“The real voyage of discovery consists not in seeking new landscapes, but in having new eyes.”
Marcel Proust
SAS 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 SAS software, refer to http://support.sas.com/thirdpartylicenses.

**Technology License Notices**

- **Scintilla** - Copyright © 1998-2014 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.

- **Telerik RadControls**: Copyright © 2002-2012, Telerik. Usage of the included Telerik RadControls outside of JMP is not permitted.


- **Made with Natural Earth. Free vector and raster map data @ naturalearthdata.com.**

- **Packages** - 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 - 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.

• bzip2, the associated library “libbzip2“, and all documentation, are Copyright ©
  1996-2010, 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:
Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

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.

Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.

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.


• 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 licence 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:

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 - 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.

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
• a blog with tips, tricks, and stories from JMP staff
• a forum to discuss JMP with other users

http://www.jmp.com/getstarted/
## Contents

**JSL Syntax Reference**

### 1 Learn about JMP

**Documentation and Additional Resources** .................................................. 13
- Formatting Conventions ................................................................. 14
- JMP Documentation ..................................................................... 15
  - JMP Documentation Library ......................................................... 15
  - JMP Help ....................................................................................... 21
- Additional Resources for Learning JMP ........................................ 21
  - Tutorials ....................................................................................... 22
  - Sample Data Tables ................................................................. 22
  - Learn about Statistical and JSL Terms ........................................ 22
  - Learn JMP Tips and Tricks ......................................................... 22
  - Tooltips ....................................................................................... 23
- JMP User Community ................................................................. 23
- JMPer Cable ................................................................................. 23
- JMP Books by Users ................................................................. 24
  - The JMP Starter Window ........................................................... 24
- Technical Support ......................................................................... 24

### 2 JSL Functions

**Summary of Functions, Operators, and Messages** .......................... 25
- Assignment Functions ............................................................... 26
- Character Functions ................................................................. 29
- Character Pattern Functions ....................................................... 42
- Comment Functions ....................................................................... 51
- Comparison Functions ............................................................... 52
- Conditional and Logical Functions .................................................. 57
- Constant Functions ................................................................. 64
- Date and Time Functions .............................................................. 64
- Discrete Probability Functions .......................................................... 72
- Display Functions .......................................................................... 75
- Expression Functions ................................................................. 95
- File Functions ............................................................................. 98
- Financial Functions ...................................................................... 114
- Graphic Functions ....................................................................... 119
- List Functions ............................................................................... 134
<table>
<thead>
<tr>
<th>MATLAB Integration Functions</th>
<th>140</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB JSL Function Interfaces</td>
<td>141</td>
</tr>
<tr>
<td>Matrix Functions</td>
<td>147</td>
</tr>
<tr>
<td>Numeric Functions</td>
<td>164</td>
</tr>
<tr>
<td>Optimization Functions</td>
<td>167</td>
</tr>
<tr>
<td>Probability Functions</td>
<td>170</td>
</tr>
<tr>
<td>Programming Functions</td>
<td>187</td>
</tr>
<tr>
<td>R Integration Functions</td>
<td>199</td>
</tr>
<tr>
<td>Random Functions</td>
<td>204</td>
</tr>
<tr>
<td>Row Functions</td>
<td>211</td>
</tr>
<tr>
<td>Row State Functions</td>
<td>215</td>
</tr>
<tr>
<td>SAS Integration Functions</td>
<td>218</td>
</tr>
<tr>
<td>SQL Functions</td>
<td>235</td>
</tr>
<tr>
<td>Statistical Functions</td>
<td>238</td>
</tr>
<tr>
<td>Transcendental Functions</td>
<td>248</td>
</tr>
<tr>
<td>Trigonometric Functions</td>
<td>254</td>
</tr>
<tr>
<td>Utility Functions</td>
<td>257</td>
</tr>
</tbody>
</table>

3 **JSL Messages**

**Summary of Messages for Objects and Display Boxes**

| Alpha Shape | 282 |
| Associative Arrays | 282 |
| Data Tables | 283 |
| Columns | 300 |
| Rows | 304 |
| Data Filter | 305 |
| Databases | 308 |
| Datafeed | 308 |
| Display Boxes | 309 |
| All Display Boxes | 309 |
| Axis Boxes | 317 |
| Border Boxes | 320 |
| Data Grid Boxes | 321 |
| Frame Boxes | 321 |
| Matrix Boxes | 323 |
| Nom Axis Boxes | 324 |
| Number Col Boxes | 324 |
| Outline Boxes | 325 |
| Panel Boxes | 326 |
| Plot Col Boxes | 326 |
| Slider Boxes and Range Slider Boxes | 326 |
| String Col Boxes | 328 |
| Tab Boxes | 328 |
This chapter includes the following information:

- book conventions
- JMP documentation
- JMP Help
- additional resources, such as the following:
  - other JMP documentation
  - tutorials
  - indexes
  - Web resources
  - technical support options
Formatting Conventions

The following 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 font.
- Code appears in Lucida Sans Typewriter font.
- Code output appears in Lucida Sans Typewriter italic font and is indented farther than the preceding code.
- Helvetica bold formatting 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
  - script output
- Features that are for JMP Pro only are noted with the JMP Pro icon . For an overview of JMP Pro features, visit http://www.jmp.com/software/pro/.

Note: Special information and limitations appear within a Note.

Tip: Helpful information appears within a Tip.
JMP Documentation

JMP offers documentation in various formats, from print books and Portable Document Format (PDF) to electronic books (e-books).

- Open the PDF versions from the Help > Books menu.
- All books are also combined into one PDF file, called JMP Documentation Library, for convenient searching. Open the JMP Documentation Library PDF file from the Help > Books menu.
- You can also purchase printed documentation and e-books on the SAS website: http://www.sas.com/store/search.ep?keyWords=JMP

JMP Documentation Library

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

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Document Purpose</th>
<th>Document Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovering JMP</td>
<td>If you are not familiar with JMP, start here.</td>
<td>Introduces you to JMP and gets you started creating and analyzing data.</td>
</tr>
<tr>
<td>Using JMP</td>
<td>Learn about JMP data tables and how to perform basic operations.</td>
<td>Covers general JMP concepts and features that span across all of JMP, including importing data, modifying columns properties, sorting data, and connecting to SAS.</td>
</tr>
<tr>
<td>Basic Analysis</td>
<td>Perform basic analysis using this document.</td>
<td>Describes these Analyze menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fit Y by X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Tabulate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Text Explorer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Covers how to perform bivariate, one-way ANOVA, and contingency analyses through Analyze &gt; Fit Y by X. How to approximate sampling distributions using bootstrapping and how to perform parametric resampling with the Simulate platform are also included.</td>
</tr>
<tr>
<td>Document Title</td>
<td>Document Purpose</td>
<td>Document Content</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Essential Graphing</td>
<td>Find the ideal graph for your data.</td>
<td>Describes these Graph menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Graph Builder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Overlay Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Scatterplot 3D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Contour Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bubble Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Parallel Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cell Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Treemap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Scatterplot Matrix</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ternary Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Chart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The book also covers how to create background and custom maps.</td>
</tr>
<tr>
<td>Profilers</td>
<td>Learn how to use interactive profiling tools, which enable you to view cross-sections of any response surface.</td>
<td>Covers all profilers listed in the Graph menu. Analyzing noise factors is included along with running simulations using random inputs.</td>
</tr>
<tr>
<td>Design of Experiments Guide</td>
<td>Learn how to design experiments and determine appropriate sample sizes.</td>
<td>Covers all topics in the DOE menu and the Specialized DOE Models menu item in the Analyze &gt; Specialized Modeling menu.</td>
</tr>
<tr>
<td>Document Title</td>
<td>Document Purpose</td>
<td>Document Content</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Fitting Linear Models</td>
<td>Learn about Fit Model platform and many of its personalities.</td>
<td>Describes these personalities, all available within the Analyze menu Fit Model platform:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Standard Least Squares</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Stepwise</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Generalized Regression</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Mixed Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MANOVA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Loglinear Variance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Nominal Logistic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ordinal Logistic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Generalized Linear Model</td>
</tr>
<tr>
<td>Document Title</td>
<td>Document Purpose</td>
<td>Document Content</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><em>Predictive and Specialized Modeling</em></td>
<td>Learn about additional modeling techniques.</td>
<td>Describes these Analyze &gt; Predictive Modeling menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Modeling Utilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Neural</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Partition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bootstrap Forest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Boosted Tree</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• K Nearest Neighbors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Naive Bayes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Model Comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Formula Depot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Describes these Analyze &gt; Specialized Modeling menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fit Curve</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Nonlinear</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Gaussian Process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Time Series</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Matched Pairs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Describes these Analyze &gt; Screening menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Response Screening</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Process Screening</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Predictor Screening</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Association Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The platforms in the Analyze &gt; Specialized Modeling &gt; Specialized DOE Models menu are described in <em>Design of Experiments Guide</em>.</td>
</tr>
<tr>
<td>Document Title</td>
<td>Document Purpose</td>
<td>Document Content</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Multivariate Methods</strong></td>
<td>Read about techniques for analyzing several variables simultaneously.</td>
<td>Describes these Analyze &gt; Multivariate Methods menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Multivariate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Principal Components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Discriminant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Partial Least Squares</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Describes these Analyze &gt; Clustering menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hierarchical Cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• K Means Cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Normal Mixtures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Latent Class Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cluster Variables</td>
</tr>
<tr>
<td><strong>Quality and Process Methods</strong></td>
<td>Read about tools for evaluating and improving processes.</td>
<td>Describes these Analyze &gt; Quality and Process menu platforms:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Control Chart Builder and individual control charts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Measurement Systems Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Variability / Attribute Gauge Charts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Process Capability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Pareto Plot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Diagram</td>
</tr>
<tr>
<td>Document Title</td>
<td>Document Purpose</td>
<td>Document Content</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **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 and Destructive Degradation  
  - Reliability Forecast  
  - Reliability Growth  
  - Reliability Block Diagram  
  - Repairable Systems Simulation  
  - Survival  
  - Fit Parametric Survival  
  - Fit Proportional Hazards |
| **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  
  - Multiple Correspondence Analysis  
  - Multidimensional Scaling  
  - Factor Analysis  
  - Choice  
  - MaxDiff  
  - Uplift  
  - Item Analysis |
| **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. |
Chapter 1
Learn about JMP

JSL Syntax Reference
Additional Resources for Learning JMP

<table>
<thead>
<tr>
<th>Document Title</th>
<th>Document Purpose</th>
<th>Document Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSL Syntax Reference</td>
<td>Read about many JSL functions on functions and their arguments, and messages that you send to objects and display boxes.</td>
<td>Includes syntax, examples, and notes for JSL commands.</td>
</tr>
</tbody>
</table>

**Note:** The Books menu also contains two reference cards that can be printed: The Menu Card describes JMP menus, and the Quick Reference describes JMP keyboard shortcuts.

**JMP Help**

JMP Help is an abbreviated version of the documentation library that provides targeted information. You can open JMP Help in several ways:

- On Windows, press the F1 key to open the Help system window.
- 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.
- Search the Help at http://jmp.com/support/help/ (English only).

**Additional Resources for Learning JMP**

In addition to JMP documentation and JMP Help, you can also learn about JMP using the following resources:

- Tutorials (see “Tutorials” on page 22)
- Sample data (see “Sample Data Tables” on page 22)
- Indexes (see “Learn about Statistical and JSL Terms” on page 22)
- Tip of the Day (see “Learn JMP Tips and Tricks” on page 22)
- Web resources (see “JMP User Community” on page 23)
- JMPer Cable technical publication (see “JMPer Cable” on page 23)
- Books about JMP (see “JMP Books by Users” on page 24)
- JMP Starter (see “The JMP Starter Window” on page 24)
• Teaching Resources (see “Sample Data Tables” on page 22)

Tutorials

You can access JMP tutorials by selecting Help > Tutorials. The first item on the Tutorials menu is Tutorials Directory. This opens a new window with all the tutorials grouped by category.

If you are not familiar with JMP, then 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 Library to open the sample data directory.

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

Sample data tables are installed in the following directory:

On Windows: C:\Program Files\SAS\JMP\13\Samples\Data

On Macintosh: \Library\Application Support\JMP\13\Samples\Data

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

To view examples using sample data, select Help > Sample Data and navigate to the Teaching Resources section. To learn more about the teaching resources, visit http://jmp.com/tools.

Learn about Statistical and JSL Terms

The Help menu contains the following indexes:

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.

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. See the Using JMP book for details.

**Tooltips**

JMP provides descriptive tooltips when you place your cursor 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 Macintosh.

**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 User Community or visit https://community.jmp.com/.

**JMPer Cable**

The JMPer Cable is a yearly technical publication targeted to users of JMP. The JMPer Cable is available on the JMP website:

http://www.jmp.com/about/newsletters/jmpercable/
JMP Books by Users

Additional books about using JMP that are written by JMP users are available on the JMP website:


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 the Macintosh) > 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 Macintosh, select JMP > Preferences > Initial JMP Starter Window.

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 http://www.jmp.com/support, including the technical support phone number.
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 Platforms chapter in the Scripting Guide.
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, 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 **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 \( a \).

**See Also**

The Data Structures chapter in the *Scripting Guide*. 
Assign(a, b)

\[ a = b \]

**Description**
Placing 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.

**See Also**
The Data Structures chapter in the *Scripting Guide*.

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 \( a \).

See Also
The Data Structures chapter in the Scripting Guide.

---

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.

---

PostIncrement(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 PostIncrement(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 \( a \).

See Also

The Data Structures chapter in the Scripting Guide.

Character Functions

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

The Types of Data chapter in the Scripting Guide provides more details about some of the functions.
**Blob To Char(blob, <"encoding">)**

**Description**
Reinterpret binary data as a Unicode string.

**Returns**
A string.

**Arguments**
- **blob** a binary large object.
- **encoding** Optional 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 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.

**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** A quoted string that contains a named type that indicates whether the first byte is the most significant. Options are as follows:
  - "big" indicates that the first byte is the most significant.
  - "little" indicates that the first byte is the least significant.
  - "native" indicates that the machine’s native format should be used.
- **nCols** Optional. Specify the number of columns in the matrix. The default value is 1.

**Char(x, <width>, <decimal>)**

**Description**
Converts an expression or numeric value into a character string.
## Returns
A string.

## Arguments
- **x**: an expression or a numeric value. An expression must be quoted with `Expr()`. Otherwise, its evaluated value is converted to a string.
- **width**: optional number that sets the maximum number of characters in the string.
- **decimal**: optional number that sets the maximum number of places after the decimal that is included in the string.

### Note
The `width` argument overrides the `decimal` argument.

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

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

### Char To Blob(string, <"encoding">)

#### Description
Converts a string of characters into a binary (blob).

#### Returns
A binary object.

#### Arguments
- **string**: Quoted string or a reference to a string.
- **encoding**: Optional 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">)
Hex(value, "<"integer" | "encoding">)

Description
Converts the given value and encoding into its hexadecimal representation.

Returns
A hexadecimal string.

Arguments
value  Any number, quoted string, or blob.
integer  Switch that causes the value to be interpreted as an integer.
encoding  Optional 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.

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.

Arguments
text  A quoted string.

Concat(a, b)
Concat(A, B)
a||b
A||B

Description
For 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 strings: A 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 strings, string variables, lists, or matrices.

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

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]
```

Concat Items
See “Concat Items({string1, string2, ...}, <delimiter>)” on page 135.

Concat To(a, b)
Concat To(a, b)
```
a||=b
A||=B
```

Description
For 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.

Returns
For strings: A string composed of the string a directly followed by the string b.
For matrices: A matrix.

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

Notes
More than two arguments can be strung together. Each additional string or matrix 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];

Contains(whole, part, <start>)

Description
Determines whether part is contained within whole.

Returns
If part is found: For lists, 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.

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

part For a 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.

start An optional 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 delimiter (delimiter) string. A comma, "", character 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." ), "No match." 
    );
    *The specified items were found.*

**Ends With(string, substring)**

**Description**
Determine whether substring appears at the end of string.

**Returns**
1 if string ends with substring, otherwise 0.

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

**Equivalent Expression**
Right(string, Length(substring)) == substring

**Hex(value, <"integer" | encoding="enc">)**
See “Char To Hex(value, <"integer" | "encoding">)” on page 32.

**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 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)

Description

Converts the quoted hexadecimal string to its integer or its floating number equivalent.

Example

Hex To Number( "80" );
   128

Note

16-digit hexadecimal numbers are converted as IEEE 754 64-bit floating point numbers.
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>)” on page 135.

Insert Into

See “Insert Into(source, item, <position>)” on page 136.

Item(n, string, <delimiters>)

Description

Extracts the n\textsuperscript{th} word from a quoted string according to the quoted string delimiters given. The default delimiter is space. If you include a third argument, any and all characters in that argument are taken to be delimiters.

Note

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.

Item( 4,"the quick, brown fox", ", " );
   "brown"

Word( 4,"the quick, brown fox", ", " );
   "fox"

Left(string, n, <filler>)

Left(list, n, <filler>)

Description

Returns a truncated or padded version of the original 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**
Calculates the number of characters (length) of the quoted `string`.

### Lowercase(string)

**Description**
Converts any upper case character found in quoted `string` to the equivalent lowercase character.

### Munger(string, offset, find|length)

**Description**
Computes new 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.

- **Offset** is a numeric expression indicating the starting position to search in the string. If the offset is greater than the position of the first instance of the find argument, the first instance is disregarded. If the offset is greater than the search string’s length, Munger uses the string’s length as the offset.

### Num(string)

**Description**
Converts a character string into a number.

### Regex("source", "pattern", (<replacementString>, <GLOBALREPLACE>), <format>, <IGNORECASE>)

**Description**
Searches for the `pattern` within the `source` string.

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

**Arguments**
- **source** A quoted string.
- **pattern** A quoted regular expression.
- **format** Optional. A backreference to the capturing group. The default is `/0`, which is the entire matched string. `/n` returns the `n`th match.
- **IGNORECASE** Optional. The search is case sensitive, unless you specify IGNORECASE.
GLOBALREPLACE  Optional with a replacement string. Applies the regular expression to the source string repeatedly until all matches are found.

Remove

See “Remove(source, position, <n>)” on page 137.

Remove From

See “Remove From(source, position, <n>)” on page 137.

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)” on page 138.

Reverse Into

See “Reverse Into(source)” on page 138.

Right(string, n, <filler>)

Right(list, n, <filler>)

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

Shift

See “Shift(source, <n>)” on page 138.

Shift Into

See “Shift Into(source, <n>)” on page 138.
Starts With(string, substring)

Description
Determines whether substring appears at the start of 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, ...)” on page 139.

Substitute Into
See “Substitute Into(string, substring, replacementString, ...)” on page 139.

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.”

Titlecase("text")

Description
Converts the string to title case, that is, each word in the string has an initial uppercase character and subsequent lowercase characters.

Returns
A quoted string.
Arguments
  text A quoted string.

Example
  For example, the following function:
  \texttt{Titlecase( "veronica layman ")}
  returns the following string:
  "Veronica Layman"

\texttt{Trim("text",<left|right|both>)}

Description
  Removes leading and trailing whitespace.

Results
  A quoted string.

Arguments
  text A quoted string.
  left|right|both The second argument determines if whitespace is removed from the
                  left, the right, or both ends of the string. If no second argument is used, whitespace is
                  removed from both ends.

Example
  For example, the following command:
  \texttt{Trim( " John ", both )}
  returns the following string:
  "John"

\texttt{Uppercase(string)}

Description
  Converts any lower case character found in the quoted string to the equivalent uppercase
  character.

\texttt{Word(n, "text", <"delimiters">)}

Description
  Extracts the \( n \)\textsuperscript{th} word from a character string according to the delimiters given. The default
delimiter is space. If you include a third argument, any and all characters in that argument
are taken to be delimiters.
Examples

This example returns the last name:

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

Note

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

```javascript
Item( 4,"the quick brown fox" );
"brown"
Word( 4,"the quick brown fox" );
"fox"
```

Words

See “Words("text", <"delimiters">)” on page 140.

**XPath Query( xml, "xpath_expression")**

Description

Runs an XPath expression on an XML document.

Returns

A list.

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.

```javascript
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

See the Types of Data chapter in the Scripting Guide book for more detailed information on constructing and using pattern matching expressions.

Pat Abort()

Description
Constructs a 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, <pattern 2, ...>)

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

Returns
A pattern.

Argument
One or more patterns.

Pat Any("string")

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

Returns
A pattern.

Argument
string a string.

Pat Arb()

Description
Constructs a pattern that matches an arbitrary string. It initially matches the null string. It then matches one additional character each time the pattern matcher backs into it.
Chapter 2
JSL Syntax Reference

JSL Functions
Character Pattern Functions

### Pat Arb No(pattern)

**Description**

Constructs a pattern that matches zero or more copies of `pattern`.

**Returns**

A pattern.

**Argument**

- `pattern` a 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 pattern that matches the null 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 pattern.

**Argument**

- `varName` the name of a variable to store the result in.

**Example**

```
p = ":" + Pat At(listStart) + Expr(
   If(listStart == 1,
   ...)
```

---

Returns

A 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 ";
```

---

Returns

A pattern.

Argument

- `pattern` a 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 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 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 source string without finding a break character).

Returns
A pattern.

Argument
string a string.

Pat Concat(pattern1, pattern2 <pattern 3, ...>)
Pattern1 + Pattern2 + ...
Pat Conditional(pattern, varName)

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

**Returns**
A pattern.

**Arguments**
- pattern a pattern to match against.
- varName the name of a variable to store the result in.

**Example**
```javascript
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 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 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 pattern.

**Arguments**
- **pattern** a 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 pattern that matches \( n \) characters.

**Returns**
A pattern.

**Argument**
- **int** an integer that specifies the number of characters.

---

**Pat Match(SourceText, Pattern, <ReplacementText>, <NULL>, <ANCHOR>, <MATCHCASE>, <FULLSCAN>)**

**Description**
Pat Match executes the Pattern against the SourceText. 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.

**Arguments**
- **SourceText** A string or string variable that contains the text to be searched.
Pattern  A pattern or pattern variable that contains the text to be searched for.

ReplacementText  Optional 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  Optional command to start the pattern match to the beginning of the 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 pattern.

**Argument**

string  a string.

---

**Pat Pos(int)**

**Description**

Constructs patterns that match the null string if the current position is \textit{int} from the left end of the string, and fail otherwise.

**Returns**

A pattern.

**Argument**

int  an integer that specifies a position in a string.

---

**Pat R Pos(int)**

**Description**

Constructs patterns that match the null string if the current position is \textit{int} from the right end of the string, and fails otherwise.

**Returns**

A pattern.
**JSL Functions**

Character Pattern Functions

---

**Pat R Tab(int)**

**Description**
Constructs a pattern that matches up to position \( n \) from the end of the 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 string.

---

**Pat Regex(string)**

**Description**
Constructs a pattern that matches the regular expression in the quoted \( string \) argument.

**Returns**
A pattern.

**Argument**

string a string.

---

**Pat Rem()**

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

**Returns**
A pattern.

**Argument**

none

---

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

**Description**
Matches \( pattern \) between \( minimum \) and \( maximum \) times.

**Returns**
A pattern.
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.
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 string. It fails rather than matching zero characters.

Returns
A pattern.

Argument
string  a string.

Pat String("string")

Description
Constructs a pattern that matches its string argument.

Returns
A pattern.

Argument
string  a string.

Pat Succeed()

Description
Constructs a pattern that always succeeds, even when the matcher backs into it. It matches the null string.
Returns
1 when the match succeeds.

Argument
none

Pat Tab(int)

Description
Constructs a pattern that matches forward to position \textit{int} in the 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
\textit{int} an integer that specifies a position in a string.

Pat Test(expr)

Description
Constructs a pattern that succeeds and matches the null string if \textit{expr} is not zero and fails otherwise.

Returns
A pattern.

Argument
\textit{expr} An expression.

Note
Usually the argument is wrapped with \texttt{expr()} because the test needs to be made on the current value of variables set by \texttt{Pat Immediate}, \texttt{Pat Conditional}, and \texttt{Pat At}. Without \texttt{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
\begin{verbatim}
  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;
    ))
\end{verbatim}


```plaintext
) 
),  
"dog"
);
Show( rc, string, nCats );
rc = 1
string = "catch a catnapping dog in a catsup factory"
ncats = 3

Regex Match(source, pattern, <MATCHCASE>)

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

Returns
A pattern.

Argument
source a string.
pattern a pattern.
MATCHCASE Optional. The search is case insensitive unless you specify MATCHCASE.

---

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`.

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

is valid JSL; the comments are both ignored.

```js
//!
```

**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.

```js
/*debug step*/
/*debug run*/
```

**Description**

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

**Notes**

All letters must be lower case. 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, 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, 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:

```js
a = 1;
Show( 1 <= a < 3 );
```
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 \(<...<=\)...\(<=...<\). 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 string comparisons.
Greater(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, Greater0rEqual, and Less0rEqual can also be strung together. 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, ...)

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, Greater0rEqual, and Less0rEqual can also be strung together. 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, ...)

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, GreaterOrEqual, and LessOrEqual can also be strung together. 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 LessEqual(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.
Comparison Functions

JSL Functions

Chapter 2

JSL Syntax Reference

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

\[ a \leq b \leq \ldots \]

Description
Compared 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, GreaterOrEqual, and LessOrEqual can also be strung together. 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.

LessEqual Less(a, b, c, ...)

\[ a \leq b < c < \ldots \]

Description
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)

\[ a \neq b \]

**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)

\[ a \land b \]

**Description**
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. \( a \land b \) returns 1 (true) only if all arguments evaluate to true.

AndMZ(a, b)
AndV3(a, b)

\[ a : \land b \]

**Description**
Logical And with JMP 3 behavior, which treats missing values as 0.
JSL Functions
Chapter 2
Conditional and Logical Functions

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. When opening a JMP 3 data table, this function is automatically used for any And function.

Break()

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

Note
Break works with For and While loops, and also with For Each Row.

Choose(expr, r1, r2, r3, ..., rElse)

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 (rElse) 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.

Continue()

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

Note
Continue works with For and While loops, and also with For Each Row.
**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 Row**(dt, script)

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

**Returns**
Null.

**Argument**
- **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.
- **script** Any valid JSL expressions.

**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.
```
dt1 = Open( "$SAMPLE_DATA/Big Class.jmp" );
dt2 = Open( "$SAMPLE_DATA/San Francisco Crime.jmp" );
For Each Row( dt1, If( :age > 15, Show( :age ) ) );
```
If(condition, result, condition, ..., <elseResult>)

Description
Returns result when condition evaluates true. Otherwise returns next result when that condition evaluates as true. The optional elseResult is used if none of the preceding conditions are true. If no elseResult is given, and none of the conditions are true, then nothing happens.

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

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 result>)

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(condition, result, condition, ..., <elseResult>)
IfV3(condition, result, condition, ..., <elseResult>)

Description
Logical If with version 3.x behavior, which treats missing values as 0; used automatically when opening v3 data tables.
Interpolate(x, x1, y1, x2, y2)
Interpolate(x, xmatrix, ymatrix)

**Description**
Linearly interpolates the \( y \)-value corresponding to a given \( x \)-value between two points \((x_1, y_1)\), and \((x_2, y_2)\) or by matrices \(xmatrix\) and \(ymatrix\). The points must be in ascending order.

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 does not have a value (is uninitialized), 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)” on page 136.

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 string, or 0 otherwise.

Match(a, value1, result1, value2, result2, ...)

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

MatchMZ(a, value1, result1, value2, result2, ...)
MatchV3(a, value1, result1, value2, result2, ...)

Description
Match with version 3.x behavior, which treats missing values as 0; used automatically when opening v3 data tables.

Not(a)

!a

Description
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.
0r(a, b)

\(a \mid b\)

**Description**
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.

0rMZ(a, b)
0rV3(a, b)

\(a :| b\)

**Description**
Logical Or with version 3.x behavior, which treats missing values as 0.

**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. When opening a JMP 3 data table, this function is automatically used for any \(0r\) function.

0r() returns missing if any evaluated argument is missing. 0rMZ() returns 0 if any evaluated argument is missing.

**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.

**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 $\pi$, 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 string.

**Returns**
A 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
```

**See Also**
The Types of Data chapter in the *Scripting Guide*.

---

**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.

**See Also**
The Types of Data chapter in the *Scripting Guide*.

---

**Date Difference(datetime1, datetime2, "interval_name", <"alignment">)**

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

**Returns**
A number.

**Arguments**
datetime1, datetime2  Datetime values.
interval_name  A quoted string that contains a date-time interval, such as "Month", "Day", or "Hour".
alignment  An optional string. Options are as follows:
- "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 number, day of month, 1-31. Note that there is no error-checking, so you can enter February 31.
- month number of month, 1-12.
- year number of 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.

