698999404907
      -0.0525696985423565, ...]
```

dt<<Get As Report

Returns the data table as a report. If rows and columns are selected in the data table, only those rows and columns are in the report.

Example

The following script returns `Big Class.jmp` as a report and displays it and a distribution in one window.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dtRpt = dt << Get As Report;
distRpt = V List Box(
   dt << Distribution(
      Continuous Distribution( Column( :weight ) ),
      Nominal Distribution( Column( :age ) )
   )
);
New Window( "Example", H List Box( dtRpt, distRpt ) );
```

`dt<<Get Cell Height`

Returns the data table cell height in pixels.

`dt<<Get Column Names( <data type>, <modeling type>, <string>)`

Description

Returns a list of column names that match the specified type. More than one type can be specified. If no arguments are provided, all column names are returned.

Optional Arguments

data type Valid values: `Numeric`, `Character`, and `RowState`.

modeling type Valid values: `Continuous`, `Ordinal`, and `Nominal`.

string Returns a list of quoted strings.

`dt<<Get Column Reference({list of column names}|[matrix of column numbers])`

`dt<<Get Column References({list of column names}|[matrix of column numbers])`

Returns the column references of the quoted strings in the list or matrix. If no list or matrix is used, JMP returns a list of references to all columns.

`dt<<Get Display Width`

Returns the column display width in pixels.

`dt<<Get Edit Lock`

Returns a list of the disallowed operations on the data table (if cells cannot be edited; rows cannot be added or deleted; and columns cannot be added or deleted).

`dt<<Get Excluded Columns`

Returns the currently excluded columns in the data table.

`dt<<Get Excluded Rows`

Returns the rows that are excluded in the data table.

`dt<<Get Header Height`

Returns the column header's display height (in pixels).

dt<<Get Hidden Columns

Returns the columns that are hidden in the data table.

dt<<Get Hidden Rows

Returns the currently hidden rows in the data table.

dt<<Get Journal

Returns a quoted string that contains journal source for the display box.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
biv = dt << Bivariate( Y( :weight ), X( :height ) );
rbiv = biv << Report;
Print( rbiv << Get Journal );
```

dt<<Get Label Columns

dt<<Get Labeled Columns

dt<<Get Labelled Columns

Returns the currently labeled columns in the data table.

Example

In PopAgeGroup.jmp, the Country and Year columns are labeled. The following script returns a list of the labeled column names.

```
dt = Open( "$SAMPLE_DATA/PopAgeGroup.jmp" );
dt << Get Labeled Columns;
   {:Country, :Year}
```

dt<<Get Labeled Rows

dt<<Get Labelled Rows

Returns the currently labeled rows in the data table.

dt<<Get Name

Returns the name of the data table.

Optional Argument

Ignore Extension Returns the name of the data table without the extension.

dt<<Get Path

Returns the absolute path for the JMP data table. Note that this function is not for imported data that is not saved yet.

dt<<Get Property(*name*)

Returns the script from the quoted property *name*.

dt<<Get Row Change Function

Returns the expression that is evaluated when a row is selected.

dt<<Get Row ID Width

Returns the row ID display width in pixels.

dt<<Get Row States

Returns a vector containing the row state for every row in the data table or data filter.

dt<<Get Rows Where(*where clause*)

Returns the rows in the data table that match the specified Where criteria. Here are some examples:

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Get Rows Where( :sex == "M" );
dt << Get Rows Where( :sex == "M" & :age < 15 );
```

dt<<Get Script(<*script name*>)

Returns the script specified by the quoted *script name*. If the *script name* is omitted, Get Script returns a text representation of the data table and all scripts in the table.

dt<<Get Script Group(<*group name*>)

Description

Returns the list of table scripts in the quoted *group name*. If no group name is specified, a list of all table scripts in all groups is returned.

dt<<Get Script Group Names

Description

Returns the list of names of table script groups.

dt<<Get Scroll Locked Columns

Returns a list of columns that are locked from scrolling.

`dt<<Get Selected Columns(<quoted string>)`

Description

Returns a list of selected columns as column references. Include the quoted *string* argument to return the selected column names as a list of strings.

`dt<<Get Selected Properties(<{list of properties}>)`

Description

Returns the selected table properties in a list.

Optional Argument

list of properties Specifies the properties to get.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Select Properties( {2, 4} );
proplist = dt << Get Selected Properties();
// returns the second and fourth table scripts and highlights them
// in the data table
```

`dt<<Get Selected Rows()`

Returns the selected rows.

`dt<<Get Table Script Names()`

Returns a list of the names of all the scripts and properties in the data table.

`dt<<Get Table Variable(name)`

Returns the value of the quoted *name* variable.

`dt<<Get Table Variable Names`

Returns a list of the names of all the variables in the data table.

`dt<<Go To Row(n)`

Locates and selects row number *n* in *dt*.

---

dt<<Group Columns(*{column1, column2, ...}*)

dt<<Group Columns(*group name*, *column*, *n*)

dt<<Group Columns(*first column*, *n*)

Description

Groups the columns under the specified quoted *group name*. You can provide either a list of columns to group, or a column name and the number of columns to group. In the latter case, the number *n* specifies to group the column given with the *n*-1 columns that follow.

---

dt<<Group Scripts(*{script1, script2, ...}*)

Description

Groups a list of table scripts in the data table.

---

dt<<Hide

dt<<Unhide

Toggles selected rows in *dt* from hidden to unhidden or *vice versa*.

---

dt<<Hide and Exclude

Hides the selected rows from graphs and excludes them from contributing to calculations.

---

dt<<Invert Column Selection(*<{list of columns}>*)

Description

Selects any column that is currently deselected and deselects any column that is currently selected. If the *list of columns* is specified, the columns that are not in the list are selected.

---

dt<<Invert Row Selection

Selects any row that is currently deselected and deselects any row currently selected.

---

dt<<Is Dirty

Returns 1 if the table has been modified from its saved state. Otherwise, returns 0.

---

dt<<Join(With(*Data Table(name)*), (<"Private">|<"Invisible">), Select(*columns*), Select With(*columns*), (By Matching Columns(*column1=column2, ...*)|"Cartesian"|"By Row Number"), <"Merge Same Name Columns">, <"Match Flag">, <Copy Formula(*Boolean*)>, <Suppress Formula Evaluation(*Boolean*)>, <"Update">, <Drop

`Multiples(Boolean, Boolean)>, <Include Non Matches(Boolean, Boolean)>, <"Preserve Main Table Order">, <Output Table Name(name))>`

Description

Combines data tables *dt* and *Data Table* side to side.

Returns

A data table.

Required Arguments

`With(Data Table(name))` Specifies the data table to join with the active table.

`"Private"` Specifies a quoted keyword that opens the data table without displaying it in a data table window.

`"Invisible"` Specifies a quoted keyword that hides the data table. Use this argument to keep the data table hidden but use it in a subsequent expression. The data table is displayed in the Home Window's Window List and the Window > Unhide list.

`Select(columns)` Selects the data table to join with the active table.

`Select With(columns)`

`By Matching Columns(column1=column2)` Selects columns in both tables whose values and data types match.

`"Cartesian"` Joins two tables using a Cartesian fashion, where it forms a new data table consisting of all possible combinations of the rows from two original data tables. JMP crosses the data in the first table with the data in the second to display all combinations of the values in each set.

`"By Row Number"` Joins the two tables side by side.

`"Merge Same Name Columns"` Data from the second table replaces the data of the same name columns in the original table. Note that missing values in the first table are replaced by nonmissing values in the second.

`"Match Flag"` Determines whether the Match Flag column is created when you are matching by column.

`Copy Formula(Boolean)` Includes formulas from the main table and/or the second table in the output columns.

`Suppress Formula Evaluation(Boolean)` Prevents JMP from evaluating columns' formulas during the creation of the new table.

`"Update"` Column data from the second table change the data of the same name columns in the original table. The results are displayed in a new data table. Note the following: JMP does not replace data with missing values; the output table uses the same columns as the original table. Thus, when you use `"Update"`, `Select Columns`, the `"Update"` option is available only when joining by row number or by matching columns.

`Drop Multiples(Boolean, Boolean)` Specifies that you want the new table to contain only one row for each name. Applies only when matching by columns.

`Include Non Matches(Boolean, Boolean)` Includes non-matching columns in the main table and new data table. Applies only when matching by columns.

`"Preserve Main Table Order"` Maintains the order of the original data table in the joined table, instead of sorting by the matching columns.

`Output Table Name(`*`name`*`)` Specifies the name of the joined table. If you do not specify a name, JMP names the data table Untitled # (for example, Untitled 1).

`dt<<Journal`

Makes a journal from the data table. Only the data grid is included, not notes, variables, or scripts.

Note

Journals that are created in JMP 14 or later might contain compressed matrix data for large matrices. If you have JSL scripts that open journals and extract data from them, you might need to use the `Get Journal` message (which does not compress the matrices) rather than saving the journal to disk with the `Journal` message.

See Also

See "dt<<Get Journal"

`dt<<Journal Link(<Save(<`*`path`*`>)|Embed()>, <Button Name(<`*`name`*`>)>))`

Adds a link to the data table in the current journal. If a journal does not exist, a new one is created.

Optional Arguments

`path` Specifies the quoted *`path`* where the table is saved. If omitted, the data table should already have a disk location (previously saved or loaded), otherwise the journal link is incomplete and will not reload the table.

`Embed` Embeds a JSL script to recreate the data table.

`Button Name(name)` Specifies the name that is displayed on the button. The *`name`* argument is quoted. If the button name is not specified, the button is named after the data table.

`dt<<Label`

`dt<<Unlabel`

Toggles selected rows in *`dt`* from labeled to unlabeled or *vice versa*.

`dt<<Last Modified`

Returns the date on which the data table was last saved.

`dt<<Layout`

`Layout` is deprecated and will be removed in a future release. Use `Journal` instead.

dt<<Lock Data Table

Description

Locks the data table so that data and column properties cannot be added or changed.

See Also

"dt <<Set Edit Lock(<"Modify Cells">, <"Add Rows">, <"Add Columns">, <"Delete Rows">, <"Delete Columns">)"

dt<<Make Indicator Columns(<Append Column Name(*Boolean*)>, <Include Missing(*Boolean*)>)

Creates indicator columns of 0 and 1 values for the specified categorical columns.

Example

The following example creates indicator columns for the `sex` column. `Append Column Name` creates columns named `sex_F` and `sex_M`. Otherwise, the columns are named after each level (`F` and `M`). `Include Missing` includes missing values.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Make Indicator Columns(
   Columns( :sex ),
   Append Column Name( 1 ),
   Include Missing( 1 )
);
```

dt<<Make RowState Handler

Creates a row state handler function. The argument of the function holds the rows whose row states get changed.

dt<<Make SAS Data Step

Returns the data table as a SAS Data Step.

dt<<Make SAS Data Step Window

Returns the data table as a SAS Data Step and places it in a SAS script window.

dt<<Make Validation Column

Creates a column that is used to divide the data into training and validation sets.

dt<<Marker by Column(*column*)

Description

Assigns markers according to the values of the specified data table *column*.

See Also

"dt<<Marker By Column(column, <named arguments>);"

`dt<<Markers(n)`

Assigns marker *n* to the selected rows.

`dt<<Maximize Display`

Deprecated. Use `Optimize Display` instead.

Forces the data table to remeasure all of its columns and zoom to the best-sized window.

`dt<<Move Script Group(group name, "To First"|"To Last"|After(table script name)|After(group))`

Description

Rearranges the table script groups that are specified by the quoted *group name*.

`dt<<Move Selected Column(name(s), "To First"|"To Last"|After(name))`

`dt<<Move Selected Columns(name(s), "To First"|"To Last"|After(name))`

Description

Moves the selected column or columns in the data table to the specified position. The *name* argument is quoted.

Example

The following example moves the age column to the last column in Big Class.jmp:

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Go To( :age );
dt << Move Selected Columns( "To Last" );
```

You may also use a list to specify the column names.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
list = {"name", "sex"};
dt << Move Selected Columns( list, "To Last" );
```

`dt<<Move Rows("At Start"|"At End"|After(n))`

Moves the selected rows in the data table to the specified position. *n* represents a row number.

`dt<<New Column(`*`name`*`, <`*`data type`*`>, <`*`modeling type`*`>, <Format(`*`format`*`, `*`width`*`)>, <Formula()>, <Set Values({..., ..., })>, <Set Property(`*`properties`*`)>)`

Description

Adds a new column titled with the quoted name after the last column in *dt*. Unless otherwise specified, columns are numeric, continuous, and 12 characters wide.

Returns

A column reference.

Required Argument

`name` The name of the new column.

Optional Arguments

`data type` A quoted string that specifies the data type. The options are `"Numeric"`, `"Character"`, `"Row State"`, or `"Expression"`.

`modeling type` A quoted string that describes the modeling type (`"Continuous"`, `"Nominal"`, `"Ordinal"`, `"Multiple Response"`, `"Unstructured Text"`, `"None"`, or `"Vector"`).

`Format(`*`format`*`, `*`width`*`)>` Sets the format type and column width.

`Set Values({})` Specifies the data in the column.

`Formula` Specifies the column formula.

`Set Property(`*`properties`*`)` Specifies any messages that data table columns support. Action arguments are found in the Column Properties menu in the New Column window. `Axis` and `Link Reference` are action argument.

See Also

For examples of setting other numeric format properties, see "col<<Format(<width>, <decimal places>, <"Use Thousands Separator">)".

`dt<<New Data Box()`

Makes a data table view in a display box tree. Useful for displaying the data table and report in one window. A data browser box is created when you send the `New Data Box` message to the data table object.

Example

The following script creates a data table view and report in one window. The data table is placed in a data browser box. The width of that box is set to 800 pixels. Because auto stretch is turned off, the data table view remains 800 pixels wide even if you stretch the right border of the window.

```
dtA = Open( "$SAMPLE_DATA/Semiconductor Capability.jmp", invisible );
nw = New Window( "Example",
  H List Box(
     V List Box( dtbox = dtA << New Data Box() ),
```

```
        dtA << Distribution(
           Continuous Distribution( Column( :NPN1 ) ),
           Continuous Distribution( Column( :PNP1 ) )
        )
     )
  );
  dtbox << Set Stretch( "Off", "Off" ) << Set Width( 800 );
```

`dt<<New Data View`

Opens a duplicate of the data table. The second data table is identical to and linked to the original data table, so that any changes made in one are reflected in the other. Closing either data table also closes the other and all references to the data tables are deleted.

This can be useful to show an invisible data table.

`dt<<New Script(`*`name`*`, `*`script`*`)`

`dt<<Set Property(`*`name`*`, `*`script`*`)`

Creates a new table property (also called a *table script*) using the quoted *`name`* that stores the specified *`script`*.

Use `New Script()` or `Set Property()` rather than the deprecated `New Property()` and `New Table Property()`.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Set Property( "Bivariate Example", Bivariate( Y( :weight ), X( :height
  ), Fit Line ) );
```

`dt<<New Table Variable(`*`name`*`, `*`number`*`)`

`dt<<Set Table Variable(`*`name`*`, `*`number`*`)`

Creates a new table variable with the quoted *`name`* and the *`number`*.

`dt<<Next Selected`

Scrolls data table down to show the next selected row that is not already in view.

`dt<<Optimize Display`

Forces the data table to remeasure all of its columns and zoom to the best-sized window.

`dt<<Original Order`

Restores saved order of columns in *`dt`*.

dt<<Paste Column Properties

Pastes multiple lists of column properties to multiple columns. Optionally, you can specify a list of target columns instead of selecting them in the data table.

Example

```
dt = Open( "$SAMPLE_DATA/Tiretread.jmp" );
dt << Copy Column Properties( {:MODULUS, :ELONG} );
dt2 = New Table( "test it",
   New Column( "T1", numeric, continuous ),
   New Column( "T2", numeric, continuous ),
   New Column( "T3", numeric, continuous ),
   Add Rows( 10 )
);
dt2 << Paste Column Properties( {:T1, :T3} );
// pastes the column properties from MODULUS and ELONG to T1 and T3
```

dt<<Predictor Screening

dt<<Screen Predictors

Description

Used to identify strong predictors.

Example

```
dt = Open( "$SAMPLE_DATA/Boston Housing.jmp" );
obj = dt << Predictor Screening(
   Y( :mvalue ),
   X( :crim, :indus, :chas, :nox, :rooms, :age, :distance, :radial )
);
```

dt<<Previous Selected

Scrolls data table up to show the previous selected row that is not already in view.

dt<<Print Window(<"Show Dialog">)

Prints the window. If the optional named argument "Show Dialog" is specified, the print window is displayed. Otherwise, the window is printed to the default printer using the current settings, and no print window is displayed.

dt<<Rename Script Group(*old name*, *new name*)

Description

Renames the table script group.

Example

```
dt << Rename Script Group( "Maps", "Street Maps" );
```

`dt<<Reorder By Data Type`

Reorders columns in *dt*, row state first, then character, then numeric.

`dt<<Reorder By Modeling Type`

Reorders columns in *dt* to continuous, then ordinal, then nominal.

`dt<<Reorder By Name`

Reorders columns in *dt* to alphanumeric order by name.

`dt<<Rerun Formulas`

Recalculates all formula-based data table variables. Recalculations are performed in the proper dependency order.

`dt<<Reverse Order`

Reverses columns in *dt* from current order.

`dt<<Revert`

Reverts to the most recently saved version of *dt*.

`dt<<Row Selection(Select Where(condition), <current selection("Extend"|"Restrict"|"Clear")> <Dialog("Keep Dialog Open")>)`

Description

Selects all rows that meet the specified condition.

Required Argument

`Select Where(condition)` Specifies the condition by which the rows are selected.

Optional Arguments

`current selection("Extend"|"Restrict"|"Clear")` Extends, restricts, or clears the existing selections. `Clear` is the default value.

`Dialog("Keep Dialog Open")` Shows the dialog so that the user can edit the options.

`dt<<Run Formulas`

Performs all pending formula evaluations, including evaluations that are pending as a result of evaluating other formulas.

`dt<<Run Script(name)`

Finds the table property with the quoted name and runs it as a JSL script.

`dt<<Save(path)`

`dt<<Save As(path)`

Description

Saves the table in the specified quoted *path*.

`dt<<Save Database(connection information, table name, <"Replace">)`

Saves the data table to the database named using the quoted connection information and quoted table name. The "Replace" option replaces the existing database with the current database.

`dt<<Save Script to Script Window`

Saves a script to reproduce the data table in a script editor window. Appends the script to any script that currently appears in the script editor.

`dt<<Select All Rows`

Selects all rows in the data table.

`dt<<Select Columns(<column1>, <column2>,...|"All")`

Selects the specified columns (or all columns) in the data table.

`dt<<Select Duplicate Rows`

Description

Selects the second and subsequent duplicate rows. If columns are selected, duplicate values are found in the rows of those columns. The duplicate values are case sensitive.

`dt<<Select Excluded`

Selects only those rows in the data table that are currently excluded.

`dt<<Select Hidden`

Selects only those rows in the data table that are currently hidden.

dt<<Select Labeled

Selects only those rows in the data table that are currently labeled.

dt<<Select Randomly(*n*|*p*|Sample Size(*n*)|Sampling Rate(*p*))

Randomly selects the given percentage *p* of the rows in the data table, or the number of rows *n*. If you specify an argument with a keyword, numbers between 0 and 1 represent a percentage, and numbers larger than 1 represent a number of rows.

dt<<Select Rows([*row1*, *row2*, ...])

Selects the rows given in the list of row numbers.

dt<<Select Script Group(*<group name|{group1, group2, ...}>*)

Selects the table script group specified as a quoted *group name* or a list of quoted strings. If no argument is provided, all groups are selected.

dt<<Select Where(*condition*, <Current Selection("Extend"|"Restrict"|"Clear")>)

Description

Selects the rows in *dt* where the condition evaluates as true.

dt<<Set Dirty(*Boolean*)

Marks the data table as changed, even if no changes have been made.

dt <<Set Edit Lock(<"Modify Cells">, <"Add Rows">, <"Add Columns">, <"Delete Rows">, <"Delete Columns">)

Description

Prevents cells from being modified; rows from being added or deleted; and columns from being added or deleted.

dt <<Set Cell Height(*n*)

Sets the cell height to the specified number of pixels.

dt<<Set Header Height(*n*)

Sets the column header's height to the specified number of pixels.

dt<<Set Label Columns(*column1, columns2, ...*)

Assigns the specified columns as label columns.

dt<<Set Matrix(*[matrix]*)

Inserts the specified matrix into a data table, adding new columns and rows as necessary.

dt<<Set Name(*name*)

Description

Specifies a name for the table. The *name* argument is quoted.

Returns

The data table name as a quoted string.

Notes

A change was made to the `Set Name` message so that now the new table name is returned as a quoted string. In previous releases, `Set Name` returned a scriptable data table object. As a result of this change, JMP scripts might need to be updated for the desired result to be returned. For example, rewrite the following script:

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" ) << Set Name( "Test" );
```

Separate the messages so that *dt* represents the data table instead of "Test":