**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.
- increment An optional number that specifies the number of intervals. The default value is 1.
- alignment An optional 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  number of month, 1-12.
- day    number, day of month, 1-31. Note that there is no error-checking, so you can enter February 31.
- year   number of year.

Day(datetime)

Description
Determine 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  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
Determine 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.

Arguments
- datetime  Number of seconds since midnight, 1 January 1904. This can also be an expression.
Day Of Year(datetime)

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

Format(x, "format", <"currency code">, <decimal>)
Format Date(x, "format", <"currency code">, <decimal>)

Description
Converts the value of `x` into the "format" that you specify in the second argument. Typically used for formatting datetime values from a number of seconds to a formatted date. Format choices are those shown in the data table column properties.

Returns
Returns the converted date in the format specified.

Arguments
x Can be a column, a number, or a datetime.
format Any valid format, as a quoted string: "Best", "Fixed Decimal", "Percent", "PValue", "Scientific", "Currency", or any of the Date/Time formats.
currency code An optional ISO 4217 code for a specific currency (for example, "GBP" for Great Britain, Pound). This argument is valid only if Currency is specified as `format`.
decimal An optional integer that specifies the number of decimal places to be shown.

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 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.

Note
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 Format(string, "format")
Parse Date(string, "format")

Description
Parses a string of a given "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. Format choices are those shown in the data table column properties. See “Utility Functions” on page 257 for details on As Date.

Example
Informat( "07152000", "MMDDYYYY" );
15Jul2000

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.

Long Date(date)

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

MDYHMS(date)

Description
Returns a 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 \( date \) supplied.

Parse Date()

See "In Format(string, "format")" on page 69.

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 string representation for the `date` supplied, in the format mm/dd/yy. For example, “2/29/04” for the next Leap Day.

---

Tick Seconds()

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

**Note**
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, <rule_n>)

**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 probability that a beta binomially distributed random variable is less than or equal to `k`.

**Beta Binomial Probability**

(k, p, n, delta)

**Description**

Returns the probability that a beta binomially distributed random variable is equal to `k`.

**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`.

**Binomial Distribution**

(p, n, k)

**Description**

The probability that a binomially distributed random variable is less than or equal to `k`.

**Returns**

The cdf for the binomial distribution with `n` trials, probability `p` of success for each trial, and `k` successes.

**Arguments**

- `p`  probability of success for each trial
- `n`  number of trials
- `k`  number of successes
**Binomial Probability** \( (p, n, k) \)

**Description**

The probability that a binomially distributed random variable is equal to \( k \).

**Returns**

The probability of getting \( k \) successes out of \( n \) trials if the probability is \( p \).

**Arguments**

- \( p \) probability of success for each trial
- \( n \) number of trials
- \( k \) number of successes

**Binomial Quantile** \( (p, n, \text{cumprob}) \)

**Description**

The quantile function of binomial distribution.

**Returns**

The smallest integer quantile for which the cumulative probability of the Binomial \( (p, n) \) distribution is larger than or equal to \( \text{cumprob} \).

**Arguments**

- \( p \) probability of success for each trial
- \( n \) number of trials
- \( \text{cumprob} \) cumulative probability

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

**Description**

Returns the probability that a gamma-Poisson distributed random variable is less than or equal to \( k \).

**Arguments**

- \( k \) The count of interest.
- \( \lambda \) The mean argument.
- \( \sigma \) The overdispersion argument.

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

**Description**

Returns the probability that a gamma-Poisson distributed random variable is equal to \( k \).

**Arguments**

- \( k \) The count of interest.
lambda The mean argument.
sigma The overdispersion argument.

**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*.

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

**Description**
Returns the cumulative distribution function at *x* for the hypergeometric distribution with population size *N*, *K* items in the category of interest, sample size *n*, count of interest *x*, and optional odds ratio *r*.

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

**Description**
Returns the probability that a hypergeometrically distributed random variable is equal to *x*.

**Arguments**
- *N* Population size.
- *k* The Number of items in the category of interest.
- *n* Sample size.
- *x* Count of interest.
- *r* Optional odds ratio.

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

**Description**
Returns the probability that a negative binomially distributed random variable is less than or equal to *k*, where the probability of success is *p* and the number of successes is *n*.

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

**Description**
Returns the probability that a negative binomially distributed random variable is equal to *k*, where the probability of success is *p* and the number of successes is *n*. 
Poisson Distribution($\lambda$, $k$)

**Description**
Returns the cumulative distribution function at $k$ for the Poisson distribution with mean $\lambda$.

Poisson Probability($\lambda$, $k$)

**Description**
Returns the probability that a Poisson distributed random variable with mean $\lambda$ is equal to $k$.

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.

---

**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(0)>, 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.

**Arguments**
- **Left**: An integer that measures pixels.
- **Right**: An integer that measures pixels.
- **Top**: An integer that measures pixels.
- **Bottom**: An integer that measures pixels.
Sides  An integer that maps to settings for the display box.

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

Notes
The formula for deriving the integer for Sides is: 1*top + 2*left + 4*bottom + 8*right +
16*hightlightcolor + 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.

Button Box("title", <<Set Icon("path"), "script" <<Set Icon Location("left,
"right")

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

Returns
The display box (button box).

Arguments

  title  A quoted string or a string variable.

  script  A quoted string or a reference to a string where the string is a valid JSL script.

  <<Set Icon("path")  Displays the image in the 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("right" or "left")  Allows the position of the icon on a button
to be either left or right of the text.

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).

Arguments

  title  A quoted string or a string variable.

  <<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.

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

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

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

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

Check Box(list, <script>)

Description

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

Returns

The display box (Check Box).

Arguments

list a list of quoted strings or a reference to a list of strings.
script an optional JSL script.

Messages

<<Get(n) Returns 1 if the check box item specified by n is selected, or 0 otherwise.
<<Set(n, 0|1) Sets the check box item specified by n as either selected (1) or cleared (0).
<<Get Selected Returns a list of strings that contain the names of the check box items that are selected.
<<Enable Item(n, 0|1) 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 List Box(<"all"|"character"|"numeric">, <width(pixels)>, <maxSelected(n)>, <nlines(n)>, <MaxItems(n)>, <MinItems(n)>, <On Change(expr)>,<script>)

Description

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

Returns

The display box (ColListBox).

Arguments

all | character | numeric an optional command that adds all columns of the current data table into the list. Omitting "all" results in an empty collistbox with the “optional” label. To display “optional character”, specify "character". To display “optional numeric”, specify "numeric".

width(pixels) an optional command that sets the width of the list box to pixels. Pixels is a number that measures pixels.

maxSelected(n) an optional command that sets whether only one item can be selected. For n>1, n is ignored.

nlines(n) an optional command that sets the length of the list box to n number of lines. n is an integer.

script an optional script.

MaxItems(n) An optional number that only allows n columns to be added to the list.

MinItems(n) An optional 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(expression) An optional command that evaluates the expression each time the contents of the column list box 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.

Messages

Set Tips ( {"Tip text 1", "Tip text 2", ...} ) Sets tool tips for items in the list box. A 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" ) 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.

**Note**
The `maxSelected` argument only affects whether one or more than one item can be selected. It does not enforce a limit greater than 1.

**Column Dialog(ColList("rolename"), specifications)**

**Description**
Draws a dialog box for column role assignments.

**Returns**
A list of commands that were sent and the button that was clicked.

**Arguments**
- `ColList` Specifies the name of at list one list to add variables to.
- `specifications` Any additional Dialog items (for example, `Max Col`, `Datatype`).

**Combo Box(list, <script>)**

**Description**
Constructs a display box to show a combo box with a menu.

**Returns**
The display box (`ComboBox`).

**Arguments**
- `list` a list of quoted strings or a reference to a list of strings.
- `script` an optional JSL script.

**Context Box(displayBox, ...)**

**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.

**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.

Dialog(contents)

Dialog is deprecated. Use New Window() with the Modal argument instead. See the Display Trees chapter in the Scripting Guide for details.

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.

Argument

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

width(pixels) an optional command that 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.

Global Box(global)

Description
Constructs a box for editing global value directly.

Graph()

See “Graph Box(properties, script)” on page 81.

Graph 3D Box(properties)

Description
Constructs a display box with 3-D content.

Returns
The display box.

Arguments
Properties can include: framesize(x, y), Xname("title"), Yname("title"), Zname("title").

Note
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(display box)**

**Description**

Returns a display box with the display box argument centered horizontally with respect to all other sibling display boxes.

**H List Box(<Align("center"|"bottom"), display box, <arguments>)**

**Description**

Creates a display box that contains other display boxes and displays them horizontally.

**Arguments**

Align Specify bottom or center 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 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, <arguments>)**

**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.

**Arguments**

display box Any number of display box arguments can be contained in the splitter box.

**Optional Arguments**

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.

<<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 Text Edit Box.

---

**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 | 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.

---

**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; if 0, does not display them.
- display boxes Any display box tree.
Journal Box("Journal Text")

Description

Constructs a display box that displays the quoted string *journal box*. We recommend that you do not generate the journal text by hand.

Line Seg(x, y, <Row States(dt | dt, [rows] | dt, {{rows}, ...} | {states} ) >, <Sizes(s)>))

Description

Creates a display seg of connected line segments. The optional third argument enables row state assignments from either a data table or independently.

Lineup Box(<NCol(n)>, <Spacing(pixels, <vspace>), display boxes, ...>)

Description

Constructs a display box to show an alignment of boxes in \( n \) columns.

ListBox({"item", ...}, <width(pixels)>, <maxSelected(n)>, <nLines(n)>, <script>)

Description

Creates a display box to show a list box of selection items. The argument can be a list of two-item lists containing the item name and a 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}, ...} | {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(matrix)

Description

Displays the *matrix* given in the usual array form.

Mouse Box(displayBoxArgs, messages)

Description

Returns a box that can make JSL callbacks for dragging and dropping, marking, or clicking and tracking mouse actions.
Arguments

displayBoxArgs  Specifies the object that the user interacts with, such as a Text Box or Button Box. See the Scripting Index in the Help menu for details.

New Image()
New Image(width, height)
New Image("filepath")
New Image(picture)
New Image(matrix)
New Image("rgb"|"r"|"rgba", {matrix, ...})

Description
  Creates a new image in a report. PNG, JPG, GIF, BMP, or TIF file types are supported.

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"  A filepath to an image.
  picture  A JSL picture object.
  matrix  The image as a matrix of JSL color pixels.
  "rgb"|"r"|"rgba", {matrix, ...}  Specify the channels (rgb, r, or rgba) and provide 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 Window("title", <arguments>, displayBox)

Description
  Makes a new window with the indicated title (a required argument) and a display box tree.

Additional Arguments
  <<Script(<"script">)  Creates a new script window. The optional quoted string script is placed inside the script window.
  <<Journal  Creates an empty journal.
  <<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.

**Examples**

- `Show Toolbars (0|1)` Show or hide the toolbar. The default value is 1. (Windows only.)
- `Show Menu (0|1)` Show or hide the menu bar. The default value is 1. (Windows only.)
- `Suppress AutoHide` Suppress or use the auto-hide feature for menus and toolbars. The default value is 1. (Windows only.)

**Notes**

`Dialog()` was deprecated in JMP 10. Use `New Window()` with the Modal argument instead. See the Display Trees chapter in the *Scripting Guide* for details about using `New Window()`.

---

### Number Col Box("title", numbers)

**Description**

Creates a column named `title` with numeric entries given in list or matrix form.

### Number Col Edit Box("title", numbers)

**Description**

Creates a column named `title` with numeric entries given in list or matrix form. The numbers can be edited.

### Number Edit Box(value)

**Description**

Creates an editable number box that initially contains the `value` argument.

**Returns**

The display box object.

**Argument**

`value` Any number.
Outline Box("title", display box, ...)  
Description  
Creates a new outline named \textit{title} 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 \textit{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  
\texttt{Open}  
Opens the directory that contains the picture.  
\texttt{picture}  
The pathname for the picture to include.  
\texttt{format}  
The graphic file format.  
Example  
\texttt{New Window( "Example",  
Picture Box( Open( "$SAMPLE\_IMAGES/pi.gif", gif ) ) );}

Plot Col Box("title", numbers)  
Description  
Returns a display box labeled with the quoted string \textit{title} to graph the \texttt{numbers}. The numbers can be either a list or a matrix.

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 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. **Note:** Pressing ALT and right-clicking the red triangle menu opens a window with check boxes for the commands. See the JMP Reports chapter in the *Using JMP* book for details.

---

**Radio Box({"item", ...}, <script>)**

**Description**

Constructs a display box to show a set of radio buttons. 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 (`RangeSliderBox`).

**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` by `y`-sized scene box for 3-D graphics.

---

**Script Box("script"), <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.
Arguments

- **script** An optional quoted string that appears in the script box.
- **width** An optional integer that sets the width of the script box.
- **height** An optional integer that sets the height of the script box.

**Scroll Box**(<size(h,v)>, <flexible(Boolean)>, display box, ...)

Description

Creates a display box that positions a larger child box using scroll bars.

Returns

A reference to the scroll box object.

Arguments

- **size(h,v)** Optional. The h and v arguments specify the size of the box in pixels.
- **flexible(Boolean)** Optional. True (1) sets the box to be resizable with the window. False (0) sets the box to remain the same size when the window is resized.
- **display box** Any number of display box arguments can be contained in the scroll box.

Note

You can send a scroll box object a message to set the background color:

```
<<Set Background Color( {R, G, B} | <color> )
```

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
sb << Set Scroll Position( 0, 20 ); // position the scrollbars on the scrollbar

Sheet Part("title", display box)

Description
Returns a display box containing the display box argument with the quoted string title as its title.

Slider Box(minValue, maxValue, variable, script, <set width(n)>, <rescale slider(min, max)>)

Description
Creates an interactive slider control.

Returns
The display box (SliderBox).

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 Any valid JSL commands that is run as the slider box is moved.
set width(n) specify the width of the slider box in pixels.
rescale slider(l, u) resets the max and min values for the slider box.

Notes
You can send Set Width and Rescale Slider as commands 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.

Arguments
- size(h,v) Optional. The h and v arguments specify the size of the box in pixels.
- color(color) Optional. Sets the color of the box to the JSL color argument.

String Col Box("title", {"string", ...})

Description
Creates column in the table containing the string items listed.

String Col Edit Box("title", {"string", ...})

Description
Creates column in the table containing the string items listed. The string boxes are editable.

Note
To retrieve the data, use this message:

data = obj << Get;

Tab Box(Tab Page Box("page title 1", <options>, contents of page 1), Tab Page Box("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.

Note
Certain messages you can send to Tab Page Box have been renamed, as follows:
- Set Title to Title
- Set Tip to Tip
- Set Icon to Icon
- Set Closeable to Closeable

Example
New Window( "Example",
Tab Box(
    t1 = Tab Page Box( "alpha", Panel Box( "panel", Text Box( "text" ) ) ),
    t2 = Tab Page Box( "beta", Popup Box( {"x", ex = 1, "y", ex = 2} ) ),
)
);

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.

Tree Box(<{rootnodes}>, <size(width, height)>, <MultiSelect>)  
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 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.

### 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**

```julia
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)

**Description**

Returns a display box with the display box argument centered vertically with respect to all other sibling display boxes.

### 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 Sheet Box(<<Hold(report), display boxes)

**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
- display box  Any number of display box arguments can be contained in the splitter box.

Notes
For details about the optional arguments, see “H Splitter Box(<size(h,v)>, display box, <arguments>)” on page 82.

Web Browser Box("url")

Description
Creates a display box that contains a web page. Available only on Windows.

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.
```javascript
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 Auto Stretching( 1, 1 ); // auto stretch horizontally and vertically
wb << Set Max Size( 10000, 10000 ); // maximum size in pixels
```

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 defined previously in the JSL script.
i an integer denoting which argument to return.
Notes
Arg Expr() was deprecated in a previous release of JMP. 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 Any JSL expression.
**Expr(x)**

**Description**
Returns the argument unevaluated (expression-quoting).

**Returns**
The argument, unevaluated.

**Argument**
- x  Any valid JSL expression.

**Extract Expr(expr, pattern)**

**Description**
Find `expr` matching `pattern`.

**Returns**
A pattern that matches the specified `pattern`.

**Arguments**
- expr  Any expression.
- pattern  Any pattern.

**Head(exprArg)**

**Head Expr(exprArg)**

**Description**
Returns the head of the evaluated expression, without its arguments.

**Note**
Head Expr() is deprecated. Use Head() instead.

**Head Name(expr)**

**Head Name Expr(expr)**

**Description**
Returns the head of the evaluated expression as a string.

**Note**
Head Name Expr() is deprecated. Use Head Name() instead.
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.

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.

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 path before closing or to close the file without saving it.

Close All(type, <invisible|private>, <noSave|save>)

Description
Closes all open resources of type.

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.
invisible Optional. Specifies whether to close all invisible data tables.
private Optional. Specifies whether to close all private data tables.
noSave or Save An optional argument that specifies whether to save the specified types of windows before closing or to close without saving.

Close Log(Boolean)

Description
Closes the log window.

Convert File Path(path, <"absolute"|"relative">, <"posix"|"windows">, <base(path)>)

Description
Converts a file path according to the arguments.

Returns
The converted path.
Arguments

path  A pathname that can be either Windows or POSIX.

absolute|relative Optional quoted string, specifies whether the returned pathname is in absolute or relative terms. The default value is absolute.

posix|windows Optional quoted string, specifies whether the returned pathname is in Windows or POSIX style. The default is POSIX.

base(path) Optional, specifies the base pathname, useful if relative is specified. The default is the default directory.

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.

Arguments

from path  Specifies the directory containing the files to copy to the new directory.

to path  Specifies the path where the new directory should be created and to which the files are copied.

<recursive(Boolean)> Indicates whether to copy the from path subdirectory structure to the to path.

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.

to path  Specifies the path and file name for the new file.

Create Database Connection( (string, <DriverPrompt(1)> ) | "Connect Dialog")

Description
Creates a connection to the specified database or prompts the user to provide database log in information.
File Functions JSL Syntax Reference

Returns
A handle to the database connection.

Arguments
string The server connection string that contains information such as the data source name and user name.
Driver Prompt An optional Boolean argument that enables the ODBC driver to prompt for the connection information if necessary.
"Connect Dialog" A string that opens the Select Data Source window, from which the user selects the database.

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 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 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 directory or path and file name for the query.
Delete Directory("path")

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.

Delete File("path")

Description
Deletes the specified file.

Returns
Returns 1 if the file is deleted; otherwise, returns 0.

Arguments
path Specifies the path and file name for deletion.

Directory Exists("path")

Description
Verifies the specified directory exists.

Returns
Returns 1 if the directory exists; otherwise returns 0.

Arguments
path Specifies the 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 path and file name for verification.

Files In Directory(path, <"recursive">)

Description
Returns a list of filenames in the path given.
Returns
List of filenames. If "recursive" is not specified, directory names are included in the list.

Arguments
- **path** A valid pathname.
- **recursive** An optional quoted keyword that causes all folders in the path (and all folders that they contain, and so on) to be searched for files.

---

### Get Current Directory()

**Description**
Retrieves the current directory, which is used in the Open File window. **Note:** The operating system can change the current directory outside of JMP. The function is deprecated. On Macintosh and Windows, the initial current directory is the user’s Documents folder. Previously, Windows returned the folder that contained jmp.exe or the working folder set by the operating system when JMP is launched.

- If JMP was opened using the JMP shortcut, the function returns the JMP Installation directory. For example, on Windows, the current directory is "/C:/Program Files/SAS/JMPPRO/13/" or "/C:/Program Files/SAS/JMP/13/".
- If JMP was opened by double-clicking a JMP file (.jmp, .jsl, .jrn, and so on), the command returns the path of the opened file.
- If the current directory was configured using the Set Current Directory() function, JMP returns the specified path.

See “Set Current Directory("path")” on page 112.

**Returns**
The absolute pathname as a string.

**Arguments**
none

---

### 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")” on page 113.

**Returns**
The absolute pathname as a string.
**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.

**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 or list of paths})” on page 113.

**Returns**
A list of pathnames as strings.

**Get Path Variable("name")**

**Description**
Retrieves the value of name, a path variable.

**Returns**
The absolute pathname as a string.

**Argument**
`name`  A quoted string that contains a path variable. (Examples: SAMPLE_DATA, SAMPLE_SCRIPTS)

**Is Directory(path)**

**Description**
Returns 1 if the path argument is a directory and 0 otherwise.

**Is Directory Writable(path)**

**Description**
Returns 1 if the directory specified in the path argument is writable and 0 otherwise.
Is File(path)

Description
Returns 1 if the path argument is a file and 0 otherwise.

Is File Writable(path)

Description
Returns 1 if the file specified in the path argument is writable and 0 otherwise.

Is Log Open()

Description
Returns a Boolean value that indicates whether the log window is open.

Last Modification Date("path")

Description
Returns the last modification date of the specified file or directory.

Returns
Last modification date.

Arguments
path Specifies the directory or file name.

Load Text File(path, <arguments>)

Description
Reads the text file at path into a JSL variable.

Returns
A string.

Arguments
path A pathname that points to a text file. The path can be a URL.
Charset Optional argument that determines the character set. Arguments include the following:
  – "best guess" attempts to detect the character set.
  – force("throw"|"alert"|"silent") is an optional argument that determines what happens if the character set cannot be detected.
Line Separator("character") Optional argument that specifies the end-of-line character. For example, "\n" specifies a line feed character. "\t" specifies a tab.
XMLParse Optional argument that converts an XML file into JSL.
SASODSXML  Optional argument that parses the text file as SAS ODS default XML.

BLOB(<arguments>) Optional argument that returns data from the file as a blob rather than a 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(0|1) 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 specified path to another specified 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 specified path to another specified path with the same or different file name.

Returns
Returns 1 if the file is moved; otherwise returns 0.

Arguments
from path  Specifies the path and file name for relocation.
to path  Specifies the 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 Macintosh, 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. Refer to the examples in the JMP Scripting Index for details about which arguments apply to specific file types.

Arguments

**Charset("option")** 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)**: Indicates whether the specified column contains character or numeric data.
- **colWidth(n)**: Indicates the integer width of the specified column.

**Columns(<arguments>)** For ESRI shapefiles (.shp), this argument and its settings indicate the following:
- **Shape=numeric(n)**: Indicates the column number in the imported ESRI shapefile that contains the shape number.
- **Part=numeric(n)**: Indicates the column number in the imported ESRI shapefile that contains the part number.
- **X=numeric(n)**: Indicates the column number in the imported ESRI shapefile that contains the decimal degree for the longitude (range of ±180°).
- **Y=numeric(n)**: Indicates the column number in the imported ESRI shapefile that contains the decimal degree for the latitude (range of ±90°).

**Compress Allow List Check(Boolean)** Indicates that JMP can compress data table created from the imported text file.

**Compress Character Columns(Boolean)** Indicates that JMP should compress data table columns that contain character data from the imported text file.

**Compress Numeric Columns(Boolean)** Indicates that JMP should compress data table columns that contain numeric data from the imported text file.

**Concatenate Worksheets(Boolean)** Indicates that JMP should combine the imported Excel worksheets into one data table.
Create Concatenation Column(Boolean) Indicates 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 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 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.

File Type An optional string that specifies the type of file that you are opening (for example, xls, bmp, jpg, or gif). If you do not specify this string, the file opens in the default program for the file type.

Note: The path argument should be used for a zip archive. The extension (.zip) is not required. See “Zip Archives” on page 382 in the “JSL Messages” chapter 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 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 The Column Info Window chapter in the Using JMP book for details about expression columns.

Ignore Columns( "col", ... ) Indicates the 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 the files: data table, JMP file, external, text, Excel, SAS, ESRI shapefile, or HTML. The data table appears only in the JMP Home Window and the Window menu.

Labels(Boolean) Indicates 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 password for a password-protected Microsoft Excel .xls or SAS files to avoid entering it manually. The password is not encrypted. (Password-protected Microsoft Excel .xlsx files cannot be 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. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful because the data requires a lot of memory.

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( "col", ... ) Indicates the 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) Indicates 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 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) - Indicates 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) - Indicates the Excel file contains spanned columns that 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.

Year Rule | Two digit year rule
("1900-1999"|"1910-2009"|"1920-2019"|"1930-2029"|"1940-2039"|
Indicates the year format used in the imported text file. For example, if the earliest date is 1979, use “1970-2069”.

Open Database("connectInfo" "sqlStatement", "outputTableName")

Description
Opens the database indicated by connectInfo with the sqlStatement, returning a data table named outputTableName.

Pick Directory(<"prompt">, <path>, <Show Files>)

Description
Prompts the user for a directory, returning the directory path as a string.

Returns
The path for the directory that the user selects.

Arguments

prompt An optional quoted string. If provided, that string appears at the top of the Browse window on Windows.

path An optional quoted string that specifies the initial directory that appears in the Pick Directory window.

Show Files (Boolean) Specify 1 to show files in the Pick Directory window, or zero to hide files. The default is zero.

Pick File(<"prompt">, <"initial directory">, {{filter list}>}, <first filter>, <save flag>, <"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.
Arguments

prompt  An optional quoted string. If provided, that string appears at the top of the Open window.

initial directory  An optional 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  An optional list of quoted strings that define the filetypes to show in the Open window. See the following example for syntax.

first filter  An optional 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  An optional 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 name of the file that appears in the window by default.

multiple  An optional argument that lets the user select multiple files if the save flag 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  Specifies the path and old directory name.
new name  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 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 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)

Description
Saves the JSL variable text into the file specified by path.

Set Current Directory("path")

Description
Sets the current directory for Open File operations. Note: The operating system can change the current directory outside of JMP. The function is deprecated. Setting the current directory saves or returns the new value for backwards compatibility though the value is not used.

To avoid having to set the current directory for each file operation, define a filepath string and concatenate it with the filename. See the Path Variables section in the Scripting Guide for details.

See “Get Current Directory()” on page 102.
Set Default Directory("path")

**Description**
Sets the default directory, which is used for resolving relative paths.

See “Get Default Directory()” on page 102.

Set File Search Path({path or 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.

See “Get File Search Path()” on page 103.

**Example**
```
Set File Search Path( {"C:/JMP/13/source", "C:/Program Files/SAS/JMPPRO/13/Samples"} );
```

Set Path Variable("name")

**Description**
Sets the path stored in the variable.

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.

**Arguments**
- *path*  Quoted string that contains the full path to the .xml or .sss file.
- *Invisible*  Optional. 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 Columns*  Optional Boolean. Converts the label names to column headings. The default value is true.

**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.

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.
- factor An optional number that is the rate at which the balance declines. The default value is 2.

Note
This function is equivalent to the Excel function DDB.

Future Value(rate, nper, pmt, <pv>, <type>)

Description
Returns the future value of an investment that is based on periodic, constant payments and a constant interest rate.

Arguments
- rate The interest rate.
- nper The number of periods.
- pmt The constant payment.
- pv An optional number that is the present value. The default value is 0.
- type An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Note
This function is equivalent to the Excel function FV.

Interest Payment(rate, per, nper, pv, <fv>, <type>)

Description
Returns the interest payment for a given period for an investment that is based on periodic, constant payments and a constant interest rate.
Arguments
  rate  The interest rate.
  per  The period for which you want the interest.
  nper  The total number of periods.
  pv  The present value.
  fv  An optional number that is the future value. The default value is 0.
  type  An optional switch. 0 specifies end-of-period payments, and 1 specifies
       beginning-of-period payments. The default value is 0.

Note
This function is equivalent to the Excel function IPMT.

Interest Rate(nper, pmt, pv, <fv>, <type>, <guess>)

Description
Returns the interest rate per period of an annuity.

Arguments
  nper  The total number of periods.
  pmt  The constant payment.
  pv  The present value.
  fv  An optional number that is the future value. The default value is 0.
  type  An optional switch. 0 specifies end-of-period payments, and 1 specifies
        beginning-of-period payments. The default value is 0.
  guess  An optional number that is what you think the rate will be. The default value is 0.1
        (10%).

Note
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  The number that you think is near the result. The default value is 0.1 (10%).

Note
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.

Note
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.

Note
This function is equivalent to the Excel function NPV.

Number of Periods(rate, pmt, pv, <fv>, <rate>)

Description
Returns the number of periods for an investment that is based on periodic, constant payments and a constant interest rate.

Arguments
- rate The interest rate.
- pmt The constant payment.
- pv The present value.
- fv An optional number that is the future value. The default value is 0.
type  An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Note
This function is equivalent to the Excel function NPER.

Payment(rate, nper, pv, <fv>, <type>)

Description
Returns the payment for a loan that is based on constant payments and a constant interest rate.

Arguments
  rate  The interest rate.
  nper  The total number of periods.
  pv    The present value.
  fv    An optional number that is the future value. The default value is 0.
  type  An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Note
This function is equivalent to the Excel function PMT.

Present Value(rate, nper, pmt, <fv>, <type>)

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    An optional number that is the future value. The default value is 0.
  type  An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Note
This function is equivalent to the Excel function PV.

Principal Payment(rate, per, nper, pv, <fv>, <type>)

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.
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.
fv An optional number that is the future value. The default value is 0.
type An optional switch. 0 specifies end-of-period payments, and 1 specifies beginning-of-period payments. The default value is 0.

Note
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.

Note
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.

Note
This function is equivalent to the Excel function SYD.
Graphic 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.

**Arguments**
- **name** The name of the color theme.
- **flags** An optional 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
- **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.

**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.

```julia
Add Color Theme("Blue to Purple", Continuous, Sequential, colors=[RGB(0,0,255), RGB(0,0,0), RGB(255,0,0)], positions=[0, 0.5, 1])
```
{"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("name").

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.

Arguments
pixellength Optional: specifies the length of the arrowhead in pixels.
{x1, y1}, {x2, y2} Two lists of two numbers that each specify a point in the graph.

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 string to a matrix.

Returns
A matrix.

Arguments
path A 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.

Arguments
{x, y} A number that describes a point in the graph
radius A number that describes 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) A number that describes the length of the circle’s radius in pixels. If the vertical axis is resized, the circle is not resized.
"fill" Optional string. Indicates that all circles defined in the function are filled with the current fill color. If "fill" is omitted, the circle is empty.

Note
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:

```javascript
graphbox(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:
```julia
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:
```julia
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, <zColors>)

Description
Draws contours given a grid of values.

Returns
None.

Arguments
- xVector The n values that describe zGridMatrix.
- yVector The m values that describe zGridMatrix.
- zGridMatrix An n x m matrix of values on some surface.
- zContour Optional: Definition of values for the contour lines.
- zColors Optional: Definition of colors to use for the contour lines.

Contour Function(expr, xName, yName, z, < <<XGrid(min, max, incr)>, < <<YGrid(min, max, incr)>, < <<zColor(color)>, < <<zLabeled>, < <<Filled>, < <<FillBetween>, < <<Ternary>, < <<Transparency(alpha|vector))

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
- expr Any expression. For example, Sine(y)+Cosine(x).
- xName, yName Values to use in the expression.
- z A z-value or a matrix of z-values.

Optional Arguments
- <<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  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.

Arguments

xMatrix  A matrix of x-coordinates.
yMatrix  A matrix of y-coordinates.
dragScript  Any valid JSL script; it is run at drag.
mouseUpScript  Any valid JSL script; it 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; it is run at drag.
mouseUpScript  Any valid JSL script; it 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.

Arguments

xMatrix  A matrix of x-coordinates.
yMatrix  A matrix of y-coordinates.
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.

Arguments
xMatrix A matrix of 2 x-coordinates.
yMatrix A matrix of 2 y-coordinates.
dragScript Any valid JSL script; it is run at drag.
mouseupScript Any valid JSL script; it is run at mouseup.

Note
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, and 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, "text", <dragScript>, <mouseupScript>)

Description
Draws the text (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.
text A quoted string to be drawn in the graph.
dragScript Any valid JSL script; it is run at drag.
mouseupScript Any valid JSL script; it is run at mouseup.
Fill Color(n)

Description
Sets the color used for filling solid areas.

Returns
None.

Argument
n Index for a color or a quoted color name.

Fill Pattern()

Description
Sets the pattern for filled areas. See the Scripting Graphs chapter in the Scripting Guide for examples.

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.

Example
Gradient Function(Log(a * a + b * b),
a, b, [2 10],
Z Color([4, 6]));

Zexpr is a function in terms of the two following variables (a and b), whose values range from zlow to zhigh (2 to 10). Zcolor defines the two colors that are blended together (4 is green, 6 is orange).

H Line(<x1, x2>, 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(a, b, dragScript, mouseupScript)

Description
Places draggable marker at coordinates given by a, b. The first script is executed at drag and the second at mouseup.

Heat Color(n, <"color theme">)

Description
Returns the JMP color that corresponds to n in the color "theme".

Returns
An integer that is a JMP color.

Arguments
n A value between 0 and 1.
theme Any 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, path)

Description
Determines if 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.
path Either a matrix or a string describing a path.
In Polygon(x, y, xx, yy)

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 xx and yy vector arguments.

The vector arguments (xx, yy) can also be combined into a 2-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, <n>, <"Color Theme">)

Description
Assigns a JMP color to categorical data in a graphic.

Returns
An integer that is a JMP color.

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.

"Color Theme" A color theme from the Value Color list of the Column Properties window. If not specified, the JMP Default color theme is applied.

Note
When the second argument is a character string and not n, then the second argument determines the color theme.

Line({x1, y1}, {x2, y2}, ...), <<ValueSpace(0|1)
Line([x1, x2, ...], [y1, y2, ...]), <<ValueSpace(0|1)

Description
Draws a line between points.

Arguments
{x1, y1}, {x2, y2} or [x1, x2, ...], [y1, y2, ...] Can be any number of lists of two points, separated by commas; or a matrix of x’s and a matrix of y’s.

<<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(n)

Description
Sets the line style used to draw the graph.

Argument
n  Can be either a style name or the style’s number:
  – 0 or Solid
  – 1 or Dotted
  – 2 or Dashed
  – 3 or DashDot
  – 4 or DashDotDot

Marker(<markerState>, {x1, y1}, {x2, y2}, ...)
Marker(<markerState>, [x1, x2, ...], [y1, y2, ...])

Description
Draws one or more markers at the points described either by lists or matrices. The optional <markerState> 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 at mouseup.

Normal Contour(prob, meanMatrix, stdMatrix, corrMatrix, <colorsMatrix>, <fill=x>)

Description
Draws normal probability contours for k populations and two variables.

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.
colorsMatrix  Optional. 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  Optional. Specifies the amount of transparency for the contour fill color.

\[ \text{Oval}(x_1, y_1, x_2, y_2, <\text{fill}>) \]

\[ \text{Oval}([x_1, y_1], [x_2, y_2], <\text{fill}>) \]

**Description**

Draws an oval inside the rectangle whose diagonal has the coordinates \((x_1, y_1)\) and \((x_2, y_2)\). 

*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(path, <fill>)**

**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.

**Argument**

- **path**  Can be either an Nx3 matrix or a string that contains SVG syntax.
- **fill**  An optional, Boolean argument that specifies whether a line is drawn (0) or the path is filled (1). The default value is 0.

**Note**

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 string.

**Returns**

A string.

**Argument**

- **path**  An Nx3 path matrix.

**Note**

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.

Pie(x1, y1, x2, y2, 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(x, y)

**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 commands.

Pixel Move To(x, y)

**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 commands.

Polygon({x1, y1}, {x2, y2}, ...)

**Description**
Draws a filled polygon defined by the listed points.
Rect(x1, y1, x2, y2, <fill>)
Rect({x1, y1}, {x2, y2}, <fill>)

**Description**
Draws a rectangle whose diagonal has the coordinates (x1, y1) and (x2, y2). 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.

---

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 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 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).

Note
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(expr, symbol, <Min(min), Max(max), Fill(value), Inc(bound), Show Details(n)>)

Description
Draws a plot of the function as the symbol 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, and 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.

Note: Either inc() or steps() is needed if the default granularity misses details.

Y Function(expr, symbol, <Min(min), Max(max), Fill(value), Inc(bound), Show Details(n)>)

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).

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 string expressions into one string, with each item separated by a delimiter. The delimiter is a blank, if unspecified.

Returns
The concatenated string.

Arguments
- **string** any string
- **delimiter** an optional string that is placed between each item. The delimiter can be more than one character long.

Example
```javascript
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.
Arguments

source A string, list, expression, or associative array.

item or key Any value to be placed within source. For an associative array, key might or might not 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.

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 string, list, display box, expression, or associative array.

item or key Any value to be placed within source. For an associative array, key might or might not 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.

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 the matrix, the number of elements in
the matrix is returned. For the namespace, the number of items in the namespace 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 string, list, 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 string, list, 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 string, list, or expression.

Reverse Into(source)

Description
Reverses the order of elements or terms in source in place.

Argument
source  A string, list, 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 string, list, 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 string, list, 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 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 string, list, or expression in which to perform the substitution.
substring, listItem, findExpr A string, list item, or expression to be found in the source string, list, or expression.
replacementString, replacementItem, replacementExpr A 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 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 string, list, or expression in which to perform the substitution.
- `substring`, `listItem`, `findExpr` A string, list item, or expression to be found in the source string, list, or expression.
- `replacementString`, `replacementItem`, `replacementExpr` A string, list item, or expression to replace the found string, list item, or expression.

**Words("text", <"delimiters">)**

**Description**

Extracts the words from `text` according to the delimiters given. The default delimiter is space. If you include a second argument, any and all characters in that argument are taken to be delimiters.

**Examples**

```plaintext
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.

See the Extending JMP chapter in the *Scripting Guide* for details on working with MATLAB.
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, 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 \( z = x \times y \), and then gets the MATLAB variable z and returns it to JMP.

```matlab
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.

```matlab
// 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>
<thead>
<tr>
<th>MATLAB Data Type</th>
<th>JMP Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Numeric</td>
</tr>
<tr>
<td>Logical</td>
<td>Numeric (0</td>
</tr>
</tbody>
</table>
### 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", and "tif".

### 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.
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
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
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**

```javascript
// 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>
<thead>
<tr>
<th>MATLAB Data Type</th>
<th>JMP Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Numeric</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>Double Matrix</td>
<td>Matrix</td>
</tr>
<tr>
<td>Structure</td>
<td>Data Table</td>
</tr>
</tbody>
</table>

**Example**

```javascript
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 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, 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.

Example
The following example creates two vectors of random points and plots them as x and y variables.
MATLAB Init();
m = MATLAB Submit("/
    x = rand(5);
    fprintf('%f/n', x);
    y = rand(5);
    fprintf('%f/n', x);
    z = plot(x, y);
]/*");

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, 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.
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.

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×n matrix A, then after calling cholUpdate L is replaced with the Cholesky root of A+V*C*V' where C is an m×m symmetric matrix and V is an n×m matrix.

Cholesky(A)  
Description
Finds the lower Cholesky root (L) of a positive semi-definite matrix, L*L' = A.
Returns
L (the Cholesky root).

Arguments
A a symmetric matrix.

Correlation(matrix)

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.

Notes
Rows are discarded if they contain missing values.
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>)

Description
Creates design columns for 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 argument that is a list or matrix specifying 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.

Note
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 Nom(vector, <levelsList | <<levels>)

DesignF(vector, < <<levels >>)

Description
A version of Design for making full-rank versions of design matrices for nominal effects.
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 argument that is a list or matrix specifying 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.

Note
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>)

Description
A version of Design for making full-rank versions of design matrices for ordinal effects.

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 argument that is a list or matrix specifying 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.