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt << Set Name( "Test" );
```

The result is the same as in previous releases but will run successfully in both earlier and newer versions of JMP.

dt<<Set Property(*name, script*)

See "dt<<New Script(name, script)".

dt<<Set Label Columns(*column(s), ...*)

dt<<Set Label Columns

Turns on the Label attribute for the specified columns. If no columns are listed, it turns the Label attribute off.

dt<<Set Row ID Width(*n*)

Sets the row ID display width to the specified number of pixels. If *n* is set to zero, the row ID display width automatically resizes.

`dt<<Set Row States(`*`[matrix]`*`)`

Sets the row states for all rows in the data table.

`dt<<Set Scroll Lock Columns(`*`column name, ...`*`)`

Locks scrolling for the columns specified as quotes quoted strings. If no columns are listed, unlocks scrolling.

`dt<<Set Table Variable(`*`name, value`*`)`

See "dt<<New Table Variable(name, number)".

`dt<<Sort(<"Private">|<"Invisible">, <"Replace Table">, By(`*`columns`*`), Order("Descending" | "Ascending"), <Output Table Name(`*`name`*`))>`

Description

Creates a new table (named after the quoted name) by rearranging the rows of *dt* according to the values of one or more *columns*.

Returns

A reference to the sorted table.

`dt<<Split(Split(`*`columns`*`), Split By(`*`column`*`), <Group(`*`column`*`)>, <"Private">|<"Invisible">, <Remaining Columns("Keep All"|"Drop All"| Keep(`*`columns`*`)|Drop(`*`columns`*`))>, <Copy Formula(`*`Boolean`*`)>, <Suppress Formula Evaluation(`*`Boolean`*`)>, <Sort by Column Property>, <Output Table (`*`name`*`)>)`

Description

Unstacks multiple rows for each `Split` column into multiple columns as identified by the `Split by` column. The `Split` and `Split by` arguments are required.

Returns

A reference to the split data table.

Required Arguments

`Split(`*`columns`*`)` The column to split.

`Split By(`*`column`*`)` The column to split by.

Optional Arguments

`Group` Splits data within the specified groups.

`Remaining Columns("Keep All"|"Drop All"| Keep(`*`columns`*`)|Drop(`*`columns`*`))` Specifies what to do with the remaining columns in the resulting table. `Keep All` is the default setting.

Note: `Keep All` includes all columns in the output data table. However, the *values* of every column are not included. Because multiple rows are collapsed to a single row in the output data table, some values of the kept columns are dropped.

`Copy Formula(`*`Boolean`*`)` Includes column formulas from the source table in the resulting table.

`Suppress Formula Evaluation(`*`Boolean`*`)` Stops any copied formulas from being evaluated. True is the default setting.

`Sort by Column Property` Sorts the order of the output columns by the sort column property that is defined for the Split by column.

`Output Table(`*`name`*`)` Generates the output to the specified table name.

`dt<<Stack(<"Private">|<"Invisible">, Columns(`*`columns`*`), <Source Label Column(`*`quoted string`*`)>, <Stacked Data Column(`*`quoted string`*`)>, <Copy Formula(`*`Boolean`*`)>, <Suppress Formula Evaluation(`*`Boolean`*`)>, <Drop All Other Columns(`*`Boolean`*`)|Name(`*`non-stacked columns`*`)(Keep(`*`column1, column2, ...`*`))|Name(`*`non-stacked columns`*`) (Drop(`*`column1, column2, ...`*`))>, <Output Table(`*`name`*`)>), <Number of Series(`*`n`*`)>, <"Contiguous">`

Description

Creates a new table by combining the values from several columns in *dt* into one column.

Returns

A reference to the stacked data table.

`dt<<Subscribe("keyname"(<"`*`client`*`">), On Delete Columns(<`*`function`*`>|<`*`script`*`>)|On Add Columns(<`*`function`*`>|<`*`script`*`>)|On Add Rows(<`*`function`*`>|<`*`script`*`>)|On Delete Rows(<`*`function`*`>|<`*`script`*`>)|On Rename Column(<`*`function`*`>|<`*`script`*`>)|On Close(<`*`function`*`>|<`*`script`*`>)|On Save(<`*`function`*`>|<`*`script`*`>)|On Rename(<`*`function`*`>|<`*`script`*`>))`

Description

Subscribes to a data table to get messages regarding changes in the data table.

Returns

The keyname.

Arguments

`"keyname"(<`*`client`*`>)` Specifies the subscription name so that it can be referenced. The quoted client triggers a close confirmation when a close is attempted on the data table, warning that other open windows depend on the data table.

`On Delete Columns(<`*`function`*`>|<`*`script`*`>)` Returns the keyname when columns are deleted.

`On Add Columns(<`*`function`*`>|<`*`script`*`>)` Returns the keyname when columns are added.

`On Add Rows(<function>|<script>)` Returns the keyname when rows are added.

`On Delete Rows(<function>|<script>)` Returns the keyname when rows are deleted.

`On Rename Column(<function>|<script>)` Returns the keyname when columns are renamed.

`On Close(<function>|<script>)` Returns the keyname when the data table is closed. Takes one argument, a function. The function requires only one argument, the data table name.

`On Save(<function>|<script>)` Returns the keyname when the data table is saved.

`On Rename(<function>|<script>)` Returns the keyname when a rename is attempted on the data table. The function can be either the name of a previously defined function or the function itself.

Notes

Each subscription option remains in effect until you unsubscribe.

`dt<<Subset(<"Private">|<"Invisible">, <"Selected Columns">, <Columns(column list)>, <"Selected Rows">, <Rows([number, number, ...])>, <By(column list)>, <Sampling Rate(fraction)>, <Sample Size(integer)>, <Stratify(column list)>, <Link to Original Data Table(Boolean)>, <Copy Formula(Boolean)>, <Suppress Formula Evaluation(Boolean)>, <"Keep by Columns">)`

Description

Creates a new table from the rows and columns that you specify in *dt*.

Returns

A reference to the subset data table.

`dt<<Summary(<"Private">|<"Invisible">, <Group(column)>, <Subgroup(column)>, <N(column)>, <Mean(column)>, <Std Dev(column)>, <Min(column)>, <Max(column)>, <Range(column)>, <Sum(column)>, <CV(column)>, <Freq(column)>, <Weight(column)>, "Include Marginal Statistics", <Link to Original Data Table(Boolean)>, <Statistics Column Name Format(Stat(column)|Column|Stat of Column|Column Stat)>)`

Description

Creates a new table of summary statistics for the column that you specify, according to groups and subgroups. `Statistics Column Name Format` values are quoted.

Returns

A reference to the summary data table.

dt<<Suppress Formula Eval(*Boolean*)

Turns off automatic calculation of formulas for data table *dt*.

dt<<Text to Columns(Delimiters(<*separator*>, <"Tab">, <"Newline">), Columns(*column1, column2...*))

Makes a set of text columns or indicator columns from a delimited text column. "newline" includes the three forms: \r, \n, and \r\n. The separator is quoted.

Example

```
dt = Open( "$SAMPLE_DATA/Consumer Preferences.jmp" );
dt << Text To Columns(
   delimiter( "," ),
   columns( :Brush Delimited )
);
```

dt<<Transpose(Columns(*columns*), Rows([*matrix*]), Output Table Name(*name*))

Description

Creates a new table (named after the quoted *name*) from the rows and columns that you specify.

Returns

A reference to the transposed data table.

dt<<Ungroup Columns(*{column1, column2, ...}*)

Ungroups the columns defined in the list argument.

dt<<Ungroup Scripts(*Name of Script Group*|*{script1, script2,...}*)

Description

Removes the specified table scripts or group from the group. The *Name of Script Group* argument is quoted.

dt<<Unsubscribe(*keyname*, "On Delete Columns"|"On Add Columns"|"On Add Rows"|"On Delete Rows"|"On Close"|"On Col Rename"|"All")

Releases any previous subscriptions to the data table *dt*. The *keyname* argument is quoted.

dt<<Update from Database

Updates the data in the table *dt* with data reimported from the database.

Column Messages

---

col<<Add Column Properties(*name*, *expression*)

Adds the quoted column property *name* with the *expression* given. You can add any standard column property by name or a user-specified property.

---

col<<Add From Row States

Updates a row state column with any currently used row state changes that are not the default state.

---

col<<Add To Row States

Copies all row state values in a column that are not the default state to the currently used row state in the data table.

---

col<<Color Cells(*color*)

Description

Colors the cells of the column within the data table grid. Use any quoted named color or 0 to clear the color.

---

col<<Color Cell by Value(*Boolean*)

Description

Colors the cells of the column in the data table grid using the value color property.

---

col<<Copy Column Properties

Copies the column properties into the buffer.

---

col<<Copy From Row States

Copies all row state values currently used in the data table to a column.

---

col<<Copy to Row States

Copies all row state values in the column to the currently used row state in the data table.

---

col<<Data Type(*type*, <Format(*format quoted string*)>, <Input Format(*format quoted string*)>, <*width*>)

col<<Set Data Type(*type*, <Format(*format quoted string*)>, <Input Format(*format quoted string*)>, <*width*>)

Description

Sets the *data type* to *col*.

Required Argument

type Specifies the "Numeric", "Character", "Row State", or "Expression" data type.

Optional Arguments

Format(*format quoted string*) Specifies the way the data are displayed, such as h:m for hours and minutes. The *format quoted string* argument is quoted.

Input Format(*format quoted string*) Specifies the way the data are input. The *format quoted string* argument is quoted.

width (Optional for numeric data) Specifies 1, 2, or 4 (the number of bytes in the column).

---

col<<Delete Formula

Deletes the formula from a column.

---

col<<Delete Property(*name*)

col<<Delete Column Property(*name*)

Deletes the quoted property name from a column.

---

col<<Eval Formula

Forces the formula to evaluate (perhaps again). If formula suppression is enabled, the evaluation is not performed.

---

col<<Exclude(*Boolean*)

Turns the excluded or unexcluded state on, depending on the Boolean argument.

---

col<<Format(<*width*>, <*decimal places*>, <"Use Thousands Separator">)

col<<Format("Best", <*width*>, <"Use Thousands Separator">)

col<<Format(("Fixed Dec"|"Percent"), <*width*>, <*decimal places*>, <"Use Thousands Separator">)

col<<Format("Pvalue", <width>)

col<<Format(("Scientific"|"Engineering"|"Engineering SI"), <*width*>, <*decimal places*>)

col<<Format("Precision", <*width*>, <*decimal places*>, <"Use Thousands Separator">, <"Keep Trailing Zeroes">, <"Keep All Whole Digits">)

col<<Format("Currency", <"Currency Code">, <*width*>, <*decimal places*>, <"Use Thousands Separator">)

col<<Format("Datetime", <*width*>, <*input format*>)

col<<Format(("Latitude DDD"|"Latitude DDM"|"Latitude DMS"|"Longitude DDD"|"Longitude DDM"|"Longitude DDM"), <*width*>, <*decimal places*>, ("PUN"|"DIR"|"PUNDIR"))

col<<Format("Custom", Formula(...), <*width*>, <*input format*>)

Description

Sets the numeric display specified format.

Arguments

See *Using JMP* for more information about the arguments.

Examples

```
col<<Format( 10, 2, "Use thousands separator");
col<<Format( "Currency", "EUR", 20 );
col<<Format( "m/d/y", 10 );
col<<Format( "Precision", 10, 2, "Keep trailing zeroes", "Keep all whole
  digits" );
col<<Format( "Latitude DDD", "PUNDIR"); // "PUN" for punctuation, "DIR" for
  direction, PUNDIR for both
col<<Format( "Custom", Formula( Abs( value ) ), 15 );
```

---

col<<Formula(*expression*)

col<<Set Formula(*expression*)

Sets the formula for the variable and evaluates it.

---

col<<Get Column Field Width

Returns the field width used for displaying data in the column.

`col<<Get Data Type`

Returns the data type of *col*.

`col<<Get Data Type Length`

Returns the data type and length of the data column. Only the data type is returned if the data length is not fixed, as with character columns.

`col<<Get Format`

Returns the format of the column.

`col<<Get Formula`

Returns the formula.

`col<<Get Hidden`

Returns 1 if the column is hidden.

`col<<Get Input Format`

Returns the format used for input and storing of data for the column.

`col<<Get Labeled`

Returns 1 if the column is labeled.

`col<<Get List Check`

Returns the list check definition. If list check is not defined for the column, a message is sent to the log stating so.

`col<<Get Lock`

Returns the current Lock setting.

`col<<Get Modeling Type`

Returns the modeling type of the column.

`col<<Get Name`

Returns the name of the column.

`col<<Get Property("property name")`

Returns the specified property definition. If the specified property is not defined for the column, a message is sent to the log stating so.

`col<<Get Range Check`

Returns the range check definition. If range check is not defined for the column, a message is sent to the log stating so.

`col<<Get Role`

Returns the preselected role of *col*.

`col<<Get Script`

Returns the script to reproduce the column.

`col<<Get Scroll Locked`

Returns 1 if the column is scroll locked.

`col<<Get Selected`

Returns 1 if the column is selected, or 0 otherwise.

`col<<Get Stored Values`

Returns the values in the columns without considering the Missing Value Codes column property.

`col<<Get Value Labels`

Returns the value labels definition. If value labels is not defined for the column, a message is sent to the log stating so.

`col<<Get Use Value Labels`

Returns 1 if the value labels are set to be used for the column, or 0 otherwise.

`col<<Get Values`

Returns the values in the column.

`col<<Hide(Boolean)`

Turns the Hide attribute on or off according to the Boolean argument given.

`col<<Ignore Errors`

Ignores formula evaluation errors in a column, and sets the cell value to missing when a formula error occurs.

`col<<Input Format(`*`format`*`)`

Sets the quoted *format* used for input and storage for the column. The argument is the name of any JMP format (for example, `"ddmmyyyy"` for a date column).

`date_col<<Is Transformed On SAS Export`

Returns true if the data in the resulting SAS data set for the date column will be changed when it is exported to SAS.

`col<<Label(`*`Boolean`*`)`

Turns the Label attribute on or off according to the Boolean argument given.

`col<<Lock(`*`Boolean`*`)`

`col<<Set Lock(`*`Boolean`*`)`

Turns the Lock attribute on or off according to the Boolean argument given.

`col<<Preselect Role(`*`role`*`)`

Preselects the specified *role* for the column. Choices are `"Y"`, `"X"`, `"Weight"`, `"Freq"`, and `"None"`, or `"No Role"`.

`col<<Reset Transform`

Removes the cached data for the transform column. Accessing column data rebuilds the caches. Use this option to reduce memory or to allow recalculation if the formula depends on external information.

`col<<Set Display Width(`*`n`*`)`

Sets the column display width to the *n* in pixels. If *n* is set to zero, the column display width automatically resizes.

`col<<Set Each Value(n)`

Sets all the values in the column to *n*.

`col<<Set Excluded`

Excludes the column.

`col<<Set Field Width(n)`

Sets the field width for the column to *n*.

`col<<Set Hidden`

Hides the column.

`col<<Set Labeled`

`col<<Set Labelled`

Uses the column's data values for labels.

`col<<Set Modeling Type(type)`

Sets the modeling type for the variable. Choices are `"Continuous"`, `"Ordinal"`, `"Nominal"`, `"None"`, `"Row State"`, `"Unstructured"`, `"Multiple Response"`, or `"Vector"`.

`col<<Set Name(name)`

Sets the name for the column. The *name* argument is quoted.

`col<<Set Property(name, expression)`

Sets the quoted property name to the *expression* given. You can set any standard column property by name or a user-specified property.

Examples

The following example adds the Value Colors column property to the sex column, with pink for females and blue for males.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
Column( "sex" ) << Set Property( "Value Colors", {"F" = 78, "M" = 69} );
```

The following example adds a custom column property named *Date recorded* to the height column.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
Column( "height" ) << Set Property( "Date recorded", 05Jan1990 );
```

See Also

Scripting Guide

Using JMP

`col<<Set Scroll Locked(`*`Boolean`*`)`

Turns the Scroll Lock attribute on or off according to the Boolean argument given.

`col<<Set Selected(`*`Boolean`*`)`

Sets the column to be selected or not selected.

`col<<Set Use for Marker`

`col<<Use for Marker`

Uses the values in the column as markers in graphs. Designed to use with expression columns and character columns that have IDs. In the Big Class Families.jmp sample data table, the picture column is specified to use as markers in graphs. Not supported in Bubble Plot.

`col<<Set Values(`*`[matrix]`* `or` *`{list}`*`)`

`col<<Values(`*`[matrix]`* `or` *`{list}`*`)`

Sets values for the matrix (for numeric variables) or list (for character variables).

`col<<Suppress Eval(`*`Boolean`*`)`

Turns off automatic calculation of formulas for the column.

`col<<Use For Marker(`*`Boolean`*`)`

Uses the values in the column as markers in graphs or turns off the option. Designed to use with expression columns and character columns that have IDs. In the Big Class Families.jmp sample data table, the picture column is specified to use as markers in graphs.

Row Messages

`row<<Colors(`*`n`*`)`

Assigns the color *n* to the selected rows.

`row<<Exclude(`*`Boolean`*`)`

`row<<Unexclude(`*`Boolean`*`)`

Turns the excluded or unexcluded state on for the selected rows according to the Boolean argument given. Omit the argument to toggle the row state.

`row<<Hide(`*`Boolean`*`)`

`row<<Unhide(`*`Boolean`*`)`

Turns the Hide attribute on or off according to the Boolean argument given. Omit the argument to toggle the row state.

`row<<Hide and Exclude`

Shows or hides the selected rows from appearing on graphs, and excludes or unexcludes them from contributing to calculations.

`row<<Label(`*`Boolean`*`)`

`row<<Unlabel(`*`Boolean`*`)`

Turns the Label attribute on or off according to the Boolean argument given. Omit the argument to toggle the row state.

`row<<Markers(`*`marker`*`)`

Assigns the quoted "*marker*" to the selected rows.

`row<<Next Selected`

Causes the next selected row in the data table to blink.

`row<<Previous Selected`

Causes the previous selected row in the data table to blink.

`row<<Row Editor`

Opens the Row Editor window for the selected rows.

Data Filter Messages

---

dtf<<Add Favorites(*name*)

Description

Associates the current filter selection with the quoted *"name"* and saves it in the Favorites list.

Returns

The favorite as a quoted string.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
df = dt << Data Filter(
   Add Filter(
      Columns( :age, :sex, :height, :weight ),
      Where( :sex == "F" ),
      Where( :height >= 55 & :height <= 65 )
   ),
   Mode( Select (1) )
);
Wait( 1 ); // for demonstration purposes
fav1 = df << Add Favorites( "Female Average Ht" );
```

---

dtf<<Add Filter(Columns(*column1*, <*column2*>), <Where(*clause*)>)

Add one or more filter columns in a new OR group.

---

dtf<<Auto Clear(*Boolean*)

Clears all currently selected rows before setting a new selection.

---

dtf<<Clear

Clears the currently selected rows.

---

dtf<<Close

Closes the data filter window.

---

dtf<<Columns(*column1, column2, ...*)

Sets the columns to use in the data filter.

---

dtf<<Data Table Window

Shows the data table that the data filter window is using.

dtf<<Delete All

Removes all filters that are set.

dtf<<Delete(*column1*, *column2*, ...)

Removes the specified columns from the data filter.

dtf<<Display(*column*, <Size(*x*, *y*)>, "Blocks Display"|"List Display"|"Single Category Display"|"Checkbox Display")

Sets how the specified categorical column levels are displayed in the filter.

dtf<<Get Script

Returns the data filter script as text in the log.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
df = dt << Data Filter(
     Add Filter( Columns( :age, :sex ), Where( :age == 12 ) )
);
txt = df << Get Script;
Show( txt );
```

dtf<<Local Data Filter

Embeds the data filter in the specified window. See *Scripting Guide* for more information about local data filters.

dtf<<Location(*x*, *y*)

Moves the data filter window to the specified location. *x* and *y* are measured in pixels. 0,0 is the top left of the monitor.

dtf<<Make Filter Change Handler(*function*)

Creates a data filter handler to handle notification that the filter has changed. The number of rows filtered is returned in the argument to the function.

Example

```
dt = Open( "$SAMPLE_DATA/PopAgeGroupSubset.jmp" );
dist = Distribution( Automatic Recalc( 1 ), Continuous Distribution( Column(
   :POP ) ) );
filter = dist << Local Data Filter( Add Filter( Columns( :Region ) ) );
f = Function( {a}, Print( a ) );
rs = filter << Make Filter Change Handler( f );
```

`dtf<<Make Subset`

Creates a new subset data table that contains the rows that are selected in the data filter.

`dtf<<Match(Filter Columns(`*`column1`*`, column2, ...), Where(`*`clause`*`)`

Sets the filter conditions for each column. The *`Where clause`* is used for all the columns listed. To use different Where clauses for different columns, send the `Match` message separately for each column.

`dtf<<Mode(Select(`*`Boolean`*`)|Show(`*`Boolean`*`)|Include(`*`Boolean`*`))`

Sets the action, or mode, that is used when rows are selected using the data filter.

`dtf<<Save and Restore Current Row States`

Saves the current row states for the data table, and then restores those states when the data filter is closed.

`dtf<<Show Columns Selector(`*`Boolean`*`)`

Displays or hides the column selector after completing a filter.

`dtf<<To Clipboard`

Creates a Where clause from the current state of the data filter and places it on the clipboard, where it can be pasted elsewhere.

`dtf<<To Data Table`

Creates a Where clause from the current state of the data filter and saves it as a property to the data table.

`dtf<<To Journal`

Creates a Where clause from the current state of the data filter and appends it to the current journal. If there is no current journal, a new journal is opened and the Where clause is added to it.

`dtf<<To Row State Column`

Creates a row state column whose formula is the Where clause.

dtf<<To Script Window

Creates a Where clause from the current state of the data filter and appends it to the current script window. If there is no current script window, a new script window is opened and the Where clause is added to it.

dtf<<Use Floating Window(*Boolean*)

Sets whether the data filter window floats on top of its associated data table or behaves as a normal window.

dtf<<Where(*clause*)

Sets a condition for selecting rows.

Data Feed Messages (Windows Only)

feed<<Close

Closes the data feed object and its window.

feed<<Connect(*port settings*)

Sets up port settings for the connection to the device.

feed<<Disconnect

Disconnects the device from the data feed queue but leaves the data feed object active.

feed<<EOL("CR", "LF", "CRLF")

Sets the line ending value used as a separator when parsing incoming lines of data. The value is also used as the terminator in outgoing lines of data.

- "CR": ASCII character 13 (carriage return)
- "LF": ASCII character 10 (line feed)
- "CRLF": Uses both CR and LF in sequence.

feed<<Get Line

Returns and removes one line from the data feed queue.

feed<<Get Lines

Returns as a list and removes all lines from the data feed queue.

feed<<Print Queue

Prints the internal queue of messages to the log window.

feed<<Queue Line(*quoted string*)

Sends one quoted *string* (or *line*) to the end of the data feed queue. Queue Line is primarily useful for testing your script without requiring it to be attached to a device. You can essentially simulate the data coming from the device to make sure the rest of your code handles the values properly when it's really attached to a working device.

feed<<Restart

Restarts processing queued lines.

feed<<Set Script(*script*)

Assigns the *script* that is run each time a line of data is received.

feed<<Stop

Stops processing queued lines.

feed<<Write(*quoted string*)

Description

Sends a quoted *string* to the data feed device.

Example

```
/* Example - send a message to external device over the serial port to trigger
   data messages. This can be used to send control messages to a sensor or
   other attached device.*/
exfeed = Open Datafeed(
   Connect( Port( "com1" ), Baud rate( 4800 ), Parity( "even" ), DataBits( 8 )
   ),
   Set Script(
      ex = exfeed << Get Line;
      Show( ex );
   )
);
exfeed << Write( "Ready" );
```

feed<<Write Line(*quoted string*)

Description

Sends a quoted *string* to the data feed device. If EOL has been set for the data feed, the quoted strings are terminated by the specified EOL value. If EOL has not been set, the line is terminated with CRLF.

Example

```
/* Send a message to external device over the serial port to trigger data
   messages. This can be used to send control messages to a sensor or other
   attached device.*/
exfeed = Open Datafeed(
   Connect( Port( "com1" ), Baud rate( 4800 ), Parity( "even" ), DataBits( 8 )
   ),
   Set Script(
      ex = exfeed << Get Line;
      Show( ex );
   )
);
exfeed << Write Line( "Ready" );
```

feed<<Write Lines({*quoted string1, quoted string2, quoted string3*})

Description

Sends a list of quoted *strings* to the data feed device. If EOL has been set for the data feed, the quoted strings are terminated by the specified EOL value. If EOL has not been set, the line is terminated with CRLF.

Example

```
/* Send a message to external device over the serial port to trigger data
   messages. This can be used to send control messages to a sensor or other
   attached device.*/
exfeed = Open Datafeed(
   Connect( Port( "com1" ), Baud rate( 4800 ), Parity( "even" ), DataBits( 8 )
   ),
   Set Script(
      ex = exfeed << Get Line;
      Show( ex );
   )
);
exfeed << Write Lines( {"Ready", "Set", "Go"} );
```

Display Box Messages

This section contains JSL messages that apply to display boxes. For additional examples, see the JMP Scripting Index.

All Display Box Messages

`db<<Add Text Annotation(Text(`*quoted string*`), Text Box(<`*x1*`, `*y1*`, `*x2*`, `*y2*`>))`

Draws a text annotation box at the specified pixel location that contains the quoted *string*. The `Text Box` argument controls where the text annotation box is drawn in the window, from the upper left corner to the lower right corner.

Note that *x1*, *y1*, *x2*, and *y2* are not graph axis values but the specific pixel locations in the window. Exactly where the text box appears depends on the user's window size, display resolution, and so on.

`db<<Append(`*db2*`)`

Adds *db2* as the last child of the *db*.

`db<<Child`

Returns the child of the box.

`db<<Class Name`

Returns the name of the display class for the box.

`db<<Clone Box`

Makes a new copy of the display box.

`db<<Close Window`

Closes the containing window.

`db<<Copy Picture`

Puts a picture of the box on the clipboard.

db<<Delete

Deletes the display box.

db<<Enable(*Boolean*)

Controls the ability to interact with the display box. 0 disables the display box. 1 enables the display box.

db<<Get HTML

Returns a quoted string containing HTML source for the box.

db<<Get Journal

Returns a quoted string containing journal source for the box.

db<<Get Menu Item State(*index*)

Returns the popup menu item state of the index menu item. The state can be normal (0), checked (1), or disabled (-1).

db<<Get Menu Items

Description

Returns the menu items used for popup menu when the button is clicked. Menu items are returned in a list.

See Also

For submenus see "db<<Get Submenu(index)".

db<<Get Menu Script

Returns the menu script attached to the calling object.

db<<Get Page Setup()

Returns the page setup settings.

Example

The example below creates a new window and returns the page setup configuration.