Note
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];
```
\[
\begin{align*}
\text{Design Ord}(0, [. \ 0 \ 1]) &= [1 \ 0]; \\
\text{Design Ord}(1, [. \ 0 \ 1]) &= [1 \ 1]; \\
\text{Design Ord}([0 \ 0 \ 1 \ . \ 1], [. \ 0 \ 1]) &= [1 \ 0, 1 \ 0, 1 \ 1, 0 \ 0, 1 \ 1]; \\
\text{Design Ord}({0, 0, 1, ., 1}, [. \ 0 \ 1]) &= [1 \ 0, 1 \ 0, 1 \ 1, 0 \ 0, 1 \ 1];
\end{align*}
\]

---

DesignF()

See “Design Nom(vector, <levelsList | <<levels>)” on page 149.

---

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.
JSL Functions

Matrix Functions

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

Description
Element-by-element division of two matrices.

Returns
The resulting matrix.

Arguments
A, B Two matrices.

E Mult(A, B)
A*B

Description
Element-by-element multiplication of two matrices.

Returns
The resulting matrix.

Arguments
A, B Two matrices.

Eigen(A)

Description
Eigenvalue decomposition.

Returns
A list \{M, E\} such that \(E \cdot \text{Diag}(M) \cdot E = A'\).

Argument
A A symmetric matrix.
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>)

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)” on page 154.

Inv Update(A, X, 1|-1)

Description
Efficiently update an $X'X$ 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. 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\}. Position is a point that is described as a row vector for the coordinate of a row, or as the number of a row.
<<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.

Note
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(X, y, messages)

Description
Computes optionally weighted least squares estimates.

Returns
A list that contains the matrix Beta=Inverse(X'X')X'y and the estimated variance matrix of Beta.

Arguments
<<weights(optional weight vector) Specifies a vector of weights to perform weighted least squares.
Specifies the default Sweep method (much more computationally efficient) or a generalized inverse (GInv) method for solving the normal equations (more numerically stable).

---

**Loc(A)**

**Loc(list, item)**

**Description**

Creates 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 \( list \).

**Returns**

The new matrix.

**Argument**

- \( A \) a matrix
- \( list \) a list
- \( item \) the item to be found within the list

---

**Loc Max(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 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.
JSL Functions

Chapter 2
JSL Syntax Reference

JSL Syntax Reference Matrix Functions

Returns
The new matrix or list.

Loc Sorted(A, B)

Description
Creates a matrix of subscript positions where the values of A have values less than or equal to the values in B. A must be a matrix sorted in ascending order.

Returns
The new matrix.

Argument
A, B matrices

Matrix({{x11, x12, ..., x1m}, {x21, x22, ..., 2m}, {...}, {xn1, xn2, ..., xnm}})

Description
Constructs an n-by-m matrix from a list of n lists of m row values or from the number of rows and columns.

Returns
The matrix.

Arguments
A list of lists in which each list forms a row with the specified values.

Example
mymatrix = Matrix({{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}});
[ 1 2 3,
  4 5 6,
  7 8 9,
  10 11 12]

Equivalent Expression
[x11 x12 ... x1m,
 ..., 
 xn1 xn2 ... xnm ]

Matrix Mult(A, B)
C=A*B, ...

Description
Matrix multiplication.
Arguments
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).

Note
Matrix Mult() allows only two arguments, while using the * operator enables you to
multiply several matrices.

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 character strings is acceptable.
Arguments
Specify either a list or a matrix.

Multivariate Normal Impute(yVec, meanYvec, symCovMat)
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.

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.

Print Matrix(M, <named arguments>)

Description
Returns a 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.

Named Arguments
Note that the following named arguments are all optional.
<<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).
Rank Index(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 strings are supported in addition to matrices.

Rank(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 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 strings are supported in addition to matrices.

Shape(A, nrow, <ncol>)

Description
Reshapes the matrix A across rows to the specified dimensions. Each value from the matrix A is placed row-by-row into the re-shaped matrix.

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.

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 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,}
Chapter 2
JSL Functions

Matrix Functions

161

JSL Syntax Reference Matrix Functions

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]

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.

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 $\text{coef}$ 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:

$$\text{result} = a[j] + xx \cdot (b[j] + xx \cdot (c[j] + xx \cdot d[j]))$$

The following line is equivalent:

$$\text{result} = a[j] + b[j] \cdot xx + c[j] \cdot xx ^ 2 + d[j] \cdot xx ^ 3$$

**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 Min(matrix)

Description
Returns a row vector containing the minimum 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)

Description
Returns a list of the rotated matrix and the orthogonal rotation matrix.

Vec Diag(A)

Description
Creates a vector from the diagonals of a square matrix A.

Returns
The new matrix.

Arguments
A square matrix.

Note
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')

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.
Note
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
Invert Expr(expr, name)

Description
Attempts to unfold expr around name.

Mod()

See “Modulo(number, divisor)” on page 166

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.
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), \ldots\}, messages)

Description
Finds the values for the $x$ arguments, specified as a list, that maximize the $expr$ expression with optional linear constraints. You must specify lower and upper bounds in parentheses for each argument.

In the following messages, $A$ is a matrix of coefficients. $x = [x_1, x_2, \ldots]$ 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 ($A^x \leq b$).
<<Greater Than EQ({A, b}) Sets the constraint to greater than or equal to the specified values ($A^x \geq b$).
<<Equal To({A, b}) Sets the constraint as equal to the specified values ($A^x = b$).
<<Starting Values([x1Start, x2Start, \ldots]) 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 $10^5$. 
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 specify lower and upper bounds in parentheses for each argument.

In the following messages, A is a matrix of coefficients. \( x = [x_1, x_2, \ldots] \) 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.

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.
ng e 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.

Note
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.
<<Tolerance(p) p sets the tolerance for the convergence criterion. The default tolerance is 10^-8.
<<Show Details("true") Returns a list with the final values for the objective value, number of iterations, gradient, and Hessian. Shows the step-by-step results of the optimizer in the log.
Minimize(expr, \{x1(low1, up1), x2(low2, up2), \ldots\}, 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.
<<Tolerance(p) \(p\) sets the tolerance for the convergence criterion. The default tolerance is \(10^{-8}\).
<<Show Details("true") Returns a list with the final values for the objective value, number of iterations, gradient, and Hessian. Shows the step-by-step results of the optimizer in the log.

Probability Functions

Beta Density(x, alpha, beta, <theta>, <sigma>)

Description
Calculates the beta probability density function (pdf).

Returns
The density function at quantile \(x\) for the beta distribution for the given arguments.

Arguments
\(x\) A quantile between theta and theta + sigma. Theta’s default value is 0. Sigma’s default value is 1.
alpha, beta Shape parameters that must both be greater than 0.
theta optional threshold. The allowable range is \(-\infty < \theta < \infty\). The default is 0.
sigma optional scale parameter, which must be greater than 0. The default is 1.

Notes
Beta Density() is useful for modeling the probabilistic behavior of random variables such as proportions constrained to fall in the interval [0, 1].
Beta Distribution(x, alpha, beta, <theta>, <sigma>)

Description
Calculates the cumulative distribution function for the beta distribution. The Beta Distribution() function is the inverse of the Beta Quantile() function.

Returns
Returns the cumulative distribution function at quantile x for the beta distribution with shape arguments alpha and beta.

Arguments
- x A quantile between theta and theta + sigma.
- alpha, beta Shape parameters that must both be greater than 0.
- theta optional threshold. The allowable range is $-\infty < \theta < \infty$. The default is 0.
- sigma optional scale parameter, which must be greater than 0. The default is 1.

Beta Quantile(p, alpha, beta, <theta>, <sigma>)

Description
Calculates the requested quantile for the beta distribution. The Beta Quantile() function is the inverse of the Beta Distribution() function.

Returns
Returns the pth quantile from the beta distribution with shape arguments alpha and beta.

Arguments
- p The probability of the quantile desired; p must be between 0 and 1.
- alpha, beta Shape parameters that must both be greater than 0.
- theta optional threshold. The allowable range is $-\infty < \theta < \infty$. The default is 0.
- sigma optional scale parameter, which must be greater than 0. The default is 1.

Cauchy Density(q, <center>, <scale>)

Description
Returns the density at q of a Cauchy distribution with center mu and scale sigma.

Cauchy Distribution(q, <center>, <scale>)

Description
Returns the probability that a Cauchy distributed random variable is less than q.
Cauchy Quantile(p, <center>, <scale>)

Description
Returns the quantile from a Cauchy distribution, the value for which the probability is \( p \) that a random value would be lower.

ChiSquare Density(q, df, <center>)

Description
The chi-square density at \( q \) of the chi-square with \( df \) degrees of freedom and optional non-centrality parameter \( center \).

Returns
The chi-square density.

Arguments
\( q \) quantile
\( df \) degrees of freedom.
\( center \) non-centrality parameter

ChiSquare Distribution(q, df, <center>)

Description
Returns cumulative distribution function at quantile \( x \) for chi-square with \( df \) degrees of freedom centered at \( center \).

ChiSquare Log CDistribution(x, df, <nc>)

Description
Returns 1 - log (value) of the distribution function at quantile \( x \) for the chi-square distribution.

ChiSquare Log Density(x, df, <nc>)

Description
Returns the log of the value of the density function at quantile \( x \) for the chi-square distribution.

ChiSquare Log Distribution(x, df, <nc>)

Description
Returns the log of the value of the distribution function at quantile \( x \) for the chi-square distribution.
ChiSquare Noncentrality(x, df, prob)

Description
Solves the noncentrality such that prob=ChiSquare Distribution (x, df, nc)

ChiSquare Quantile(q, df, <center>)

Description
Returns the \( p \)th quantile from the chi-square distribution with \( df \) degrees of freedom, centered at center.

Dunnett P Value(q, nTrt, dfe, lambdaVec)

Description
Returns the \( p \)-value from Dunnett’s multiple comparisons test.

Returns
A number that is the \( p \)-value.

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 (.), the parameters are set to \( 1/Sqrt(2) \).

Dunnett Quantile(1-alpha, nTrt, dfe, lambdaVec)

Description
Returns quantile needed in from Dunnett’s multiple comparisons test.

Returns
A number that is the quantile.

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 (.), the parameters are set to \( 1/Sqrt(2) \).
**F Density**(x, dfnum, dfden, <center>)

Description

Returns the F density at x for the F distribution with numerator and denominator degrees of freedom **dfnum** and **dfden**, with optional noncentrality parameter **center**.

**F Distribution**(x, dfnum, dfden, <center>)

Description

Returns cumulative distribution function at quantile x for F distribution with numerator and denominator degrees of freedom **dfnum** and **dfden** and noncentrality parameter **center**.

**F Log CDistribution**(x, dfn, dfd, <nc>)

Description

Returns 1 - log (value) of the normal distribution function at quantile x for the F distribution.

**F Log Density**(x, dfn, dfd, <nc>)

Description

Returns the log of the value of the density function at quantile x for the F distribution.

**F Log Distribution**(x, dfn, dfd, <nc>)

Description

Returns the log of the value of the distribution function at quantile x for the F distribution.

**F Noncentrality**(x, ndf, ddf, prob)

Description

Solves the noncentrality such that prob=F Distribution (x, ndf, ddf, nc)

**F Power**(alpha, dfh, dfm, d, n)

Description

Calculates the power from a given situation involving an F test or a t test.

**F Quantile**(x, dfnum, dfden, <center>)

Description

Returns the pth quantile from the F distribution with numerator and denominator degrees of freedom **dfnum** and **dfden** and noncentrality parameter **center**.
**F Sample Size** (alpha, dfh, dfm, d, power)

**Description**
Calculates the sample size from a given situation involving an \( F \) test or a \( t \) test.

**Frechet Density** (x, mu, sigma)

**Description**
Returns the density at \( x \) of a Frechet distribution with location \( mu \) and scale \( sigma \).

**Arguments**
- \( x \) A number.
- \( mu \) A location.
- \( sigma \) The scale.

**Frechet Distribution** (x, mu, sigma)

**Description**
Returns the probability that the Fréchet distribution with location \( mu \) and scale \( sigma \) is less than \( x \).

**Arguments**
- \( x \) A number.
- \( mu \) A location.
- \( sigma \) The scale.

**Frechet Quantile** (p, mu, sigma)

**Description**
Returns the quantile associated with a cumulative probability \( p \) for a Fréchet distribution with location \( mu \) and scale \( sigma \).

**Arguments**
- \( p \) The probability of the quantile desired; \( p \) must be between 0 and 1.
- \( mu \) A location.
- \( sigma \) The scale.

**Gamma Density** (q, alpha, <scale>, <threshold>)

**Description**
Calculates the density at \( q \) of a Gamma probability distribution.
Returns
The density function at quantile \( q \) for the gamma density distribution for the given arguments.

Arguments
- \( q \) A quantile.
- \( \alpha \) Shape parameters that must be greater than 1.
- \( \text{scale} \) Optional scale, which must be greater than 0. The default is 1.
- \( \text{threshold} \) Optional threshold parameter. The allowable range is \( -\infty < \theta < \infty \). The default is 0.

\textbf{Gamma Distribution}(x, <shape, scale, threshold>)
\textbf{IGamma}(x, <shape, scale, threshold>)

Description
Returns cumulative distribution function at quantile \( x \) for the gamma distribution with \textit{shape}, \textit{scale}, and \textit{threshold} given.

\textbf{Gamma Log CDistribution}(x, \alpha, <scale = 1>, <threshold = 0>)

Description
Same as \( \log(1 - \text{Gamma Distribution}(x, \alpha)) \) except that it has a much greater range.

\textbf{Gamma Log Density}(x, \alpha, <scale = 1>, <threshold = 0>)

Description
Same as \( \log(\text{Gamma Density}(x, \alpha)) \) except that it has a much greater range.

\textbf{Gamma Log Distribution}(x, \alpha, <scale = 1>, <threshold = 0>)

Description
Same as \( \log(\text{Gamma Distribution}(x, \alpha)) \) except that it has a much greater range.

\textbf{Gamma Quantile}(p, <shape, scale, threshold>)

Description
Returns the \( p \)th quantile from the gamma distribution with the \textit{shape}, \textit{scale}, and \textit{threshold} parameters given.
GenGamma Density(x, mu, sigma, lambda)

Description
Returns the density at x of an extended generalized gamma probability distribution with parameters mu, sigma, and lambda.

GenGamma Distribution(x, mu, sigma, lambda)

Description
Returns the probability that an extended generalized gamma distributed random variable (with parameters mu, sigma, and lambda) is less than x.

GenGamma Quantile(p, mu, sigma, lambda)

Description
Returns the quantile from an extended generalized gamma distribution (with parameters mu, sigma, and lambda), the value for which the probability is p that a random value would be lower.

GLog Density(q, mu, sigma, lambda)

Description
Returns the density at q of a generalized logarithmic distribution with location mu, scale sigma, and shape lambda.

GLog Distribution(q, mu, sigma, lambda)

Description
Returns the probability that a generalized logarithmically distribution random variable is less than q.

GLog Quantile(p, mu, sigma, lambda)

Description
Returns the quantile for whose value the probability is p that a random value would be lower.

IGamma()

See “Gamma Distribution(x, <shape, scale, threshold>)” on page 176.
Johnson Sb Density(q, gamma, delta, theta, sigma)

Description
Returns the density at $q$ of a Johnson Sb distribution.

Arguments
- $q$ A value that is in the interval $\theta$ to $\theta + \sigma$.
- gamma Shape parameter that can be any value.
- delta Shape parameter that must be greater than 0.
- theta Location parameter that can be any value.
- sigma Scale parameter that must be greater than 0.

Johnson Sb Distribution(q, gamma, delta, theta, sigma)

Description
Returns the probability that a Johnson Sb-distributed random variable is less than $q$.

Arguments
- $q$ A value that is in the interval $\theta$ to $\theta + \sigma$.
- gamma Shape parameter that can be any value.
- delta Shape parameter that must be greater than 0.
- theta Location parameter that can be any value.
- sigma Scale parameter that must be greater than 0.

Johnson Sb Quantile(p, gamma, delta, theta, sigma)

Description
Returns the quantile whose value for which the probability is $p$ that a random value would be lower.

Arguments
- $p$ The probability of the quantile desired; $p$ must be between 0 and 1.
- gamma Shape parameter that can be any value.
- delta Shape parameter that must be greater than 0.
- theta Location parameter that can be any value.
- sigma Scale parameter that must be greater than 0.

Johnson Sl Density(q, gamma, delta, theta, sigma)

Description
Returns the density at $q$ of a Johnson Sl distribution.
Arguments

- **q**: A value that is in the interval \( \theta \) to +infinity.
- **gamma**: Shape parameter that can be any value.
- **delta**: Shape parameter that must be greater than 0.
- **theta**: Location parameter that can be any value.
- **sigma**: Parameter 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 Distribution** \( (q, \gamma, \delta, \theta, \sigma) \)

Description

Returns the probability that a Johnson Sl-distributed random variable is less than \( q \).

Arguments

- **q**: A value that is in the interval \( \theta \) to +infinity.
- **gamma**: Shape parameter that can be any value.
- **delta**: Shape parameter that must be greater than 0.
- **theta**: Location parameter that can be any value.
- **sigma**: Parameter 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 quantile whose value for which the probability is \( p \) that a random value would be lower.

Arguments

- **p**: The probability of the quantile desired; \( p \) must be between 0 and 1.
- **gamma**: Shape parameter that can be any value.
- **delta**: Shape parameter that must be greater than 0.
- **theta**: Location parameter that can be any value.
- **sigma**: Parameter 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** \( (q, \gamma, \delta, \theta, \sigma) \)

Description

Returns the density at \( q \) of a Johnson Su distribution.

Arguments

- **q**: A value that is between -infinity and +infinity.
gamma  Shape parameter that can be any value.
delta  Shape parameter that must be greater than 0.
theta  Location parameter that can be any value.
sigma  Scale parameter that must be greater than 0.

### Johnson Su Distribution(q, gamma, delta, theta, sigma)

**Description**
Returns the probability that a Johnson Su-distributed random variable is less than \( q \).

**Arguments**
- \( q \): A value that is between -infinity and +infinity.
- gamma: Shape parameter that can be any value.
- delta: Shape parameter that must be greater than 0.
- theta: Location parameter that can be any value.
- sigma: Scale parameter that must be greater than 0.

### Johnson Su Quantile(p, gamma, delta, theta, sigma)

**Description**
Returns the quantile whose value for which the probability is \( p \) that a random value would be lower.

**Arguments**
- \( p \): The probability of the quantile desired; \( p \) must be between 0 and 1.
- gamma: Shape parameter that can be any value.
- delta: Shape parameter that must be greater than 0.
- theta: Location parameter that can be any value.
- sigma: Scale parameter that must be greater than 0.

### LEV Density(x, mu, sigma)

**Description**
Returns the density at \( x \) of the largest extreme value distribution with location \( mu \) and scale \( sigma \).

### LEV Distribution(x, mu, sigma)

**Description**
Returns the probability that the largest extreme value distribution with location \( mu \) and scale \( sigma \) is less than \( x \).
LEV Quantile(p, mu, sigma)

Description
Returns the quantile associated with a cumulative probability \( p \) of the largest extreme value distribution with location \( mu \) and scale \( sigma \).

LogGenGamma Density(x, mu, sigma, lambda)

Description
Returns the density at \( x \) of a log generalized gamma probability distribution with parameters \( mu, sigma, \) and \( lambda \).

LogGenGamma Distribution(x, mu, sigma, lambda)

Description
Returns the probability that a log generalized gamma distributed random variable (with parameters \( mu, sigma, \) and \( lambda \)) is less than \( x \).

LogGenGamma Quantile(p, mu, sigma, lambda)

Description
Returns the quantile from a log generalized gamma distribution (with parameters \( mu, sigma, \) and \( lambda \)), the value for which the probability is \( p \) that a random variable would be lower.

Logistic Density(x, mu, sigma)

Description
Returns the density at \( x \) of a logistic distribution with location \( mu \) and scale \( sigma \).

Logistic Distribution(x, mu, sigma)

Description
Returns the probability that the logistic distribution with location \( mu \) and scale \( sigma \) is less than \( x \).

Logistic Quantile(x, mu, sigma)

Description
Returns the quantile associated with a cumulative probability \( p \) of the logistic distribution with location \( mu \) and scale \( sigma \).
Loglogistic Density(x, mu, sigma)
Description
Returns the density at x of a loglogistic distribution with location mu and scale sigma.

Loglogistic Distribution(x, mu, sigma)
Description
Returns the probability that the loglogistic distribution with location mu and scale sigma is less than x.

Loglogistic Quantile(x, mu, sigma)
Description
Returns the quantile associated with a cumulative probability p of the loglogistic distribution with location mu and scale sigma.

Lognormal Density(x, mu, sigma)
Description
Returns the density at x of a lognormal distribution with location mu and scale sigma.

Lognormal Distribution(x, mu, sigma)
Description
Returns the probability at x of a lognormal distribution with location mu and scale sigma.

Lognormal Quantile(x, mu, sigma)
Description
Returns the quantile at p 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>, <stddev>)

Description
Returns 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 Distribution(x, <mean>, <stddev>)

Description
Returns the cumulative distribution function at quantile x for the normal distribution with mean and stddev. The default mean is 0; the default stddev is 1.

Normal Log CDistribution(x)

Description
Returns 1 - log (value) of the distribution function at quantile x for the normal distribution.

Normal Log Density(x)

Description
Returns the log of the value of the density function at quantile x for the normal distribution.

Normal Log Distribution(x)

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 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>, <stddev>)
Probit(p, <mean>, <stddev>)

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>, <stddev>)” on page 184.

SEV Density(x, mu, sigma)

Description
Returns the density at \( x \) of the smallest extreme distribution with location mu and scale sigma.

SEV Distribution(x, mu, sigma)

Description
Returns the probability that the smallest extreme distribution with location mu and scale sigma is less than \( x \).

SEV Quantile(p, mu, sigma)

Description
Returns the quantile at \( p \) of the smallest extreme distribution with location mu and scale sigma.

Students t Density()

See “t Density(q, df)” on page 185.

Students t Distribution()

See “t Distribution(q, df, <nonCentrality>)” on page 185.
Students t Quantile()

See “t Quantile(p, df, <nonCentrality>)” on page 186.

t Density(q, df)

Students t Density(q, df)

Description
Returns the value of the density function at quantile $x$ for the Student’s $t$ distribution with degrees of freedom $df$.

Students t Distribution(q, df, <nonCentrality>)

Students t Distribution(q, df, <nonCentrality>)

Description
Returns the probability that a Student’s $t$ distributed random variable is less than $q$. NonCentrality defaults to 0.

t Log CDistribution(x, df, <nc>)

Description
Returns $1 - \log$ (value) of the normal distribution function at quantile $x$ for the $t$ distribution.

t Log Density(x, df, <nc>)

Description
Returns the log of the value of the density function at quantile $x$ for the $t$ distribution.

t Log Distribution(x, df, <nc>)

Description
Returns the log of the value of the distribution function at quantile $x$ for the $t$ distribution.

t Noncentrality(x, df, prob)

Description
Solves the noncentrality such that $prob=T Distribution (x, df, nc)$
**t Quantile(p, df, <nonCentrality>)**

**Students t Quantile(p, df, <nonCentrality>)**

**Description**

Returns the \( p \)th quantile from the Student’s \( t \) distribution with degrees of freedom \( df \).

*NonCentrality* 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.
- \( n \) 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 needed in Tukey’s HSD multiple comparisons test.

**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, threshold>)**

**Description**

Returns the value of the density function at quantile \( x \) for the Weibull distribution with the parameters given.

**Weibull Distribution(x, shape, <scale, threshold>)**

**Description**

Returns the cumulative distribution function at quantile \( x \) for the Weibull distribution with the parameters given.

**Weibull Quantile(p, shape, <scale, threshold>)**

**Description**

Returns the \( p \)th quantile from the Weibull distribution with the parameters given.
Programming Functions

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

Note
: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)” on page 134.

As Name(string)
  Description
    Evaluates argument as a 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.

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.

Arguments
name Optional: any global variable name(s).

See
“Clear Symbols(<name>, <name>, ...)” on page 189

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.

Arguments
name Optional: any global variable name(s).

See
“Clear Globals(<name>, <name>, ...)” on page 189

Close Log()

Description
Closes the log.
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.

Arguments  
name  Optional: any global variable name(s).

See  
See “Delete Symbols(<name>, <name>, ...)” on page 190.

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.

Arguments  
name  Optional: any global variable name(s).

See  
See “Delete Globals(<name>, <name>, ...)” on page 190.

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 *string*.

**Returns**

The result.

**Arguments**

- **string**  A 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)**” on page 135.

**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 Environment Variable("variable")

Description
Retrieves the value of an operating system environment variable.

Returns
A 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 string that contains the name of an environment variable.

Notes
On Macintosh, environment variable names are case-sensitive. On Windows, the names are case-insensitive.

Get Log(<n>)

Description
Returns a list of lines from the log.
Returns
A list of 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.

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.

Example
::x = 5;
Show Symbols(); // x = 5;
Include( "SAMPLE_SCRPTS/scriptInclude.jsl", <<New Context, <<Names Default To Here);
Show Symbols();
The resultant value of $x$ becomes 1, because myfile.jsl sets $x$ to 1.

Include File List()

Description
Returns a list of files that are included at the point of execution.

Is Log Open()

Description
Returns result if log window is open.
List

See “List(a, b, c, ...)” on page 136.

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 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.

LogCapture(expr)

Description
Evaluates the expr, captures the output that would normally be sent to the log, and instead returns it.

Returns
A string that contains the log output.

Argument
Any valid JSL expression.

Note
No output appears in the log.
N Items

See “N Items(source)” on page 136.

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 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()

Description
Opens the log. Include the Boolean argument (1) to make the window active, even if it is already open.

Parameter({name=value, ...}, model expression)

Description
Defines formula parameters for models for the Nonlinear platform.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse(string)</td>
<td>Converts a character string into a JSL expression.</td>
</tr>
<tr>
<td>Print(expr, expr, ...)</td>
<td>Prints the values of the specified expressions to the log.</td>
</tr>
<tr>
<td>Quit()</td>
<td>See “Exit(&lt;NoSave&gt;)” on page 191.</td>
</tr>
<tr>
<td>Recurse(function)</td>
<td>Makes a recursive call of the defining function.</td>
</tr>
<tr>
<td>Save Log(pathname)</td>
<td>Writes the contents of the log to the specified file location.</td>
</tr>
<tr>
<td>Send(obj, message)</td>
<td>Sends a message to a platform object.</td>
</tr>
<tr>
<td>Set Environment Variable( &quot;variable&quot;, &quot;value&quot;)</td>
<td>Sets the environment variable to the value specified. If the “value” argument is missing or is an empty string, then the environment variable is deleted from the JMP process environment variable table.</td>
</tr>
<tr>
<td>Show(expr, expr, ...)</td>
<td>Prints the name and value of each expression to the log.</td>
</tr>
</tbody>
</table>
ShowGlobals()

Description
Shows the values for all global symbols. Symbols in any scope other than global are not shown.

See
“ShowSymbols()” on page 197.

ShowNamespaces()

Description
Shows the contents of all user-defined namespaces, both named and anonymous.

ShowSymbols()

Description
Shows the values for all symbols in any and all scopes.

See
“ShowGlobals()” on page 197.

SortList

See “SortList(list|expr)” on page 138.

SortListInto

See “SortListInto(list|expr)” on page 139.

Throw("text")

Description
Returns a Throw. If you include text, throwing stores text in a global exception_msg. If text begins with “!” , throwing creates an error message about where the exception was caught.

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"))}

Note
Expr2 can be a character string or the global exception message (exception_msg) that
contains more information about the error returned.

Type(x)

Description
Returns a 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.

Watch(all | name1, ...)

Description
Shows variables (global, local, and variables within namespaces) and their values in a
window. If “all” is provided as the argument, all globals are placed into the window.

Note
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.

R Integration Functions

R Connect( <named_arguments> )

Description
Returns the current R connection object. If there is no connection to R, it initializes 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 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. The valid formats are "png", "bmp", "jpeg", "jpg", "tiff", and "tif".

R Init( named_arguments )
Description
Initializes the R session.
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.

Argument
name The name of a JMP variable to be sent to R.

Returns
A 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. The individual values are as follows:

- Selected = 1
- Excluded = 2
- Hidden = 4
- Labeled = 8
- Colored = 16
- Markered = 32

The named arguments for the Row States() argument are as follows:

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
1 if an active R connection exists; otherwise 0.

Arguments
None

Random Functions

Col Shuffle()

Description
Shuffles the values randomly each time it’s evaluated.

Returns
The last value placed into the last row.

Argument
none

Example
For Each Row(x=:shuffle = colshuffle());
show(x)
  x = 6
In a table with 40 rows, the above script places the values 1-40 randomly into each row of the column named shuffle. All numbers appear only once. Each time the script is run, the numbers are placed in a different random order. If the value placed into the row 40 of column shuffle is 6, that number is assigned to x, as in the above example.

Random Beta(alpha, beta, <theta>, <sigma>)

Description
Generates a pseudo-random number from a beta distribution using two shape parameters, alpha and beta. Optionally generates the lower threshold parameter, theta, and the scale parameter, sigma. The default values are theta = 0 and sigma = 1.
Random Beta Binomial(n, p, <delta>)

Description
Returns a random number from a beta binomial distribution for n trials with probability p and correlation delta. The default value for delta is 0.

Random Binomial(n, p)

Description
Returns random numbers from a binomial distribution with n trials and probability p of the event of interest occurring.

Random Category(probA, resultA, ProbB, resultB, resultElse)

Description
Returns a random category given an alternation of probability and result expressions.

Random Cauchy(<alpha>, <beta>)

Description
Returns a Cauchy distribution with given alpha (location) and beta (scale).

Random ChiSquare(df, <noncentral>)

Description
Returns a Chi-Square distribution with given df (degrees of freedom). The noncentrality parameter must be greater than or equal to zero, which is the default value.

Random Exp()

Description
Returns a random number distributed exponentially from 0 to infinity. Equivalent to the negative log of Random Uniform.

Random F(df, dfd, <noncentral>)

Description
Returns a random number from an F distribution with a given dfn (degrees of freedom numerator) and dfd (degrees of freedom denominator). The noncentrality parameter must be greater than or equal to zero, which is the default value.
Random Frechet(<mu>, <sigma>)
Description
Returns a random number from a Fréchet distribution with the location \( \mu \) and scale \( \sigma \). The default values are \( \mu = 0 \) and \( \sigma = 1 \).

Random Gamma(lambda, <scale>)
Description
Gives a gamma distribution for given \( \lambda \) and optional \( \text{scale} \). The default value for \( \text{scale} \) is 1.

Random Gamma Poisson(lambda, <sigma>)
Description
Returns a random number from a gamma Poisson distribution with parameters \( \lambda \) and \( \sigma \). The default value for \( \sigma \) is 1.

Random GenGamma(<mu=0>, <sigma=1>, <lambda=0>)
Description
Returns a random number from an extended generalized gamma distribution with parameters \( \mu \), \( \sigma \), and \( \lambda \).

Random Geometric(p)
Description
Returns random numbers 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.

Random Index(n, k)
Description
Returns a \( k \) by 1 matrix of random integers between 1 and \( n \) with no duplicates.

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 the Johnson Sb distribution.

Random Johnson Sl(gamma, delta, theta, <sigma>)

Description
Returns a random number from the Johnson Sl distribution. The default value for sigma is 1.

Random Johnson Su(gamma, delta, theta, sigma)

Description
Returns a random number from the Johnson Su distribution.

Random LEV(<mu>, <sigma>)

Description
Returns a random number from an LEV distribution with the location $mu$ and scale $sigma$. The default values are $mu = 0$ and $sigma = 1$.

Random LogGenGamma(<mu=0>, <sigma=1>, <lambda=0>)

Description
Returns a random number from a log generalized gamma distribution with parameters $mu$, $sigma$, and $lambda$.

Random Logistic(<mu>, <sigma>)

Description
Returns a random number from a logistic distribution with the location $mu$ and scale $sigma$. The default values are $mu = 0$ and $sigma = 1$.

Random Loglogistic(<mu>, <sigma>)

Description
Returns a random number from a loglogistic distribution with the location $mu$ and scale $sigma$. The default values are $mu = 0$ and $sigma = 1$. 
Random Lognormal(<mu>, <sigma>)

Description
Returns a Lognormal-distributed random number with location parameter \( mu \) and scale \( sigma \). The default values are \( mu = 0 \) and \( sigma = 1 \).

Random Negative Binomial(r, p)

Description
Generates a negative binomial distribution for \( r \) successes with probability of success \( p \).

Random Normal(<mu>, <sigma>)

Description
Generates random numbers that approximate a normal distribution with mean \( mu \) and standard deviation \( sigma \). The default values are \( mu = 0 \) and \( sigma = 1 \).

The normal distribution is bell shaped and symmetrical.

Random Normal Mixture(mean, std_dev, probabilities)

Description
Returns a random number from a normal mixture distribution with the specified arguments.

Arguments
mean A vector that contains group means.
std_dev A vector that contains the group standard deviations.
probabilities A vector that contains the group probabilities.

Random Poisson(lambda)

Description
Generates a Poisson variate for given \( lambda \).

Random Reset(seed)

Description
Restarts the random number sequences with \( seed \).

Random SEV(<mu>, <sigma>)

Description
Returns a random number from an SEV distribution with the specified location \( mu \) and scale \( sigma \). The default values are \( mu = 0 \) and \( sigma = 1 \).
Random Seed State(<seed state>)

Description
Retrieves or restores the random seed state to or from a blob object.

Random Shuffle(matrix)

Description
Returns the matrix with the elements shuffled into a random order.

Random t(df, <noncentral>)

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
\( \text{min} \)  Specifies the lower limit of the triangular distribution. The default value is 0.
\( \text{mode} \)  Specifies the mode of the triangular distribution.
\( \text{max} \)  Specifies 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(<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(beta, <alpha>)

Description
Returns a random number from a Weibull distribution. The default value for alpha is 1.

Resample Freq(<rate, <column>>)

Description
Generates a frequency count for sampling with replacement. If no arguments are specified, the function generates a 100% resample.

Arguments
rate Optional. Specifies the rate of resampling. The default value is 1.
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.

Note
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 that expected value); however, the values vary somewhat due to random selection at the rates corresponding to the old frequency column.

Example
JMP chooses the frequencies for the random sampling in the pre-evaluation phase, which happens before the seed is set. When the script is initially run, the seed hasn’t been set. If the same line of code is run more than once, then after the first run, the resample frequencies are consistent, because the seed is set in the final stage of each run of the code.

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;
Row Functions

As Table(matrix, <matrix 2, ...>, < <<invisible >, < <<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. The data table appears only in the JMP Home Window and the Window menu.
<<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  Quoted string that is the name of the column.

formatted  Quoted string that returns the formatted string of the cell value.

n  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 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
For Each Row(:colname[row()] = count(0, 6, 3, 1))
// 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, 2))
//The rows in the column named colname are filled with the series 0, 0, 3, 3, 6, 6, 0, ... until all rows are filled.

Note
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.

Data Table(n)
Data Table("name")
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.
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.

New Column("name", attributes)

Description
Adds a new column named "name" after the last column in \( dt \).

Note
Can also be used as a message: \( dt << \text{New Column("name", attributes).} \)

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; it appears only in the JMP Home Window and the Window menu. private avoids opening the data table. visible shows the data table. "visible" is the default value. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful because the data requires a lot of memory.
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, <stepSize>, <repeatTimes>)

Description
Produces an arithmetic sequence of numbers across the rows in a data table. The stepSize and repeatTimes arguments are optional, and the default value for both is 1.

Subscript(a, b, c)
list[i]
matrix[b, c]

Description
Subscripts for lists extract the ith item from the list, or the bth row and the cth column from a matrix.

Suppress Formula Eval(Boolean)

Description
Turns off automatic calculation of formulas for all data tables.

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.
   colorof(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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Hidden(rowstate)</code></td>
<td>Returns or sets the hidden index.</td>
<td></td>
</tr>
<tr>
<td><code>Hidden State(num)</code></td>
<td>Returns a row state for hiding from the <code>num</code> given.</td>
<td></td>
</tr>
<tr>
<td><code>Hue State(num)</code></td>
<td>Returns a hue state from the <code>num</code> given.</td>
<td></td>
</tr>
<tr>
<td><code>Labeled(rowstate)</code></td>
<td>Returns or sets the labeled index.</td>
<td></td>
</tr>
<tr>
<td><code>Labeled State(num)</code></td>
<td>Returns a labeled state from the <code>num</code> given.</td>
<td></td>
</tr>
<tr>
<td><code>Marker Of(rowstate)</code></td>
<td>Returns or sets the marker index of a row state.</td>
<td></td>
</tr>
<tr>
<td><code>Marker State(num)</code></td>
<td>Returns a marker state from the <code>num</code> given.</td>
<td></td>
</tr>
<tr>
<td><code>Row State(&lt;dt,&gt; &lt;n&gt;)</code></td>
<td>Returns the row state changed from the initial condition of the active row or the <code>n</code>th row.</td>
<td><code>dt</code> 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. <code>n</code> The row number.</td>
</tr>
</tbody>
</table>
Example

The following example creates the data table references and then returns the row state of row 1 in Big Class.jmp:

```julia
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 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 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 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: quoted string that contains the DNS name of the machine.
port  Required if machine is specified. Quoted string or integer that contains the port on which the metadata server listens.
authDomain  Optional: quoted string that contains the authentication domain for the credentials supplied. Not necessary unless username and password are included.
username  Optional: quoted string that contains the user name for the connection.
password  Optional: quoted string that contains the password for the connection.

Named Arguments

All named arguments are optional.
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("13" <,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. Valid values for SASVersion are “9.1.3”, “9.2”, “9.3”, and “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.

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.

Named Arguments

The following named arguments are all optional.

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).

Repository("repository") A quoted string that contains the name of the repository in
the SAS Metadata Server that this profile will connect to. This option is valid only for
connections to SAS 9.1.3 Metadata Servers.

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 9.2 or higher 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 Repositories()

Description
Gets a list of the repositories available on the current SAS Metadata Server connection.

Returns
A list of repository names as 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 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 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 Quoted string that contains a library reference (8-character maximum) to assign.
path 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: 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: 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.
Named Arguments

All the named arguments are optional.

**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("13" ,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. Valid values for SASVersion are “9.1.3”, “9.2”, “9.3”, and “9.4”. Note: Using the SASVersion argument has the same effect as changing the SAS Server Version on the SAS Integration Preferences page.
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 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.

Named Arguments
All the named arguments are optional.
Columns(list)|Columns(col1, col2, ...) A list of columns or a comma-separated list of columns.
Password("password") A 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 string that contains the password to serve as the READ password for the exported SAS data set.

WritePassword("password") A string that contains the password to serve as the WRITE password for the exported SAS data set.

AlterPassword("password") A 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.

Note
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**
List of strings.

**Arguments**
.libref 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 Quoted string that contains the full path of file on the server to be downloaded to the client machine.

dest Quoted string that contains the full path on the client machine for where to put the copy of the file downloaded from the server.

coding 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
List of strings.

Arguments
fileref 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
List of strings.

Arguments
path 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 quoted 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
List of 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 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 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
List of 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 9.2 or higher Metadata Server.

dataset Optional: quoted string that contains the name of the data set from which to retrieve variable names.

password("password") Optional, 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 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 9.2 or higher Metadata Server.

dataset Optional: quoted string that contains the name of the data set.

Named Arguments

All named arguments are optional.

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. Available arguments (all optional) to `Sample` are as follows:

- **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 for more information.
- **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** Quoted string that contains the SQL query to perform and from which to import the result.

**Named Arguments**

All named arguments are optional.

- **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.
**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 string.

**Returns**

String.

**Arguments**

"path" 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 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 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  Quoted string that contains the full path of the file on the client machine to be uploaded to the server.

dest  Quoted string that contains the full path on the server that receives the file uploaded from the client machine.

encoding  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  Quoted string that contains the SAS code to submit.

**Named Arguments**

All named arguments are optional.

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 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 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 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)

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 string.

New SQL Query(Connection("ODBC:connection_string")|("SAS:connection_string"), Select(Column("column", "t1")), From(Table("table", <Schema("schema")>, <Alias("t1")>), 13), <Options(JMP 12 Compatible(1)|Run on Open(1))>

New SQL Query(Connection("ODBC:connection_string;")|("SAS:connection_string;"), Custom("SELECT col1, col2, col3 FROM table;"), 13), <Options(JMP 12 Compatible(1)|Run on Open(1))>

Description
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 SQL query string and table scripts for modifying and updating the query.

Arguments
Connection The 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.
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 opening the resulting data table. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful because the data requires a lot of memory.
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;" );

Notes
See Appendix A, “SQL Functions Available for JMP Queries” for details about SQLite commands that Query() supports. See the Extending JMP chapter in the Scripting Guide for more examples.
Statistical Functions

**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 “Col Stored Value(<dt>, col, <row>)” on page 211.

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 “Col Stored Value(<dt>, col, <row>)” on page 211.

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 “Col Stored Value(<dt>, col, <row>)” on page 211.

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.
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). -1 means all prior 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). -1 means all prior items.
Weighting(1|0|n) 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).
Partial Window is Missing Boolean positional argument. Controls how missing values are treated. By default, missing values are ignored.
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
Col Moving Average( x, 1, 4 );
// equal weighting of a five-item lagging range
Col Moving Average( x, 0 );
// ramp weighting of all preceding items
Col Moving Average( x, 0, 2, 2 );
// triangle weighting of a five-item centered range
Col Moving Average( x, 0.25 );
// exponential weighting of all preceding items

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 “Col Stored Value(<dt>, col, <row>)” on page 211.

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 “Col Stored Value(<dt>, col, <row>)” on page 211.

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` the By group.

Example

```julia
dt = Open( "SAMPLE_DATA/Big Class.jmp" );
Col Quantile( :height, .5 );
```

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 “Col Stored Value(<dt>, col, <row>)” on page 211.

---

**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.

Argument

- `ByCar` (Optional) A By variable to compute statistics across groups of rows.
- `column` The column to be ranked.
- `<<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 55s. `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 “Col Stored Value(<dt>, col, <row>)” on page 211.
**Col Standardize(name)**

**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.

**Notes**
Standardizing centers the variable by its sample standard deviation. Thus, the following commands are equivalent:

```julia
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 “Col Stored Value(<dt>, col, <row>)” on page 211.

**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 “Col Stored Value(<dt>, col, <row>)” on page 211.
Col Sum(name,<By var, ...>)

Description
Calculates the sum across rows in a column. Calculating all missing values (Col Sum(.,.)) returns missing. 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 “Col Stored Value(<dt>, col, <row>)” on page 211.

Fit Censored(Distribution("name"), YLow(vector) | Y(Vector), <YHigh(vector)>, <Weight(vector)>, <X(matrix)>, <Z(matrix)>, <HoldParm(vector)>)

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.

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 they are to be estimated.
This is primarily used to test hypotheses that certain parameters are zero or some other specific value.

**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, ...,> parameter matrix)**

**Description**
Solves for ability in an Item Response Theory model.

**KDE(vector, <named arguments>)**

**Description**
Returns a kernel density estimator with automatic bandwidth selection.

**Argument**
- **vector** A vector.

**Named Arguments**
All the named arguments are optional.
- **<<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, ...)” on page 246.

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 either all numeric values or all strings.

Mean(var1, var2, ...)

Description
Rowwise mean of the variables specified.

Min

See “Minimum(var1, var2, ...)” on page 246.

Minimum(var1, var2, ...)
Min(var1, var2, ...)

Description
Returns the minimum value of the arguments or of the values within a single matrix or list argument. If multiple arguments are specified, they must be either all numeric values or all 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 \textit{bodyExpr} over all \( i \) until the \textit{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.

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. See the
Response Screening chapter in Predictive and Specialized Modeling.

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.

Note
Performs the same function as the Response Screening platform. See the Response
Screening chapter in the Predictive and Specialized Modeling book for details.

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
\( \frac{11605}{(n+273.15)} \)

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**
$$\frac{11605}{(n-273.15)}$$

**Argument**
n The value of the converted explanatory variable in Arrhenius model.

**Notes**
This is frequently used as a transformation.

---

**Beta(a, b)**

**Description**
Returns the beta function.

**Returns**
Returns the beta function.

**Arguments**
a, b numbers

---

**Digamma(n)**

**Description**
The derivative of the log of the gamma function ($\Gamma$). 

**Returns**
The digamma function evaluated at $n$.

**Argument**
n A number

---

**Exp(a)**

**Description**
Raises $e$ to the power $a$.

**Returns**
$$e^a.$$
ExpM1(x)

Description
Returns a more accurate calculation of $\text{Exp}(x)-1$ 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.

Named Arguments
All named arguments are optional.
<<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.

### 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.

**Note**
$\Gamma(t, \text{limit})$ is the same integral as $\Gamma(t)$ but with the limit of integration that is defined instead of infinity.

**Arguments**
- $t$ a number or a column
- $\text{limit}$ optional limit. The default is 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 $n$.

**Note**
This function is intended for internal use by derivatives. The Ln(n) function is recommended instead.
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.

Logit(p)

Description
Returns $\log(p/(1-p))$.

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.

Note
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 $a^2$.

\[ \text{Root}(n, <r>) \]

Description
Returns the $r$th root of $n$, where $r$ defaults to 2 for square root.

\[ \text{SbInv}(z, \text{gamma}, \text{delta}, \text{theta}, \text{sigma}) \]

Description
Johnson Sb inverse transformation. If argument is normal, the result is Johnson Sb.

\[ \text{SbTrans}(x, \text{gamma}, \text{delta}, \text{theta}, \text{sigma}) \]

Description
Johnson Sb transformation from a doubly bound variable to a standard normal (0, 1) distribution.

\[ \text{Scheffe Cubic}(x1, x2) \]

Description
Returns $x1 \times x2 \times (x1 - x2)$. This function supports notation for cubic mixture models.

\[ \text{SlInv}(z, \text{gamma}, \text{delta}, \text{theta}, \text{sigma}) \]

Description
Johnson Sl inverse transformation. If argument is normal, the result is Johnson Sl.

\[ \text{SlTrans}(x, \text{gamma}, \text{delta}, \text{theta}, \text{sigma}) \]

Description
Johnson Sl transformation from a doubly bound variable to a standard normal (0, 1) distribution.

\[ \text{Sqrt}(n) \]

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 $1/(1 + e^{-expr})$.

SuInv(z, gamma, delta, theta, sigma)

Description
Johnson Su inverse transformation. If argument is normal, the result is Johnson Su.

SuTrans(x, gamma, delta, theta, sigma)

Description
Johnson Su transformation from a doubly bound variable to a standard normal $(0, 1)$ distribution.

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)

**Description**
Inverse cosine.

**Returns**
The inverse cosine of $x$, an angle in radians.

**Argument**
- $x$ Any number, numeric variable, or numeric expression.

## ArcSine(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>)

**Description**
Inverse tangent.

**Returns**
The inverse trigonometric tangent of $x1/x2$, where the result is in the range $-\text{Pi}/2, \text{Pi}/2$. 
**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.

See Also
The Data Structures chapter in the Scripting Guide.

Batch Interactive(Boolean)

Description
Sets the run-time environment for JSL to either batch mode or interactive mode.

Returns
The previous setting.
Example

```javascript
Batch Interactive(1);  // Set Batch mode one
</code>
Batch Interactive(0);  // Revert to previous mode
```

**Note**

This function can be used in journal scripts, to ensure that messages are sent to the log rather than to interactive message windows.

---

**Beep()**

**Description**

Produces an alert sound.

**Returns**

Null.

---

**Blob MD5(blob)**

**Description**

Converts the `blob` argument into a 16-byte blob.

**Note**

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 comma-delimited string.
Caption({h, v}, "text", <Delayed(seconds)>, <Font(font)>, <FontSize(size)>, <TextColor("color")>, <BackColor("color")>, <Spoken(Boolean)>

**Description**
Displays a caption window at the location described by \( \{h, v\} \) that displays \textit{text}. The caption can be delayed before being displayed by \textit{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.
- \textit{text} A quoted string or a reference to a string that is to be displayed in the caption.
- \texttt{Delayed(seconds)} \textit{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.
- \texttt{Font(font)} Specify the font type.
- \texttt{FontSize(size)} Specify the font size.
- \texttt{TextColor("color")} Specify the color of text.
- \texttt{BackColor("color")} Specify the background color.
- \texttt{Spoken(Boolean)} Causes \textit{text} to be spoken as well as displayed. The current setting (on or off) remains in effect until switched by another \texttt{Caption} statement that includes a \texttt{Spoken} setting.

**Datafeed()**
See “Open Datafeed()” on page 268.

**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 Double("string")**

**Description**
Creates a floating point number from a base 64 encoded string.

**Returns**
A floating point number.
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.

See Also
The Data Structures chapter in the Scripting Guide.

empty()

Description
Does nothing. Used in the formula editor for making empty boxes.

Returns
Missing.

Arguments
None.

encode64 Double(n)

Description
Creates a base 64 encoded string from a floating point number.

Returns
A 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 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.

Get Name Info("address", <port>)

Description
Converts a numeric address to its name.
Returns

A list of 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, ...) ... >)

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Platform Preferences( )</td>
<td>Returns the current option values for all platform preferences.</td>
</tr>
<tr>
<td>Get Platform Preferences( Platform )</td>
<td>Returns the current option values for the specified platform preferences.</td>
</tr>
<tr>
<td>Get Platform Preferences( Platform( Option ) )</td>
<td>Returns the current option values for the specified platform.</td>
</tr>
</tbody>
</table>
Table 2.3 Get Platform Preferences() Syntax  (Continued)

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Platform Preferences( &lt;&lt;Changed )</td>
<td>Returns the current option values that have changed for all platforms.</td>
</tr>
<tr>
<td>Get Platform Preferences( Platform( &lt;&lt;Changed ) )</td>
<td>Returns the current option values that have changed for all platform preferences.</td>
</tr>
<tr>
<td>Get Platform Preferences( Platform( Option ( &lt;&lt;Changed ) ) )</td>
<td>Returns the current option values that have changed for the specified platform.</td>
</tr>
</tbody>
</table>

Examples

Suppose that the user modified several platform preferences through the JMP Platforms window or a script.

```julia
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 ):

```julia
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, Macintosh OS Settings, Fonts, Communications, Script Editor, and JMP Updates. For details about getting platform preferences, see “Get Platform Preferences(<platform <(option, ...) ... >)” on page 262.

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.

Note
A semicolon placed between expressions is the more commonly used form for Glue().

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. Bits32 or Bits64 tests for the specified 32-bit or 64-bit machine.

Note
Only one argument can be tested at a time. Invalid arguments return false (0).

Is Alt Key()

Description
Returns 1 if the Alt key is being pressed, or 0 otherwise.

Note
On a Macintosh, 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.

Note
On a Macintosh, 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)>)

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 Libraries (DLLs)” on page 337 in the “JSL Messages” chapter for details about these messages. The Extending JMP chapter in the Scripting Guide also includes examples.

Mail("address", "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 string or list of strings.
(Macintosh) Creates an e-mail in the user’s Mail application. The user must click Send in the e-mail. On Mountain Lion, you must enter the e-mail address and subject in the e-mail due to operating system limitations. Click the Send message button to send the e-mail.

Examples
To send an email with multiple attachments:
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 
);

Notes
On Macintosh, Mail() works on Mountain Lion 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.
Multiply(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.

See Also
The Data Structures chapter in the Scripting Guide.

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.

Parse XML(string, On Element("tagname", Start Tag(expr), End Tag(expr)))

Description
Parses an XML expression using the On Element expressions for specified XML tags.

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**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Preferences( &lt;&lt;Default )</td>
<td>Resets all platform preferences to their default values.</td>
</tr>
<tr>
<td>Platform Preferences( &lt;&lt;Factory Default )</td>
<td></td>
</tr>
<tr>
<td>Platform Preferences( Default )</td>
<td></td>
</tr>
<tr>
<td>Platform Preferences( Platform( &lt;&lt;Default ) )</td>
<td>Resets the specified platform preferences to their default values.</td>
</tr>
<tr>
<td>Platform Preferences( Platform( &lt;&lt;Factory Default ) )</td>
<td></td>
</tr>
<tr>
<td>Platform Preferences( Platform( Default ) )</td>
<td></td>
</tr>
<tr>
<td>Platform Preferences( Platform( option ( &lt;&lt;Default ) ) )</td>
<td>Resets the specified platform option to its default value.</td>
</tr>
<tr>
<td>Platform Preferences( Platform( option ( &lt;&lt;Factory Default ) ) )</td>
<td></td>
</tr>
<tr>
<td>Platform Preferences( option( value ) )</td>
<td>Sets the value of the specified platform option. Turned on by default.</td>
</tr>
<tr>
<td>Platform Preferences( option )</td>
<td>Resets the specified platform option back to its default value.</td>
</tr>
<tr>
<td>Platform Preferences( option( &lt;&lt;On ) )</td>
<td>Turns on the specified platform option.</td>
</tr>
<tr>
<td>Platform Preferences( option( &lt;&lt;Off ) )</td>
<td>Turns off the specified platform option.</td>
</tr>
<tr>
<td>Platform Preferences( option( value, &lt;&lt;On ) )</td>
<td>Sets the value of the specified platform option and turns it on.</td>
</tr>
<tr>
<td>Platform Preferences( option( value, &lt;&lt;Off ) )</td>
<td>Sets the value of the specified platform option and turns it off.</td>
</tr>
</tbody>
</table>
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**

```
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.

**Note**

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
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.
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:
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. Each thread that JMP creates has a 2MB stack by default. Resetting the maximum call depth can cause a physical stack overflow.

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. (Windows only)
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) Macintosh 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, forces JMP to interpret label names as column headings.

Examples
The following expressions reset all preferences to their default values.
Preferences("Default");
Preferences("Factory Default");

Notes
The preferences for the following areas are not accessible in JSL: Text Data Files, Windows Specific, Macintosh OS Settings, Fonts, Communications, Script Editor, and JMP Updates. See “Platform Preferences(platform(option(value)), ...)” on page 268 for information about setting platform preferences.

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.

**Note**
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 string, a blob, or a Run Program object as controlled by the Read Function argument.

**Arguments**
**Executable**  The path to the executable. On Macintosh, type the full path to the executable.
**Options**  Command line arguments for the executable.
Read Function  If Read Function( "text" ) is specified, a text 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 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 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
       )
    )
);
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.

```javascript
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:

```javascript
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 "string" argument looking for a character result, and then places the string on the clipboard.

**SetJVMOption( Version("13") )**

**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 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)), ...)” on page 268.

Set Preference()
Set Preferences()

See “Preferences(pref1(value1), ...)” on page 271.

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  Quoted string that shows or hides the toolbar.

Shortest Edit Script()

Description
Compares two lists, strings, matrices, or sequences. See the Scripting Index in the JMP Help menu for more detailed syntax options.

Arguments
limit  An optional argument that stops the evaluation when the edit list exceeds the specified number of inserted or deleted items.

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 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. Specifying fewer than two arguments produces an error.

See Also
The Data Structures chapter in the Scripting Guide.

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 string in the default system web browser. Under Microsoft Windows, you can add JMP Window as the second argument to have the HTML open in a JMP browser.

The http:// prefix in the URL is optional.

Examples
url = "www.jmp.com";
Web( url );

Web( "www.jmp.com" );
Web( "www.jmp.com", JMP Window );
XML Attr("attr name")

Description
Extracts the 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, &amp; becomes &, and &lt; becomes <.

Argument
xml A quoted string containing XML.

XML Encode("text")

Description
Prepares text for embedding in XML. For example, & becomes &amp;, and < becomes &lt;.

Argument
xml A quoted string containing plain text.

XML Text()

Description
Extracts the string text of the body of an XML tag in the context of evaluating a Parse XML command.
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 the Scripting Platforms chapter in the Scripting Guide.
Alpha Shape

For the following 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 Arrays

For the following 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.

Data Tables

dt<<Add Column Properties(property argument, ...)

Add the specified properties to the selected column.

dt<<Add Multiple Columns("prefix", n, position, attributes)

Adds n columns to dt at the position indicated.

dt<<Add Rows(count, <rownum>)
dt<<Add Rows(assignment list)

Add the number of rows specified to the bottom of the data table or starting at the rownum specified.

dt<<Add Scripts to Table(script, ...)
dt<<Add Properties to Table(script, ...)

Adds the specified scripts to the data table.
dt<<Anonymize(<columns(list)>, <Output Table Name(string)>);

Removes unique identifiers from data, some column properties, and table scripts. Applies to a data table or selected columns.

---

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. When columns are deleted, the display is refreshed.

---

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)

Closes the data table grid.

---

dt<<Close Side Panels(Boolean)

Closes the side panel in a data table.
dt<<Color or Mark by Column(col, <optional arguments>)
dt<<Color by Column(col, <optional arguments>)

Function
Assigns colors or markers according to the values of a data table column. If no optional arguments are provided, row states are used.

Arguments
Color Number(n) Uses the specified quoted JMP color.
Color Theme("theme") Uses the specified quoted color theme.
Marker Theme("named argument") Uses the specified quoted marker theme: "standard", "hollow", "paired", "classic", or "alphanumeric".
Continuous Scale For Color by Column, assigns colors in a chromatic sequential fashion based on the values in the highlighted column.
Reverse Scale Reverses the color scheme in use.
Excluded Rows 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<<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 numeric values are between -127 and 127.

dt<<Concatenate(dt2, ..., Keep Formulas, Output Table Name("name"))
Creates a new table ("name") from the rows of dt and dt2.

dt<<Copy Table Script
Copies the script to recreate the data table onto the clipboard so that it can be pasted somewhere else.
dt<<Data Filter(<Mode(...)>, <Add Filter (...)>)

Constructs a data filter. If no arguments are specified, the Add Filter Columns window appears.

Arguments for Mode() include Select(), Show(), and Include(). They are all Boolean. Select defaults to true(1), and Show and Include default to false (0).

Arguments for Add Filter() include Columns(), and Where(). Columns() takes one or more column names separated by commas. You can add one or more WHERE clauses to define the filter.

There are several additional arguments. For more information, see the JMP Reports chapter in the Using JMP book and the Data Tables chapter in the Scripting Guide.

dt<<Get Header Height

Returns the column header’s display height (in pixels).

dt<<Data View(<"options">)

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.

excluded  Quoted. The new data table includes only the rows that are marked as excluded in the original data table.

labeled | labelled  Quoted. The new data table includes only the rows that are marked as labeled in the original data table.

hidden  Quoted. The new data table includes only the rows that are marked as hidden in the original data table.

selected  Quoted. The new data table includes only the rows that are selected in the original data table.

dt<<Delete Columns(col, col2, ...)

dt<<Delete Column

Deletes column(s) from the data table dt. Specify which column or columns to delete. Without an argument, deletes the selected columns, if any. Delete Column is a synonym.

dt<<Delete Rows(<n>)

dt<<Delete Rows({n, o, p, ...})

dt<<Delete Rows({n::q})

Deletes the currently selected rows or rows specified.
dt<<Delete Table Property("name")

Deletes a table property (for example, a script).

dt<<Delete Table Variable("name")

Deletes a table variable.

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

Returns values from the numeric columns of dt in a matrix.

dt<<Get As Report

Returns the data table as a 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 Column Names("arguments")

Returns a list of column names in a data table. The "arguments" restrict the names retrieved as follows:

"Numeric", "Ordinal", "Rowstate", "Continuous", "Ordinal", and "Nominal" get only the specified types of columns. More than one can be specified. "String" returns a list of strings rather than a list of column references.

dt<<Get Column Reference(list or matrix of col names)

Returns the column reference of the strings in the list or matrix. If no list or matrix is used, JMP returns all column names.

dt<<Get Display Width

Returns the column display width in pixels.

dt<<Get Excluded Rows

Returns the currently excluded rows in the data table.

dt<<Get Hidden Rows

Returns the currently hidden rows in the data table.

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 table.
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 in the 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. Some examples are as follows:
        dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
        dt << Get Rows Where( :sex == "M" );
        dt << Get Rows Where( :sex == "M" & :age < 15 );

dt<<Get Script

    Returns as an expression a script to recreate the data table.

dt<<Get Selected Columns("string")

    Returns a list of selected columns as column references. Include the "string" argument to return the selected columns in a string.

Examples
        dt << Get Selected Columns();
        {:age, :sex, :height}
        dt << Get Selected Columns( "string" );
        {"age", "sex", "height"}

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 from the variable or script name.

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({col1, col1, ...})
dt<<Group Columns("group name", col, n)

    Groups the columns specified under the specified 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<<Hide
dt<<Unhide

    Toggles selected rows in dt from hidden to unhidden or vice versa.

dt<<Invert Row Selection

    Selects any row 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(table), <Private>, <Invisible>, Select(columns), Select
    With(columns), (By Matching Columns(col1=col2, ...)|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 \textit{dt} and \textit{table} side to side. For details about joining tables, see the Reshape Data chapter in the \textit{Using JMP} book.

Returns

A data table.

Arguments

\textbf{Private} Optional. Does not open the resulting data table. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful the data requires a lot of memory.

\textbf{Invisible} Optional. Opens the data table but hides it. The table is in the JMP Home Window’s Window List.

With(\textit{dataTable}) The secondary data table.

Select(\textit{columns}) Selects the columns from the main table to be added to the output table. Optional.

Select With(\textit{columns}) Selects the columns from the secondary table to be added to the output table.

By Matching Columns(\textit{col1=col2}, \ldots)|\textit{Cartesian}|\textit{By Row Number} Determines the method for joining the tables. By Matching Columns(\textit{col1=col2}, \ldots) updates the first table with data from second table (the default method). \textit{Cartesian} joins two tables using a Cartesian method, where it forms a new table consisting of all possible combinations of the rows from two original 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 \textit{Row Number} joins the two tables side by side. By Matching Columns is the default value.

\textbf{Merge Same Name Columns} Optional. Merges columns with the same name. Off by default.

\textbf{Match Flag} Optional. Off by default. Omits the Match Flag column from the joined data table when you are matching by column.

\textbf{Copy Formula} (Boolean for main table, Boolean for secondary table) Optional. Copies the formulas from the table that you are joining. On by default.

\textbf{Suppress Formula Evaluation()} Optional. Prevents JMP from evaluating columns’ formulas during the creation of the new table. On by default. Use \textbf{Suppress Main Table Formula Evaluation} to suppress evaluation only in the main table. Use \textbf{Suppress Second Table Formula Evaluation} to suppress evaluation only in the secondary table.

\textbf{Update} Optional. Replace the data in the main table with the corresponding data from the secondary table.
Drop Multiples(Boolean for main table, Boolean for secondary table)
Optional. Includes all rows from the data table, even when there is no matching value. If you do not check the boxes associated with Drop multiples in either table, a Cartesian join is performed within each group of matching column values. Off by default.

Include Non Matches(Boolean for main table, Boolean for secondary table)
Optional. Includes non-matching columns. Off by default.

Preserve Main Table Order Optional. Maintains the order of the original data table in the joined table, instead of sorting by the matching columns. On by default.

Output Table Name("name") Optional. The name of the resulting table.

```
dt<<Journal
   Makes a journal from the data table. Only the data grid is included, not notes, variables, or scripts.

   dt<<Journal Link
   Adds a link to the data table in the current journal. If a journal does not exist, a new one is created.

   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
   Makes a layout window from the data table. Only the data grid is included, not notes, variables, or scripts.

   dt<<Lock Data Table
   Locks the data table so that data and column properties cannot be added or changed.

   dt<<Make Indicator Columns( <Prepend Column Name(Boolean)>, <Include Missing(Boolean)>);
   Creates indicator columns of 0 and 1 values for the specified categorical column.
```
Example

The following example creates indicator columns for the sex column. Prepend 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 ),
  Prepend 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<<Marker by Column(col)

Assigns markers according to the values of a data table column.

dt<<Markers(n)

Assigns marker n to the selected rows.

dt<<Maximize Display

Forces the data table to remeasure all of its columns and zoom to the best-sized window.

dt<<Move Selected Columns("To First"|"To Last"|After(col))

Moves the selected columns in the data table to the specified position.

dt<<Move Rows("At Start"|"At End"|After(n))

Moves the selected rows in the data table to the specified position.
dt<<New Column("name", attributes)

Adds a new column named "name" after the last column in dt. Can also be used as a function: New Column("name", attributes).