```
w = New Window( "Window",
   Text Box( "Page Setup Test" )
);
w << Get Page Setup();
```

The results of the message:

```
{Margins( {0.75, 0.75, 0.75, 0.75} ), Scale( 1 ), Portrait( 1 ),
Paper Size( "Letter" )}
```

db<<Get Picture(<Scale(*n*)>)

Captures *db* as a picture object. The Scale(*n*) argument is a factor of the original picture size. For example, Scale(2) makes the picture object twice as large.

db<<Get RTF

Returns a quoted string containing RTF source for the box.

db<<Get Script

Returns the script for recreating the display box.

db<<Get Size

Returns either { *x, y* } or { *h, v* } in pixels:

```
xy = DisplayBox << Get Size;
```

Returns x and y in pixels:

```
{ x, y } = DisplayBox << Get Size;
```

db<<Get Submenu(*index*)

Returns the number of submenu items under the given menu item.

Example

The example below creates a menu containing *"A"*, *"B"*, and *"C"* with *"A"* having a submenu *"A1"* and *"A2"* and *"B"* having a submenu *"B1"*, *"B2"*, and *"B3"*. <<Get Submenu(*inc*) returns the number of submenu items under each indexed menu item.

```
New Window( "Title",
obj = Outline Box( "title" ) );
submenus = { };
obj << Set Menu Script(
   {"A", "", "A1", Print( "A1" ), "A2", Print( "A2" ),
   "B", "", "B1", Print( "B1" ), "B2", Print( "B2" ), "B3", Print( "B3" ),
   "C", Print( "C" )}
);
obj << Set Submenu( 1, 2 ); // menu A with 2 items in submenu A1 and A2
obj << Set Submenu( 4, 3 ); // menu B with 3 items in submenu B1, B2, and B3
For( inc = 1, inc <= N Items( Words( obj << Get Menu Script, "," ) )/2, inc++,
   Insert Into( submenus, obj << Get Submenu( inc ) );
);
submenus;
   {2, 0, 0, 3, 0, 0, 0, 0}
```

The log output indicates that index(1) contains two submenu items and index(3) contains three submenu items.

`db<<Get Text`

Returns a quoted string containing the text of the box.

`db<<Horizontal Alignment(`*`position`*`)`

Aligns a child display box inside the display parent box according to the specified *`position`*. The default value is `"Left"`, or you can specify `"Center"`, or `"Right"`.

Example

```
New Window( "Example",
   Outline Box( "Parent display box",
      Button Box( "OK", <<Horizontal Alignment( "Center" ) )
   )
);
```

`db<<Inval`

Invalidates the display box area in the window. The window is updated the next time the operating system has an opportunity to update windows (for example, when the user resizes the display box).

Notes

Consider including the message `<<Update Window` rather than including `Wait(0)`. The problem with using `Wait(`*`n`*`)` is knowing how large *`n`* should be.

Many display box messages, such as `<<Set Text`, automatically mark the box as invalid, so the `<<Inval` message is usually unnecessary. Some interactive scripts that use sliders with JSL callbacks might need `<<Update Window` to keep various parts of the display synchronized with the slider.

`db<<Is Enabled`

Returns the enabled state of the control. The message is supported in `Busy Light Box()`, `Button Box()`, `Calendar Box()`, `Check Box()`, `Col List Box()`, `Combo Box()`, `Completion Box()`, `Filter Col Selector()`, `gtext()`, `List Box()`, `Number Edit Box()`, `Popup Box()`, `Radio Box()`, `Range Slider Box()`, `Slider Box()`, `Spin Box()`, `Text Edit Box()`, `Tree Box()`, `Tree Map Box()`, and `Tree Map Seg()`.

`db<<Journal`

Appends the box to the journal.

`db<<Journal Window`

Appends the containing window of the display box to the journal; compare with `Journal`.

`db<<Move Window(x, y)`

Moves the window to the (*x, y*) location on your screen.

`db<<Page Break`

Inserts a page break before the box.

`db<<Parent`

Returns the parent of this display box.

`db<<Prepend(db2)`

Add *db2* to the display tree before *db*.

`db<<Prev Sib`

Returns the previous sibling of the display box.

`db<<Reshow`

Invalidates the display box's area in the window and immediate removes invalid areas from the window.

`db<<Save Capture(<path>, <format>, <Add Sibling(n)>)`

Saves the display box as a graphic to the specified quoted *path* in the specified quoted *format*. The optional `Add Sibling` argument adds the number of sibling display boxes to include in the capture. The default value is 1, which captures only the specified display box. Note that the specified portion of the report is not guaranteed to be scrolled into view or unobstructed by other windows. If the display box is not visible, the saved graphic will not contain the contents that you expect.

If you omit the path, you are prompted to name and save the file when running the path.

`db<<Save HTML(<path>, <format>)`

Saves the HTML source and folder of graphics to the quoted *path* and in the quoted *format*. If you omit the *path* argument, you are prompted to name and save the file when running the script.

`db<<Save Interactive HTML(<path>, "Is Static")`

Saves the display box as a web page (that includes interactive HTML features) in the quoted *path*. Non-JMP users can then explore the data. Note that the data is embedded in the web page.

Arguments

path A optional quoted *path* that specifies the location where the web page will be saved.

`"Is Static"` Omits the data from the web page and saves a static version of the web page.

Examples

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
biv = dt << Bivariate( y( weight ), x( height ) );
rbiv = (biv << Report);
rbiv << Save Interactive HTML( "$DOCUMENTS/MyInteractiveHTML.htm" );
```

`db<<Save Journal(<path>)`

Saves the journal source for the box in the quoted *path*. If you omit the argument, you are prompted to name and specify the graphic type.

`db<<Save MSWord(<path>)`

(Windows Only) Saves the display box as a Microsoft Word document in the quoted *path*. If you omit the *path* argument, you are prompted to name and save the file when running the script.

`db<<Save PDF(<path>, <Show Page Setup(Boolean)>, <Portrait(Boolean)>)`

Description

Saves a PDF of the display box in the quoted *path*.

Optional Arguments

`path` Saves the file in the quoted *path*. If you omit the argument, you are prompted to name and save the file when running the script.

`Show Page Setup(Boolean)` (Windows only) Displays the Page Setup window, where you can specify page orientation, headers and footers, margins, page scale, and paper size.

`Portrait(Boolean)` Displays the content in portrait or landscape orientation.

Notes

The PDF file contains headers and footers. Use `Save Picture` to omit these components.

`db<<Save Picture(<path>, <format>)`

Description

Saves a picture of the display box in the quoted *path* and with the specified quoted format.

Notes

- If you omit the quoted *path* argument, you are prompted to name and save the file when running the script.
- Valid file formats include `"PDF"`, `"PNG"`, `"GIF"`, `"JPG"` or `"JPEG"`, `"EPS"`, `"SVG"`, and `"EMF"`.
- On Windows, the Windows Specific preferences determine the resolution (or DPI), or you can run the following script:

  `Pref( Save Image DPI( number ) );`
- On macOS, the operating system determines the DPI.
- Use `Save Picture` to export a report as a PDF file with no headers or footers. Use `Save PDF` to include these components.

`db<<Save Presentation(<path>, <Template(path)>, <Insert("Begin"|"End"|n)|Replace("Begin"|"End"|n)|Append>, <Outline Titles(title location)>, <format>)`

Saves display boxes in a Microsoft PowerPoint presentation. You can open the file in any presentation software program.

Optional Arguments

`path` Saves the file in the quoted *path*. You must include the .pptx extension in the filename. If you omit the *path* argument, you are prompted to name and save the file when running the script.

`Template(path)` Specifies the quoted *path* of a custom PowerPoint template. Without this argument, JMP uses the default template located in the `pptx` folder of the installation directory.

Include a simple table in your template, or a default table format is applied to report tables. For an example on Windows, see `/pptx/JMPExportTemplate.pptx` in the JMP installation folder.

`Insert` Determines where the slides are inserted in an existing presentation.

– *n* inserts the slides as the *n*th slide number.

– `"Begin"` inserts the slides at the beginning of the presentation.

– `"End"` inserts the slides at the end of the presentation.

`Replace` Determines which slides are replaced in an existing presentation. The arguments are *n*, `"Begin"`, and `"End"` as described for `Insert`.

Append The slides are inserted at the end of an existing presentation.

Outline Titles The location of the outline title and any parent outline titles on the slide. By default, the immediate parent outline title appears as a slide title above the slide content, with any parent outline titles positioned in the bottom left corner of the slide.

– "None" omits the slide title above the graphic and the outline titles.

– "Hide" omits the outline titles.

– "TopLeft", "TopRight", "BottomLeft", "BottomRight" determine the position of any of the parent outline titles on the slide.

format The format of the embedded graphics. Options are "Native", "EMF", "PNG", "JPG", "BMP", "GIF", "TIF". On Windows, the native format is EMF. On macOS, the native format is PDF. See "Notes" for compatibility issues. Without this argument, JMP applies the "Image Format for PowerPoint" General preference.

Notes

Windows does not support the native PDF graphics produced on macOS. macOS does not support the native EMF graphics produced on Windows. For cross-platform compatibility, specify "PNG", "JPG", "GIF", or "TIF".

If no arguments are provided, the user is prompted to name and save the file.

---

db<<Save RTF(*<path>*, *format*)

Saves the file in the specified quoted *path* and with the quoted *format*. If you omit the *path* argument, you are prompted to name and save the file when running the script.

---

db<<Save Text(*<path>*, *format*)

Saves a file containing the text of the box in the quoted *path* and with the specified quoted *format*. If you omit the *path* argument, you are prompted to name and save the file when running the script.

---

db<<Scroll Window(*Display Box*|*relative-vertical-pixels*|*relative-horizontal-pixels*, *relative-vertical-pixels*|{*absolute-vertical-pixels*, *absolute-horizontal-pixels*})

Scrolls the containing window.

---

db<<Select

db<<Deselect

Selects (highlights) or deselects the box.

`db<<Set Menu Item State(`*`index`*`, 0|1|-1)`

Sets the popup menu item at *index* to be normal (0), selected (1), or disabled (-1).

`db<<Set Page Setup<Margins(`*`left`*`, `*`right`*`, `*`top`*`, `*`bottom`*`)>,`
`<Scale(`*`s`*`)>,<Portrait(`*`Boolean`*`)>, <Paper Size(`*`paper size`*`)>)`

`db<<Set Page Setup<Margins({`*`left`*`, `*`right`*`, `*`top`*`, `*`bottom`*`})>,`
`<Scale(`*`s`*`)>,<Portrait(`*`Boolean`*`)>, <Paper Size(`*`paper size`*`)>)`

Sets the page settings. Margins are set in inches. Scale variable *s* is a number in the range of 10 (for 1000%) to 0.2 (for 20%) with the default as 1 (for 100%). If `Portrait` is True the page is oriented for portrait, otherwise the page is landscape. Paper Size is a quoted string specifying the paper size, for example, `"Letter"` or `"Legal"`.

Example

The example below creates a new window and configures the page setup.

```
w = New Window( "Window",
   Text Box( "Page Setup Test" )
);
w << Set page setup(
   margins( 1, 1, 1, 1 ),
   scale( 1 ),
   portrait( 1 ),
   paper size( "Letter" )
);
```

`db<<Set Print Headers(`*`left header`*`, `*`center header`*`, `*`right header`*`)`

Description

Sets the left, center, and right header for print output.

Example

```
w = New Window( "Window", Text Box( "Header Example" ) );
w << Set Print Headers(
   "Today is: &d;", // left
   "&wt;", // center
   "Page &pn; of &pc;" // right
);
w << Print Window;
```

`db<<Set Print Footers(`*`left footer`*`, `*`center footer`*`, `*`right footer`*`)`

Description

Sets the left, center, and right footer for print output.

Example

```
w = New Window( "Window", Text Box( "Footer Example" ) );
```

```
w << Set Print Footers(
   "Today is: &d;", // left
   "&wt;", // center
   "Page &pn; of &pc;" // right
);
w << Print Window;
```

---

db<<Set Stretch(x,y)

Description

Sets the horizontal and vertical stretching of display boxes.

Arguments

"Window" The box stretches to fill the space provided by the window, The sizing depends on the minimum and maximum properties for the box.

"Fill" The box stretches to fill the available space in its container.

"Off" The box does not stretch.

"Neutral" Many boxes start in the neutral state. A leaf box (a box that has no children) that is neutral resolves to "Off". Most container boxes aggregate the stretching behavior (and minimum and maximum) of their child boxes.

---

db<<Set Submenu (*index*, *submenu count*)

Description

Sets the submenu items for the item (specified by index number) by specifying the number of items in the submenu.

Example

The example below creates a menu containing “A”, “B”, and “C” with “A” having a submenu “A1” and “A2” and “B” having a submenu “B1”, “B2”, and “B3”.

```
New Window( "title", ob = Outline Box( "title" ) );
ob << Set Menu Script(
   {"A", "", "A1", Print( "A1" ), "A2", Print( "A2" ),
   "B", "", "B1", Print( "B1" ), "B2", Print( "B2" ), "B3", Print( "B3" ),
   "C", Print( "C" )}
);
ob << Set Submenu(1, 2); // menu A with 2 items in submenu A1 and A2
ob << Set Submenu(4, 3); // menu B with 3 items in submenu B1, B2, and B3
```

---

db<<Set Report Title(*title*)

Sets a new *title*. The *title* is quoted.

Show Properties(*db*)

Shows the messages a given display box can interpret.

`db<<Sib`

Returns the sibling of the display box.

`db<<Sib Append(`*db2*`)`

Appends a display as a sibling to this one. The argument must evaluate to a display box owner or reference.

`db<<Size Window(`*x*`, `*y*`)`

Resizes the containing window.

`db<<Update Window`

Updates the window that holds the display box (and possibly other windows as well, depending on the operating system) if there are invalidated regions. Previously invalidated box areas are redrawn with their new content.

Notes

In some interactive JSL scripts that combine sliders with JSL callbacks, you might need to use `<<Update Window` to keep parts of the display synchronized with the slider.

`db<<Zoom Window`

Resizes the window to be large enough to show all of its contents.

Axis Box Messages

`Axis Box<<Axis Settings(`*<named arguments>*`)`

Opens the Axis Specification window or specifies axis settings, such as tick marks and axis labels.

If no arguments are included, the axis specification window appears.

Otherwise, specify named arguments for each axis.

- Specify the Y axis as `Axis Box(1)`.
- Specify the X axis as `Axis Box(2)`.

Optional Named Arguments

All Axes

```
Scale("Linear"|"Log"|"Power"|"Geodesic"|"Geodesic US"|"Custom
  Scale"|"Normal Probability|Weibull Probability|Frechet
  Probability|Logistic Probability|Exponential Probability|Gamma
```

Probability|Beta Probability|Mixture of 2 Normals Probabilities|Mixture of 3 Normals Probabilities) Specifies the scale of the axis. If the type is Custom Scale, this message expects two additional named arguments: Scale to Internal(*expr*) and Scale to External(*expr*).

Min(*n*) Changes the minimum value on the axis.

Max(*n*) Changes the maximum value on the axis.

Reverse Order(*Boolean*) Reverses the axes by reversing the minimum and maximum values.

Inc(*n*) Shows the numbers at the specified increments.

Set Font(*font*) Specifies the quoted *font* that is applied to the numbers. The JMP Font preferences determine the default font.

Set Font Size(*points*) Specifies the size of the font that is applied to the numbers. The JMP Font preferences determine the default font.

Set Font Style("Strikeout"|"Underline") Specifies the quoted style that is applied to the numbers.

Automatic Font Size(*Boolean*) JMP attempts to decrease the font size (down to a certain minimum) if all of the labels cannot fit at the default size. If 0, the font size is not decreased.

Automatic Tick Marks(*Boolean*) Turns on tick marks only if one or more labels are hidden (due to insufficient space).

Label Orientation("Automatic"|"Horizontal"|"Vertical"|"Perpendicular"|"Parallel"|"Angled") Rotates the axis label. The default value is "Automatic", which is based on the width of the labels.

Lower Frame(*Boolean*) Shows a frame below the labels. The default value is off.

Value Labels Displays the label that you specify instead of the data value.

Inside Ticks(*Boolean*) Shows tick marks inside or outside of the axis.

Add Ref Line({Label Row Nesting(*n*), *begin range*, <*end range*>, <"Solid"|"Dotted"|"Dashed"|"DashDot"|"DashDotDot">, <*color*>, <*label*>, <width*(n)*>, <*opacity*(%)>}) Defines the reference line range, line pattern, color, label, width, and opacity. A solid, black, 1-pixel line is the default setting. Label Row Nesting(*n*) specifies the number of nested rows on the axis. The *color* and *label* arguments are quoted.

Categorical Axes

Wrap Lines(*n*) Wraps long labels across multiple lines (*n*).

Numeric Axes

Format(*arguments*) Specifies the format of the numeric axis data. See the Format list in a numeric column's column properties for arguments. If you specify the a datetime

format, also include the `Interval` argument: `"Numeric"`, `"Year"`, `"Quarter"`, `"Month"`, `"Week"`, `"Day"`, `"Hour"`, `"Minute"`, or `"Second"`.

`Minor Ticks(`*`number`*`)` Specifies the *number* of minor tick marks between major tick marks.

`Tick Offset(`*`number`*`)` Specifies the starting point of the tick marks.

`Major Ticks(`*`Boolean`*`)` Shows or hides a major tick mark between each number.

`Minor Ticks(`*`Boolean`*`)` Shows or hides a minor tick mark between each number.

`Show Major Grid(`*`Boolean`*`)` Shows or hides a grid line at each major tick mark.

`Show Minor Grid(`*`Boolean`*`)` Shows or hides a grid line at each minor tick mark.

`Major Grid Line Color(`*`color`*`)` Sets the color for the major grid (if enabled) using the quoted *color*.

`Minor Grid Line Color(`*`color`*`)` Specifies the color of the grid line at each minor tick mark.

Example

The following example creates a bivariate plot and defines basic settings for the X and Y axes.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
biv = dt << Bivariate( X( :height ), Y( :weight ), FitLine );
rbiv = biv << Report;
xaxis = rbiv[Axis Box( 2 )];
yaxis = rbiv[Axis Box( 1 )];
xaxis << Axis Settings( Show Major Grid( 1 ) );
yaxis << Axis Settings( Decimal( 10, 3 ) );
```

`Axis Box<<Add Axis Label(`*`quoted string`*`)`

Adds an axis label with the specified quoted string.

`Axis Box<<Add Ref Line(`*`number`*`, `*`linestyle`*`, <`*`color`*`>, <`*`label`*`>, <`*`width`*`>)`

Adds a reference line at *number* in the specified *linestyle* (`"Solid"|"Dashed"|"Double"`), *color* (using the quoted *color*), quoted *label*, and *width* (in pixels).

`Axis Box<<Decimal(`*`width`*`, `*`decimal places`*`)`

Changes the numeric format for axis values.

`Axis Box<<Format(`*`name`*`)`

Changes to the numeric format given by the quoted *name*.

`Axis Box<<Get Inc(n)`

Gets the increment value of the axis.

`Axis Box<<Inc(n)`

Sets the increment between ticks.

`Axis Box<<Interval(format)`

Specifies the units used for `Inc()` with date/time formats: `"Numeric"`, `"Year"`, `"Quarter"`, `"Month"`, `"Week"`, `"Day"`, `"Hour"`, `"Minute"`, or `"Second"`.

`Axis Box<<Label Orientation(format)`

Rotates the axis label to one of the following formats: `"Automatic"` (the default setting based on the width of the label), `"Horizonal"`, `"Vertical"`, `"Perpendicular"`, `"Parallel"`, and `"Angled"`.

`Axis Box<<Major Grid Line Color(color)`

Sets the color for the major grid (if enabled) using the quoted *color*.

`Axis Box<<Max(maximum)`

Changes the maximum value on the axis.

`Axis Box<<Minor Grid Line Color(color)`

Sets the color for the minor grid (if enabled) using the quoted *color*.

`Axis Box<<Min(minimum)`

Changes the minimum value on the axis.

`Axis Box<<Minor Ticks(number)`

Specifies the *number* of minor tick marks between major tick marks.

`Axis Box<<Remove Axis Label`

Removes any label added with `Add Axis Label`.

Axis Box<<Reverse Scale(*Boolean*)

Reverses the normal scale direction so that the highest value is on the left or bottom (that is, closest to the origin).

Axis Box<<Revert Axis

Restores the axis' original settings (from time of creation).

Axis Box<<Scale(*type*)

Changes the scale of the axis to *type* ("Linear"|"Log"|"Exp Prob"|"Weibull Prob"|"Logistic Prob"|"Frechet Prob"|"Normal"|"Cube Root"|"Johnson Su Scale"|"Geodesic"|"Geodesic US"|"Custom Scale"|"Power"|"Gamma Prob"|"Beta Prob"|"Mixture of 2 Normals Prob"|"Mixture of 3 Normals Prob").

If the type is Custom Scale, this message expects two additional named arguments: Scale to Internal(expr) and Scale to External(expr).

Axis Box<<Tick Font(*name*, *<size>*, *<style|style style...>*, *<angle>*)

Sets the font name (quoted), size, and quoted properties for tick marks. To specify more than one style, include a space between each style and place them in quotes.

Axis Box<<Show Labels(*Boolean*)

Shows or hides labels for the axis values.

Axis Box<<Show Major Grid(*Boolean*)

Adds or removes grid lines at the major tick values.

Axis Box<<Show Major Ticks(*Boolean*)

Shows or hides major tick marks.

Axis Box<<Show Minor Grid(*Boolean*)

Adds or removes grid lines at the minor tick values.

Axis Box<<Show Minor Ticks(*Boolean*)

Shows or hides minor tick marks.

Axis Box<<Tick Label List(*<i>*, *{text1,text2, ...}*,*<{n1, n2, ...}>*)

Sets the values and positions of the axis tick labels.

Note: Major tick increments are automatically set to 1.0 if the tick labels are not specified.

Required Arguments

{*text1*,*text2*, ...} Specifies the quoted string titles for your labels.

Optional Arguments

i Specifies the label row index. Leaving it out clears any existing label rows and creates one new one as specified. Including it allows you to override any particular label row; using an index higher than the current number of label rows adds a new label row on to the end.

{*n1*, *n2*, ...} Specifies the values corresponding to each label. If the value list is omitted, the labels will be on integer increments starting with 1.

Border Box Messages

This section contains JSL messages that apply to border boxes.

Note: Border boxes support only one display box argument.

Border Box<<Set Background Color({*r, g, b*}|<*color*>)

Sets the background color for a border box. Specify an optional quoted list of RGB values or *color*. For example:

```
border box<<Set Background Color("red");
```

or

```
border box<<Set Background Color( {255, 192, 3} );
```

Border Box<<Set Color({*r, g, b*}|<*color*>)

Sets the border color for a border box. Specify a list of RGB values or a quoted color. For example:

```
border box<<Set Color("red");
```

Border Box<<Get Color

Gets the border color for a border box.

Border Box<<Set Style(*style*)

Sets the border style for a border box. Specify the *style* as one of the following numbers or keywords: 0 ("Solid"), 1 ("Dotted"), 2 ("Dashed"), 3 ("DashDot"), or 4 ("DashDotDot"). For example:

```
border box<<Set Style("Dotted");
```

Border Box<<Get Style

Gets the border style for a border box.

Data Browser Box Message

dbb<<Set Data Table(*<data table>*)

Sets the data table for the data browser box.

Data Filter Source Box Message

dfsb<<Set Row States(*dt*, *rs*)

Sets the row states for the given data table within the filter. Selections made in this row state will not be linked with the data table, but will be included in the reports linked to the selection filter.

See Also

Frame Box Messages

Frame Box<<Add Graphics Script(*<order>*,*<description>*, *<script>*)

Description

Adds a script to draw graphics in the frame box.

Optional Arguments

order Specifies the order in which the graphics elements are drawn. The value can be the keyword "Back" or "Forward" or an integer that specifies the drawing order for a number of graphics element. 1 means the object is drawn first.

description A quoted string that appears in the Customize Graph window next to the graphics script. The *description* argument is quoted.

script A JSL script.

Example

In the following example, the graphics script draws the line first and then draws the other graphics elements: the grid lines, references lines, and markers that create the bivariate plot. Without the 1 order argument, the line is drawn last and covers up the markers.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
obj = dt << Bivariate( Y( :weight ), X( :height ) );
Report( obj )[FrameBox( 1 )] <<
Add Graphics Script(
  1, // draws the line first
```

```
    Description( "Pen Script" ),
    Pen Color( "red" );
    Pen Size( 5 );
    Y Function( 60 + 120 / 2 * (1 + Sine( (2 * Pi() * (x - 50)) / 22.5 )), x );
);
```

`Frame Box<<Append Seg`

Adds a display seg to the specified Frame Box.