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.

```julia
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 Auto Stretching( 0, 0 ) << 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 using the specified name that stores the specified script.

Use New Script() or Set Property() rather than the deprecated New Property() and New Table Property().
dt<<New Table Variable("name", value)

dt<<Set Table Variable("name", value)

Creates a new table variable with the specified name and value.

---

dt<<Next Selected

Scrolls data table down to show the next selected row that is not already in view.

---

dt<<Original Order

Restores saved order of columns in dt.

---

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 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<<JMP Query Builder

Builds a query containing one or more data tables.

---

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.
JSL Messages
Data Tables

---

**dt<<Reverse Order**
Revokes columns in *dt* from current order.

---

**dt<<Revert**
Reverts to the most recently saved version of *dt*.

---

**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 *name* and runs it as a JSL script.

---

**dt<<Save("path")**
**dt<<Save As("path")**
Saves the table under the "*path*" given.
For information about supported formats, see the Import Your Data chapter in the *Using JMP* book.

---

**dt<<Save Database("connection_information", "table_name", <Replace>)**
Save the data table to the database named using the connection and table name specified. 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 Journal window.

---

**dt<<Select All Rows**
Selects all rows in the data table.

---

**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(p|n)

Randomly selects the given percentage $p$ of the rows in the data table, or the number of rows $n$.

dt<<Select Rows({list})

Selects the rows given in the list of row numbers.

dt<<Select Where(condition)

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 Header Height(num)

Sets the column header's display height to the number of specified pixels.

dt<<Set Label Columns(column_1, ...)

Assigns the specified columns as label columns.

dt<<Set Matrix(matrix)

Inserts matrix into a data table, adding new columns and rows as necessary.

dt<<Set Name("name")

Gives a "name" to the table.

dt<<Set Property("name", script)


Turns on the Label attribute for the specified columns. If no columns are listed, turns Label attribute off.
JSL Messages
Chapter 3
Data Tables

\[ \text{dt}<<\text{Set Row ID Width(} \text{expr} \text{)} \]

Sets the row ID display width to the expr in pixels.

\[ \text{dt}<<\text{Set Row States(} \text{matrix} \text{)} \]

Sets the row states for all rows in the data table.

\[ \text{dt}<<\text{Set Scroll Lock Columns("name", ...)} \]

Locks scrolling for the specified columns. If no columns are listed, unlocks scrolling.

\[ \text{dt}<<\text{Set Table Variable("name", value)} \]

See “\[ \text{dt}<<\text{New Table Variable("name", value)} \]” on page 295.

\[ \text{dt}<<\text{Sort(By(} \text{columns}, \text{order("Descending" or "Ascending"), Output Table Name("name")}) \]

Creates a new table (“name”) by rearranging the rows of \( \text{dt} \) according to the values of one or more columns.

\[ \text{dt}<<\text{Split (} \text{Split(} \text{column}), \text{Split by(} \text{column}), \text{Group(} \text{column}), <\text{Private}> | <\text{Invisible}>, <\text{Remaining Columns(} \text{Keep All} | \text{Drop All} | \text{Select(} \text{columns})>)>, <\text{Copy formula(0|1)}>, <\text{Suppress formula evaluation(0|1)}>, <\text{Sort by Column Property("Value Ordering" {"string", "string"} | "Row Order Levels")}>, <\text{Output Table ("name")}> \]

Un-stacks multiple rows for each Split column into multiple columns as identified by the Split by column. The Split and Split by arguments are required.

\[ \text{dt}<<\text{Stack(} \text{Stack(} \text{columns}), \text{ID(} \text{columns}), \text{Stacked(newcol)}, \text{Output Table Name("name")}) \]

Creates a new table (“name”) by combining the values from several columns in \( \text{dt} \) into one column newcol.

\[ \text{dt}<<\text{Subscribe( "keyname"("client"), On Delete Columns | On Add Columns | On Add Rows | On Delete Rows | On Rename Column | On Close | On Save | On Rename ( function ) )} \]

Subscribes to a data table to get messages regarding changes in the data table.
dt<<Subset(Columns(columns), Rows(matrix), Linked, Table Name("name"), Copy Formula(1|0), Suppress Formula Evaluation(1|0), Sampling Rate(rate))

Creates a new table ("name") from the rows that you specify in dt.

dt<<Summary(<Private>, <Invisible>, Group(column), Subgroup(column), <N>, <Mean(column)>, <Std Dev(column)>, <Min(column)>, <Max(column)>, <Range(column)>, <Sum(column)>, <CV(column)>, Freq(<freq column>), Weight(<weight column>), Include marginal statistics, Link to original data table(0), statistics column name format("stat(column)" | "column" | "stat of column" | "column stat")

Creates a new table of summary statistics for the col that you specify, according to groups and subgroups.

dt<<Suppress Formula Eval(Boolean)

Turns off automatic calculation of formulas for data table dt.

dt<<Text to Columns

Makes a set of text columns or indicator columns from a delimited text column.

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"))

Creates a new table ("name") from the rows and columns that you specify.

dt<<Ungroup Columns({col1, col1, ...})

Ungroups the columns defined in the list argument.

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.
Columns

`col<<Add Column Properties ("name", expression)`

Adds the 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")`

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)`

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")`

Sets the "type" for col; choices are "Numeric", "Character", "Rowstate".

`col<<Delete Formula`

Delete the formula from a column.
col<<Delete Property(name)
Delete the property name from a column.

col<<Delete Column Property(name)

__________________________
col<<Eval Formula
Forces the formula to evaluate (perhaps again).

__________________________
col<<Exclude(0|1)
Turns the excluded or unexcluded state on, depending on the Boolean argument.

__________________________
col<<Format("format", <width>, <decimal>, <"Use Thousands Separator">)
Sets the numeric display format specified.

__________________________
col<<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 Format
Returns the format of the column.

__________________________
col<<Get Formula
Returns the formula.

__________________________
col<<Get Input Format
Returns the format used for input and storing of data for the column.

__________________________
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 ("Continuous" or "Ordinal" or "Nominal").

col<<Get Name
    Returns the name of the column.

col<<Get Property
    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 Selected
    Returns 1 if the column is selected, or 0 otherwise.

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 Formula Error
    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 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 role for the column. Choices are "Y", "X", "Weight", "Freq", and "None", or
    "No Role".

col<<Set Scroll Locked(Boolean)
    Turns the Scroll Lock attribute on or off according to the Boolean argument given.

col<<Set Display Width(n)
    Sets the column display width to the n in pixels.

col<<Set Each Value(n)
    Sets all the values in the column to n.
**col**«Set Field Width(n)**

Sets the field width for the column to *n*.

**col**«Set Modeling Type("type")

Sets the modeling type for the variable. Choices are "Continuous", "Ordinal", or "Nominal".

**col**«Set Name("name")

Sets the name for the column.

**col**«Set Property ("name", {argument list})

Sets the property "name" to the expression given. You can set any standard column property by name or a user-specified property.

**col**«Set Selected(Boolean)

Sets the column to be selected (true or 1) or not selected (false or 0).

**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.

**Rows**

**row**«Colors(n)

Assigns the color *n* to the selected rows.

**row**«Exclude(0|1)
**row**«Unexclude(0|1)

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(0|1)
row<<Unhide(0|1)

Turns the Hide attribute on or off according to the Boolean argument given. Omit the argument to toggle the row state.

row<<Label(0|1)
row<<Unlabel(0|1)

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 "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

dtf<<Add Filter(columns(column <,col>), <Where(clause)>)

Add one or more filter columns in a new OR group.

dtf<<Auto Clear(0|1)

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(col1, col2, ...)
    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(col1, col2, ...)
    Removes the specified columns from the data filter.

dtf<<Display(col, <Size(x, y)>, "Blocks Display"|"List Display"|"Single Category Display"|"Checkbox Display")
    Sets how the specified categorical column levels are displayed in the filter. The specified column must be categorical.

dtf<<Get Script
    Returns a scriptable object that you can send messages to.

Example
    Open( "$SAMPLE_DATA/Big Class.jmp" );
    df = dt << Data Filter(
        Add Filter( Columns( :age, :sex ), Where( :age == 12 ) )
    );

    txt = df;
    txt << ( Filter Column( :sex ) << Where( :sex == "M" ) );

dtf<<Local Data Filter
    Embeds the data filter in the specified window.

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 Subset
    Creates a new subset data table that contains the rows that are selected in the data filter.
dtf<<Match(Filter Columns(:col1, :col2, ...), Where(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(0|1)|Show(0|1)|Include(0|1))

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(0|1)

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(0|1)
    Sets whether the data filter window floats on top of its associated data table or behaves as a normal window.

dtf<<Where(WHERE clause)
    Sets a condition for selecting rows.

Databases

dt<<Save Database("connectInfo", "TableName")
    Saves a data table to a database.

Datafeed

feed<<Close
    Closes the Datafeed object and its window.

feed<<Connect(portSettings)
    (Windows only) Sets up port settings for the connection to the device.

feed<<Disconnect
    (Windows only) Disconnects the device from the Datafeed queue but leaves the Datafeed object active.

feed<<Get Line
    Returns and removes one line from the Datafeed queue.

feed<<Get Lines
    Returns as a list and removes all lines from the Datafeed queue.

feed<<Queue Line(string)
    Sends one line to the end of the Datafeed queue.
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.

---

**Display Boxes**

For additional examples, see the Display Trees chapter in the *Scripting Guide*.

**All Display Boxes**

db<<Add Text Annotation(Text("string"), Text Box(x1, y1, x2, y2))

Draws a text annotation box that contains the string. The Text Box argument controls where the text annotation box is drawn in relation to the display box, from the upper left corner to the lower right corner.

db<<Append(db2)

Add db2 to the display tree after 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.
**JSL Messages**

**Display Boxes**

**JSL Syntax Reference**

---

**db<<Copy Picture**

Puts a picture of the box on the clipboard.

---

**db<<Delete**

Delete 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 string containing HTML source for the box.

---

**db<<Get Journal**

Returns a 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**

Returns the menu items used for popup menu when the button is clicked. For submenus see **<<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.

```plaintext
w = New Window( "Window",
    Text Box( "Page Setup Test" )
);
w << Get Page Setup();
```

The results of the message:
JSL Messages

{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 is a factor of the original picture size. For example, Scale(2) makes the picture object twice as large.

db<<Get RTF

Returns a 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:

{ x, y } = 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(

  "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, "," ) ), inc++,
  Insert Into( submenus, obj << Get Submenu( inc ) );
);

submenus;

{2, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 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 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", db<<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 new 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 may 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<<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 "path" in the specified "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.

db<<Save HTML("" | "path", "format")

Saves HTML source and folder of graphics in "format" specified.

db<<Save Interactive HTML( "path" )

Saves the display box as a web page that includes interactive HTML 5 features. Non-JMP users can then explore the data.

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" );

\( db\ll<\text{Save Journal}(""")\) \\
Saves journal source for the box.

\( db\ll<\text{Save MSWord}("pathname", \text{Native}) \) \\
Saves the display box as a Microsoft Word document. (Windows Only)

\( db\ll<\text{Save Picture}(""") \) \\
Saves a picture of the box.

**Notes**

Valid file formats include PNG, GIF, JPG or JPEG, and EMF.

On Windows, the Windows Specific preferences determine the resolution (or DPI), or you can run the following script:

\[ \text{Pref( Save Image DPI( <number> ) )}; \]

On Macintosh, the operating system determines the DPI.

\( db\ll<\text{Save Presentation}("path", \text{<template>}, \text{<Insert>}, \text{<Outline>}, \text{<format>}) \) \\
Saves display boxes in a Microsoft PowerPoint presentation. You can open the file in any presentation software program.

**Arguments**

- **path** The quoted location of the saved file. You must include the .pptx extension in the filename. An existing file with the same name is overwritten.
- **template** Optional. The quoted path name and file name 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** Optional. 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  Optional. Determines which slides are replaced in an existing presentation. The arguments are \texttt{n}, \texttt{"Begin"}, and \texttt{"End"} as described for \texttt{Insert}.

Append  Optional. The slides are inserted at the end of an existing presentation.

Outline Titles  Optional. The location of the outline titles on the slide. By default, outline titles appear in the bottom left corner of the slide.
- \texttt{"None"} omits the slide title above the graphic and the outline titles.
- \texttt{"Hide"} omits the outline titles.
- \texttt{"TopLeft"}, \texttt{"TopRight"}, \texttt{"BottomLeft"}, \texttt{"BottomRight"} determine the position of the outline titles on the slide. The outline title and its parent titles are included.

format  Optional. The quoted format of the embedded graphics. Options are \texttt{"Native"}, \texttt{"EMF"}, \texttt{"PNG"}, \texttt{"JPG"}, \texttt{"BMP"}, \texttt{"GIF"}, \texttt{"TIF"}. On Windows, the native format is EMF. On Macintosh, 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 Macintosh. Macintosh does not support the native EMF graphics produced on Windows. For cross-platform compatibility, specify \texttt{"PNG"}, \texttt{"JPG"}, \texttt{"GIF"}, or \texttt{"TIF"}.

```
db<<Save RTF("" | "pathname", "format")
```
Saves RTF source with graphics in \texttt{"format"} specified.

```
db<<Save Text("" | "pathname", "format")
```
Saves a file containing the text of the box.

```
db<<Scroll Window(x, y)
```
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), checked (1), or disabled (-1).
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 string specifying the paper size, for example, "Letter" or "Legal".

**Example**

The example below creates a new window and configures the page setup.

```julia
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" )
);
```

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".

```julia
New Window( "title", ob = Outline Box( "title" ) );

ob << Set Menu Script(
      "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
```

Sets a new title.

**Show Properties(db)**

Shows the messages a given display box can interpret.
db<<Sib

Returns the sibling of the 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 Boxes

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).

The following example creates a bivariate plot and defines 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("string")

   Adds an axis label with the specified string.

axis box<<Add Ref Line(number, "linestyle", <"color">, <"label">, <"width">)

   Adds a reference line at number in the linestyle("Solid" | "Dashed" | "Double"),
   "color" (using "color name" or index number), "label", and width (in pixels) specified.
   Note: When a reference line is added that uses the same "label" as an existing reference
   line, the existing reference line is removed and the new line added.

axis box<<Decimal(width, decimalplaces)

   Changes the numeric format for axis values.

axis box<<Format("name")

   Changes to the numeric format given by "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")

   Description
   Rotates the axis label to one of the following quoted formats: "Automatic" (based on the
   width of the label), "Horizontal", "Vertical", "Perpendicular", "Parallel", and
   "Angled".

axis box<<Major Grid Line Color("color")

   Sets the color for the major grid (if enabled) using either the "color name" or index
   number.
axis box<<Max(max)
    Changes the maximum value on the axis.

axis box<<Minor Grid Line Color("color")
    Sets the color for the minor grid (if enabled) using either the "color name" or index number.

axis box<<Min(min)
    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").
    If the type is Custom Scale, this message expects two additional clauses: Scale to Internal(expr) and Scale to External(expr).

axis box<<Tick Font("name", <size>, <"style" | "style style...">, "<angle>")
    Sets the font name and 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.

   The first (optional) integer, i, is 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.

   The second list argument is the string titles for your labels.

   The third list argument is optional and specifies the values corresponding to each label; if the value list is omitted, the labels will be on integer increments starting with 1.

**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 a "color name" or a list of RGB values.
   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 "color name" or a list of RGB values. 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. For example:
border box<<Get Style;

Data Grid Boxes

dgb<<Set Data Table(<data_table>)
Sets the data table for the data grid box.

Frame Boxes

frame box<<Add Graphics Script(<order>,<"description">, <script>)
Description
Adds a script to draw graphics in the frame box.
Arguments
order An optional argument that 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 elements. 1 means the object is drawn first.
description An optional string that appears in the Customize Graph window next to the graphics script.
script An optional 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(
    // draws the line first
    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( {R, G, B} | <"color">)***

Changes the background color. Specify a "color name" or a list of RGB values.

**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.
**Frame Boxes**

- `frame box<<Row Colors(color)`
- `frame box<<Row Markers(marker)`
- `frame box<<Row Exclude`
- `frame box<<Row Hide`
- `frame box<<Row Label`

  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<<X Axis(min, max, inc, named arguments)`

  Scales the X coordinate system.

- `frame box<<Y Axis(min, max, inc, named arguments)`

  Scales the Y coordinate system.

**Matrix Boxes**

- `matrix box<<Get`

  Returns the matrix contents.

- `matrix box<<Make Into Data Table(<Invisible(Boolean)>)`

  Turns the matrix into a new data table. The optional argument `Invisible()`, if True, hides the data table from view but shows it in the Window List pane. If `False`, the `Invisible()` argument shows the data table and it is listed in the Window List pane.

- `matrix box<<Set Format(width, decimalplaces)`

  Sets the numeric format for matrix elements.

- `matrix box<<Sort(column_num, ascending_bool)`

  Sorts the rows of the matrix based on the column number specified by `column_num`. The default sort order is ascending.

  If `column_num` is 0, the sort is removed.

  If `ascending_bool` is "True", the sort is performed in ascending order. If `ascending_bool` is "False", the sort is in descending order.
Nom Axis Boxes

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 Boxes

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 of numbers.

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 \((\text{width}, \text{decimal places})\). A \( \text{decimal places} > 100 \) indicates date/time values.

number col box<<Get Heading
 Returns the column heading text.
number col box<<Set Format(width, decimalplaces)
number col box<<Set Format("format", width)

Sets the format. Set decimalplaces > 100 for date/time values. Set decimalplaces = 97 for p-value format.

For example,
<< Set Format( 10, 2)
<< Set Format( "Scientific", 10 )

number col box<<Set Heading("string")

Changes the column heading text.

Outline Boxes

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(\"string1\", script1,\"string2\", script2, ...")
   Adds an entry to the menu when the red triangle on an outline box is selected.

outline box<<Set Title(\"text\")
   Changes the title of the outline box.

Panel Boxes

panel box<<Get Title
   Gets the title of the panel box.

panel box<<Set Title(\"text\")
   Changes the title of the panel box.

Plot Col Boxes

plot col box<<Get As Matrix
   Gets the values in a matrix, specifically a column vector.

plot col box<<Set Values([[matrix] or {list}])
   Sets values for the matrix (for numeric variables) or list (for character variables).

Slider Boxes and Range Slider Boxes

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<<Set(n, <index>, <run script(0|1)>)
range slider box<<Set Lower(n, <index>, <run script(0|1)>)
range slider box<<Set Upper(n, <index>, <run script(0|1)>)

Sets the value of the slider. run script(0|1) controls whether an on-change script runs after the <<Set, <<Set Lower, or <<Set Upper message.

slider box<<Get Min()

Returns the minimum 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<<Get Max()

Returns the maximum 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 Max(float, <index>)

Sets the maximum 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_var)
range slider box<<Set Lower Var(slider_var)
range slider box<<Set Upper Var(slider_var)

Sets the variable name associated with the slider.
String Col Boxes

string col box<<Add Element(item)

Adds the item to the String Col Box. Item can be a single quoted string, a list of quoted strings, or a matrix of quoted strings.

string col box<<Get

string col box<<Get(i)

Gets the values in a list, or the ith value.

string col box<<Get Heading

Returns the column heading text.

string col box<<Set Heading("text")

Changes the column heading text.

string col box<<Set Justify("right" | "left" | "center")

Specifies the alignment of the contents in the string col box to right, left, or center.

Tab Boxes

tab box<<Get Tab Margin()

Returns a list of the current margins in pixels for the tab box in this order: {left, top, right, 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. See the Display Trees chapter in the Scripting Guide for an example.

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\}.

\texttt{tab box\<Show Tabs(0|1)}

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.

\textbf{Table Boxes}

\texttt{table box\<Get}

Gets the entries of the table in list form.

\texttt{table box\<Get As Matrix}

Gets the numeric entries of the table in matrix form.

\texttt{table box\<Get Locked Columns}

Returns the number of columns that cannot be dragged with the hand cursor or have any
columns dropped before them.

\texttt{table box\<Get Row Change Function}

Returns the expression that is evaluated when a row is selected.

\texttt{table box\<Get Selectable Rows}

Returns True if the table box currently allows row selection.

\texttt{table box\<Get Selected Row Color}

Returns the index number of the background color of the selected rows in the table box.

\texttt{table box\<Make Combined Data Table}

Same as \texttt{Make Data Table}, but also searches the report for report tables with the same
columns and combine all of these into the new data table.

\texttt{table box\<Make Data Table(name)}

Turns the table entries into a new data table.
**table box<<Set Row Change Function**

Sets the expression that is evaluated when a row is selected.

**table box<<Set Cell Changed Function(Function({this, col box, row},<script>;))**

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.

```javascript
New Window( "Mountains",
  tb = Table Box(
    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 Locked Columns(n)**

Locks the first n columns. You cannot drag the locked columns or drag columns before them.

**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 for the selected rows.

**Text Boxes**

text box<<Font Color(n)

Sets the color for Text strings.

text box<<Get Hidden State

Returns the current state of a text box.

text box<<Get Text

Returns the 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.

```julia
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 the horizontal.

text box<<Set Font("name", <size>, <style | "style style...">, <angle>)

Sets the font name and properties for text 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 strings.

**text box**<<Set Font Style("style" | "style style...")

Sets the font style for text strings. To specify more than one style, include a space between each style and place them in quotes. Integers are backwards compatible but not recommended for new scripts.

**text box**<<Set Script(script)

Associate a script to a text box. The script executes when the user hits enter (or the text edit box otherwise loses focus).

**text box**<<Set Text("string")

Changes the string content of the box.

**text box**<<Set Tip("string")

Sets the tooltip for the text box (or a text edit box).

**text box**<<Set Wrap(n)

Set the wrap point, in pixels, in pixels (n).

### Tree Node and Tree Box

For the following messages, node stands for a tree node or a reference to one and root stands for a tree box or a reference to one.

**node** <<Append(<node_ref>)

Inserts a referenced tree node after this node’s children.

**root** <<Collapse(<node>)

Collapses the given tree node.

**root** <<Expand(<node>)

Expands the given tree node.
JSL Syntax Reference Display Boxes

Chapter 3

JSL Messages

333

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

<table>
<thead>
<tr>
<th>Items Selected in Tree</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>no items selected</td>
<td>empty</td>
</tr>
<tr>
<td>single item selected</td>
<td>list of one tree node</td>
</tr>
<tr>
<td>multiple items selected</td>
<td>list of selected tree nodes</td>
</tr>
</tbody>
</table>

node <<Get Tip

Returns the tooltip for the node.

root <<Is Multiselect

Returns 1 for a MultiSelect tree and 0 for a single-select tree.

node <<Prepend(<node_ref>)

Inserts a tree node before this node’s children.

node <<Remove(<node>)

Removes the given tree node and all its children from the tree display box.

root <<Set Selected(node|{nodes}, <0|1>)

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 Macintosh, <<Set Selected() does not change the tree expansion state.
node <<Set Tip("text")
   Specifies a tooltip for the node.

Triangulation

For the following 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 Nx2 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.

Windows

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 <<Reshow for another method.

window<<Maximize Display

Maximizes the window.

window<<Maximize Window(Boolean)

Maximizes the window.

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<<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.

window<<Set Window Size(x, y)

Resizes the window.

window<<Show Window(0|1)

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 Macintosh), 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 also <<Inval and <<Reshow messages 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, Layout, 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 Libraries (DLLs)

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>)

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.

Named Arguments

Alias("name") A name you can include if you don't like the name encoded in the DLL.

Arg(type, <"description">, <access_mode>, <array>) Arg is optional and 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 an optional 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) An optional keyword that specifies the calling convention: STDCALL or PASCAL, or CDECL. The default value is STDCALL. STDCALL and PASCAL are equivalent.

MaxArgs(n) An optional integer that specifies the maximum number of arguments that can be supplied.

MinArgs(n) An optional integer that specifies the minimum number of arguments that can be supplied.

Returns(type) Optional. 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) Optional keyword that specifies the order in which arguments are to be 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) Optional keyword that 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">) Optional and 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 DeclareFunction (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>)

Loads the DLL from the specified path.

path A quoted 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).

dll object<<Show Functions

Sends the declared functions for the DLL object to the log.

dll object<<Unload DLL

Unloads the DLL.

Images

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.
**Images JSL Syntax Reference**

```plaintext
img<<Crop(Left(pix), Right(pix), Top(pix), Bottom(pix))
```

Creates a new image from an existing image to the specified dimensions (in pixels).

```plaintext
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 Macintosh.

**Argument**

- **name** A quoted string contains the 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 smoothes 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 N Frames

Returns the number of frames in a multi-frame TIFF or animated GIF file, where the number of frames begins with frame 0.

#### Example

The following example places a four-frame TIFF file in a new window and shows the image that is in the first frame.

```plaintext
img = New Image( "$DOWNLOADS/Multiframe.tif" );
frames = 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("filepath")

Saves the image to the specified path and filename (filepath).

### img<<Scale(n, <n>)

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

```plaintext
img = New Image( "$SAMPLE_IMAGES/tile.jpg" );
x = 2;
img << Scale( x );
```
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

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 “img<<Get N Frames” on page 341 for an example.

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).

MATLAB

The MATLAB interfaces are scriptable using a MATLAB connection object. Use the MATLAB Connect() JSL function to obtain a scriptable MATLAB connection object. See the Extending JMP chapter in the Scripting Guide for details.

mlconn << Control( <named arguments> )

Controls the execution of MATLAB.

Returns

None.

Named Arguments

Echo(Boolean) Global. Echo MATLAB source lines to the JMP Log window.

Visible(Boolean) Global. 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, <named arguments> )

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.

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, 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 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.

Arguments
format Positional. The 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.

Arguments
None.
mlconn << Get( name )

Description
Gets a named variable from MATLAB to JMP.

Returns
Value of named variable.

Arguments
name Positional. 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.

Arguments
None.

mlconn << JMP Name To MATLAB Name( JMP name)

Maps a JMP variable name to its corresponding MATLAB variable name using MATLAB variable name naming rules.

Returns
String, a mapped MATLAB name.

Arguments
names 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.

Arguments
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

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”.

\[
\text{mlconn} \ll \text{Submit( mCode, <named arguments> )}
\]

Submits MATLAB code to the active global MATLAB integration interface connection.

Returns
0 if successful, otherwise nonzero.

Arguments
mCode  Positional, 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.

\[
\text{mlconn} \ll \text{Submit File( pathname )}
\]

Submits statements to MATLAB using a file pointed to by pathname.

Returns
0 if successful, otherwise nonzero.
Arguments
  Pathname  Positional, string. Pathname to file containing MATLAB source lines to be
            executed.

Platforms

obj<<Action
  Evaluates expressions. Useful for 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.

  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, {col 1, col 2, ...})
  Adds a control panel to a platform for switching variables.

obj<<Copy ByGroup Script
  Create a script to produce this analysis containing By variables and place it on the
  clipboard.

obj<<Copy Script
  Create 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.

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.

obj<<Get Timing

Times the launch of the platform.

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 <<Save Interactive HTML() 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

Ignores the current settings of the platform’s preferences.

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

Launches the platform again with the same options.

obj<<Redo ByGroup Analysis

Rerun this same analysis involving By groups in a new window.

obj<<Relaunch Analysis

Return to the launch window for this analysis involving By groups.

obj<<Relaunch ByGroup

Return to the launch window for this analysis.

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. For details, see the Display Trees chapter in the Scripting Guide.

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

Creates a script to produce the analysis involving By variables and saves it as a table property in the data table.

obj<<Save ByGroup Script to Journal

Creates a 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 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 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 Script Window**

Saves a script to reproduce analysis in the Script Journal.

**obj<<Scroll Window(x, y)**

Scrolls the window $x$ pixels to the left and $y$ pixels down from the current position. Negative coordinates go right 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(0|1)**

1 shows the window (brings it to the front). 0 hides the window. If the window is also minimized (on Windows) or docked (on Macintosh), 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**

Returns the 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.

**obj<<Zoom Window**

Resizes the window to be large enough to show all of its contents.

### Response Screening

**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

\texttt{obj<<Display Column Width(<Data Column(Column Table(n),"colname_path"), Row Label(Row Table(n), "colname_path"), <width>)}

Returns or sets the display pixel width of a column in a Tabulate table.

**Argument**

- **data column** Use Column Table and column references to define columns in the main body of the table.
- **row label** Use Row Table and heading for columns in the row labels area.
- **colname_path** An optional Column Table or Row Table, and the series of column headings that traces the path of the column. \textbf{Note:} Column table or row table can be omitted if the table referenced is the first table.
- **width** The pixel width of the column.

**Examples**

```julia
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
);
```
The R interfaces are also scriptable using an R connection object. A scriptable R connection object can be obtained using the `R Connect()` JSL function.

```julia
rconn << Control(Interrupt(1))
```

If `Async` is set to true (1) for `R Submit()`, this message immediately stops the execution of the R code that was submitted.

```julia
rconn << Disconnect()
```

Disconnects this R connection.

```julia
rconn << Is Connected()
```

Returns 1 if the R connection is active, 0 otherwise.

```julia
rconn << Send("name", named_arguments)
```

Send the specified JMP variable to R.

**Returns**

0 if successful, nonzero otherwise.

**Argument**

name A quoted string contains the name of a JMP variable to send to R.

**Arguments for Data Tables**

Selected(0|1) Optional, Boolean. If true, sends only the selected rows from the referenced data table to R.

Excluded(0|1) Optional, Boolean. If true, sends only the excluded rows from the referenced data table to R.

Labeled(0|1) Optional, Boolean. If true, sends only labeled rows from the referenced data table to R.

Hidden(0|1) Optional, Boolean. If true, sends only hidden rows from the referenced data table to R.

Colored(0|1) Optional, Boolean. If true, sends only colored rows from the referenced data table to R.

Markered(0|1) Optional, Boolean. If true, sends only markered rows from the referenced data table to R.

Row States(0|1, <named arguments>) Optional. 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

<table>
<thead>
<tr>
<th>Multiple row states are created by adding together individual settings.</th>
<th>Selected = 1</th>
<th>Excluded = 2</th>
<th>Labeled = 4</th>
<th>Hidden = 8</th>
<th>Colored = 16</th>
<th>Markered = 32</th>
</tr>
</thead>
</table>
| Arguments | Colors(Boolean) Optional, Boolean. If true, sends row colors and adds an additional data column named “RowStateColor”. | Markered(Boolean) Optional, Boolean. If true, sends row markers and adds an additional data column named “RowStateMarker”.

rconn<<Send("name")

Send the specified JMP data file to R.

rconn<<Get("name")

Gets the specified variable from R.

Returns

The value of the specified variable.

Arguments

name  A quoted string that contains the 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.

Arguments

type  The format the R graph display window contents are to be converted to. Valid formats are "png", "bmp", "jpeg", "jpg", "tiff", and "tif".

Table 3.2 Row States
rconn<<Submit("code", named_arguments)

Submits the R code.

Returns
0 if successful, nonzero otherwise.

Arguments
code A quoted string that contains the R code to submit.
Expand(0|1) Optional, Boolean. Performs an Eval Insert() on the R code before submitting the code.
Echo(0|1) Optional, Boolean. Echoes the R source lines to the JMP log. The default value is true.

Rconn<<Submit File("filepath")

Submits statements to R using the file in the "filepath".

Arguments
filepath A quoted string that contains the pathname to the file that contains R code to be executed.

rconn<<Execute({ list of inputs }, { list of outputs }, "code", named_arguments)

Submits the 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.

Arguments
{ 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.
code A quoted string that contains the R code to submit.
named_arguments See rconn<<Submit("code", named_arguments) on page 355.

rconn<<Control(named_arguments)

Controls the execution of R.

Returns
Void.

Argument
Echo(Boolean) Optional, 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 JMP Name to its corresponding R Name using R variable name naming rules.

Returns
    A string that contains the R name.

Arguments
    name  A quoted string that contains the name of a JMP variable to be sent to R.

SAS Integration Messages

Metadata Server Objects

metaserver<<Disconnect()

Function
    Disconnects the metadata server.

Returns
    Void.

metaserver<<Get Display Name()

Function
    Gets the display name of the metadata server.

Returns
    A string.

metaserver<<Get Host Name()

Function
    Gets the host (machine) name of the metadata server.

Returns
    A string.
**metaserver<<Get Port()**

*Function*

Gets the port used for the metadata server connection.

*Returns*

An integer.

**metaserver<<Get User Identity()**

*Function*

Gets the identify of the connected user as defined in metadata.

*Returns*

A string.

**metaserver<<Get User Name()**

*Function*

Gets the user name (login ID) that was used for the metadata server connection.

*Returns*

A string.

### SAS Server Objects

**sasconn<<Assign Libref("libref", "path", engine, engine options)**

*Function*

Assign a SAS libref on this SAS server connection.

*Returns*

Void.

*Arguments*

See “SAS Assign Lib Refs("libref", "path", <"engine">, <"engine options">)” on page 223 in the “JSL Functions” chapter.

**sasconn<<Cancel Submit()**

*Function*

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()  
**Function**  
Clears the SAS Log history for this server.  
**Returns**  
Void.

sasconn<<Clear Output History()  
**Function**  
Clears the SAS Output history for this server.  
**Returns**  
Nothing.

sasconn<<Connect(<named arguments>)  
**Function**  
Attempt to reconnect a SAS server connection object that has become disconnected.  
**Returns**  
1 if the connection was successful, 0 otherwise.  
**Named Arguments**  
All named arguments are optional.  
**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.  
**Prompt(Always|Never|IfNeeded)** A 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")  
**Function**  
De-assign a SAS libref on this SAS server connection.  
**Returns**  
Void.  
**Arguments**  
**libref** Quoted string that contains the library reference.
sasconn<<Disconnect()

Function
Disconnect this SAS server connection.

Returns
Void.

sasconn<Does Module Exist("moduleName")

Function
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.
Because MODEXIST is new for SAS 9.2, this function throws an exception if it is called for a
SAS connection that is not version SAS 9.2 or later.

Returns
1 if the specified module is found to exist, 0 if it does not exist.

Argument
moduleName The quoted SAS module, the existence of which should be checked. Do not
include any extension.

sasconn<<Export Data(dt, "library", "dataset", <named arguments>)

Function
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.

Arguments
See “SAS Export Data(dt, "library", "dataset", <named arguments>)” on page 225 in the
“JSL Functions” chapter.

sasconn<<Get Data Sets("libref")

Function
Returns a list of the data sets defined in a SAS library on this SAS server connection.

Returns
List of strings.
Arguments
libref  Quoted string that contains the 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()
Function
Gets the count of the number of errors encountered in the previous SAS Submit.
Returns
An integer.

sasconn<<Get File("source", "dest")
Function
Download a file from this SAS server connection.
Returns
Void.
Arguments
See “SAS Get File("source", "dest", "encoding")” on page 226 in the “JSL Functions” chapter.

sasconn<<Get File Names("fileref")
Function
Get a list of filenames found in the given "fileref" on this SAS server connection.
Returns
List of strings.
Arguments
fileref  Quoted string that contains the name of fileref from which to retrieve filenames.

sasconn<<Get File Names In Path("path")
Function
Get a list of filenames found in the given "path" on this SAS server connection.
Returns
List of strings.
Arguments
path  Quoted string that contains the directory path on the server from which to retrieve filenames.


sasconn<<Get File Refs()

Function
Get a list of the currently defined SAS filerefs on this SAS server connection.

Returns
List of strings.

sasconn<<Get Librefs(<named arguments>)

Function
Get a list of the currently defined SAS librefs on this SAS server connection.

Returns
List of strings.

Arguments
See “SAS Get Lib Refs(<named arguments>)” on page 228 in the “JSL Functions” chapter.

sasconn<<Get Log()

Function
Retrieve the SAS Log from the last SAS Submit from this SAS server connection.

Returns
String.

sasconn<<Get Option Name()

Function
Query SAS for the value of a SAS option variable.

Returns
String.

Example
The following script iterates through the define variables and prints out the values:

```julia
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()

Function
Retrieve the listing output from the last submission of SAS code to this SASServer object.

Returns
String.

sasconn<<Get Results()

Function
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.

sasconn<<Get Submit Status()

Function
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">)

Function
Returns information about the variables the specified SAS data set.

Arguments
libref Quoted string that contains the library reference to de-assign.
dataset Quoted string that contains the name of the data set from which to retrieve variable names.
Password("password") A quoted string that contains the password for the connection.

sasconn<<Get Var Names("libref", "dataset", <named arguments>)

Function
Retrieves the variable names contained in the specified data set on this SAS server connection.

Returns
List of strings.
Arguments
See “SAS Get Var Names(string, <"dataset">, <password("password")>)” on page 228 in the “JSL Functions” chapter.

sasconn<<Get Version(<"long">)
Function
Returns the SAS version as a string such as “9.1” or “9.2”.
Returns
A string that contains the SAS version.
Argument
long An optional 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()
Function
Returns the full path of the folder corresponding to the WORK library for this server.
Returns
A string that contains the work folder path.

sasconn<<Import Data("library", "dataset", <named arguments>)
Function
Import a SAS data set from this SAS server connection into a JMP table.
Returns
JMP Data Table object.
Arguments
See “SAS Import Data(string, <"dataset">, <named arguments>)” on page 229 in the “JSL Functions” chapter.

sasconn<<Import Query("sqlquery", <named arguments>)
Function
Execute the requested SQL query on this SAS server connection, importing the results into a JMP data table.
Returns
JMP data table object.
Arguments
   See “SAS Import Query("sqlquery", <named arguments>)” on page 231 in the “JSL Functions” chapter.

sasconn<<Is Connected()

Function
   Determine whether this SAS Server object is currently connected to SAS.

Returns
   1 if sasconn is connect, 0 otherwise.

sasconn<<Is Product Available("productName")

Function
   Determine 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.

Argument
   productName  The quoted SAS product for which licensing should be checked. The product name can be specified with or without the “SAS/” prefix.

Note
   The MODEXIST function is new in SAS 9.2. For SAS 9.1.3, this function only checks the license, not the installed status. In other words, for SAS 9.1.3, this function operates the same way as Is Product Licensed().

sasconn<<Is Product Licensed("productName")

Function
   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.
Argument

productName  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>)

Function
Immediately terminates the SAS connection.

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>)

Function
Download the file specified in "path" from the active SAS server connection and retrieve its contents as a string.

Returns
String.

Arguments
See “SAS Load Text File("path")” on page 232 in the “JSL Functions” chapter.

sasconn<<Open Log Window()

Function
Opens (or brings to the front) the SAS Log window for this server.

Returns
Void.

sasconn<<Open Output Window()

Function
Opens (or brings to the front) the SAS Output window for this server.

Returns
Void.
sasconn<<Open SAS Results()
Function
   Open 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()
Function
   Opens all the results from the last SAS Submit command.
Returns
   Void.

sasconn<<Send File("source", "dest")
Function
   Upload a file to this SAS server connection.
Returns
   Void.
Arguments
   See “SAS Send File("source", "dest", "encoding")” on page 233 in the “JSL Functions” chapter.

sasconn<<Submit(sasCode, <named arguments>)
Function
   Submit some SAS code to this SAS server connection.
Returns
   Void.
Arguments
   See “SAS Submit("sasCode", <named arguments>)” on page 233 in the “JSL Functions” chapter.

sasconn<<Submit File("filename", <named arguments>)
Function
   Submit a SAS code file to this SAS server connection.
Returns

Void.

Arguments

See “SAS Submit File(“filename”, <named arguments>)” on page 235 in the “JSL Functions” chapter.

Stored Processes

stp<<Begin Run(<named arguments>)

Function

Start 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.

Arguments

Same as Run, except AutoOpenResults and NoAlerts are not supported. They are available on EndRun.

AutoResume(<"filename">) Optional, quoted string. If specified with no argument, it specifies that the stored process results should be auto-opened when the stored process completes. If filename is specified, filename is opened rather than all results of the stored process being auto-opened.

AutoResumeScript("script") Optional, quoted string that specifies that after stored process execution completes, 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>)

Function

Delete all results from the execution of this stored process.
Returns
1 if deletion is successful, 0 otherwise (error message to JMP log).

Arguments
NoAlerts(0|1) Optional, Boolean. If True, the user is not prompted for confirmation before the attempt is made to delete results.
DeleteDirectory(0|1) Optional, 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()
Function
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>)
Function
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.
Arguments
AutoOpenResults(0|1) 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) Optional, an integer. 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(0|1) Optional, Boolean. If True, error messages are sent to the JMP log rather than message boxes. The default value is False.
stp<<Get Metadata Id()
Function
    Returns the metadata ID of the stored process.
Returns
    String.

stp<<Get Metadata Path()
Function
    Returns the full metadata path of the stored process.
Returns
    String.

Stp<<Get Name()
Function
    Returns the name of the stored process.
Returns
    String.

stp<<Get Param Enum Labels("name")
Function
    Get the enumeration labels for a parameter.
Returns
    List of strings.
Arguments
    name  Quoted string that contains the name of the parameter whose enumeration labels to retrieve.

stp<<Get Param Enum Values("name")
Function
    Get the possible enumerated values for a parameter.
Returns
    List of strings.
Arguments
    name  Quoted string that contains the name of the parameter whose possible enumerated values to retrieve.
stp<<Get Param Names(<named arguments>)

Function
Get a list of parameter names for this stored process of specific types.

Returns
List of strings.

Arguments
Visible(0|1) Optional, Boolean. If True, get only visible parameters. If False, get only non-visible parameters. If not specified, get both visible and non-visible parameters.

Modifiable(0|1) Optional, Boolean. If True, get only modifiable parameters. If False, get only non-modifiable parameters. If not specified, get both modifiable and non-modifiable parameters.

Required(0|1) Optional, Boolean. If True, get only required parameters. If False, get only non-required parameters. If not specified, get both required and non-required parameters.

Expert(0|1) Optional, Boolean. If True, get only expert parameters. If False, get only non-expert parameters. If not specified, get both expert and non-expert parameters.

stp<<Get Param Value("name")

Function
Get the current value of the specified parameter.

Returns
String.

Arguments
name Quoted string that contains the name of the parameter whose value to retrieve.

stp<<Get Results()

Function
Get the results generated by the execution of this stored process as a scriptable object.

Returns
SAS Results scriptable object.

stp<<Get Status()

Function
Get 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()
Function
Get the message associated with the failure of the stored process, if any.
Returns
String.

stp<<Reset Param Values()
Function
Reset all parameter values to their metadata-defined default values.
Returns
Void.

stp<<Run(<named arguments>)
Function
Execute 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.
Arguments
AutoOpenResults(0|1) Optional, 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") Optional, quoted string that contains the user name under which to run the stored process.
Password("password") Optional, quoted string that contains the password for Username.

AuthDomain("authDomain") Optional, quoted string that contains the authentication domain of the credentials (username, password) given.

ODSDest("dest") Optional, quoted string that contains an 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") Optional, quoted string that contains the 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("styleType") Optional, quoted string that contains an ODS style to apply to the results. This requires the stored process SAS code to call %STPBEGIN. There is no default value.

ODSStyleSheet("cssFile") Optional, quoted string that contains the full 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(0|1) Optional, 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")

Function
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 Quoted string that contains the name of the parameter whose value to set.
value Quoted string that contains the value to which to set the parameter.

stp<<Set Results Directory("directory")

Function
Sets the directory on the client machine where stored process results are placed.

Returns
String.

Arguments
directory Quoted string that contains the full 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

results<<Delete All Result Files()

Function

   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()

Function

   Gets the directory where the results generated by the stored process or SAS submit are located.

Returns

   String.

results<<Get Log()

Function

   Get the SAS Log from the execution of the stored process or SAS submit.

Returns

   String.

results<<Get Main Result File Name(<named arguments>)

Function

   Gets the full path of the main result file generated by the stored process or SAS submit.

Returns

   String.

Arguments

   FullPath(0|1) Optional, Boolean. If True, the main result filename is returned as a full path. The default value is False.

results<<Get Output()

Function

   Gets the SAS Listing output from the execution of the stored process or SAS submit.

Returns

   String.
results<<Get Output Datasets()

Function
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>)

Function
Get information about result files that were generated by the execution of the stored process or SAS submit.

Returns
List of two lists of strings. The first list is filenames, and the second list is the MIME-type of the corresponding file from the first list.

Arguments
MIMEType("mime_type") Optional, quoted string that restricts the set of files for which information is returned to only those files with the specified MIME-type. If not specified, information about all generated files is returned.
FullPath Optional, 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()

Function
Parses the ODS XML results and creates a JMP report.

Returns
The display box for the report.

results<<Open All Results()

Function
Opens all results generated by the execution of the stored process or SAS submit.

Returns
Void.

results<<Open Result File("filename", <named arguments>)

Function
Attempts to open the result file with the given name.
Returns
JMP Data Table if one was opened.

Arguments
filename Quoted string that contains the 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.

RunScript(0|1) Optional, Boolean. If True, and if filename is a JSL script, the script is executed. If False, filename is just opened, even if it is JSL.

```
results<<Run Script("filename")
```

Function
Looks for a JSL file in the results with the given filename and runs it if it finds it.

Returns
Void.

Arguments
filename Quoted string that contains the name of the JSL file from the generated results to open. Filename should just be the name of the file, not the full path, and it does not need to include the .jsl extension.

---

**Schedule**

See also “Schedule(n, script)” on page 276 in the “JSL Functions” chapter.

For more information about scheduling actions, see the Programming chapter in the Scripting Guide.

```
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.

Sockets

skt<<Accept(<callback, timeout>)

Function
    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 string that echoes the command ("accept"). The second is a string, either "ok" or an error. The third is a 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.

Arguments
    callback  an optional argument that 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)

Function
    Associates a port on the local machine with the socket.

Returns
    A list of two strings. The first string is the command name ("bind") and the second is "ok" if successful or an error.

Argument
    localhost  Specifies the local machine. You cannot bind to another machine.
    port  The port that should be used.

skt<<Close()

Function
    Closes a socket.
**skt<<Connect(socketname, port)**

**Function**

Connects to a listening socket.

**Returns**

A list of two 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 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 the port of the other socket to connect through.

**skt<<GetPeerName()**

**Function**

Retrieves the address and port of the socket at the other end of the connection.

**Returns**

A list of four 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<<GetSockName()**

**Function**

Retrieves the address and port of the socket at this end of the connection.

**Returns**

A list of four 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, 1)**

**Function**

Controls the socket’s blocking behavior.

**Returns**

A list of two 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. 1 turns on the behavior and the argument.

skt<<Listen()

Function
Tells the server socket to listen for connections.

Returns
A list of two 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>)

Function
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 strings. The first string is the command name ("recv" or "recvfrom"). 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.

Arguments
n specifies the number of bytes to receive from the other socket.
callback an optional argument that 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)

Function
Sends the data in the argument to the other socket. Send sends a stream and sendto sends a datagram.

Returns
A list of three 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  the command to send to the other socket.

dgram  the command to send to the other socket.

Note
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

---

obj<<Custom SQL(sql)

Changes the query to a custom SQL query and sets the SQL.

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 the JSL script that runs after the query runs each time.

obj<<Query Name(<newName>)

Gets (without the newName argument) or sets (with the newName 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>, <UpdateTable(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).
Arguments

private  Optional. Opens the data table that the query produces without displaying it in a data table window. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful if the data requires a lot of memory.

invisible  Optional. Hides the data table that the query produces. Use this argument to keep the query result hidden but use it in a subsequent query.

UpdateTable  Optional. Updates the specified data table. Runs the query in the foreground.

OnRunComplete  Optional. Specifies a script to run after the query is complete.

OnRunCanceled  Optional. Specifies a script to run after the user cancels the query.

OnError  Optional. 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.

```julia
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.

```julia
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.

```julia
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;WID=D79255;DATABASE=SQB;" ),
   QueryName( "movies_to_update" ),
   Select( Column( "YearMade", "t1" ), Column( "Rating", "t1" ) ),
```
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).

**Arguments**
- **OnRunComplete** Optional. Specifies a script to run after the query is complete. To get the resulting data table, include OnRunComplete.
- **private** Optional. 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. Using private data tables saves memory for smaller tables. However, for large tables (for example, 100 columns and 1,000,000 rows), using a private data table is not helpful because the data requires a lot of memory.
- **invisible** Optional. Hides the data table. Use this argument to keep the query result hidden but use it in a subsequent query.
- **OnRunCanceled** Optional. Specifies a script to run after the user cancels the query.
- **OnError** Optional. 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.

```julia
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 “Run Background(<OnRunComplete(script), <private|invisible>>, <OnRunCanceled(script)>, <OnError(script)>>” on page 381 for details about the arguments.

---

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, <ReplaceExisting(Boolean)>)

Saves the query to the specified file. If the file already exists, the save fails unless Replace Existing is true.

---

Other Objects

Zip Archives

---

list = za<<dir

Returns a list of member names

---

data = za<<read(membername, <format(blob)>)

Returns a string that contains the entire member data. If the optional argument is specified, the message returns a blob.

Note

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(membername, textstring|blobdata)

Writes a text or blob to a zip archive member file. If the specified membername isn’t in the current zip file, the returned actualname is the same as membername. This member name will be changed to prevent overwriting an existing member; the name actually used is returned. For example,

name1 = za<<write(name, blobdata)

---

Journals

---

jnl<<Save HTML(<"path">, <"format">)

Saves the journal as HTML.
path  A quoted string that contains the filepath for the saved HTML file (for example, "c:/myFile.html").

format The quoted graphic file format. JPG, PNG, and TIFF formats are supported. The graphics are saved in a subdirectory named gfx.

\[\text{jnl<<Save RTF(<"path">, <"format">)}\]
Saves the journal as an RTF file.

\textbf{Arguments}

path  A quoted string that contains the filepath for the saved RTF file (for example, "c:/myFile.rtf").

format The quoted file format for the embedded graphics. JPG, PNG, and TIF formats are supported.

\[\text{jnl<<Save PDF(<"path">, <Show Page Setup(0|1)>, <Portrait(0|1)>)}\]
Saves the journal as a PDF file.

\textbf{Arguments}

path  A quoted string that contains the filepath 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.

```julia
dt = Open( "$SAMPLE_DATA/Big Class.jmp" );
Open( 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:

```julia
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 for details.
## Numeric SQL Functions

<table>
<thead>
<tr>
<th>Numeric Function</th>
<th>Native SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS( <code>number</code>)</td>
<td></td>
<td>Returns the absolute value of the specified number.</td>
</tr>
<tr>
<td>ACOS( <code>cosine</code>)</td>
<td></td>
<td>Returns the angle in radians for the specified cosine.</td>
</tr>
<tr>
<td>ASIN( <code>sin</code>)</td>
<td></td>
<td>Returns the angle in radians for the specified sine.</td>
</tr>
<tr>
<td>ATAN( <code>tangent</code>)</td>
<td></td>
<td>Returns the angle in radians for the specified tangent.</td>
</tr>
<tr>
<td>ATAN2( <code>x</code>, <code>y</code>)</td>
<td>Two-argument arctangent function.</td>
<td></td>
</tr>
<tr>
<td>CEILING( <code>number</code>)</td>
<td>Returns the smallest integer larger than the specified number.</td>
<td></td>
</tr>
<tr>
<td>CEIL( <code>number</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COS( <code>radians</code>)</td>
<td></td>
<td>Returns the cosine of the specified angle in radians.</td>
</tr>
<tr>
<td>COT( <code>radians</code>)</td>
<td></td>
<td>Returns the cotangent of the specified angle in radians.</td>
</tr>
<tr>
<td>DEGREES( <code>radians</code>)</td>
<td>Converts an angle in radians to an angle in degrees.</td>
<td></td>
</tr>
<tr>
<td>EXP( <code>number</code>)</td>
<td></td>
<td>Returns the constant e raised to the specified power.</td>
</tr>
<tr>
<td>FLOOR( <code>number</code>)</td>
<td></td>
<td>Returns the largest integer smaller than the specified number.</td>
</tr>
<tr>
<td>LN( <code>number</code>)</td>
<td></td>
<td>Returns the natural logarithm of the specified number.</td>
</tr>
<tr>
<td>LOG( <code>number</code>)</td>
<td></td>
<td>Returns the common logarithm of the specified number.</td>
</tr>
<tr>
<td>LOG10( <code>number</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX( <code>n1</code>, <code>n2</code>, ...)</td>
<td>Yes</td>
<td>Returns the largest of the specified numbers. A minimum of two numbers must be specified.</td>
</tr>
<tr>
<td>MIN( <code>n1</code>, <code>n2</code>, ...)</td>
<td>Yes</td>
<td>Returns the smallest of the specified numbers. A minimum of two numbers must be specified.</td>
</tr>
<tr>
<td>MOD( <code>dividend</code>, <code>divisor</code>)</td>
<td></td>
<td>Returns the remainder when <code>dividend</code> is divided by <code>divisor</code>. Floating-point values are truncated to integers before the modulus operation is performed.</td>
</tr>
<tr>
<td>PI()</td>
<td></td>
<td>Returns the value of the constant pi (π).</td>
</tr>
<tr>
<td>POWER( <code>number</code>, <code>power</code>)</td>
<td>Raises <code>number</code> to the specified <code>power</code>.</td>
<td></td>
</tr>
<tr>
<td>POW( <code>number</code>, <code>power</code>)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RADIANS( <code>degrees</code>)</td>
<td></td>
<td>Converts an angle in degrees to an angle in radians.</td>
</tr>
</tbody>
</table>
## JSL Syntax Reference 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:

```sql
Query( Scalar, "SELECT CURRENT_TIMESTAMP;" );
```

returns:

<table>
<thead>
<tr>
<th>Numeric Function</th>
<th>Native SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RANDOM()</code></td>
<td></td>
<td>Returns a random number. <code>RANDOM()</code> returns a number between 0 and 1. <code>RANDOM( max )</code> returns a number between 0 and <code>max</code>. <code>RANDOM( min, max )</code> returns a number between <code>min</code> and <code>max</code>. This function is equivalent to the <code>Random Uniform()</code> JSL function, and its seed can be controlled using the <code>Random Reset()</code> JSL function. <code>RANDOM</code> can be shortened to <code>RAND</code>.</td>
</tr>
<tr>
<td><code>RAND</code></td>
<td></td>
<td>Returns a number between 0 and 1.</td>
</tr>
<tr>
<td><code>RAND( max )</code></td>
<td></td>
<td>Returns a number between 0 and <code>max</code>.</td>
</tr>
<tr>
<td><code>RAND( min, max )</code></td>
<td></td>
<td>Returns a number between <code>min</code> and <code>max</code>.</td>
</tr>
<tr>
<td><code>RANDOMBLOB( length )</code></td>
<td>Yes</td>
<td>Returns an N-byte blob that contains pseudo-random bytes. See the SQLite Online documentation at <a href="https://www.sqlite.org/lang.html">https://www.sqlite.org/lang.html</a> for details.</td>
</tr>
<tr>
<td><code>ROUND( number, &lt;precision&gt; )</code></td>
<td></td>
<td>Rounds <code>number</code> to the number of decimal places given by <code>precision</code>. The default value of <code>precision</code> is 0, and <code>precision</code> can be negative.</td>
</tr>
<tr>
<td><code>SIGN( number )</code></td>
<td></td>
<td>Returns 1 if <code>number</code> is positive, -1 if <code>number</code> is negative, or 0 if <code>number</code> is zero.</td>
</tr>
<tr>
<td><code>SIN( radians )</code></td>
<td></td>
<td>Returns the sin of the specified angle in radians.</td>
</tr>
<tr>
<td><code>SQRT( number )</code></td>
<td></td>
<td>Returns the square root of <code>number</code>.</td>
</tr>
<tr>
<td><code>TAN( radians )</code></td>
<td></td>
<td>Returns the tangent of the specified angle in radians.</td>
</tr>
<tr>
<td><code>TRUNCATE( number, &lt;precision&gt; )</code></td>
<td></td>
<td>Truncates <code>number</code> at the number of decimal places given by <code>precision</code>. The default value of <code>precision</code> is 0, and <code>precision</code> can be negative. <code>TRUNCATE()</code> can be shortened to <code>TRUNC()</code>.</td>
</tr>
</tbody>
</table>
"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:

```julia
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:

```julia
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.