`Frame Box<<Background Color({`*`RGB values`*`}|<`*`color`*`>)`

Changes the background color. Specify a list of RGB values or a quoted *color*.

`Frame Box<<Child Seg`

Returns the display seg child of the Frame Box.

`Frame Box<<Edit Graphics Script`

Brings up a dialog box to view, edit, or delete the current graphics scripts.

`Frame Box<<Find Seg`

Returns a display seg with the specified argument (for example, the name of a seg).

`Frame Box<<Frame Size(`*`x, y`*`)`

Resets the size of the frame, in pixel units.

`Frame Box<<Make Table of Graphs Like This`

Creates a data table of graphs.

`Frame Box<<Marker Size(`*`size`*`)`

Changes the marker size. The values are 0 (dot), 1 (small), 2 (medium), and so on.

`Frame Box<<Row Colors(`*color*`)`

`Frame Box<<Row Markers(`*marker*`)`

`Frame Box<<Row Exclude(`*Boolean*`)`

`Frame Box<<Row Hide(`*Boolean*`)`

`Frame Box<<Row Label(`*Boolean*`)`

Forwards commands to the data table associated with the report, so that the row states of selected rows can be manipulated. For `Row Exclude`, `Row Hide`, and `Row Label`, omitting the argument toggles the option. If the option is off, the message turns it on. If the option is on, the message turns it off.

`frame box<<Set Background Fill(`*Boolean*`)`

Enables or disables filling the background with the background color. Use this option when you want to paste a graph and make the background transparent.

Example

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
biv = Bivariate( y( weight ), x( height ) );
rbiv = biv << Report;
framebox = rbiv[Frame Box( 1 )];
// set background color
framebox << Background Color( "red" );
// for demonstration purposes: wait to see the color change
Wait( 1 );
// turn off background fill color
framebox << Set Background Fill( 0 );
```

`frame box<<X Axis(<Min(`*minimum*`)>, <Max(`*maximum*`)>, <Inc(`*n*`)>, <`*named arguments*`>)`

Scales the X coordinate system.

`frame box<<Y Axis(<`*Min(min)*`>, <`*Max(max)*`>, <Inc(`*n*`)>, <`*named arguments*`>)`

Scales the Y coordinate system.

Display 3D Box Message

`Graph 3D Box()`

Sends display commands to the 3D plot.

Excerpt Box Message

```
Excerpt Box(rptnum, lstSubscripts)
```

Returns a display box containing the excerpt designated by the report held at number `rptnum` and the list of display subscripts `lstSubscripts`. The subscripts reflect the current state of the report after previous excerpts have been removed.

Filter Col Selector Message

```
Filter Col Selector(<Data Table(name)>, <width(pixels)>, <nLines(n)>,
<script>, <onChange(expr)>)
```

Returns a display box that contains a list of items. The control supports column filtering.

Global Box Message

```
Global Box(value)
```

Creates a display box that shows the value of a global variable.

Hier Box Message

```
Hier Box(title, Hier Box(...), Hier Box(...), ...)
```

Returns a display box with the *title* (quoted) that contains a hierarchy of quoted strings.

Matrix Box Messages

```
Matrix Box<<Get
```

Returns the matrix contents.

```
Matrix Box<<Make Into Data
Table(<Invisible(Boolean)|Private(Boolean)>)
```

Description

Turns the matrix into a new data table. `Invisible(1)` hides the data table from view. An invisible data table can be open from the JMP Home Window or the Window menu. `Private(1)` opens the data table without displaying it in a data table window. A private

data table is generally for scripts that want better control of the data table by not exposing it to general use.

Returns

A reference to the new data table.

`Matrix Box<<Set Format(<width>, <decimal places>, <"Use Thousands Separator">)`

Description

Sets the numeric format for matrix elements.

Arguments

A number of other formats can be set on matrix boxes. See "Number Col Box<<Set Format(<width>|<width, decimal places>, <"Use Thousands Separator">)" for more information about the syntax.

`Matrix Box<<Sort(column number, ascending)`

Sorts the rows of the matrix based on the column number specified by `column_num`. The default sort order is ascending.

If `column number` is 0, the sort is removed.

ascending is a Boolean value. If `ascending` is `"True"`, the sort is performed in ascending order. If `ascending` is `"False"`, the sort is in descending order.

Nom Axis Box Messages

`Nom Axis Box<<Divider Lines(Boolean)`

Adds or removes divider lines between labels in the axis box.

`Nom Axis Box<<Lower Frame(Boolean)`

Adds or removes a lower frame around the axis.

`Nom Axis Box<<Rotated Tick Labels(Boolean)`

Rotates or unrotates the labels at each tick value.

Number Col Box Messages

`Number Col Box<<Add Element(`*`item`*`)`

Adds the *item* to the Number Col Box. *item* can be a single number, a list of numbers, or a matrix.

`Number Col Box<<Bootstrap(`*`nsample`*`, Random Seed(`*`number`*`), Fractional Weights(`*`Boolean`*`), Split Selected Column(`*`Boolean`*`), Discard Stacked Table if Split Works(`*`Boolean`*`)`

Description

Bootstraps the analysis, repeating it many times with different resampling weights and collecting tables as selected.

Arguments

`nsample` Sets the number of times that you want to resample the data and compute the statistics. A larger number results in more precise estimates of the statistics' properties. By default, the number of bootstrap samples is set to 2,500.

`Random Seed(`*`number`*`)` Sets a random seed that you can re-enter in subsequent runs of the bootstrap analysis to duplicate your current results. By default, no seed is set.

`Fractional Weights(`*`Boolean`*`)` Performs a Bayesian bootstrap analysis. In each bootstrap iteration, each observation is assigned a weight that is calculated as described in *Basic Analysis*. The weighted observations are used in computing the statistics of interest. By default, the fractional weights option is not selected and a simple bootstrap analysis is conducted.

`Split Selected Column(`*`Boolean`*`)` Places bootstrap results for each statistic in the column that you selected for bootstrapping into a separate column in the Bootstrap Results table. Each row of the Bootstrap Results table (other than the first) corresponds to a single bootstrap sample.

If you exclude this option, a Stacked Bootstrap Results table appears. For each bootstrap iteration, this table contains results for the entire report table that contains the column that you selected for bootstrapping. Results for each row of the report table appear as rows in the Stacked Bootstrap Results table. Each column in the report table defines a column in the Stacked Bootstrap Results table.

`Discard Stacked Table if Split Works(`*`Boolean`*`)` (Applicable only if the Split Selected Column option is included.) Determines the number of results tables produced by Bootstrap. If the `Discard Stacked Table if Split Works` option is not selected, then two Bootstrap tables are shown. The Stacked Bootstrap Results table, which contains bootstrap results for each row of the table containing the column that you selected for bootstrapping, gives bootstrap results for every statistic in the report, where each column is defined by a statistic. The unstacked Bootstrap Results table,

which is obtained by splitting the stacked table, provides results only for the column that is selected in the original report.

`Number Col Box<<Get`

`Number Col Box<<Get(`*i*`)`

Gets the values in a list, or the *i*th value.

`Number Col Box<<Get As Matrix`

Gets the values in a matrix, specifically a column vector.

`Number Col Box<<Get Format`

Returns the current format.

`Number Col Box<<Get Heading`

Returns the column heading text.

`Number Col Box<<Remove Element(`*row number*`)`

Removes an element from the column at the specified position.

Number Col Box<<Set Format(*<width>*|*<width, decimal places>*, <"Use Thousands Separator">)

Number Col Box<<Set Format("Best", *<width>*, <"Use Thousands Separator">)

Number Col Box<<Set Format(("Fixed Dec"|"Percent"), *<width>*|*<width, decimal places>*, <"Use Thousands Separator">)

Number Col Box<<Set Format("Pvalue", *<width>*)

Number Col Box<<Set Format(("Scientific"|"Engineering"|"Engineering SI"), *<width>*|*<width, decimal places>*)

Number Col Box<<Set Format("Precision", *<width>*|*<width, decimal places>*, <"Use Thousands Separator">, <"Keep Trailing Zeroes">, <"Keep All Whole Digits">)

Number Col Box<<Set Format("Currency", *<currency code>*, *<width>*|*<width, decimal places>*, <"Use Thousands Separator">)

Number Col Box<<Set Format(*datetime*, *<width>*, *<input format>*)

Number Col Box<<Set Format(("Latitude DDD"|"Latitude DDM"|"Latitude DMS"|"Longitude DDD"|"Longitude DDM"|"Longitude DDM"), *<width>*|*<width, decimal places>*, ("PUN"|"DIR"|"PUNDIR"))

Number Col Box<<Set Format("Custom", Formula(...), *<width>*, *<input format>*)

Description

Sets the column format.

Arguments

Using JMP describes the arguments. Note that `Matrix Box()`, `Number Col Box()`, `Number Col Edit Box()`, `Number Edit Box()` have the same `Set Format` syntax.

Examples

```
<<Set Format( 10, 2, "Use thousands separator");
<<Set Format( "Currency", "EUR", 20, );
<<Set Format( "m/d/y", 10 );
<<Set Format( "Precision", 10, 2, "Keep trailing zeroes", "Keep all whole
  digits" );
<<Set Format( "Latitude DDD", "PUNDIR"); // "PUN" for punctuation, "DIR" for
  direction, PUNDIR for both
<<Set Format( "Custom", Formula( Abs( value ) ), 15 );
```

Notes

- For a list of currency codes, see *Scripting Guide*. The currency code is based on the locale if the code is omitted.

- If you don't specify the format, set the decimal places to greater than 100 for datetime values and to 97 for *p*-values.
- You must always precede the number of decimal places with the width.
- Options can be defined in a list or a variable, or they can be in a `Function()` that is evaluated.

```
ncbFunc = Function({}, {"Fixed", 12, 5});
```

`number col box<<Set Heading(`*quoted string*`)`

Changes the column heading text.

Number Col Edit Box Messages

`Number Col Edit Box<<Set Format(`*<width>*`, `*<decimal places>*`, <"Use Thousands Separator">|`*<other options>*`)`

Description

Sets the column format.

Arguments

A number of other formats can be set on number col edit boxes.

See Also

"Number Col Box<<Set Format(<width>|<width, decimal places>, <"Use Thousands Separator">)"

`Number Col Edit Box<<Remove Element(`*x position*`, `*y position*`, `*i*`)`

Removes an element from the column at the specified position.

Number Edit Box Message

`Number Edit Box<<Set Format(`*<width>*`, `*<decimal places>*`, <"Use Thousands Separator">|`*<other options>*`)`

Description

Sets the column format.

Arguments

A number of other formats can be set on number edit boxes.

See Also

"Number Col Box<<Set Format(<width>|<width, decimal places>, <"Use Thousands Separator">)"

Outline Box Messages

Outline Box<<Close(*Boolean*)

Closes the outline box.

Outline Box<<Close All Below

Closes all the node's child nodes.

Outline Box<<Close All Like This

Closes all nodes similar to this outline box.

Outline Box<<Close Where No Outlines

Closes all nodes that do not have children.

Outline Box<<Get Title

Gets the title of the outline box.

Outline Box<<Horizontal(*Boolean*)

Horizontally aligns the node's children.

Outline Box<<Open All Below

Opens all the node's child nodes.

Outline Box<<Open All Like This

Opens all nodes similar to this outline box.

Outline Box<<Set Menu Script(*{quoted string1, script1, quoted string2, script2, ...}*)

Adds an entry to the menu when the red triangle on an outline box is selected.

Outline Box<<Set Title(*title*)

Specifies the quoted *title* of the outline box.

Panel Box Messages

`Panel Box<<Get Title`

Gets the title of the panel box.

`Panel Box<<Set Title(`*`title`*`)`

Specifies the *`title`* (quoted) of the panel box.

Plot Col Box Messages

`Plot Col Box<<Get As Matrix`

Gets the values in a matrix, specifically a column vector.

`Plot Col Box<<Get Labels`

Gets the labels for each row.

`Plot Col Box<<Remove Element(`*`row number`*`)`

Removes an element from the column at the specified position.

`Plot Col Box<<Set Labels(`*`{list}`*`)`

Sets the labels for each row.

`Plot Col Box<<Set Scale(`*`minimum`*`, `*`maximum`*`, <"format">, `*`<width>`*`, `*`<decimal places>`*`, <"Use Thousands Separator">)`

Description

Specifies the minimum and maximum values of the horizontal axis.

Arguments

The format arguments specify a format that is used when the first two arguments are 0 and 1. A number of formats can be set on plot col boxes. See "Number Col Box<<Set Format(<width>|<width, decimal places>, <"Use Thousands Separator">)" for more information about the syntax.

`Plot Col Box<<Set Values(`*`[matrix]`*` or `*`{list}`*`)`

Sets values for the matrix (for numeric variables) or list (for character variables).

Slider Box and Range Slider Box Messages

Slider Box<<Get(*<index>*)

Range Slider Box<<Get Lower(*<index>*)

Range Slider Box<<Get Upper(*<index>*)

Returns the current value of the slider.

Slider Box<<Get Max()

Returns the maximum value possible for the range slider and slider.

Slider Box<<Get Min()

Returns the minimum value possible for the range slider and slider.

Slider Box<<Get Var

Range Slider Box<<Get Lower Var

Range Slider Box<<Get Upper Var

Returns the variable name associated with the slider.

Slider Box<<Set(*float*, *<index>*, <Run Script(*Boolean)>*)

Range Slider Box<<Set Lower(*float*, *<index>*, <Run Script(*Boolean*)>)

Range Slider Box<<Set Upper(*float*, *<index>*, <Run Script(*Boolean*)>)

Sets the value of the slider. Run Script(*Boolean*) controls whether an on-change script runs after the Set, Set Lower, or Set Upper message.

Slider Box<<Set Max(*float*, *<index>*)

Sets the maximum value possible for the range slider and slider.

Slider Box<<Set Min(*float*, *<index>*)

Sets the minimum value possible for the range slider and slider.

Slider Box<<Set Script(*<script>*)

Sets a script to be run when the range sliders and slider is updated.

`Slider Box<<Set Var(`*slider variable*`)`

`Range Slider Box<<Set Lower Var(`*slider variable*`)`

`Range Slider Box<<Set Upper Var(`*slider variable*`)`

Sets the variable name associated with the slider.

String Col Box Messages

`string Col Box<<Add Element(`*item*`)`

Adds the item to the quoted string Col Box. Item can be a single quoted string or a list of quoted strings.

`String Col Box<<Get`

`string Col Box<<Get(`*i*`)`

Gets the values in a list or the *i*th value.

`String Col Box<<Get Heading`

Returns the column heading text.

`String Col Box<<Remove Element(`*row number*`)`

Removes an element from the column at the specified position.

`String Col Box<<Set Allow Text Search(`*Boolean*`)`

Description

In table boxes with selectable rows, allows a quoted string column that has focus to respond to keyboard input to change the selected row.

Example

```
// Run the example.
// Select K2.
// Type the letter g. Notice the last row is selected.
// Type the letters ki. Notice the third row is selected.
New Window( "Mountains",
   tb = Table Box(
      sb =
      String Col Box( "Mountain",
         {"K2", "Delphi", "Kilimanjaro",
         "Grand Teton"}
      ),
```

```
        Number Col Box( "Elevation (meters)",
           {8611, 681, 5895, 4199}
        ),
        Plot Col Box( "", {8611, 681, 5895, 4199} )
      )
   );
   tb << Set Selectable Rows( 1 );
   sb << Set Allow Text Search( 1 );
```

`String Col Box<<Set Heading(`*`title`*`)`

Changes the column heading specified in the quoted *`title`*.

`String Col Box<<Set Justify(`*`justification`*`)`

Specifies the alignment of the contents in the quoted string col box to `"Right"`, `"Left"`, or `"Center"`.

Tab Box Messages

`Tab Box<<Get Tab Margin()`

Returns a list of the current margins in pixels for the tab box in this order: `Left, Top, Right`, and `Bottom`.

`Tab Box<<Set Style("Tab"|"Combo" |"Outline"|"Vertical Spread"|"Horizontal Spread"|"Minimize Size")`

Changes the appearance of the tab box from a tab to a combo box or outline node.

`"Vertical Spread"` and `"Horizontal Spread"` change the orientation of the tab title.

`"Minimize Size"` bases the tab style on the width of the tab title.

`Tab Box<<Set Tab Margin(`*`n`*`|{...})`

Sets the tab margin for the tab box. If a single number is specified, all four margins are set to that number of pixels. If a list of two numbers is specified, the left and right margins are set to the first number, and the top and bottom margins are set to the second number. If a list of four numbers is specified, the margins are set in this order: `{left, top, right, bottom}`.

`Tab Box<<Show Tabs(`*`Boolean`*`)`

Shows or hides the tabs for tab boxes. If you hide the tabs, you need to provide another way to select and show tabs. For example, a list box that contains a list of references to the tabs. The default value is 1.

Table Box Messages

`Table Box<<Bootstrap(`*`nsample`*`, Random Seed(`*`number`*`), Fractional Weights(`*`Boolean`*`), Split Selected Column(`*`Boolean`*`), Discard Stacked Table if Split Works(`*`Boolean`*`)`

Description

Bootstraps the analysis, repeating it many times with different resampling weights and collecting tables as selected.

See Also

"Number Col Box Messages"

`Table Box<<Get`

Gets the entries of the table in list form.

`Table Box<<Get As Matrix(<"Visible">)`

Gets the numeric entries of the table in matrix form. `"Visible"` means that only visible columns will be included.

`Table Box<<Get Click Sort`

Returns 1 if the table can be sorted by clicking a column header and 0 otherwise.

`Table Box<<Get Locked Columns`

Returns the number of columns that cannot be dragged with the cursor or have any columns dropped before them.

`Table Box<<Get Row Change Function`

Returns the expression that is evaluated when a row is selected.

`Table Box<<Get Selectable Rows`

Returns True if the table box currently allows row selection.

Table Box<<Get Selected Row Color

Returns the index number of the background color of the selected rows in the table box.

Table Box<<Make Combined Data Table

Returns a reference to the data table. Same as `Make Data Table`, but also searches the report for report tables with the same columns and combines all of these into the new data table.

Table Box<<Make Data Table(*name*)

Returns a reference to the data table. Turns the table entries into a new data table with the quoted *name* argument.

Table Box<<Reorder Columns(*from column index*, *to column index*)

Puts the column specified with *from column index* in the place of the column specified with *to column index*. The indexes are 0-based. For example, indicate the first column with "0", and indicate the second column with "1".

Table Box<<Set Cell Changed Function(Function({*this*, *col box*, *row*},<*script*>))

Description

Sets a function that is called whenever the user edits a cell in a column in a table.

Example

This example prints the new values for the changed cell to the log.

```
New Window( "Mountains",
   tb = Table Box(
      quoted string Col Edit Box(
         "Mountain",
         {"K2", "Delphi", "Kilimanjaro",
         "Grand Teton"}
      ),
      Number Col Edit Box(
         "Elevation (meters)",
         {8611, 681, 5895, 4199}
      ),
      Plot Col Box( "", {8611, 681, 5895, 4199} )
   )
);
tb <<
Set Cell Changed Function(
   Function( {this, col, row},
```

```
        Print(
           (col << Get Heading) || ": row:" ||
           Char( 3 ) || " is now " ||
           Char( col << Get( row ) )
        )
     )
  );
```

`Table Box<<Set Click Sort(`*Boolean*`)`

Specifies if the table can be sorted by clicking a column header.

`Table Box<<Set Column Borders(`*Boolean*`)`

Draws a line on each side of the column.

`Table Box<<Set Heading Column Borders(`*Boolean*`)`

Draws a line on each side of the column headings.

`Table Box<<Set Locked Columns(`*n*`)`

Locks the first *n* columns. You cannot drag the locked columns or drag columns before them.

`Table Box<<Set Row Borders(`*Boolean*`)`

Draws a line above and below each row.

`Table Box<<Set Row Change Function(`*function*`)`

Sets the expression that is evaluated when a row is selected.

`Table Box<<Set Selectable Rows(`*Boolean*`)`

Makes the rows of the table box selectable or not.

`Table Box<<Set Selected Row Color(`*color*`)`

If the rows of the table box are selectable (`Set Selectable Rows(True)`), sets the background color (specified in the quoted *color* argument) for the selected rows.

`Table Box<<Set Shade Cells(`*Boolean*`)`

Shades the background of every cell in the table.

Table Box<<Set Shade Alternate Rows(*Boolean*)

Shades the background of every other row in the table.

Table Box<<Set Shade Heading(*Boolean*)

Shades the background in column headings.

Table Box<<Set Underline Headings(*Boolean*)

Draws a line underneath the column headings.

Table Box<<Sort By Column(*<column number|column title>*, <Ascending(*Boolean*)>

Sorts all rows based on the values in the specific column number or quoted column title. The default order sorting is descending.

Text Box Messages

Text Box<<Font Color(*n*)

Sets the color for `Text` quoted strings.

Text Box<<Get Hidden State

Returns the current state of a text box.

Text Box<<Get Text

Returns the quoted string content of the box.

Text Box<<Get Tip

Returns the tooltip for the text box (or a text edit box).

Text Box<<Markup

Returns text formatted with the specified HTML tags. The HTML must be well-formed; make sure you close nested tags correctly.

The following example returns text formatted in bold, italic, and underlined.

```
w = New Window( "Formatted Text",
   Text Box( "This is <b>bold</b> text. This is <b><i>bold italic</i></b>
   text. This is <u>underlined</u> text.",
   <<Markup) );
```

`Text Box<<Rotate Text(direction)`

Rotates the text 90 degrees `"Left"` or `"Right"`, or returns it to horizontal.

`Text Box<<Set Font(name, <size>, <style|style style...>, <angle>)`

Sets the font specified in the quoted *name* argument and the properties for text quoted strings. To specify more than one style, include a space between each *style* and place them in quotes.

`Text Box<<Set Font Size(n)`

Sets the font size in points for text quoted strings.

`Text Box<<Set Script(script)`

Associate a script with a text box. The script executes when the user presses Enter (or the text edit box otherwise loses focus).

`Text Box<<Set Text(quoted string)`

Sets the text in the box as specified in the quoted *string* argument.

`Text Box<<Set Tip(quoted string)`

Sets the tooltip for the text box (or a text edit box) as specified in the quoted *string* argument.

`Text Box<<Set Wrap(n)`

Set the wrap point, in pixels, in pixels (*n*).

Tree Node and Tree Box Messages

For the following JSL messages, `node` stands for a tree node or a reference to one and `root` stands for a tree box or a reference to one.

Caution: If you send a root node that contains one or more nodes with the `Set Node Select Script` defining a collapse message, then macOS runs the script twice. Windows doesn't run the script. This behavior on macOS doesn't just affect increments. Any script runs twice. It will print to the log twice, create a column twice, try to delete something twice, and so on.

`node<<Append(<node>)`

Inserts a referenced tree node after this node's children.

`node<<Collapse`

Closes the node. The behavior is not guaranteed if the node has a collapsed parent.

`node<<Expand`

Opens the node. The behavior is not guaranteed if the node has a collapsed parent.

`node<<Get Dimmed(<node>)`

Gets the option to dim text (decrease the opacity) for the node.

`node<<Get Font Style(<node>)`

Gets the font style for the node.

`node<<Get Tip`

Returns the tooltip for the node.

`node<<Prepend(<node>)`

Inserts a tree node before this node's children.

`node<<Remove`

Removes the given tree node and all its children from the tree display box.

`node<<Set Dimmed(Boolean)`

Sets the option to dim text (decrease the opacity)) for the node.

`node<<Set Font Style("Plain"|"Bold")`

Specifies the font style for the node.

`node<<Set Selected(<node>)`

Selects the node. The behavior is not guaranteed if the node has a collapsed parent.

`node<<Set Tip(tooltip)`

Sets a tooltip for the node. The *tooltip* argument is quoted.

`root<<Collapse(<node>)`

Collapses the given tree node.

`root<<Expand(<node>)`

Expands the given tree node.

`root<<Get Selected(<node>)`

Gets the currently selected tree node.

- In a single-item tree, the currently selected tree node or `Empty` is returned.
- Table 3.1 shows the results for a `Tree Box()` that contains the `MultiSelect` argument.

Table 3.1 Multi-Select Tree Results

Items Selected in Tree	Result
no items selected	empty
single item selected	list of one tree node
multiple items selected	list of selected tree nodes

`root<<Is Multiselect`

Returns 1 for a `MultiSelect` tree and 0 for a single-select tree.

`root<<Set Selected(node|{nodes}, <Boolean>)`

Selects the given tree node in the tree display box. In list of tree nodes, all nodes in the list are selected for `MultiSelect` trees. Otherwise, the first node in the list is selected. Specify the Boolean argument to indicate whether the node or nodes should be selected or unselected. The default value is 1, which selects the nodes.

Notes

- On Windows, the `Set Selected` message expands all nodes between the selected node or nodes and the root of the tree; items that are selected deep within the tree are shown. The expansion state does not change for nodes that were previously selected.
- On macOS, the `Set Selected` message does not change the tree expansion state.

Triangulation Messages

For the following JSL messages, `tri` stands for a triangulation or a reference to one.

`tri<<Get N Points`

Returns the number of unique points in the triangulation.

`tri<<Get Points`

Returns the coordinates of the unique points in the triangulation.

`tri<<Get Y`

Returns the Y values of the unique points in the triangulation.

`tri<<Get N Hull Points`

Returns the number of points on the boundary of the triangulation.

`tri<<Get Hull Points`

Returns the indices of the points on the boundary of the triangulation.

`tri<<Get N Hull Edges`

Returns the number of edges on the boundary of the triangulation.

`tri<<Get Hull Edges`

Returns the indices of the edges on the boundary of the triangulation.

`tri<<Get Hull Path`

Returns the boundary of the triangulation as a path.

`tri<<Get N Triangles`

Returns the number of triangles.

`tri<<Get Triangles`

Returns the indices of the triangles in the form of an Nx3 matrix.

`tri<<Get N Edges`

Returns the number of edges in the triangulation.

`tri<<Get Edges`

Returns the indices of the edges in the form of an *N*x2 matrix.

`tri<<Subset(`*{indices}*`)`

Returns a triangulation resulting from the given subset of points.

`tri<<Peel`

Peel the boundary layer of a triangulation, returning a new triangulation.

Window Messages

`window<<Bring Window to Front`

Brings the window to the front.

`window<<Close Window(<nosave>)`

Closes the window. If the optional argument nosave is specified, the window (journal, report, and so forth) is closed without saving or prompting.

`window<<Get Content Size`

Returns the size of the window's contents.

`window<<Get Window Icon`

Returns the name of the window's icon.

`window<<Get Window Position`

Returns the position of the window.

`window<<Get Window Size`

Returns the size of the window.

`window<<Get Window Title`

Returns the title of the window.

`window<<Inval`

Invalidate the display box. The window updates either when the `<<Update Window` message is sent or when the operating system has time for the update. See "window<<Reshow" for another method.

`window<<Maximize Display`

Maximizes the window. Deprecated.

`window<<Maximize Window(Boolean)`

Maximizes the window. Deprecated.

`window<<Minimize Window(Boolean)`

Minimizes the window.

`window<<Move Window(x, y)`

Moves the window to the specified position.

`window<<On Close(script)`

Runs the script when the window is closed.

`window<<Pad Window(Boolean)`

Turns padding around a window's contents on (1) or off (0). The default value is off.

`window<<Print Window`

Prints the window to the default printer. Note that the Print window is not opened and user input is not required.

`window<<Reshow`

Invalidates the display box and updates the window with the new content. See `<<Inval` and `<<Update Window` messages if more control over timing of the update is required.

`window<<Save Window to Report(pathname, <Embed Data(Boolean)>)`

Saves the current report window to a JMP report file (.jrp).

`window<<Set Main Window`

Sets the specified window as the default window that appears when JMP is run.

`window<<Set Window Icon(icon name)`

Sets the window's icon as specified in the quoted *icon name* argument.

`window<<Set Window Size(x, y)`

Resizes the window.

`window<<Show Window(Boolean)`

1 shows the window (only if the window is not currently open). 0 hides the window. If the window is also minimized (on Windows) or docked (on macOS), showing the window restores it to the normal state and brings it to the front.

`window<<Size Window(x, y)`

Resizes the window.

`window<<Update Window`

Updates or refreshes the window holding the display box if there are invalidated regions. See "window<<Inval" and "window<<Reshow" for additional methods.

`window<<Window Class Name`

Returns the name of the window class for the display box. Valid responses include: DataTable, FormulaEditor, Starter, Journal, Launcher, Report, Dialog, DialogWithMenu, ModalDialog, FindReplace, User, Generic, ToolWindow, FindReplace, AppBuilder, and Debugger.

`window<<Zoom Window`

Resizes the window to be large enough to show all of its contents.

Dynamic Link Library (DLL) Messages

`dll object<<Call DLL(function name, signature, arguments)`

Calls the specified function in the DLL with the specified signature and arguments.

`dll object<<Declare Function(name, <named arguments>)`

Description

Declares the return type and argument types for the specified function so that it can be successfully invoked. You can use one of the named arguments for `Convention`: `"STDCALL"` or `"PASCAL"`, or `"CDECL"`. The *type* argument for `Returns` takes the same named arguments as `Arg`. The *name* argument is quoted.

Optional Named Arguments

`Alias(name)` Specifies a quoted *name* that you can include if you don't like the name encoded in the DLL.

`Arg(type, <description>, <access mode>, <array>)` `Arg` can appear multiple times, once for each argument to be sent to the function.

`type` is one of these keywords that specifies the argument type: `"Int8"`, `"UInt8"`, `"Int16"`, `"UInt16"`, `"Int32"`, `"UInt32"`, `"Int64"`, `"UInt64"`, `"Float"`, `"Double"`, `"AnsiString"`, `"UnicodeString"`, `"Struct"`, `"IntPtr"`, `"UIntPtr"`, or `"ObjPtr"`.

`description` is a quoted string that describes the argument for reference.

`access mode` is an optional keyword that specifies how the argument is passed. `"input"` specifies that the argument is passed by value. `"output"` specifies that the argument is passed by address with the initial value undefined. `"update"` specifies that the argument is passed by reference and the value of the JSL variable is set as the initial value. The default value is `"input"`.

`array` is an optional keyword. It is valid only if the *type* is specified as `"Double"` and the `access mode` is specified as either `"input"` or `"update"`. Specifies that the exported function expects an array of doubles.

`Convention(calling convention)` Specifies the calling convention: `"STDCALL"` or `"PASCAL"`, or `"CDECL"`. The default value is `"STDCALL"`. `STDCALL` and `PASCAL` are equivalent.

`MaxArgs(n)` Specifies the maximum number of arguments that can be supplied.

`MinArgs(n)` Specifies the minimum number of arguments that can be supplied.

`Returns(type)` Specifies the data type that the function returns: `"Int8"`, `"UInt8"`, `"Int16"`, `"UInt16"`, `"Int32"`, `"UInt32"`, `"Int64"`, `"UInt64"`, `"Float"`, `"Double"`, `"AnsiString"`, `"UnicodeString"`, `"Struct"`, `"IntPtr"`, `"UIntPtr"`, or `"ObjPtr"`.

`StackOrder(order)` Specifies the order in which arguments are placed on the stack when calling the function. Valid values are `"L2R"` (left-to-right) and `"R2L"` (right-to-left). The default value is `"R2L"`.

`StackPop(pop)` Specifies how the exported function expects the stack to be cleared after the function returns. Valid values are `"CALLER"` and `"CALLEE"`. The default value is `"CALLEE"`.

`StructArg(Arg(...), <Arg(...)>, ..., <access mode>, <pack mode>, <description>)` Can appear multiple times. If an exported DLL function requires that a structure argument be passed in as an argument, use `StructArg` to declare the structure members. The `Arg` arguments use the same syntax as for `Arg` arguments to `Declare Function` (one for each structure member), an `access mode` indicator and a `pack mode` indicator.

`access mode` is an optional keyword that indicates whether the struct argument should be passed by value (`input`) or by reference (`update`).

`pack mode` is an optional integer that determines how the structure is packed. Valid values are 1, 2, 4, 8, and 16. The default value is 8.

`description` is an optional, quoted string that contains a description of the structure for reference.

`dll object<<Get Declaration JSL`

Sends the declaration JSL from the DLL object to log.

`dll object<<Load DLL(`*`path`*`, <AutoDeclare(`*`Boolean`*`|"Quiet"|"Verbose")>)`

`dll object<<Load DLL(`*`path`*`, <"Quiet"|"Verbose">)`

Description

Loads the DLL from the specified path.

Required Argument

`path` A quoted path that specifies where to load the DLL.

Optional Named Arguments

`AutoDeclare(`*`Boolean`*`|"Quiet"|"Verbose")` `AutoDeclare(1)` and `AutoDeclare("Verbose")` write verbose messages to the log. `AutoDeclare("Quiet")` turns off log window messages. If you omit this option, verbose messages are written to the log.

`Quiet|Verbose` When you use `Declare Function`, this option turns off log window messaging (`"Quiet"`) or turns on log window messaging (`"Verbose"`).

`dll object<<Show Functions`

Sends the declared functions for the DLL object to the log.

`dll object<<Unload DLL`

Unloads the DLL.

Image Messages

This section contains JSL messages that apply to images.

Related Information

- The Scripting Index provides examples for processing images. In JMP, select **Help > Scripting Index** to view this interactive resource.
- Additional resources are available from the JMP File Exchange at https://community.jmp.com/community/file-exchange.

`img<<Crop(Left(`*`pix`*`), Right(`*`pix`*`), Top(`*`pix`*`), Bottom(`*`pix`*`))`

Creates a new image from an existing image to the specified dimensions (in pixels).

`img<<Filter(name, <n>)`

Filters the image based on the specified algorithm. Filtering is useful for cleaning up noise in the image.

Note: All of the JMP image filters are supported at the operating system level. Images that are processed on Windows might differ from images processed on macOS.

Argument

`name` Specifies the quoted name of a JMP image filter. The following filters are available:

- `"Despeckle"` removes defects (that is, speckles) from a scanned or captured image (for example, scratches, dust, etc.).
- `"Edge"` identifies pixels in an image where the brightness changes sharply and darkens pixels with no sharp change. Edge detection is used to detect changes in surface, depth, material, and lighting.
- `"Enhance"` reduces the contrast between pixels in a noisy image.
- `"Median"` reduces noise (that is, the random variation) and smooths an image by comparing each pixel's brightness with its neighbors' and, if the value is very different, replaces it with the average of the neighbors' values.
- `"Negate"` creates the negative of the color or gray-scale image by changing each pixel color to its complementary color.
- `"Normalize"` changes a color image's pixels to use the full range of the file format's number system. Normalization will make the image's colors more intense.
- `"Sharpen"` reduces blur by sharpening edges of an image.
- `"Contrast"`, *n* brightens or darkens an image. A higher number (>0.0) brightens an image; a lower number (<0.0) darkens an image.
- `"Gamma"`, *n* corrects the image visual display (brightness and intensity) to account for differences in monitor hardware. A higher number (> 1.0) lightens the image; a lower number (< 1.0) darkens the image.
- `"Reduce Noise"`, *n* reduces the random variation (or noise) that occurs with higher ISO sensitivity or longer exposure times.
- `"Gaussian Blur", radius, sigma` reduces image noise and detail creating a smoother image. Radius is equal to the blur radius around each pixel and sigma is the standard deviation of the Gaussian distribution. Gaussian blur is commonly used when resizing or performing edge detection.

`img<<Flip Both`

Flips the image from left to right and top to bottom.

`img<<Flip Horizontal`

Flips the image from left to right.

`img<<Flip Vertical`

Flips the image from top to bottom.

`img<<Get EXIF`

Returns EXIF data from the image (such as the shutter speed and aperture value) in an associative array.

`img<<Get N Frames`

Description

Returns the number of frames in a multi-frame TIF or animated GIF file, where the number of frames begins with frame 0.

Example

The following example places a four-frame TIF file in a new window and shows the image that is in the first frame.

```
img = New Image( "$DOWNLOADS/Multiframe.tif" );
nframes = img << Get N Frames(); // return 4
img << Set Current Frame( 1 ); // show image 1
win = New Window( "Multi-Frame TIFF", img );
```

`img<<Get Size`

`img<<Size`

Returns a list containing the width and height (in pixels) of the image.

`img<<Rotate(`*`degrees`*`)`

Rotates the image by the specified number of degrees.

`img<<Save Image(`*`path`*`)`

Saves the image to the quoted *path*.

`img<<Scale(`*`scale|xscale, yscale`*`)`

Resizes the image by the specified dimensions. Provide one argument to resize both the width and height. Provide two arguments to resize the width and height separately.

Examples

```
img = New Image( "$SAMPLE_IMAGES/tile.jpg" );
xs = 2;
img << Scale( xs );
New Window( "Tilex 2", img );

img = New Image( "$SAMPLE_IMAGES/tile.jpg" );
img << Scale( 2, 0.5 ); // scale image width by 2 and height by 1/2
New Window( "Tile squished", img );
```

Notes

Using `Scale` is an alternative to getting the size of the image, multiplying by the scale factor, and then setting the size.

`img<<Set Current Frame`

Description

Sets the frame that shows in a multi-frame TIFF or animated GIF file. Specify 0 through the number of frames minus 1. For example, with four frames, you can specify frame 0 through frame 3.

See Also

"img<<Get N Frames"

`img<<Set Size(`*`width`*`, `*`height`*`)`

Resizes the image to the specified dimensions (in pixels). To scale the image proportionally, specify a width and height that correspond to the aspect ratio in the original image.

`img<<Transparency(`*`fraction`*`)`

Sets the transparency for the image where the fraction is between 0.0 (full transparency) to 1.0 (no transparency).

Interactive HTML Messages

This section contains JSL messages for interactive HTML web reports.

Web Report Messages

`webreport<<Publish(<Add Image(...)>, <Add Report(...)>, <Add Reports(...)>, <Public(`*`Boolean`*`)>, <Index(...), <User Name(...)>,`

<Password(...)|*password function)*>,
<Prompt("IfNeeded"|"Always"|"Never")>, <URL(...)>, <Publish
Data(*Boolean*)>, Replace(<*id*>, Prompt("IfNeeded"|"Always"|"Never"))>)

Description

Publishes the web report to the JMP server.

Returns

On success, the URL of the published report is returned.

Optional Arguments

`Add Image` Inserts an image at the top of the index page. Valid formats are `"png"`, `"bmp"`, `"jpeg"`, `"jpg"`, `"tiff"`, and `"tif"`. `Title` and `Description` are optional. `Title` appears above the image. `Description` appears below the image. Use `File(`*`filepath`*`)` or just a quoted string. Here is an example:

```
webrpt << Add Image( File( "C:\Users\Public\JMP\Projects\WebJMP\atlas.jpg" ),
  Title( "Atlas" ), Description( "Holding up the world as always." ) );
```

`Add Report` Adds a report to publish within the web report.

`Public(`*`Boolean`*`)` Specifies whether the public has access to the report. By default, the report is private.

`Index` The name of the index page for multiple reports. You can also specify the description.

`User Name` Specifies the user name registered on the JMP server.

`Password` Specifies the user's password. You can also define a password function.

`Prompt` Displays a window in which the user types the server URL, user name, and password.

`URL` The location that you are publishing to.

`Publish Data(`*`Boolean`*`)` Includes the data in the HTML. Reports contains static rather than interactive images. In a public report, you might not want to share the data.

`Replace` Replaces the report. Get the URL from the address field in the browser where the page is displayed.

JMP Application Messages

This section contains JSL messages that apply to the JMP Application Builder and the JMP Dashboard Builder. Both of these applications use the same infrastructure to design and execute applications and dashboards. Because a dashboard is a special form of an application, this section uses the term *application* to describe how both dashboard and application objects use scripting.

See the Scripting Index in the JMP Help menu for examples.

JMP App Messages

The `JMP App` object is the main controller for JMP applications built by Application Builder or Dashboard Builder. Scripts both inside and outside of a JMP application can use a `JMP App` object.

A JMP application can have one of three states: initial (with no editor, and the application is not running), running, or editing. A `JMP App` object only exists in one state at a time; if you are editing a JMP Application and choose to run it, a copy of the JMP application is created before it is run.

`app<<Combine Windows({`*`reports or data tables`*`})`

Description

Combines the given list of platform reports or data tables into a new module. The application should be in the initial state when this message is sent.

`app<<Debug`

Invokes the JSL Debugger on a JMP application. The application script will run first. The Debugger then breaks as each module is created, invoking the module scripts. In the Debugger, set breakpoints to debug the scripts that are associated with the application or modules.

`app<<Edit`

Starts Application Builder on a JMP application that is in the initial state. There is no editor, and the application is not running.

`app<<Get Modules`

Gets the list of modules associated with an application. In Application Builder, each module corresponds to a tab in the workspace, which describes the layout and behavior for one type of window in the application.

`app<<Get Namespace`

The `JMP App()` object automatically creates an anonymous namespace for the variables created within the application script. Use this message to get a handle to this namespace to inspect or modify variables. There is a default symbol in this namespace called `thisApplication`, which holds a reference to the application itself.

`app<<Get Windows`

Gets a list of all windows created as instances of JMP app modules. Some modules might create more than one instance. All windows might not exist at the same time, so the number of windows might vary and might differ from the number of modules.

`app<<Open File(`*`path`*`)`

Resets the state of an existing application from a the .jmpapp or .jmappsource file. *`path`* is quoted.

`app<<Relaunch Analysis`

Creates a new copy of a running application and runs the new instance.

`app<<Run`

Runs the application. The application script runs first, and depending on settings, one or more JMP app module instance objects might be created automatically.

`app<<Save Script to Add-In`

`app<<Save Script to Data Table`

`app<<Save Script to Journal`

`app<<Save Script to Script Window`

Saves the script for the application to the given destination. An application script consists of a `JMP App()` object that contains the definition for the application. Scripts saved to an add-in, data table, or journal include a `Run` message to run the application. A script saved to the script window includes an `Edit` message to open Application Builder.

JMP App Module Messages

A JMP application module is a definition of the display box layout and behavior for a single component in a JMP application or dashboard. Depending on the module type, the component might represent a window in the application or just part of a window.

`module<<Create Instance`

Use `Create Instance` within a `JMP App()` or `JMP App Module()` script to create an instance of a JMP app module. By default, one instance of each JMP application module is created when an application is run. For more complex applications with multiple windows, such as a launcher and report combination, it might be necessary to change the default settings and control how the module instance is created.

`module<<Get Instance`

Returns a handle to the application that owns a module.

JMP App Module Instance Messages

The JMP application module instance is a running realization of a JMP app module, a window on the screen, or a collection of display box elements that can be inserted into another window.

`inst<<Create Objects`

Appears in the default template for a JMP app module script. This message controls the point at which the display and window for a module instance are created. The message appears in the script so that the script writer can choose to do certain setup before the objects are created. One example of this setup is for a parameterized application.

`inst<<Get Box`

Returns a handle to the top-most display box associated with a module instance. This might be useful to issue display or window commands, such as the `Save to PDF` or `Close Window` messages.

`inst<<Get Namespace`

Like `JMP App()`, each JMP application module instance also creates an anonymous namespace for all variables created in the module script. The namespace also includes all the variables that represent the display boxes in the module. This namespace contains a default symbol named `thisModuleInstance` that refers to itself.

`inst<<Get User Data`

`inst<<Set User Data`

Stores and retrieves a JSL value in the JMP application module instance. The value could be a number, quoted string, list, associative array, or other JSL type that is returned with the `Type()` function.

MATLAB Messages

The MATLAB interfaces are scriptable using a MATLAB connection object. Use the MATLAB Connect() JSL function to obtain a scriptable MATLAB connection object.

`mlconn<<Control(<Echo(`*`Boolean`*`)>, <Visible(`*`Boolean`*`)>)`

Controls the execution of MATLAB.

Returns

None.

Optional Global Arguments

`Echo(`*`Boolean`*`)` Echo MATLAB source lines to the JMP Log window.

`Visible(`*`Boolean`*`)` Determine whether to show or hide the active MATLAB workspace.

`mlconn<<Disconnect()`

Description

Disconnects this MATLAB integration interface connection.

`mlconn<<Execute({`*`list of inputs`*`}, {`*`list of outputs`*`}, `*`mCode`*`, <Expand(`*`Boolean`*`)>, <Echo(`*`Boolean`*`)>)`

Submits MATLAB code to the active global MATLAB integration interface connection given a list of inputs and upon completion a list of outputs are retrieved.

Returns

0 if successful, otherwise nonzero.

Required Arguments

`{`*`list of inputs`*`}` Positional, name list. List of JMP variable names to be sent to MATLAB as inputs.

`{`*`list of outputs`*`}` Positional, name list. List of JMP variable names to be retrieved from MATLAB as outputs.

`mCode` Positional, quoted string. The MATLAB code to submit.

Optional Named Arguments

`Expand(`*`Boolean`*`)` Performs an Eval Insert on the MATLAB code prior to submission.

`Echo(`*`Boolean`*`)` Echos MATLAB source lines to the JMP Log window. Default is true.

`mlconn<<Get Graphics(`*`format`*`)`

Gets the last graphic object written to the MATLAB graph display window. The graphic object can be returned in several graphic formats.

Returns

JMP Picture object.

Optional Argument

`format` Positional. The quoted format the MATLAB graph display window contents are to be converted to. Valid formats are `"png"`, `"bmp"`, `"jpeg"`, `"jpg"`, `"tiff"`, and `"tif"`.

`mlconn<<Get Version()`

Gets the current version of the installed MATLAB.

Returns

Matrix, returns a vector of length 3 containing the MATLAB version number.

`mlconn<<Get(`*name*`)`

Description

Gets a named variable from MATLAB to JMP.

Returns

Value of named variable.

Required Argument

name The name of a JMP variable to be retrieved from MATLAB.

`mlconn<<Is Connected()`

Description

Determines whether connection is active.

Returns

1 if connected, otherwise 0.

`mlconn<<JMP Name To MATLAB Name(`*jmp name*`)`

Description

Maps a JMP variable name to its corresponding MATLAB variable name using MATLAB variable name naming rules.

Returns

A quoted string, a mapped MATLAB name.

Required Argument

`jmp name` Positional. The name of a JMP variable to be sent to MATLAB.

`mlconn<<Send(`*name, <named arguments>*`)`

Description

Sends the named variable from JMP to MATLAB.

Returns

0 if successful, otherwise nonzero.

Required Argument

`name` Positional. The name of a JMP variable to be sent to MATLAB.

Named Arguments

The following arguments are for data tables only:

`Selected(`*`Boolean`*`)` Send selected rows from the referenced data table to MATLAB.

`Excluded(`*`Boolean`*`)` Send only excluded rows from the referenced data table to MATLAB.

`Labeled(`*`Boolean`*`)` Send only labeled rows from the referenced data table to MATLAB.

`Hidden(`*`Boolean`*`)` Send only hidden rows from the referenced data table to MATLAB.

`Colored(`*`Boolean`*`)` Send only colored rows from the referenced data table to MATLAB.

`Markered(`*`Boolean`*`)` Send only markered rows from the referenced data table to MATLAB.

`Row States (`*`Boolean`*`, <named arguments>)` Send row states from referenced data table to MATLAB by adding an additional data column named "RowState". Create multiple selections by adding together individual settings. The row state consists of individual settings with the following values:

- Selected = 1
- Excluded = 2
- Labeled = 4
- Hidden = 8
- Colored = 16
- Markered = 32

Row State Optional Named Arguments

The following optional, named Row States arguments are supported:

`Colors(`*`Boolean`*`)` Send row colors. Adds additional data column named "RowStateColor".

`Markers(`*`Boolean`*`)` Send row markers. Adds additional data column named "RowStateMarker".

`mlconn<<Submit(`*`mCode`*`, <named arguments>)`

Description

Submits MATLAB code to the active global MATLAB integration interface connection.

Returns

0 if successful, otherwise nonzero.

Required Argument

`mCode` Positional quoted string. The MATLAB code to submit.

Named Arguments

`Expand(`*`Boolean`*`)` Perform an Eval Insert on the MATLAB code prior to submission.

`Echo(`*`Boolean`*`)` Echo MATLAB source lines to the JMP log. The default is true.

mlconn<<Submit File(*path*)

Description

Submits statements to MATLAB using a quoted *path*.

Returns

0 if successful, otherwise nonzero.

Arguments

path Positional quoted string. The path to a file containing the MATLAB source lines to be executed.

Namespace Messages

ns<<Contains(*string*)

Returns 1 or 0, depending on whether the specified quoted string exists within the namespace.

ns<<Delete Namespace

Removes this namespace from the internal global list.

To delete variables in the namespace, use the Remove(*variable name*) message.

ns<<First

Returns a quoted string that contains the first variable name used within the namespace.

ns<<Get Contents

Returns a list of key-value pairs, which are each enclosed in a list. Each key is a quoted string that contains a variable name, and each value is the unevaluated expression that the variable contains.

ns<<Get Keys

Returns a list of variable names.

ns<<Get Name

Returns the name of this namespace.

`ns<<Get Value(variable name);`

Returns the unevaluated expression that the quoted *variable name* contains in this namespace.

`ns<<Get Values`

Returns a list of unevaluated expressions that each variable in the namespace contains.

`ns<<Get Values({variable name1, variable name2, ... });`

Returns a list of unevaluated expressions that each quoted variable in the namespace specified in the list argument contains. If a requested variable name is not found, an error is returned.

`ns<<Insert(variable name, expr);`

Inserts into this namespace a quoted variable named *variable name* that holds the expression *expr*.

`ns<<Lock Namespace(<variable name, ...>)`

Locks all specified variables in the namespace and prevents quotes variables from being added or removed. If no variables are specified, all variables in the namespace are locked.

`ns<<N Items`

Returns the number of variables contained in the namespace.

`ns<<New Namespace(name, <{list of expressions}>)`

Creates a namespace where all functions and variables created are defined only within the optional quoted *name* argument.

`ns<<Next(variable name)l`

Returns the name of the variable that follows the specified quoted variable.

`ns<<Remove(variable name, ...)`

Removes the specified quoted variable or list of variables.

`ns<<Show Contents`

Shows the contents of a namespace in the log.

`ns<<Unlock Namespace(variable name, ...);`

Unlocks the specified quoted variables in the namespace. If no variables are specified, all variables are unlocked.

Platform Messages

`obj<<Action`

Evaluates expressions. Useful for quoted stringing together multiple platforms interrupted by user input.

`obj<<Automatic Recalc`

Redoes the analysis automatically for exclude and data changes. If automatic recalc is on, you should use `Wait(0)` commands to let the triggers take effect and do the recalculation.

Note: Not supported on all platforms.

`obj<<Bring Window To Front`

Brings the current window to the front.

`obj<<Close Window`

Closes window identified by *obj*, typically a platform surface.

`obj<<Column Switcher(default column, {column1, column2, ...})`

Adds a control panel to a platform for switching variables.

`obj<<Copy ByGroup Script`

Creates a script to produce this analysis containing By variables and place it on the clipboard.

`obj<<Copy Script`

Creates a script to produce this analysis and place it on the clipboard.

`obj<<Data Table Window`

Makes the associated data table window active (front-most).

`obj<<Get Data Table`

Returns a reference to the data table.

`obj<<Get Script`

Returns script to reproduce the analysis as an expression in the log.

`obj<<Get Script With Data Table`

Creates a script to reproduce the analysis, specifically referencing the source data table, and returns it as an expression in the log.

`obj<<Get Timing`

Times the launch of the platform and returns it in the log.

`obj<<Get Web Support`

Returns the score for the display tree that is about to be saved as interactive HTML. Possible values are -1 (unsupported), 0 (supported), and 1 (supported). If the score does not equal -1, interactive HTML is supported and the `Save Interactive HTML` message can be used.

`obj<<Get Window Position`

Gets the position of the window. Returns an ordered pair.

`obj<<Get Window Size`

Gets the window size, in pixels. Returns an ordered pair.

`obj<<Ignore Platform Preferences(`*`Boolean`*`)`

Ignores the current settings of the platform's preferences. The message is ignored when sent to the platform after creation.

`obj<<Journal Window`

Appends the contents of the window to the journal.

`obj<<Local Data Filter`

Filters data to specific groups or ranges, but stays local to the platform.

`obj<<Maximize Window`

Maximizes the window. Equivalent to pushing the maximize button in the corner of the window. This message takes an optional Boolean argument:

```
// maximize the window:
obj<<Maximize Window(1)
// restore the window:
obj<<Maximize Window(0)
```

`obj<<Minimize Window`

Minimizes the window. Equivalent to pushing the minimize button in the corner of the window. This message takes an optional Boolean argument:

```
// minimize the window
obj<<Minimize Window( 1 )
// restore the window
obj<<Minimize Window( 0 )
```

`obj<<Move Window(x, y)`

Moves the window to the (*x, y*) location on your screen.

`obj<<Print Window`

Sends the selected window to the printer.

`obj<<Redo Analysis`

Reruns the analysis with the same options.

`obj<<Redo ByGroup Analysis`

Reruns the same analysis involving By groups.

`obj<<Relaunch Analysis`

Returns to the launch window for this analysis.

`obj<<Relaunch ByGroup`

Returns to the launch window for this analysis involving By groups.

`obj<<Remove Column Switcher`

Removes all Column Switchers that were added to the platform.

`obj<<Remove Local Data Filter`

Removes all Local Data Filters that were added to the platform.

`obj<<Report`

`Report(`*`obj`*`)`

Returns a display box reference for the report in the platform window.

`obj<<Report View`

Determines the level of detail visible in a platform report. Full shows all detail and Summary shows only select content, dependent upon the specific platform. For customized behavior, use the `Set Summary Behavior` message with display boxes.

`obj<<Save ByGroup Script to Data Table(<`*`name`*`>, <Append Suffix(`*`Boolean`*`)>, <Prompt(`*`Boolean`*`)>, <Replace(`*`Boolean`*`)>)`

Description

Creates a table script to produce the analysis involving By variables and saves it as a table script in the data table.

Optional Arguments

`name` The name of the script. *`name`* is quoted. If omitted, the platform names the script. For example, in Tabulate, the script is named "Tabulate". In Bivariate, the script might be named "Bivariate of height by weight" to reflect the platform and column names.

`Append Suffix(`*`Boolean`*`)` If true, appends a numerical suffix to the script name. This suffix differentiates the script from an existing script with the same name.

`Prompt(`*`Boolean`*`)` If true, prompts the user to specify a script name.

`Replace(`*`Boolean`*`)` If true, replaces an existing script with the same name.

`obj<<Save ByGroup Script to Journal`

Creates a table script to produce the analysis involving By variables and adds a button to the journal containing this script.

`obj<<Save ByGroup Script to Script Window`

Creates a script to produce the analysis involving By variables and appends it to the current Script window.

`obj<<Save Script for All Objects`

Saves script to reproduce all analyses found within the object's window in the Script Journal window.

`obj<<Save Script for All Objects to Data Table`

Saves a script for all report objects to the current data table. The script is named after the platform unless you specify the script name in quotes.

```
obj << Save Script for All Objects To Data Table("My Script")
```

`obj<<Save Script to Data Table`

Saves script to reproduce analysis as a property in the associated data table.

`obj<<Save Script to Journal`

Creates a script to produce the analysis and adds a button to the journal containing this script.

`obj<<Save Script to Report`

Saves script to reproduce analysis as a text box at the top of the report.

`obj<<Save Script to Script Window`

Saves a script to reproduce analysis in the Script Journal.

`obj<<Scroll Window(`*x*`, `*y*`)`

`obj<<Scroll Window({`*x*`, `*y*`})`

Scrolls the window *x* pixels to the right and *y* pixels down from the current position. Negative coordinates go left and up. If the coordinates are a list in braces { }, they are absolute coordinates. The window scrolls to the point *x* pixels from the left and *y* pixels from the top.

`obj<<SendToReport`

Used with the Dispatch function to customize the appearance of a report.

`obj<<SendToByGroup`

Sends messages to open platforms or turn on platform features to each level of a by-group.

`obj<<Show Window(`*Boolean*`)`

1 shows the window (brings it to the front). 0 hides the window. If the window is also minimized (on Windows) or docked (on macOS), showing the window restores it to the normal state and brings it to the front.

`obj<<Size Window(x, y)`

Resizes the window to *x* pixels wide by *y* pixels high.

`obj<<Title(new title)`

Sets the quoted title of the platform.

`obj<<Top Report`

Returns a reference to the top display box in the report. Useful for By groups or other cases when several platform reports are in one window.

`obj<<View Web XML`

Returns the XML used to create the interactive HTML report. The XML code appears in the log.

`obj<<Zoom Window`

Resizes the window to be large enough to show all of its contents.

Bubble Plot Messages

`bp<<Set Shape("Circle"|"Triangle"|"Square"|"Diamond"|"Arrow"|"Custom")`

Description

Sets the shape for the bubble. When specifying a custom shape, use the `Set Custom Path` message to specify the path for the bubbles.

See Also

"bp<<Set Custom Path(path matrix|path text)".

`bp<<Set Custom Path(path matrix|path text)`

Sets a path for custom bubbles.

Arguments

path matrix An *N*x3 matrix.

path text A quoted string that contains SVG code.

See Also

"bp<<Set Shape("Circle"|"Triangle"|"Square"|"Diamond"|"Arrow"|"Custom")".

DOE Messages

obj<<Get Prediction Variances

Description

Returns a vector of the MC variances used for the FDS plot.

Example

```
dt = Open( "$SAMPLE_DATA/Design Experiment/Bounce Data.jmp" );
d = DOE( Evaluate Design, X( :Silica, :Sulfur, :Silane ), Y( :Stretch ) );
d << Get Prediction Variances;
```

obj<<Set Number of FDS Points()

Description

Enables you to specify the number of runs used to generate an FDS plot.

Example

```
dt = Open( "$SAMPLE_DATA/Design Experiment/Bounce Data.jmp" );
d = DOE( Evaluate Design, X( :Silica, :Sulfur, :Silane ), Y( :Stretch ) );
d << Set Number of FDS Points( 20000 );
```

Partition Messages

obj<<Initial Splits(*condition*, *{left}*, *{right}*)

Description

Describes the splits that are performed.

Example

```
dt = Open( "$SAMPLE_DATA/Car Poll.jmp" );
obj = Partition(
   Y( :country ),
   X( :sex, :marital status, :age, :type, :size ),
   Method( "Decision Tree" ),
   Initial Splits( :size == {"Large"}, {}, {:size == {"Medium"}} )
);
```

Notes

The condition is for the left side and is either `[name compareoperator value]` or `[name == list of values]`. The left is an empty list if the right has splits. Omit the right side if there are no splits. The left and right continue recursively in this format.

Response Screening Messages

`obj<<Get PValues`

Returns a reference to a PValues table.

`obj<<Save PValues`

Stores the *p*-values in an output data table.

`obj<<Save Compare Means`

Stores the means comparisons in an output data table.

`obj<<Save Mean`

Stores the means in an output data table.

`obj<<Save Outlier Indicator`

Saves Outlier Indicator for each fit.

`obj<<Save Std Residuals`

Saves the residual formula for each fit.

`obj<<Select Columns`

Select columns in the original table corresponding to selected rows in this table.

Tabulate Messages

`obj<<Display Column Width(<Data Column(Column Table(n),<column name path>), Row Label(Row Table(n), <column name path>)>, <width>)`

Returns or sets the display pixel width of a column in a Tabulate table.

Required Argument

`Row Label` Use `Row Table` and heading for columns in the row labels area.

Optional Arguments

`Data Column` Use `Column Table` and column references to define columns in the main body of the table.

`column name path` Specifies the `Column Table` or `Row Table` (both quoted), and the series of column headings that traces the path of the column. **Note:** `Column Table` or `Row Table` can be omitted if the table referenced is the first table.

`width` Specifies the pixel width of the column.

Examples

```
dt = Open( "$SAMPLE_DATA/Car Poll.jmp" );
obj = dt << Tabulate(
   Add Table(
      Column Table(
         Grouping Columns( :sex, :marital status ),
         Analysis Columns( :age ),
         Statistics( Sum, "% of Total" )
      ),
      Row Table( Grouping Columns( :type ) ),
      Row Table( Grouping Columns( :country, :size ) )
   )
);
Wait( 3 ); // for demonstration purposes
obj << Display Column Width( Row Label( Row Table( 2 ), "country" ), 150 );
Wait( 3 ); // for demonstration purposes
obj << Display Column Width(
   Data Column(
      Column Table( 1 ),
      "sex",
      "Female",
      "marital status",
      "Married",
      "age",
      "Sum"
   ),
   150
);
```

Python Integration Messages

The Python interfaces are also scriptable using a Python connection object. A scriptable Python connection object can be obtained using the `Python Connect()` function. See "Python Connect(<Echo(Boolean),> <Path(path),> <Use Python Version(`string`),> <Python System Path(list)>)".

`pythconn<<Control(<Interactive(`*Boolean*`)>|<Echo(`*Boolean*`)>)`

Description

Controls the execution of Python.

Optional Named Arguments

`Interactive(`*`Boolean`*`)` Enables interactive mode in the Python matplotlib package. Determines whether the graphics window is released or closed when graphics rendering is complete.

`Echo(`*`Boolean`*`)` Global argument. Prints the Python source lines to the JMP log. The default value is true.

`pythconn<<Disconnect`

Disconnects the Python integration interface connection.

`pythconn<<Execute(`*`{list of inputs}, {list of outputs}, Python code, <named arguments>`*`)`

Description

Submits Python code to the active global Python integration interface connection given a list of inputs. On completion, returns a list of outputs.

Returns

Returns 0 if successful and 1 otherwise. The results are returned using the list of outputs. Given each element of the JMP output list, the corresponding Python variable value is returned.

Positional Arguments

`{list of inputs}` A list of JMP variable names to be sent to Python as inputs.

`{list of outputs}` A list of JMP variable names to be retrieved from Python as outputs.

`Python code` The quoted Python code to submit.

Optional Named Arguments

See "pythconn<<Submit(`Python code`, <Expand`(Boolean)`>, <Echo`(Boolean)`>)".

`pythconn<<Get(`*`name`*`)`

Description

Gets a named variable from Python.

Returns

Returns the value of the named variable.

Argument

`name` The name of the JMP variable to be received from Python. The argument can represent any of the following Python data types: numeric, quoted string, matrix, list, or data frame.

`pythconn<<Get Graphics(format)`

Description

Gets the last graphics object written to the Python graph display window in the specified graphics format. The graphics object can be returned in several different graphic formats.

Returns

Returns a JMP picture object.

Argument

`format` The format that the Python graph display window contents are to be converted to. Valid formats are PNG, BMP, JPEG, JPG, TIFF, and TIF.

`pythconn<<Get Version`

Description

Gets the current version of the Python installation.

Returns

Returns a list of length 5 that contains the five components of the version number: major, minor, micro, releaselevel, and serial. The releaselevel value is a quoted string.

`pythconn<<Is Connected`

Description

Determines if the connection is active.

Returns

Returns 1 if connected and 0 otherwise.

`pythconn<<JMP Name to Python Name(name)`

Description

Maps a JMP variable name to its corresponding Python variable name using Python variable name naming rules.

Argument

`name` The name of the JMP variable to be sent to Python. Some variable names allowed by JMP are not allowed by Python. When you send using these variables from JMP to Python (the `Send` message), their names get changed. Use `JMP Name to Python Name` to determine what the variable name was changed to.

`pythconn<<Send(name)`

Description

Sends a named variable from JMP to Python.

Returns

Returns 0 if successful and non-zero otherwise.

Argument

`name` The name of the JMP variable to be sent to Python.

`pythconn<<Submit(`*`Python code`*`, <Expand(`*`Boolean`*`)>, <Echo(`*`Boolean`*`)>)`

Description

Submits Python code to the active global Python integration interface connection.

Returns

Returns 0 if successful and 1 otherwise.

Required Arguments

`Python code` The quoted Python code to submit.

Optional Arguments

`Expand(`*`Boolean`*`)` Performs an `Eval Insert()` on the Python code before submission.

`Echo(`*`Boolean`*`)` Prints the Python source lines to the JMP log. The default value is true.

`pythconn<<Submit File(`*`path`*`)`

Description

Submits statements to Python using the quoted path name.

Argument

`path` The quoted path to the file that contains the Python source lines to be executed.

R Integration Messages

The R interfaces are also scriptable using an R connection object. A scriptable R connection object can be obtained using the `R Connect()` function.

`rconn<<Control(Interrupt|Async(`*`Boolean`*`)|Echo(`*`Boolean`*`))`

Changes the control options for R. If `Async` is set to true (`1`) for `R Submit()`, this message immediately stops the execution of the R code that was submitted.

`rconn<<Disconnect()`

Disconnects this R connection.

`rconn<<Is Connected()`

Returns 1if the R connection is active, 0 otherwise.

`rconn<<Send File(name, <named arguments>)`

Send the specified JMP variable to R.

Returns

0 if successful, nonzero otherwise.

Required Argument

name A quoted string contains the name of a JMP variable to send to R.

Optional Named Arguments for Data Tables

`Selected(Boolean)` If true, sends only the selected rows from the referenced data table to R.

`Excluded(Boolean)` If true, sends only the excluded rows from the referenced data table to R.

`Labeled(Boolean)` If true, sends only labeled rows from the referenced data table to R.

`Hidden(Boolean)` If true, sends only hidden rows from the referenced data table to R.

`Colored(Boolean)` If true, sends only colored rows from the referenced data table to R.

`Markered(Boolean)` If true, sends only markered rows from the referenced data table to R.

`Row States(Boolean, <named arguments>)` Includes a Boolean argument and optional named arguments. Sends row state information from the referenced data table to R by adding an additional data column named "RowState". The row state value consists of individual settings with the values shown in Table 3.2.

Table 3.2 Row States

Multiple row states are created by adding together individual settings.	Selected = 1 Excluded = 2 Labeled = 4 Hidden = 8 Colored = 16 Markered = 32
Arguments	`Colors(Boolean)` (Optional) If true, sends row colors and adds an additional data column named "RowStateColor". `Markers(Boolean)` (Optional) If true, sends row markers and adds an additional data column named "RowStateMarker".

`rconn<<Send(name, <R Name(name)>)`

Sends the quoted JMP data file to R. The *name* argument can represent any of the following data types: numeric, quoted string, matrix, list, or data table.

`rconn<<Get(name)`

Returns data from R. The *name* argument can represent any of the following data types: numeric, quoted string, matrix, list, or data table.

Returns

The value of the specified variable.

Arguments

name Specifies the quoted name of a JMP variable to be retrieved from R.

`rconn<<Get Graphics(type)`

Gets the last graphics object written to the R graph display window. The graphics object can be returned in different graphic formats.

Returns

A JMP picture object.

Required Argument

`type` The format the R graph display window contents are converted to. Valid formats are `"png"`, `"bmp"`, `"jpeg"`, `"jpg"`, `"tiff"`, and `"tif"`.

`rconn<<Submit(R code, Expand(Boolean), Echo(Boolean))`

Submits the quoted R code.

Returns

0 if successful, nonzero otherwise.

Required Argument

`code` Specifies the quoted R code to submit.

Optional Named Arguments

`Expand(Boolean)` Performs an `Eval Insert()` on the R code before submitting the code.

`Echo(Boolean)` Echoes the R source lines to the JMP log. The default value is true.

`Rconn<<Submit File(path)`

Submits statements to R using the file in the quoted *path*.

Arguments

`path` Specifies the quoted path to the file that contains R code to be executed.

`rconn<<Execute({list of inputs}, {list of outputs}, R code, <named arguments>)`

Submits the quoted R code to the R connection using the list of inputs. Upon completion, a list of outputs is returned.

Returns

0 if successful, nonzero otherwise.

Required Arguments

R code Specifies the quoted R code to submit.

`{list of inputs}` List of JMP variable names to be sent to R as inputs.

`{list of outputs}` List of JMP variable names to be retrieved from R as outputs.

Optional Named Arguments

See "rconn<<Submit(R code, Expand(Boolean), Echo(Boolean))".

`rconn<<Control(<Echo(Boolean)>)`

Controls the execution of R.

Returns

Void.

Optional Named Argument

`Echo(Boolean)` Echoes the R source lines to the JMP log.

`rconn<<Get Version()`

Gets the current version of R that is installed.

Returns

A vector of length 3 containing the R version number.

`rconn<<JMP Name To R Name(name)`

Maps a quoted JMP Name to its corresponding R Name using R variable name naming rules.

Returns

A quoted string that contains the quoted R name.

Arguments

name A quoted string that specifies the name of a JMP variable to be sent to R.

SAS Integration Messages

This section contains JSL messages that apply to SAS integration objects.

Metadata Server Object Messages

metaserver<<Disconnect()

Description

Disconnects the metadata server.

Returns

Void.

metaserver<<Get Display Name()

Description

Gets the display name of the metadata server.

Returns

A quoted string.

metaserver<<Get Host Name()

Description

Gets the host (machine) name of the metadata server.

Returns

A quoted string.

metaserver<<Get Port()

Description

Gets the port used for the metadata server connection.

Returns

An integer.

metaserver<<Get User Identity()

Description

Gets the identify of the connected user as defined in metadata.

Returns

A quoted string.

`metaserver<<Get User Name()`

Description

Gets the user name (login ID) that was used for the metadata server connection.

Returns

A quoted string.

SAS Server Object Messages

`sasconn<<Assign Libref(`*`libref`*`, `*`path`*`, `*`engine`*`, `*`engine options`*`)`

Description

Assign a SAS libref on this SAS server connection.

Returns

Void.

Arguments

See "SAS Assign Lib Refs("libref", "path", <"engine">, <"engine options">)".

`sasconn<<Cancel Submit()`

Description

Cancels the currently running SAS Submit for this server that is presumably running asynchronously.

Returns

1 if a running submit was found and canceled; 0 otherwise.

`sasconn<<Clear Log History()`

Description

Clears the SAS Log history for this server.

Returns

Void.

`sasconn<<Clear Output History()`

Clears the SAS Output history for this server.

`sasconn<<Connect(<User Name(name)>, <Password(password)>, <Prompt("Always"|"Never"|"IfNeeded")>)`

Description

Attempt to reconnect a SAS server connection object that has become disconnected.

Returns

1 if the connection was successful, 0 otherwise.

Optional Named Arguments

`User Name(name)` Specifies the quoted user name for the connection.

`Password(password)` Specifies the quoted password for the connection.

`Prompt` A quoted keyword. `"Always"` means always prompt before attempt to connect. `"Never"` means never prompt even if the connection attempt fails (just fail with an error message going to the log), and `"IfNeeded"` (the default) means prompt if the attempt to connect with the given arguments fails (or is not possible with the information given).

`sasconn<<Deassign Libref(libref)`

Description

De-assign the quoted SAS libref on this SAS server connection.

Returns

Void.

Arguments

`libref` Specifies the quoted library reference.

`sasconn<<Disconnect()`

Description

Disconnect this SAS server connection.

Returns

Void.

`sasconn<Does Module Exist(module name)`

Description

Determines whether the specified SAS module exists in the SAS installation represented by the SAS connection. This can be helpful in determining whether certain SAS products are installed. The SAS DATA Step function MODEXIST is used to determine module existence.

Returns

1 if the specified module is found to exist, 0 if it does not exist.

Argument

`module name` Specifies the quoted SAS module, the existence of which should be checked. Do not include any extension.

`sasconn<<Export Data(`*`dt`*`, `*`library`*`, `*`dataset`*`, `*`<named arguments>`*`)`

Description

Exports a JMP data table to the specified SAS data set in the specified library on the active SAS server connection.

Returns

`1` if the data table was exported successfully; `0` otherwise.

Optional Named Arguments

See "SAS Export Data(dt, "library", "dataset", <named_arguments>)".

`sasconn<<Get Data Sets(`*`libref`*`)`

Description

Returns a list of the data sets defined in a SAS library on this SAS server connection.

Returns

A list of quoted strings.

Arguments

`libref` Specifies the quoted SAS libref or friendly library name associated with the library for which the list of defined SAS data sets will be returned.

`sasconn<<Get Error Count()`

Description

Gets the count of the number of errors encountered in the previous SAS Submit.

Returns

An integer.

`sasconn<<Get File(`*`source`*`, `*`dest`*`)`

Description

Downloads a file from this SAS server connection.

Returns

Void.

Arguments

See "SAS Get File("source", "dest", "encoding")".

sasconn<<Get File Names(*fileref*)

Description

Gets a list of filenames found in the quoted *fileref* on this SAS server connection.

Returns

A list of quoted strings.

Arguments

fileref A quoted string that contains the name of *fileref* from which to retrieve filenames.

sasconn<<Get File Names In Path(*path*)

Description

Gets a list of filenames found in the quoted *path* on the current SAS server connection.

Returns

A list of quoted strings.

Arguments

path The quoted directory path on the server from which to retrieve filenames.

sasconn<<Get File Refs()

Description

Gets a list of the currently defined SAS filerefs on this SAS server connection.

Returns

A list of quoted strings.

sasconn<<Get Librefs(*<named arguments>*)

Description

Gets a list of the currently defined SAS librefs on this SAS server connection.

Returns

A list of quoted strings.

Optional Named Arguments

See "SAS Get Lib Refs(<named arguments>)".

sasconn<<Get Log()

Description

Retrieves the SAS Log from the last SAS Submit from this SAS server connection.

Returns

A quoted string.

`sasconn<<Get Option Name()`

Description

Queries SAS for the value of a SAS option variable.

Returns

A quoted string.

Example

The following script iterates through the define variables and prints out the values:

```
option_names = sasconn << Get Option Names();
For(i=1, i <= N Items(option_names), i++,
  option_value = sasconn << Get Option Value (option_names[i]);
  output = option_names[i] || "=" || char(option_value) || "/!n";
  Write(output);
);
```

`sasconn<<Get Output()`

Description

Retrieves the listing output from the last submission of SAS code to this SASServer object.

Returns

A quoted string.

`sasconn<<Get Results()`

Description

Retrieves the results of the previous SAS Submit as a scriptable object, which allows significant flexibility in what to do with the results.

Returns

A SAS Results Scriptable Object.

`sasconn<<Get Submit Status()`

Description

Gets the current status of a SAS Submit for this server that is presumably running asynchronously.

Returns

1 if the submit has not started; 2 if the submit is running; 3 if the submit has been canceled; 10 if the submit has completed successfully; 11 if the submit has completed with errors.

`sasconn<<Get Var Info(libref, dataset, <Password(password)>)`

Description

Returns information about the variables the specified SAS data set.

Required Arguments

libref The library name.

dataset The quoted name of the data set from which to retrieve variable names.

Optional Argument

Password(*password*) The quoted *password* for the connection.

`sasconn<<Get Var Names(libref, dataset, <named arguments>)`

Description

Retrieves the variable names contained in the specified data set on this SAS server connection.

Returns

A list of quoted strings.

Arguments

See "SAS Get Var Names(string, <"dataset">, <password("password")>)".

`sasconn<<Get Version(<"Long">)`

Description

Returns the SAS version as a quoted string such as "9.4".

Returns

A quoted string that contains the SAS version.

Optional Argument

Long A quoted keyword that specifies to return the long SAS version, which corresponds to the SYSVLONG SAS macro (for example, "9.02.02M0P01152009").

`sasconn<<Get Work Folder()`

Description

Returns the full path of the folder corresponding to the WORK library for this server.

Returns

A quoted string that specifies the work folder path.

`sasconn<<Import Data(library, dataset, <named arguments>)`

Description

Imports a SAS data set from this SAS server connection into a JMP table.

Returns

A JMP Data Table object.

Arguments

See "SAS Import Data(`string`, <"dataset">, <named arguments>)".

`sasconn<<Import Query(`*`sqlquery`*`, `*`<named arguments>`*`)`

Description

Executes the requested SQL query on this SAS server connection, importing the results into a JMP data table.

Returns

A JMP data table object.

Arguments

See "SAS Import Query("sqlquery", <named arguments>)".

`sasconn<<Is Connected()`

Description

Determines whether this SAS Server object is currently connected to SAS.

Returns

`1` if *`sasconn`* is connect, `0` otherwise.

`sasconn<<Is Product Available(`*`product name`*`)`

Description

Determines whether the quoted SAS product is both licensed and installed in the session represented by the SAS connection. The SAS DATA Step functions SYSPROD and MODEXIST are used to determine the licensed and installed status of the product.

Returns

1 if the specified product is licensed, 0 if the product is not licensed, or -1 if the specified product is not recognized by SAS. This function throws an exception if the requested product is not one for which JMP knows how to check the installed status.

Required Argument

`product name` The quoted SAS product for which licensing should be checked. The product name can be specified with or without the "SAS/" prefix.

sasconn<<Is Product Licensed(*product name*)

Description

Determines whether the quoted SAS product is licensed in the session represented by the SAS connection. The SAS DATA Step function SYSPROD is used to determine the licensing status of the product.

Returns

1 if the specified product is licensed, 0 if the product is not licensed, or -1 if the specified product is not recognized by SAS.

Required Argument

`product name` The quoted SAS product for which licensing should be checked. The product name can be specified with or without the "SAS/" prefix.

sasconn<<Kill Session(*<n>*)

Description

If no argument is provided, the SAS connection is immediately terminated.

Returns

Void.

Arguments

n An optional number. The system waits n seconds for a normal shut-down before immediately terminating the SAS connection.

sasconn<<Load Text File(*path*, *<named arguments>*)

Description

Downloads the file specified in the quoted *path* from the active SAS server connection and retrieve its contents as a quoted string.

Returns

A quoted string.

Arguments

See "SAS Load Text File("path")".

sasconn<<Open Log Window()

Description

Opens (or brings to the front) the SAS Log window for this server.

Returns

Void.

sasconn<<Open Output Window()

Description

Opens (or brings to the front) the SAS Output window for this server.

Returns

Void.

sasconn<<Open SAS Results()

Description

Opens the results from the previous SAS Submit. Intended to be used with asynchronous SAS submits or the use of the OnSubmitComplete option to SAS Submit to give the JSL author a way to conditionally open the results of a submit.

Returns

Void.

sasconn<<Open Submit Results()

Description

Opens all the results from the last SAS Submit command.

Returns

Void.

sasconn<<Send File(*source*, *dest*)

Description

Uploads a file to this SAS server connection.

Returns

Void.

Arguments

See "SAS Send File("source", "dest", "encoding")".

sasconn<<Submit(*sas code*, *<named arguments>*)

Description

Submits quoted SAS code to this SAS server connection.

Returns

Void.

Arguments

See "SAS Submit("sasCode", <named arguments>)".

`sasconn<<Submit File(`*`filename`*`, `*`<named arguments>`*`)`

Description

Submits a SAS code file to this SAS server connection.

Returns

Void.

Arguments

See "SAS Submit File("filename", <named arguments>)".

Stored Processes Messages

`stp<<Begin Run(`*`<named arguments>`*`)`

Description

Starts this stored process executing in the background. This message is paired with `End Run`, which should also be called at some point after `Begin Run` to wait for the stored process to complete.

Returns

`-1` = execution failed.

`1` = not started.

`2` = running.

`3` = canceled.

`10` = completed successfully.

`11` = completed with errors.

Optional Named Arguments

Same as `Run`, except `AutoOpenResults` and `NoAlerts` are not supported. They are available on `EndRun`.

`AutoResume(`*`<filename>`*`)` If specified with no argument, it specifies that the stored process results should be auto-opened when the stored process completes. If a quoted filename is specified, filename is opened rather than all results of the stored process being auto-opened.

`AutoResumeScript(`*`script`*`)` Specifies that after stored process execution completes, the quoted script should be evaluated. If the script is a function taking at least one argument, the function is evaluated with the scriptable stored process object passed as the first (and only) argument. `AutoResume` and `AutoResumeScript` are mutually exclusive.

stp<<Delete Results(*<named arguments>*)

Description

Deletes all results from the execution of this stored process.

Returns

1 if deletion is successful, 0 otherwise (error message to JMP log).

Optional Named Arguments

NoAlerts(*Boolean*) If True, the user is not prompted for confirmation before the attempt is made to delete results.

DeleteDirectory(*Boolean*) If true, deletes the directory containing the stored process results along with the result files themselves. The default value is true.

stp<<Edit Param Values()

Description

Opens the stored process window for interactively setting parameter values.

Returns

1 if the user clicks OK to dismiss the window, 0 if the user clicks Cancel.

stp<<End Run(*<named arguments>*)

Description

Waits a specified amount of time (or forever) for a stored process started with Begin Run to complete. If the stored process is complete, retrieves the results, and opens them.

Returns

-1 = execution failed.

1 = not started.

2 = running.

3 = canceled.

10 = completed successfully.

11 = completed with errors.

Optional Named Arguments

AutoOpenResults(*Boolean*) Optional, Boolean. If True, results are automatically opened if the stored process completes in the time specified by *MaxWait*. If False, results are not automatically opened, and can be manually opened via the object returned by the Get Results message. Default is True.

MaxWait(*milliseconds*) An integer that specifies the maximum amount of time in milliseconds to wait for the stored process to complete. If MaxWait is not specified, End Run waits forever for the stored process to complete.

NoAlerts(*Boolean*) If True, error messages are sent to the JMP log rather than message boxes. The default value is False.

---

stp<<Get Metadata Id()

Description

Returns the metadata ID of the stored process.

Returns

A quoted string.

---

stp<<Get Metadata Path()

Description

Returns the full metadata path of the stored process.

Returns

A quoted string.

Stp<<Get Name()

Description

Returns the name of the stored process.

Returns

A quoted string.

---

stp<<Get Param Enum Labels(*name*)

Description

Gets the enumeration labels specified by the quoted *name* for a parameter.

Returns

A list of quoted strings.

Arguments

name Specifies the quoted name of the parameter whose enumeration labels to retrieve.

---

stp<<Get Param Enum Values(*name*)

Description

Gets the possible enumerated values for a parameter.

Returns

A list of quoted strings.

Arguments

`name` Specifies the quoted name of the parameter whose possible enumerated values to retrieve.

`stp<<Get Param Names(<named arguments>)`

Description

Gets a list of parameter names for this stored process of specific types.

Returns

A list of quoted strings.

Optional Named Arguments

`Visible(Boolean)` If true, gets only visible parameters. If `False`, gets only non-visible parameters. If not specified, gets both visible and non-visible parameters.

`Modifiable(Boolean)` If true, gets only modifiable parameters. If `False`, gets only non-modifiable parameters. If not specified, gets both modifiable and non-modifiable parameters.

`Required(Boolean)` If true, gets only required parameters. If `False`, gets only non-required parameters. If not specified, gets both required and non-required parameters.

`Expert(Boolean)` If true, gets only expert parameters. If `False`, gets only non-expert parameters. If not specified, gets both expert and non-expert parameters.

`stp<<Get Param Value(name)`

Description

Gets the current value of the specified parameter.

Returns

A quoted string.

Required Argument

name Specifies the name of the parameter whose value to retrieve.

`stp<<Get Results()`

Description

Gets the results generated by the execution of this stored process as a scriptable object.

Returns

A SAS Results scriptable object.

`stp<<Get Status()`

Description

Gets the execution status of the stored process.

Returns

-1 = execution failed.

1 = not started.

2 = running.

3 = canceled.

10 = completed successfully.

11 = completed with errors.

`stp<<Get Status Message()`

Description

Gets the message associated with the failure of the stored process, if any.

Returns

A quoted string.

`stp<<Reset Param Values()`

Description

Resets all parameter values to their metadata-defined default values.

Returns

Void.

`stp<<Run(<named arguments>)`

Description

Executes this stored process object in the foreground.

Returns

-1 = execution failed.

1 = not started.

2 = running.

3 = canceled.

10 = completed successfully.

11 = completed with errors.

Optional Named Arguments

`AutoOpenResults(`*`Boolean`*`)` If `True`, results are automatically opened when the stored process completes. If `False`, results are not auto-opened, and can be manually opened via the object returned by the `GetResults` message. The default value is `True`.

`UserName(`*`username`*`)` Specifies the quoted *`user name`* under which to run the stored process.

`Password(`*`password`*`)` Specifies the quoted *`password`* for *`UserName`*.

`AuthDomain(`*`authDomain`*`)` Specifies the quoted authentication domain of the credentials (*`username`*, *`password`*) given.

`ODSDest(`*`dest`*`)` Specifies the quoted ODS destination (`"HTML"`, `"PDF"`, `"tagsets.SASReport12"`) for any ODS-generated results from the stored process. This requires the stored process SAS code to call `%STPBEGIN`. The default value is `"HTML"`.

`GraphicsDevice(`*`device`*`)` Specifies the quoted SAS graphics device to use when generating graphics in ODS results. This requires the stored process SAS code to call `%STPBEGIN`. The default value is `"GIF"`.

`ODSStyle(`*`style name`*`)` Specifies the quoted ODS style to apply to the results. This requires the stored process SAS code to call `%STPBEGIN`. There is no default value.

`ODSStyleSheet(`*`path`*`)` Specifies the quoted *`path`* to a CSS file on the client machine that is to be applied to generated ODS results. This requires the stored process SAS code to call `%STPBEGIN`. There is no default value.

`NoAlerts(`*`Boolean`*`)` If `True`, error messages are sent to the JMP log rather than message boxes. The default value is `False`.

`stp<<Set Param Value(`*`name`*`, `*`value`*`)`

Description

Sets the value of the specified stored process parameter to the specified value.

Returns

`1` if successful, `0` otherwise (value can violate the parameter's constraints).

Arguments

`name` Specifies the quoted name of the parameter whose value to set.

`value` Specifies the quoted string that you want to set the parameter to.

`stp<<Set Results Directory(`*`path`*`)`

Description

Sets the quoted *`path`* on the client machine where stored process results are placed.

Returns

A quoted string.

Arguments

`path` Specifies the full quoted path of the directory where results of the stored process execution should be placed. The directory must exist or be creatable. If the results directory is not set, a temporary location appropriate for the operating system will be used, and that directory can be retrieved from the stored process Results scriptable object after the stored process executes.

SAS Results Messages

`results<<Delete All Result Files()`

Description

Deletes all files created by the SAS Submit or Stored Process execution. Note that any result files that are still in use are not deleted.

Returns

1 if the deletion was successful; 0 if some of the files could not be deleted.

`results<<Get Directory()`

Description

Gets the directory where the results generated by the stored process or SAS submit are located.

Returns

A quoted string.

`results<<Get Log()`

Description

Get the SAS Log from the execution of the stored process or SAS submit.

Returns

A quoted string.

`results<<Get Main Result File Name(<Fullpath(Boolean)>)`

Description

Gets the full path of the main result file generated by the stored process or SAS submit.

Returns

A quoted string.

Optional Named Argument

`Fullpath(Boolean)` If true, the main result filename is returned as a full path. The default value is false.

`results<<Get Output()`

Description

Gets the SAS Listing output from the execution of the stored process or SAS submit.

Returns

A quoted string.

`results<<Get Output Datasets()`

Description

Get a list of output data set generated by the SAS Submit that created this SAS Results object.

Returns

A list of data set names in the form "libname.membername".

`results<<Get Result File Info(<Mimetype(`*`mime-type`*`)>, <Fullpath(`*`Boolean`*`)>)`

Description

Get information about result files that were generated by the execution of the stored process or SAS submit.

Returns

List of two lists of quoted strings. The first list is filenames, and the second list is the MIME-type of the corresponding file from the first list.

Optional Arguments

`Mimetype(`*`mime_type`*`)` Restricts the set of files for which information is returned to only those files with the specified quoted MIME-type. If not specified, information about all generated files is returned.

`Fullpath(Boolean)` If true, the filename returned for each result file is returned as a full path; if false, only the name of the file is returned. The default value is false.

`results<<Make JMP Report()`

Description

Parses the ODS XML results and creates a JMP report.

Returns

The display box for the report.

`results<<Open All Results()`

Description

Opens all results generated by the execution of the stored process or SAS submit.

Returns

Void.

`results<<Open Result File(`*`filename`*`, <Run Script(`*`Boolean`*`)>)`

Description

Attempts to open the result file with the given name.

Returns

JMP Data Table if one was opened.

Required Argument

`filename` Specifies the quoted name of the file from the generated results to open. filename should just be the name of the file, not the full path. If *`filename`* is a filename with no extension, both JMP data tables and JSL scripts in the results are searched for a match, and if both exist, both are opened.

Optional Argument

`Run Script(`*`Boolean`*`)` If true, and if *`filename`* is a JSL script, the script is executed. If false, *`filename`* is just opened, even if it is a JSL script.

`results<<Run Script(`*`filename`*`)`

Description

Looks for the JSL file in the results with the given *`filename`* and runs it if it finds it.

Returns

Void.

Argument

`"filename"` Specifies the quoted name of the JSL file from the generated results to open. The *`filename`* argument should just be the name of the file, not the full path, and it does not need to include the .jsl extension.

Schedule Messages

This section contains JSL messages that apply to schedules.

Related Information

- See also "Schedule(n, script)".

sch<<Clear Schedule()

Cancels all scheduled events.

sch<<Close()

Closes the scheduler.

sch<<Restart()

Restarts the scheduler after it was stopped from running all scheduled events.

sch<<Show Schedule()

Shows a list of all scheduled events.

sch<<Stop()

Stops the scheduler from running all scheduled events.

Segment Message

Pie Seg(*<style>*, {*x*, *y*}, *radius*, [*values*])

Description

Creates a pie seg at the specified origin, with the specified radius, based on given values.

Required Arguments

{*x*, *y*} Specifies the x and y coordinates at which the pie seg is displayed.

radius Specifies the radius.

values Specifies the values specified in matrix format.

Optional Argument

style A quoted string that specifies the style: "Pie" (traditional pie chart with each slice sized by the Summary Statistic), "Ring" (each variable or level of a stratifying variable is represented by a concentric ring), or "Coxcomb" (the central angles for all slices are equal).

Sockets Messages

---

skt<<Accept(*<callback, timeout*>)

Description

Tells the server socket to accept a connection and return a new connected socket.

Returns

A list of up to four items. The first is a quoted string that echoes the command ("accept"). The second is a quoted string, either "ok" or an error. The third is a quoted string that specifies the name of the machine that just connected. The fourth is a reference to the socket that you can send more messages.

Optional Arguments

callback Specifies the name of a function to receive the data.

timeout If you use a *callback*, *timeout* specifies how long the function should wait for an answer. For a server socket, 0 is an acceptable value because a server should not shut down because no one has connected to it recently.

---

skt<<bind(*localhost*, *port*)

Description

Associates a port on the local machine with the socket.

Returns

A list of two quoted strings. The first string is the command name (“bind”) and the second is “ok” if successful or an error.

Required Arguments

localhost Specifies the quoted local machine. You cannot bind to another machine.

port Specifies the port that should be used.

---

skt<<Close()

Description

Closes a socket.

Returns

A list of two quoted strings. The first string is the command name (“close”) and the second is “ok” if successful.

---

skt<<Connect(*socketname*, *port*)

Description

Connects to a listening socket.

Returns

A list of two quoted strings. The first string is the command name (“connect”) and the second is “ok” for a successful connection or an error sent back by the other socket.

Arguments

socketname Specifies the name of the other socket. If you are connecting to a web server, this is the web address (the name is preferred to the IP address).

port Specifies the port of the other socket to connect through.

`skt<<GetPeerName()`

Description

Retrieves the address and port of the socket at the other end of the connection.

Returns

A list of four quoted strings. The first echoes the command (`"getpeername"`). The second is either “ok” or an error. The third and fourth are the address and the port.

`skt<<Get Sock Name()`

Description

Retrieves the address and port of the socket at this end of the connection.

Returns

A list of four quoted strings. The first echoes the command (`"getsockname"`). The second is either “ok” or an error. The third and fourth are the address and the port.

`skt<<ioctl(FIONBIO, `*`Boolean`*`)`

Description

Controls the socket’s blocking behavior.

Returns

A list of two quoted strings. The first string is the command name (“ioctl”) and the second is “ok” if successful or an error.

Arguments

`FIONBIO, 1` `FIONBIO` means Non-Blocking I/O. If true, turns on the behavior and the argument.

`skt<<Listen()`

Description

Tells the server socket to listen for connections.

Returns

A list of two quoted strings. The first echoes the command (`"listen"`) and the second is `"ok"` or an error message.

`skt<<recv(`*n*`, <`*callback*`, `*timeout*`>)`

`skt<<recvfrom(`*n*`, <`*callback*`, `*timeout*`>)`

Description

Receives either a stream message (`recv`) or a datagram message (`recvfrom`) from the other socket. If the two optional arguments are used, the data is not received immediately. Instead, the data is received when the function *callback* is called.

Returns

A list of three quoted strings. The first string is the command name (“recv” or “recvto”). The second is “ok” if successful or an error message if not. The third string is the data that was received. If a callback function is used, a fourth element is the socket that was used in the original `recv` or `recvfrom` message.

Required Argument

n Specifies the number of bytes to receive from the other socket.

Optional Arguments

callback Specifies the name of a function to receive the data.

timeout If you use a *callback*, *timeout* specifies how long the function should wait for an answer.

`skt<<Send(`*stream*`)`

`skt<<SendTo(`*dgram*`)`

Description

Sends the data in the argument to the other socket. `Send` sends a stream and `sendto` sends a datagram.

Returns

A list of three quoted strings. The first string is the command name (“send” or “sendto”). The second is “ok” if successful or an error message if not. The third string is any portion of the stream that could not be sent, or empty if all the data was sent correctly.

Arguments

stream Specifies the command to send to the other socket.

dgram Specifies the command to send to the other socket.

Notes

Either argument might need to contain binary data. JMP represents non-printable ASCII characters with a tilde (~) followed by the hexadecimal number. For example,

```
skt<<send(("GET / HTTP/1.0~0d~0a~0d~0a");
```

sends a “get request” to an HTTP server.

SQL Messages

`obj<<Custom SQL(`*`sql`*`)`

Description

Changes the query to a custom SQL query and sets the SQL.

Required Argument

`sql` The quoted SQL query.

`obj<<Generate SQL`

Returns the SQL that the query generates when you run it.

`obj<<Modify`

Opens the query in Query Builder.

`obj<<PostQueryScript(`*`script as text`*`)`

Sets a JSL script that runs after the query finishes executing. *`script as text`* is quoted JSL code.

`obj<<Query Name(<`*`new name`*`>)`

Gets (without the *`new name`* argument) or sets (with the *`new name`* argument) the name of the query. The name of the query is used as the name of the data table that results from running the query.

`obj<<Run(<"Private"|"Invisible">, <Update Table(`*`table`*`)>, <OnRunComplete(`*`script`*`)>, <OnRunCanceled(`*`script`*`)>, <OnError(`*`script`*`)>)`

Description

Runs the SQL query in the background or foreground depending on the Query Builder preference “Run queries in the background when possible”.

Returns

Null (if the query runs in the background) or a data table (if the query runs in the foreground).

Optional Named Arguments

`"Private"` A quoted keyword that opens the data table that the query produces without displaying it in a data table window. `"Private"` is available only if `OnRunComplete` is included in the script.

`"Invisible"` A quoted keyword that hides the data table that the query produces. Use this argument to keep the query result hidden but use it in a subsequent query. The data table is displayed in the Home Window's Window List and the Window > Unhide list.

`Update Table` Updates the specified data table. Runs the query in the foreground.

`OnRunComplete` Specifies a script to run after the query is complete. To get the resulting data table, include `OnRunComplete`. The `OnRunComplete` script needs to be defined in the global namespace, as indicated by the double colons in this example:

```
Names Default To Here( 1 );
::onComplete = Function( {dt},
  {default local},
  Write(
     "\!NQuery is complete!  Result name: \!"",
     dt << Get Name,
     "\!", Number of rows: ",
     N Rows( dt )
  )
);

query = Include( "rentals_fam_romcom.jmpquery" );
query << Run Background( On Run Complete( ::onComplete ) );
```

`OnRunCanceled` Specifies a script to run after the user cancels the query.

`OnError` Specifies a script to run if an error occurs.

Notes

If you want the data table that results from the background query, use the `OnRunComplete` optional argument. You can include a script that runs when the query completes and then assigns a data table reference to the resulting data table. Or you might pass the name of a function that accepts a data table as its first argument. That function is called when the query completes.

Examples

The following example opens a query that you previously saved from Query Builder. The query opens privately, that is, without opening Query Builder. The query runs, and the resulting data table opens.

```
query = Open( "C:\My Data\Movies.jmpquery", "Private");
dt = query << Run();
```

You can include a .jmpquery file in a script and run the query in the background using the `<<Run Background` message.

```
query = Include( "C:\Queries\movies.jmpquery");
```

```
query <<Run Background();
```

The following example queries the database, opens the resulting data table, and prints the number of data table rows to the log.

```
confirmation = Function( {dtResult},
  Write( "\!NNumber of rows in query result: ", N Rows( dtResult ) )
);
query = New SQL Query(
  Connection(
     "ODBC:DSN=SQL
  Databases;APP=MYAPP;TrustedConnection=yes;WSID=D79255;DATABASE=SQB;"
  ),
  QueryName( "movies_to_update" ),
  Select( Column( "YearMade", "t1" ), Column( "Rating", "t1" ) ),
  From( Table( "g6_Movies", Schema( "SQB" ), Alias( "t1" ) ) ),

);
query << Run( OnRunComplete( confirmation ) );
```

`Run Background(<OnRunComplete(`*script*`), <"Private"|"Invisible">>, <OnRunCanceled(`*script*`)>, <OnError(`*script*`)>)`

Description

Runs the SQL query in the background. The running query is not displayed.

Returns

Null (or the data table object, if `OnRunComplete` is included).

Optional Named Arguments

`OnRunComplete(`*script*`)` Specifies a script to run after the query is complete. To get the resulting data table, include `OnRunComplete`. The `OnRunComplete` script needs to be defined in the global namespace, as indicated by the double colons in this example:

```
Names Default To Here( 1 );
::onComplete = Function( {dt},
  {default local},
  Write(
     "\!NQuery is complete!  Result name: \!"",
     dt << Get Name,
     "\!", Number of rows: ",
     N Rows( dt )
  )
);

query = Include( "rentals_fam_romcom.jmpquery" );
query << Run Background( On Run Complete( ::onComplete ) );
```

`"Private"` Does not open the resulting data table. Specify only with `OnRunComplete`. If you include `private` in a background query, JMP opens the data table as invisible instead.

`"Invisible"` Hides the data table. Use this argument to keep the query result hidden but use it in a subsequent query. The data table is displayed in the Home Window's Window List and the Window > Unhide list.

`OnRunCanceled` Specifies a script to run after the user cancels the query.

`OnError` Specifies a script to run if an error occurs.

Notes

All queries except for SAS queries run in the background based on the Query Builder preference "Run the queries in the background when possible", which is selected by default. For SAS queries, `Run Background()` is ignored.

You can include a .jmpquery file in a script and run the query in the background using the `Run Background` message.

```
query = Include( "C:\Queries\movies.jmpquery");
query <<Run Background();
```

`Run Foreground(<OnRunComplete(`*`script`*`), <"Private"|"Invisible">>, <OnRunCanceled(`*`script`*`)>, <OnError(`*`script`*`)>)`

Description

Runs the SQL query in the foreground.

Returns

A data table that opens when the query is finished.

See Also

"Run Background(<OnRunComplete(`script`), <"Private"|"Invisible">>, <OnRunCanceled(`script`)>, <OnError(`script`)>)"

`obj<<Save`

Saves the query to its associated file. The save fails if the query does not yet have an associated file.

`obj<<Save As(`*`path`*`, <Replace Existing(`*`Boolean`*`))`

Saves the query to the specified file. If the file already exists, the save fails unless `Replace Existing` is true.

Messages for Other Objects

This section contains JSL messages that apply to other objects, such as zip archives and journals.

Zip Archives

This section contains JSL messages that apply to zip archives.

`list = za<<Dir`

Returns a list of member names

`data = za<<Read(`*`member name`*`, <Format("`*`blob`*`")>)`

Returns a quoted string that contains the entire quoted *member name*. A zip file consists of filenames, also called "member names".

Notes

For remote files, JMP copies the URL data to the local disk. When the zip archive is no longer accessible, the local data file is deleted.

`actualname = za<<Write(`*`member name`*`, `*`member data`*`, <"`*`replace`*`">)`

Writes a quoted string or quoted blob to a zip archive member file. If the quoted *member name* isn't in the current zip file, the returned `actualname` is the same as *member name*. This member name will be changed to prevent overwriting an existing member; the name actually used is returned. The quoted *member data* argument is the data to write into the zip file's member of that name. *replace* creates the file with a temporary name, deletes the old file, and renames the temporary file to the existing name.

Journals

This section contains JSL messages that apply to JMP journals.

`jnl<<Save HTML(<`*`path`*`>, <`*`format`*`>)`

Saves the journal as HTML.

Optional Arguments

`"path"` Specifies the quoted path for the saved HTML file (for example, `"C:\myFile.html"`).

`format` Specifies the quoted graphic file format. JPG, PNG, and TIFF formats are supported. The graphics are saved in a subdirectory named `gfx`.

`jnl<<Save RTF(<path>, <format>)`

Saves the journal as an RTF file.

Optional Arguments

`"path"` Specifies the quoted path for the saved RTF file (for example, `"C:\myFile.rtf"`).

`"format"` Specifies the quoted file format for the embedded graphics. `"JPG"`, `"PNG"`, and `"EMF"` formats are supported on Windows. All journals are saved as PDF files by default on macOS.

Notes

If no *path* or *format* are provided, you are prompted to name the file and specify the format on Windows. On macOS, you are prompted to name the file. The file is saved as a PDF file by default.

`jnl<<Save PDF(<path>, <Show Page Setup(Boolean)>, <Portrait(Boolean)>)`

Saves the journal as a PDF file.

Optional Arguments

`"path"` The quoted *path* for the saved PDF file (for example, `"C:\myFile.pdf"`).

`Show Page Setup` If set to true, opens the Page Setup window to let the user change the margin, magnification level, and other page layout options.

`Portrait` Determines whether the page orientation is portrait or landscape. Overrides the user's selection in the Show Page Setup window.

Appendix A

SQL Functions Available for JMP Queries

The `Query()` JSL function performs a SQL query on selected tables and exports the data to a data table. The following example first assigns the t1 alias to Big Class.jmp. name, age greater than 13, and height are then selected from the t1 table.

```
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
Query( Table( dt, "t1" ),
       "SELECT t1.name, t1.age, t1.height FROM t1
       WHERE t1.age > 13" );
```

You can use SQL functions in a query. For example, `SELECT CURRENT_TIMESTAMP` returns the current UTC/GMT time stamp as a SQLite time string:

```
Query( Scalar, "SELECT CURRENT_TIMESTAMP;" );
```

This appendix lists the numeric, date-time, string, system SQL, and aggregate functions that you can use in SQL queries. “Yes” in the SQLite column indicates native SQLite functions. See the Online SQLite documentation at https://www.sqlite.org/lang.html.

Contents

Numeric SQL Functions 523
Date-Time SQL Functions 524
String SQL Functions 527
System SQL Functions 528
Aggregate SQL Functions 528

Numeric SQL Functions

The numeric SQL functions are described here.

Numeric Function	Native SQLite	Description
ABS(*number*)		Returns the absolute value of the specified number.
ACOS(*cosine*)		Returns the angle in radians for the specified cosine.
ASIN(*sin*)		Returns the angle in radians for the specified sine.
ATAN(*tangent*)		Returns the angle in radians for the specified tangent.
ATAN2(*x*, *y*)		Two-argument arctangent function.
CEILING(*number*) CEIL(*number*)		Returns the smallest integer larger than the specified number.
COS(*radians*)		Returns the cosine of the specified angle in radians.
COT(*radians*)		Returns the cotangent of the specified angle in radians.
DEGREES(*radians*)		Converts an angle in radians to an angle in degrees.
EXP(*number*)		Returns the constant *e* raised to the specified power.
FLOOR(*number*)		Returns the largest integer smaller than the specified number.
LN(*number*) LOG(*number*)		Returns the natural logarithm of the specified number.
LOG10(*number*)		Returns the common logarithm of the specified number.
MAX(*n1*, *n2*, ...)	Yes	Returns the largest of the specified numbers. A minimum of two numbers must be specified.
MIN(*n1*, *n2*, ...)	Yes	Returns the smallest of the specified numbers. A minimum of two numbers must be specified.
MOD(*dividend*, *divisor*)		Returns the remainder when *dividend* is divided by *divisor*. Floating-point values are truncated to integers before the modulus operation is performed.
PI()		Returns the value of the constant pi (π).
POWER(*number*, *power*) POW(*number*, *power*)		Raises *number* to the specified *power*.

Numeric Function	Native SQLite	Description
RADIANS(*degrees*)		Converts an angle in degrees to an angle in radians.
RANDOM() RANDOM(*max*) RANDOM(*min*, *max*)		Returns a random number. RANDOM() returns a number between 0 and 1. RANDOM(max) returns a number between 0 and *max*. RANDOM(min, max) returns a number between *min* and *max*. This function is equivalent to the Random Uniform() JSL function, and its seed can be controlled using the Random Reset() JSL function. RANDOM can be shortened to RAND.
RANDOMBLOB(*length*)	Yes	Returns an N-byte blob that contains pseudo-random bytes. See the SQLite Online documentation at https://www.sqlite.org/lang.html.
ROUND(*number*, *<precision>*)		Rounds *number* to the number of decimal places given by *precision*. The default value of *precision* is 0, and *precision* can be negative.
SIGN(*number*)		Returns 1 if *number* is positive, -1 if *number* is negative, or 0 if *number* is zero.
SIN(*radians*)		Returns the sin of the specified angle in radians.
SQRT(*number*)		Returns the square root of *number*.
TAN(*radians*)		Returns the tangent of the specified angle in radians.
TRUNCATE(*number*, *<precision>*)		Truncates *number* at the number of decimal places given by *precision*. The default value of *precision* is 0, and *precision* can be negative. TRUNCATE() can be shortened to TRUNC().

Date-Time SQL Functions

Using date-time functions in JMP queries is complicated by the fact that the SQL engine that handles JMP queries (SQLite) uses different formats for storing dates than JMP does. SQLite stores date-times as strings. However, JMP stores date-times as the number of seconds since January 1, 1904. When you have columns in your table that contain date-times, the conversions are handled automatically. However, when you use functions that return date-times, you might need to let JMP know when a conversion is required.

Consider the CURRENT_TIMESTAMP function. CURRENT_TIMESTAMP is a built-in SQLite function that returns the current UTC/GMT time stamp as a SQLite time string:

```
Query( Scalar, "SELECT CURRENT_TIMESTAMP;" );
```

returns:

```
"2016-02-16 15:44:42"
```

The string could perhaps be parsed as a date to return it as a JMP date. To prevent the need to do so, wrap the CURRENT_TIMESTAMP function in the JMPDATE() function:

```
Query( Scalar, "SELECT JMPDATE( CURRENT_TIMESTAMP );" );
```

returns:

```
3538482531
```

The string is an unformatted JMP date. However, if you pass a SQLite time string to another SQL date-time function, you do not need to use JMPDate(); the value will be converted to a JMP date automatically. Here is an example:

```
Query( Scalar, "SELECT EXTRACT('YEAR', CURRENT_TIMESTAMP);" );
```

Using native SQLite date-time functions (date(), time(), datetime(), julianday(), strftime()) in JMP queries is not recommended because JMP date-time values are not compatible with those functions.

Date-Time Function	Naive SQLite	Description
CURRENT_DATE	Yes	Returns the current date (UTC/GMT) as a SQLite time string.
CURRENT_TIME	Yes	Returns the current time (UTC/GMT) as a SQLite time string.
CURRENT_TIMESTAMP	Yes	Returns the current date and time (UTC/GMT) as a SQLite time string.
DATEDIFF(*date1*, *date2*, *interval*, <*alignment = "Start"*>)		Computes the difference between two dates in units specified by *interval*, based on *alignment*. This function works the same as the Date Difference() JSL function. Valid values for *interval* are: "Year", "Quarter", "Month", "Week", "Day", "Hour", "Minute" and "Second". Valid values for *alignment* are "Start", "Actual" and "Fractional". If *alignment* is not specified, "Start" is used.

Date-Time Function	Naive SQLite	Description	
EXTRACT(*datepart*, *datetime*, *<use_locale = 1>*)		Extracts a specific part of a date or date-time value. *Datetime* is a JMP date-time value or a SQLite time string. *Use_locale* is optional and applies only to date name parts such as "MonthName" and "DayName" and determines whether values from the current language or English are returned. The following values of *datepart* are supported:	
		"Year"	Returns the year as a number.
		"Month"	Returns the numeric month (1-12).
		"MonthName"	Returns the full name of the month in the current language (*use_locale = 1*) or English (*use_locale = 0*).
		"Mon", "MMM"	Returns the abbreviated name of the month.
		"Day"	Returns the day of the month (1-31).
		"DayName"	Returns the name of the day of the week.
		"DayOfWeek"	Returns the numeric day of the week (1-7).
		"DayOfYear"	Returns the numeric day of the year (1-366).
		"Quarter"	Returns the numeric quarter (1-4).
		"Hour"	Returns the hour (0-23).
		"Minute"	Returns the minute (0-59).
		"Second"	Returns the seconds, including any fractional part.
		"Date"	Returns just the date portion of a date-time value as a JMP date-time value.
		"Time"	Returns just the time portion of a date-time value as a JMP date-time value.
JMPDATE(*SQLite time string*)		Converts a SQLite time string to the equivalent JMP date-time value.	
NOW()		A synonym for TODAY().	
TODAY()		Returns the JMP date-time value of the current moment in *local* time, which matches the JMP Today() function.	

String SQL Functions

The string SQL functions are described here
.

Function	Native SQLite	Description
HEX(*binary*)	Yes	SQLite built-in function that converts a BLOB to a string of hexadecimal characters. Useful when paired with the RANDOMBLOB() function.
JLEFT(*string*, *len*, *<pad>*)		Like the JSL Left() function. Returns *len* characters from the beginning of *string*. If *pad* is specified and fewer than *len* characters are present in *string*, the result is padded with *pad* out to length *len*.
JRIGHT(*string*, *len*, *<pad>*)		Like the JSL Right() function. Returns *len* characters from the end of *string*. If *pad* is specified and fewer than *len* characters are present in *string*, the result is padded with *pad* at the front out to length *len*.
LENGTH(*string*)	Yes	SQLite equivalent of the ANSI standard CHAR_LENGTH() function. Returns the length of its string argument in characters.
LOCATE(*string1*, *string2*) POSITION(*string1*, *string2*)		Returns the (1-based) starting position of *string1* within *string2*, returning 0 if *string1* is not found within *string2*.
LOWER(*string*)		Returns a copy of *string* with all uppercase characters converted to lowercase.
LTRIM(*string*, *<trimchars>*)	Yes	Trims any characters contained in *trimchars* from the beginning of *string* and returns the result. If *trimchars* is omitted, spaces are trimmed.
PRINTF(*format*, *<arg1, ..., argN>*)	Yes	Allows constructing strings using placeholders and arguments. See the SQLite Online documentation at https://www.sqlite.org/lang.html.
REPLACE(*string*, *find*, *replace*)	Yes	Replaces all instances of *find* in *string* with *replace* and returns the result. If *replace* is numeric, it is converted to a string.
REVERSE(*string*)		Returns a copy of *string* with the order of the characters reversed.

Function	Native SQLite	Description
RTRIM(*string*, *<trimchars>*)	Yes	Trims any characters contained in *trimchars* from the end of *string* and returns the result. If *trimchars* is omitted, spaces are trimmed.
SPACE(*length*)		Returns a string consisting of *length* space characters.
SUBSTR(*string*, *start*, *<length>*)	Yes	Returns the substring of *string* starting at *start* (1-based) that is *length* characters long. If *length* is omitted, the substring starting at *start* and continuing to the end of *string* is returned.
TRIM(*string*, *<trimchars>*)		Trims any characters contained in *trimchars* from the end of *string* and returns the result. If *trimchars* is omitted, spaces are trimmed.
UPPER(*string*)		Returns a copy of *string* with all lowercase characters converted to uppercase.

System SQL Functions

The system SQL functions are described here.

Function	SQLite	Description
COALESCE(*arg1*,..., *argN*)	Yes	Returns the first argument passed to it that is non-NULL. Returns NULL if all arguments are NULL. Requires at least two arguments.
IFNULL(*arg1*, *arg2*)	Yes	Returns *arg1* if not NULL, otherwise *arg2*. Basically, IFNULL is a two-argument version of COALESCE().
NULLIF(*arg1*, *arg2*)	Yes	Returns *arg1* if *arg1* and *arg2* are different and returns NULL if the arguments are equal. Used when you have non-NULL values in your database that you want to treat as NULL.

Aggregate SQL Functions

When passing a single argument to an aggregate function, that argument can be preceded by the keyword DISTINCT, which filters out duplicate values.

For all aggregations other than COUNT(*), NULL and missing values are ignored.

Function	SQLite	Description
AVG(*num_expr*)		Computes the average of *num_expr* for the rows in the group. *Num_expr* must be numeric.
COUNT(*expr*) COUNT(*)		Counts the number of times *expr* is not NULL in the group. COUNT(*) returns the total number of rows in the group.
GROUP_CONCAT(*expr*, *<separator = ','>*)	Yes	Concatenates all non-NULL values of *expr* and returns them as a string. Numeric values of *expr* are converted to character. If *separator* is present, it is placed between the values. The default separator is a comma. DISTINCT can be used only with GROUP_CONCAT() if *separator* is not specified.
MAX(*expr*)		Returns the maximum value of *expr* in the group. *Expr* can be character or numeric.
MIN(*expr*)		Returns the minimum value of *expr* in the group. *Expr* can be character or numeric.
STDDEV_POP(*num_expr*)		Computes the population standard deviation of *num_expr* for the group.
STDDEV_SAMP(*num_expr*)		Computes the sample standard deviation of all *num_expr* for the group.
SUM(*num_expr*)		Returns the sum of *num_expr* for the group. If no non-NULL values are found, SUM() returns NULL.
TOTAL(*num_expr*)	Yes	Same as SUM(*num_expr*), except TOTAL() returns 0.0 if no non-NULL values are found.
VAR_POP(*num_expr*)		Computes the population variance of *num_expr* for the group.
VAR_SAMP(*num_expr*)		Computes the sample variance of *num_expr* for the group.

Appendix B

References

The following sources are referenced in *JSL Syntax Reference*.

Gander, W., and Gautschi, W. (2000). "Adaptive Quadrature–Revisited." *BIT Numerical Mathematics* 40:84–101.

Moore, W. A., and Parks, D. R. (2012). "Update for the logicle data scale including operational code implementations." *Cytometry Part A* 81A:273–277.

Appendix C

Technology License Notices

- Scintilla is Copyright © 1998–2017 by Neil Hodgson <neilh@scintilla.org>.

  All Rights Reserved.

  Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

  NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
- Progressë® Telerik® UI for WPF: Copyright © 2008-2019 Progress Software Corporation. All rights reserved. Usage of the included Progress® Telerik® UI for WPF outside of JMP is not permitted.
- ZLIB Compression Library is Copyright © 1995-2005, Jean-Loup Gailly and Mark Adler.
- Made with Natural Earth. Free vector and raster map data @ naturalearthdata.com.
- Packages is Copyright © 2009–2010, Stéphane Sudre (s.sudre.free.fr). All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  Neither the name of the WhiteBox nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- iODBC software is Copyright © 1995–2006, OpenLink Software Inc and Ke Jin (www.iodbc.org). All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  – Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  – Neither the name of OpenLink Software Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL OPENLINK OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This program, "bzip2", the associated library "libbzip2", and all documentation, are Copyright © 1996–2019 Julian R Seward. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.

  3. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Julian Seward, jseward@acm.org

bzip2/libbzip2 version 1.0.8 of 13 July 2019

- R software is Copyright © 1999–2012, R Foundation for Statistical Computing.
- MATLAB software is Copyright © 1984-2012, The MathWorks, Inc. Protected by U.S. and international patents. See www.mathworks.com/patents. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.
- libopc is Copyright © 2011, Florian Reuter. All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  – Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and / or other materials provided with the distribution.
  – Neither the name of Florian Reuter nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF

USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- libxml2 - Except where otherwise noted in the source code (e.g. the files hash.c, list.c and the trio files, which are covered by a similar license but with different Copyright notices) all the files are:

  Copyright © 1998–2003 Daniel Veillard. All Rights Reserved.

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

  Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

- Regarding the decompression algorithm used for UNIX files:

  Copyright © 1985, 1986, 1992, 1993

  The Regents of the University of California. All rights reserved.

  THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

- Snowball is Copyright © 2001, Dr Martin Porter, Copyright © 2002, Richard Boulton.

  All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS \"AS IS\" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- Pako is Copyright © 2014–2017 by Vitaly Puzrin and Andrei Tuputcyn.

  Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- HDF5 (Hierarchical Data Format 5) Software Library and Utilities are Copyright 2006–2015 by The HDF Group. NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities Copyright 1998-2006 by the Board of Trustees of the University of Illinois. All rights reserved. DISCLAIMER: THIS SOFTWARE IS PROVIDED BY THE HDF GROUP AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall The HDF Group or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.
- agl-aglfn technology is Copyright © 2002, 2010, 2015 by Adobe Systems Incorporated. All Rights Reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  – Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  – Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  – Neither the name of Adobe Systems Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- dmlc/xgboost is Copyright © 2019 SAS Institute.

  Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- libzip is Copyright © 1999–2019 Dieter Baron and Thomas Klausner.

  This file is part of libzip, a library to manipulate ZIP archives. The authors can be contacted at <libzip@nih.at>.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE AUTHORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- OpenNLP 1.5.3, the pre-trained model (version 1.5 of en-parser-chunking.bin), and dmlc/xgboost Version .90 are licensed under the Apache License 2.0 are Copyright © January 2004 by Apache.org.

  You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- You must give any other recipients of the Work or Derivative Works a copy of this License; and
- You must cause any modified files to carry prominent notices stating that You changed the files; and
- You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.
- You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

- LLVM is Copyright © 2003–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- clang is Copyright © 2007–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS", WITHOUT WARRANTIES OR CONDITIONS OF

ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- lld is Copyright © 2011–2019 by the University of Illinois at Urbana-Champaign.

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:

  http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS", WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

- libcurl is Copyright © 1996–2021, Daniel Stenberg, daniel@haxx.se, and many contributors, see the THANKS file. All rights reserved.

  Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

  Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

- On the Windows operating system, JMP utilizes the OpenBLAS library. OpenBLAS is licensed under the 3-clause BSD license. Full license text follows: Copyright © 2011-2015, The OpenBLAS Project

  All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

  3. Neither the name of the OpenBLAS project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OPENBLAS PROJECT OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.