<table>
<thead>
<tr>
<th>Date-Time Function</th>
<th>Naive SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CURRENT_DATE</code></td>
<td>Yes</td>
<td>Returns the current date (UTC/GMT) as a SQLite time string.</td>
</tr>
<tr>
<td><code>CURRENT_TIME</code></td>
<td>Yes</td>
<td>Returns the current time (UTC/GMT) as a SQLite time string.</td>
</tr>
<tr>
<td><code>CURRENT_TIMESTAMP</code></td>
<td>Yes</td>
<td>Returns the current date and time (UTC/GMT) as a SQLite time string.</td>
</tr>
<tr>
<td><code>DATEDIFF</code></td>
<td></td>
<td>Computes the difference between two dates in units specified by <code>interval</code>, based on <code>alignment</code>. This function works the same as the <code>Date Difference()</code> JSL function. Valid values for <code>interval</code> are: “Year”, “Quarter”, “Month”, “Week”, “Day”, “Hour”, “Minute” and “Second”. Valid values for <code>alignment</code> are “Start”, “Actual” and “Fractional”. If <code>alignment</code> is not specified, “Start” is used.</td>
</tr>
</tbody>
</table>
### SQL Functions Available for JMP Queries

#### Date-Time SQL Functions

<table>
<thead>
<tr>
<th>Date-Time Function</th>
<th>Naive SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EXTRACT( datepart, datetime, &lt;use_locale = 1&gt; )</strong></td>
<td></td>
<td>Extracts a specific part of a date or date-time value. <strong>Datetime</strong> is a JMP date-time value or a SQLite time string. <strong>Use_locale</strong> is optional and applies only to date name parts such as &quot;MonthName&quot; and &quot;DayName&quot; and determines whether values from the current language or English are returned. The following values of <strong>datepart</strong> are supported:</td>
</tr>
<tr>
<td>&quot;Year&quot;</td>
<td></td>
<td>Returns the year as a number.</td>
</tr>
<tr>
<td>&quot;Month&quot;</td>
<td></td>
<td>Returns the numeric month (1-12).</td>
</tr>
<tr>
<td>&quot;MonthName&quot;</td>
<td></td>
<td>Returns the full name of the month in the current language (<strong>use_locale = 1</strong>) or English (<strong>use_locale = 0</strong>).</td>
</tr>
<tr>
<td>&quot;Mon&quot;, &quot;MMM&quot;</td>
<td></td>
<td>Returns the abbreviated name of the month.</td>
</tr>
<tr>
<td>&quot;Day&quot;</td>
<td></td>
<td>Returns the day of the month (1-31).</td>
</tr>
<tr>
<td>&quot;DayName&quot;</td>
<td></td>
<td>Returns the name of the day of the week.</td>
</tr>
<tr>
<td>&quot;DayOfWeek&quot;</td>
<td></td>
<td>Returns the numeric day of the week (1-7).</td>
</tr>
<tr>
<td>&quot;DayOfYear&quot;</td>
<td></td>
<td>Returns the numeric day of the year (1-366).</td>
</tr>
<tr>
<td>&quot;Quarter&quot;</td>
<td></td>
<td>Returns the numeric quarter (1-4).</td>
</tr>
<tr>
<td>&quot;Hour&quot;</td>
<td></td>
<td>Returns the hour (0-23).</td>
</tr>
<tr>
<td>&quot;Minute&quot;</td>
<td></td>
<td>Returns the minute (0-59).</td>
</tr>
<tr>
<td>&quot;Second&quot;</td>
<td></td>
<td>Returns the seconds, including any fractional part.</td>
</tr>
<tr>
<td>&quot;Date&quot;</td>
<td></td>
<td>Returns just the date portion of a date-time value as a JMP date-time value.</td>
</tr>
<tr>
<td>&quot;Time&quot;</td>
<td></td>
<td>Returns just the time portion of a date-time value as a JMP date-time value.</td>
</tr>
<tr>
<td><strong>JMPDATE( SQLite time string )</strong></td>
<td></td>
<td>Converts a SQLite time string to the equivalent JMP date-time value.</td>
</tr>
<tr>
<td><strong>NOW()</strong></td>
<td></td>
<td>A synonym for <strong>TODAY()</strong>.</td>
</tr>
<tr>
<td><strong>TODAY()</strong></td>
<td></td>
<td>Returns the JMP date-time value of the current moment in local time, which matches the JMP <strong>Today()</strong> function.</td>
</tr>
</tbody>
</table>
## String SQL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Native SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEX( binary )</td>
<td>Yes</td>
<td>SQLite built-in function that converts a BLOB to a string of hexadecimal characters. Useful when paired with the RANDOMBLOB() function.</td>
</tr>
<tr>
<td>JLEFT( string, len, &lt;pad&gt; )</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>JRIGHT( string, len, &lt;pad&gt; )</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>LENGTH( string )</td>
<td>Yes</td>
<td>SQLite equivalent of the ANSI standard CHAR_LENGTH() function. Returns the length of its string argument in characters.</td>
</tr>
<tr>
<td>LOCATE( string1, string2 )</td>
<td></td>
<td>Returns the (1-based) starting position of string1 within string2, returning 0 if string1 is not found within string2.</td>
</tr>
<tr>
<td>POSITION( string1, string2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOWER( string )</td>
<td></td>
<td>Returns a copy of string with all uppercase characters converted to lowercase.</td>
</tr>
<tr>
<td>LTRIM( string, &lt;trimchars&gt; )</td>
<td>Yes</td>
<td>Trims any characters contained in trimchars from the beginning of string and returns the result. If trimchars is omitted, spaces are trimmed.</td>
</tr>
<tr>
<td>PRINTF( format, &lt;arg1, ..., argN&gt; )</td>
<td>Yes</td>
<td>Allows constructing strings using placeholders and arguments. See the SQLite Online documentation at <a href="https://www.sqlite.org/lang.html">https://www.sqlite.org/lang.html</a> for details.</td>
</tr>
<tr>
<td>REPLACE( string, find, replace )</td>
<td>Yes</td>
<td>Replaces all instances of find in string with replace and returns the result. If replace is numeric, it is converted to a string.</td>
</tr>
<tr>
<td>REVERSE( string )</td>
<td></td>
<td>Returns a copy of string with the order of the characters reversed.</td>
</tr>
</tbody>
</table>
Appendix A
JSL Syntax Reference

SQL Functions Available for JMP Queries
Aggregate SQL Functions

### 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.

---

<table>
<thead>
<tr>
<th>Function</th>
<th>Native SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RTRIM( string, &lt;trimchars&gt; )</code></td>
<td>Yes</td>
<td>Trims any characters contained in <code>trimchars</code> from the end of <code>string</code> and returns the result. If <code>trimchars</code> is omitted, spaces are trimmed.</td>
</tr>
<tr>
<td><code>SPACE( length )</code></td>
<td></td>
<td>Returns a string consisting of <code>length</code> space characters.</td>
</tr>
<tr>
<td><code>SUBSTR( string, start, &lt;length&gt; )</code></td>
<td>Yes</td>
<td>Returns the substring of <code>string</code> starting at <code>start</code> (1-based) that is <code>length</code> characters long. If <code>length</code> is omitted, the substring starting at <code>start</code> and continuing to the end of <code>string</code> is returned.</td>
</tr>
<tr>
<td><code>TRIM( string, &lt;trimchars&gt; )</code></td>
<td></td>
<td>Trims any characters contained in <code>trimchars</code> from the end of <code>string</code> and returns the result. If <code>trimchars</code> is omitted, spaces are trimmed.</td>
</tr>
<tr>
<td><code>UPPER( string )</code></td>
<td></td>
<td>Returns a copy of <code>string</code> with all lowercase characters converted to uppercase.</td>
</tr>
</tbody>
</table>

### System SQL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>COALESCE( arg1,..., argN )</code></td>
<td>Yes</td>
<td>Returns the first argument passed to it that is non-NULL. Returns NULL if all arguments are NULL. Requires at least two arguments.</td>
</tr>
<tr>
<td><code>IFNULL( arg1, arg2 )</code></td>
<td>Yes</td>
<td>Returns <code>arg1</code> if not NULL, otherwise <code>arg2</code>. Basically, <code>IFNULL</code> is a two-argument version of <code>COALESCE()</code>.</td>
</tr>
<tr>
<td><code>NULLIF( arg1, arg2 )</code></td>
<td>Yes</td>
<td>Returns <code>arg1</code> if <code>arg1</code> and <code>arg2</code> 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.</td>
</tr>
</tbody>
</table>
### Aggregate SQL Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>SQLite</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AVG( num_expr )</strong></td>
<td></td>
<td>Computes the average of num_expr for the rows in the group. num_expr must be numeric.</td>
</tr>
<tr>
<td><strong>COUNT( expr )</strong></td>
<td></td>
<td>Counts the number of times expr is not NULL in the group. COUNT( * ) returns the total number of rows in the group.</td>
</tr>
<tr>
<td><strong>GROUP_CONCAT( expr, separator = ',' )</strong></td>
<td>Yes</td>
<td>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.</td>
</tr>
<tr>
<td><strong>MAX( expr )</strong></td>
<td></td>
<td>Returns the maximum value of expr in the group. Expr can be character or numeric.</td>
</tr>
<tr>
<td><strong>MIN( expr )</strong></td>
<td></td>
<td>Returns the minimum value of expr in the group. Expr can be character or numeric.</td>
</tr>
<tr>
<td><strong>STDDEV_POP( num_expr )</strong></td>
<td></td>
<td>Computes the population standard deviation of num_expr for the group.</td>
</tr>
<tr>
<td><strong>STDDEV_SAMP( num_expr )</strong></td>
<td></td>
<td>Computes the sample standard deviation of all num_expr for the group.</td>
</tr>
<tr>
<td><strong>SUM( num_expr )</strong></td>
<td></td>
<td>Returns the sum of num_expr for the group. If no non-NULL values are found, SUM() returns NULL.</td>
</tr>
<tr>
<td><strong>TOTAL( num_expr )</strong></td>
<td>Yes</td>
<td>Same as SUM( num_expr ), except TOTAL() returns 0.0 if no non-NULL values are found.</td>
</tr>
<tr>
<td><strong>VAR_POP( num_expr )</strong></td>
<td></td>
<td>Computes the population variance of num_expr for the group.</td>
</tr>
<tr>
<td><strong>VAR_SAMP( num_expr )</strong></td>
<td></td>
<td>Computes the sample variance of num_expr for the group.</td>
</tr>
</tbody>
</table>
Symbols

- 267, 279
-- 28
; 264
: 187
:: 187
:| 63
! 62
!= 57
* 268
*== 27
/* */ 51
// 51, 260
//! 52
//== 27
^ 252
+ 257
++ 28
+= 26
< 55
< ... <= 55
< <= 53
<= 56
<= ... < 56
<= < 53
-= 29
= 27
== 53
> 54
>= 54
| 63
|| 32
||= 33

A

Abbrev Date 65
Abs 164

Add 257
Add Color Theme 119
Add To 26
All 147
Alpha Shape 75
alpha shape messages 282
And 57
AndMZ 57
AndV3 57
Any 147
Arc 120
Arc Cos H 254
Arc Cosine 255
ArCos 255
ArcSine 255
ArcSinH 255
ArcTan 255
ArcTangent 255
ArcTanH 256
Arg 95
Arg Expr 95
ARIMA Forecast 238
Arrhenius 248
Arrhenius Inv 249
Arrow 120
ArSin 255
As Column 187
As Constant 187
As Date 65
As Global 187
As List 134, 188
As Name 188
As Namespace 188
As Row State 215
As SAS Expr 218
As Scoped 188
As SQL Expr 236
As Table 211
Assign 27
assignment functions 26–29
Associative Array 188
ATan 255
ATangent 255
axis box messages 317–320

B
Back Color 120
Batch Interactive 257
Beep 258
Best Partition 238
Beta 249
Beta Binomial Distribution 72
Beta Binomial Probability 72
Beta Binomial Quantile 72
Beta Density 170
Beta Distribution 171
Beta Quantile 171
Binomial Distribution 72
Binomial Probability 73
Binomial Quantile 73
Blob MD5 258
Blob Peek 258
Blob to Char 30
Blob To Matrix 30
Border Box 75
border box messages 320
Break 58
Build Information 258
Button Box 76

C
Calendar Box 76
Caption 259
Cauchy Density 171
Cauchy Distribution 171
Cauchy Quantile 172
CDF 147
Ceiling 165
Char 30
Char to Blob 31
Char to Hex 32
Char To Path 121
character functions 29–41
character pattern functions 42–51
Check Box 77
ChiSquare Density 172
ChiSquare Distribution 172
ChiSquare Log CDistribution 172
ChiSquare Log Density 172
ChiSquare Log Distribution 172
ChiSquare Noncentrality 173
ChiSquare Quantile 173
Chol Update 147
Cholesky 147
Choose 58
Circle 121
Clear Globals 189
Clear Log 189
Clear Symbols 189
Close 98
Close All 98
Close Log 98
Close Side Panel 284
Col Cumulative Sum 238
Col List Box 78
Col Max 239
Col Maximum 239
Col Mean 239
Col Min 239
Col Minimum 239
Col Moving Average 240
Col N Missing 241
Col Number 241
Col Quantile 241
Col Rank 242
Col Shuffle 204
Col Standardize 243
Col Std Dev 243
Col Stored Value 211
Col Sum 244
Collapse Whitespace 32
Color Of 215
Color State 216
Color to HLS 122
Color to RGB 122
Column 211
Column Dialog 79
column messages 300–304
Column Name 212
Combine States 216
Combo Box 79
comment functions 51
comparison functions 52
comparison operators 52
Concat 32
Concat Items 135
Concat To 33
conditional and logical functions 57–64
connections, SAS metadata server objects 356
constant functions 64
Contains 34
Contains Item 34
Context Box 79
Continue 58
Contour 123
Contour Function 123
Convert File Path 98
Copy Directory 99
Copy File 99
Copy Picture 310
Correlation 148
Cos 256
Cosh 256
Cosine 256
Count 212
Covariance 148
Create Database Connection 99
Create Directory 100
Creation Date 100
Cumulative Sum 238
Current Data Table 213
Current Journal 79
Current Metadata Connection 218
Current Report 80
Current SAS Connection 219
Current Window 80

D
Data Browser Box 321
data filter messages 305–308
data grid box messages 321
Data Table 213
data table messages 283–299
database messages 308
Datafeed 268
datafeed messages 308
Date Difference 65
Date DMY 66
Date Increment 66
Date MDY 67
date-time functions 64–72
date-time operators 64
Day 67
Day of Week 67
Day of Year 68
DDB, Microsoft Excel 114
Debug Break 259
debug run 52
debug step 52
Decode64 Double 259
Delete Database Connection 99
Delete Directory 101
Delete File 101
Delete Globals 190
Delete Symbols 190
Derivative 165
Design 149
Design F 149
Design Nom 149
Design Ord 150
DesignF 149
Desirability 168
Det 151
Diag 151
Dialog 80
deprecation 86
New Window() 86
Dif 213
Digamma 249
Direct Product 151
Directory Exists 101
discrete probability functions 72–75
display box messages 309–337
display box messages for all boxes 309–317
display functions 75–95
Distance 151
Divide 260
Divide To 27
DLL messages 337
Double Declining Balance 114
DPI 314
Drag Line 124
Drag Marker 124
Drag Polygon 124
Drag Rect 125
Drag Text 125
Dunnett P Value 173
Dunnett Quantile 173

E

e 64
E Div 152
E Mult 152
Eigen 152
eliding operator 53
Empty 260
Encode64 Double 260
Ends With 35
Equal 53
equality operators 52
Eval 190
Eval Expr 95
Eval Insert 190
Eval Insert Into 191
Eval List 135
Excerpt Box 80
Excluded 216
Excluded State 216
Exit 191
Exp 249–250
Expr 96
Expr As Picture 80
expression functions 95–97
Extended Attributes, SAS 226
Extract Expr 96

F

F Density 174
F Distribution 174
F Log CDistribution 174
F Log Density 174
F Log Distribution 174
F Noncentrality 174
F Power 174
F Quantile 174
F Sample Size 175
Factorial 250
Faure Quasi Random Sequence 261
FFT 250
File Exists 101
file functions 98–113
Files in Directory 101
Fill Color 126
Fill Pattern 126
financial functions 114–118
First 191
Fit Censored 244
Fit Transform To Normal 251
Floor 165
For 59
For Each Row 59
Format 68
Format Date 68
frame box messages 321
Frechet Density 175
Frechet Distribution 175
Frechet Quantile 175
Function 192
Future Value 114
FV, Microsoft Excel 114

G

G Inverse 153
Gamma 251
Gamma Density 175
Gamma Distribution 176
Gamma Log CDistribution 176
Gamma Log Density 176
Gamma Log Distribution 176
Gamma Poisson Distribution 73
Gamma Poisson Probability 73
Gamma Poisson Quantile 74
Gamma Quantile 176
GenGamma Density 177
GenGamma Distribution 177
GenGamma Quantile 177
Get Addin 261
Get Addins 261
Get Addr Info 261
Get Clipboard 261
Get Current Directory 102
Get Default Directory 102
Get Environment Variable 192
Get Excel Worksheets 103
Get File Search Path 103
Get Log 192
Get Name Info 261
Get Path Variable 103
Get Platform Preference 262
Get Platform Preferences 262
Get Preference 263
Get Preferences 263
Get SAS Version Preference 219
Global Box 81
GLog Density 177
GLog Distribution 177
GLog Quantile 177
Glue 264
Gradient Function 126
Graph 81
Graph 3D Box 81
Graph Box 81
graphic functions 119–134
graphics resolution 314
Greater 54
Greater or Equal 54

H
H Center Box 82
H Direct Product 153
H Line 126
H List Box 82, 95
H Sheet Box 82
H Size 126
Handle 127
Head 96
Head Expr 96
Head Name 96
Head Name Expr 96
Heat Color 127
Hex 32
Hex to Blob 35
Hex to Char 35
Hex to Number 36
Hidden 217
Hidden State 217
Hier Box 83
Hier Clust 245
HLS Color 127
Host Is 264
Hough Line Transform 153
Hour 68
HTML
importing 107
save as interactive 313
HTML, save as 382
Hue State 217
Hypergeometric Distribution 74
Hypergeometric Probability 74

I
Icon Box 83
Identity 153
If 60
If Box 83
IfMax 60
IfMin 60
IfMZ 60
IfV3 60
IGamma 176
image messages 339
image resizing 341
importing data 106
In Days 69
In Format 69
In Hours 69
In Minutes 69
In Path 127
In Polygon 128
In Weeks 70
In Years 70
Include 193
Include File List 193
Index 153
Insert 135
Insert Into 136
interactive HTML, saving 313
Interest Payment 114
Interest Rate 115
Internal Rate of Return 115
Interpolate 61
Inv 154
Inv Update 154
Inverse 154
Invert Expr 166

Index
JSL Syntax Reference
J

J 154
JMP Product Name 265
JMP Version 265
JMP6 SAS Compatibility Mode 219
Johnson Sb Density 178
Johnson Sb Distribution 178
Johnson Sb Quantile 178
Johnson Sl Density 178
Johnson Sl Distribution 179
Johnson Sl Quantile 179
Johnson Su Density 179
Johnson Su Distribution 180
Johnson Su Quantile 180
Journal Box 84
journal messages 382
JSL Set JVM Version 276
JVM 276

K

KDE 245
KDTable 155

L

Labeled 217
Labeled State 217
Lag 214
Last Modification Date 104
Left 36
Length 37
LenthPSE 245
Less 55
Less LessEqual 55
Less or Equal 56
LessEqual Less 56
LEV Density 180
LEV Distribution 180
LEV Quantile 181
Level Color 128
LGamma 251
Line 128
Line Style 129
Lineup Box 84
List 136
List Box 84
Ln 251
LnZ 251
Load DLL 266
Load Text File 104
Loc 156
Loc Max 156
Loc Min 156
Loc Sorted 157
Local 194
Local Here 194
Lock Globals 194
Lock Symbols 194
Log 252
Log10 252
Log1P 252
LogCapture 194
LogGenGamma Density 181
LogGenGamma Distribution 181
LogGenGamma Quantile 181
Logistic Density 181
Logistic Distribution 181
Logistic Quantile 181
Logit 252
Loglogistic Density 182
Loglogistic Distribution 182
Loglogistic Quantile 182
Lognormal Density 182
Lognormal Distribution 182
Lognormal Quantile 182
Long Date 70
Lowercase 37
LPSolve 169

Maximum 246
MDYHMS 70
Mean 246
messages
alpha shapes 282
associative arrays 282
axis box 317–320
axis boxes 317
border boxes 320
columns 300–304
data filter 305–308
data grid boxes 321
data tables 283–299
databases 308
datafeed 308
display boxes 309–337
display boxes (for all boxes) 309–317
DLL 337
frame boxes 321
images 339
journals 382
MATLAB 342–346
matrix boxes 323
nom axis boxes 324
number col boxes 324
outline boxes 325
panel boxes 326
platforms 346–351
plot col boxes 326
R 353–356
range slider box 326
Response Screening 351
rows 304
SAS 356–375
SAS results 373
SAS server objects 357
schedule 375
slider box 326
sockets 376
SQL 379
stored processes (SAS) 367
string col boxes 328
tab boxes 328
table boxes 329
Tabulate 352
text boxes 331
tree boxes 332
  tree nodes 332
  triangulation 334
windows 335
zip archives 382
Meta Connect 219
Meta Create Profile 220
Meta Delete Profile 221
Meta Disconnect 222
Meta Get Repositories 222
Meta Get Servers 222
Meta Is Connected 222
Meta Set Repository 223
Meta Stored Process 222
metadata server objects (SAS) connections 356
Microsoft Excel file
  open in Wizard 107
  open worksheets 110
  worksheet settings 109
Microsoft Powerpoint, saving as 314
Min 246
Minimize 168–170
Minimum 246
Minus 267
Minute 70
MIRRR, Microsoft Excel 116
Mod 166
Mode 158
Modified Internal Rate of Return 116
Modulo 166
Month 70
MouseButton 84
Mousetrap 129
Move Directory 105
Move File 105
Moving Average 240
Multiply 268
Multiply To 27
Multivariate Normal Impute 158
Munger 37

N
  N Arg 97
  N Arg Expr 97
  N Choose K 252
N Col 158
N Cols 158
N Items 136
N Missing 246
N Row 214
N Rows 214
N Table 214
Name Expr 97
Names Default To Here 195
Namespace 195
Namespace Exists 195
NChooseK Matrix 158
Neg Binomial Distribution 74
Neg Binomial Probability 74
Net Present Value 116
New Column 214
New Image 85
New Namespace 195
New SQL Query 236
New Table 214
New Window 85
Normal Biv Distribution 182
Normal Contour 129
Normal Density 183
Normal Distribution 183
Normal Integrate 166
Normal Log CDistribution 183
Normal Log Density 183
Normal Log Distribution 183
Normal Mixture Density 183
Normal Mixture Distribution 183
Normal Mixture Quantile 184
Normal Quantile 184
Not 62
Not Equal 57
NPER, Microsoft Excel 116
NPV, Microsoft Excel 116
Num 37
Num Deriv 166
Num Deriv2 167
Number 246
Number Col Box 86
Number Col Edit Box 86
Number Edit Box 86
Number of Periods 116
numeric functions 164
<table>
<thead>
<tr>
<th>O</th>
<th>Pat Span 49</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open 106</td>
<td>Pat String 49</td>
</tr>
<tr>
<td>Open Database 110</td>
<td>Pat Succeed 49</td>
</tr>
<tr>
<td>Open Datafeed 268</td>
<td>Pat Tab 50</td>
</tr>
<tr>
<td>Open Log 195</td>
<td>Pat Test 50</td>
</tr>
<tr>
<td>optimization functions 167</td>
<td>Path 130</td>
</tr>
<tr>
<td>Or 63</td>
<td>Path To Char 130</td>
</tr>
<tr>
<td>OrMZ 63</td>
<td>pattern functions 42–51</td>
</tr>
<tr>
<td>Ortho 159</td>
<td>Payment 117</td>
</tr>
<tr>
<td>Ortho Poly 159</td>
<td>PDF, save as 383</td>
</tr>
<tr>
<td>OrV3 63</td>
<td>Pen Color 131</td>
</tr>
<tr>
<td>Outline Box 87</td>
<td>Pen Size 131</td>
</tr>
<tr>
<td>outline box messages 325</td>
<td>Pi 64</td>
</tr>
<tr>
<td>Oval 130</td>
<td>Pick Directory 110</td>
</tr>
<tr>
<td>P</td>
<td>Pick File 110</td>
</tr>
<tr>
<td>Page Break Box 87</td>
<td>Picture Box 87</td>
</tr>
<tr>
<td>Panel Box 87</td>
<td>Pie 131</td>
</tr>
<tr>
<td>panel box messages 326</td>
<td>Pixel Line 131</td>
</tr>
<tr>
<td>Parameter 195</td>
<td>Pixel Move 131</td>
</tr>
<tr>
<td>Parse 196</td>
<td>Pixel Origin 131</td>
</tr>
<tr>
<td>Parse Date 69–70</td>
<td>platform messages 346–351</td>
</tr>
<tr>
<td>Parse XML 268</td>
<td>Platform Preferences 268</td>
</tr>
<tr>
<td>password, opening protected file 108</td>
<td>Plot Col Box 87</td>
</tr>
<tr>
<td>Pat Abort 42</td>
<td>plot col box messages 326</td>
</tr>
<tr>
<td>Pat Altern 42</td>
<td>PMT, Microsoft Excel 117</td>
</tr>
<tr>
<td>Pat Any 42</td>
<td>Poisson Distribution 75</td>
</tr>
<tr>
<td>Pat Arb 42</td>
<td>Poisson Probability 75</td>
</tr>
<tr>
<td>Pat Arb No 43</td>
<td>Poisson Quantile 75</td>
</tr>
<tr>
<td>Pat At 43</td>
<td>Polygon 131</td>
</tr>
<tr>
<td>Pat Break 44</td>
<td>Polytope Uniform Random 270</td>
</tr>
<tr>
<td>Pat Concat 44</td>
<td>Popup Box 87</td>
</tr>
<tr>
<td>Pat Conditional 45</td>
<td>Post Decrement 28</td>
</tr>
<tr>
<td>Pat Fail 45</td>
<td>Post Increment 28</td>
</tr>
<tr>
<td>Pat Fence 45</td>
<td>Power 252</td>
</tr>
<tr>
<td>Pat Immediate 46</td>
<td>PowerPoint, saving as 314</td>
</tr>
<tr>
<td>Pat Len 46</td>
<td>PPMT, Microsoft Excel 117</td>
</tr>
<tr>
<td>Pat Match 46</td>
<td>Pref 271</td>
</tr>
<tr>
<td>Pat Not Any 47</td>
<td>Preference 271</td>
</tr>
<tr>
<td>Pat Pos 47</td>
<td>preference for DPI 314</td>
</tr>
<tr>
<td>Pat R Pos 47</td>
<td>Preferences 271</td>
</tr>
<tr>
<td>Pat R Tab 48</td>
<td>Present Value 117</td>
</tr>
<tr>
<td>Pat Regex 48</td>
<td>Principal Payment 117</td>
</tr>
<tr>
<td>Pat Rem 48</td>
<td>Print 196</td>
</tr>
<tr>
<td>Pat Repeat 48</td>
<td>Print Matrix 159</td>
</tr>
<tr>
<td></td>
<td>probability functions 170–186</td>
</tr>
<tr>
<td></td>
<td>Probi 184</td>
</tr>
</tbody>
</table>
Probit 184
Product 246
programming functions 187–199
PV, Microsoft Excel 117

Q
QR 159
Quantile 247
Quarter 70
Query 237
Quit 191

R
R Connect 199
R Execute 199
R functions 199–204
R Get 200
R Get Graphics 200
R Init 200
R Is Connected 201
R JMP Name to R Name 201
R messages 353–356
R Send 201
R Send File 203
R Submit 203
R Submit File 203
R Term 204
Radio Box 88
Random Beta 204
Random Beta Binomial 205
Random Binomial 205
Random Category 205
Random Cauchy 205
Random ChiSquare 205
Random Exp 205
Random F 205
Random Frechet 206
random functions 204–210
Random Gamma 206
Random Gamma Poisson 206
Random GenGamma 206
Random Geometric 206
Random GLog 206
Random Index 206
Random Integer 206
Random Johnson Sb 207
Random Johnson Sl 207
Random Johnson Su 207
Random LEV 207
Random LogGenGamma 207
Random Logistic 207
Random Loglogistic 207
Random Lognormal 208
Random Negative Binomial 208
Random Normal 208
Random Normal Mixture 208
Random Poisson 208
Random Reset 208
Random Seed State 209
Random SEV 208
Random Shuffle 209
Random t 209
Random Triangular 209
Random Uniform 209
Random Weibull 210
Range Slider Box 88
range slider box messages 326
Rank 160
Rank Index 160
Ranking 160
Ranking Tie 160
RATE, Microsoft Excel 115
Rect 132
Recurse 196
Regex 37
Regex Match 51
Register Addin 273
Remove 137
Remove Color Theme 120
Remove From 137
Rename Directory 111
Rename File 112
Repeat 38
Report 88
Resample Freq 210
Response Screening messages 351
Reverse 138
Reverse Into 138
Revert Menu 274
RGB Color 132
Right 38
Root 253
Round 167
Index
JSLSyntaxReference

Row 215
row functions 211
row messages 304
Row State 217
row state functions 215–218
RTF, save as 383
Run 379
Run Background 381
Run Foreground 381
Run Program 274
run script without opening 52

S
SAS Assign Lib Refs 223
SAS Connect 223
SAS Deassign Lib Refs 225
SAS Disconnect 225
SAS Export Data 225
SAS functions 218–235
SAS Get Data Sets 226
SAS Get File 226
SAS Get File Names 227
SAS Get File Names In Path 227
SAS Get File Refs 227
SAS Get Lib Refs 228
SAS Get Log 228
SAS Get Output 228
SAS Get Results 228
SAS Get Var Names 228
SAS Import Data 229
SAS Import Query 231
SAS Is Connected 232
SAS Is Local Server Available 232
SAS Load Text File 232
SAS messages 356–375
SAS metadata server object connections 356
SAS Name 232
SAS Open For Var Names 233
SAS results, messages 373
SAS Send File 233
SAS server object messages 357
SAS stored process messages 367
SAS Submit 233
SAS Submit File 235
Save HTML 382
Save Log 196
Save PDF 383
Save RTF 383
Save Text File 112
SBInv 253
SbTrans 253
scale an image 341
Scene Box 88
Schedule 276
schedule messages 375
Scheffe Cubic 253
Script Box 88
Scroll Box 89
Second 71
Selected 218
Selected State 218
Send 196
Sequence 215
Set Clipboard 276
Set Current Directory 112
Set Default Directory 113
Set File Search Path 113
Set Path Variable 113
Set Platform Preferences 268
Set Preference 271
Set Toolbar Visibility 277
SetJVMOption 276
SEV Density 184
SEV Distribution 184
SEV Quantile 184
Shade State 218
Shape 160
Sheet Part 90
Shift 138
Shift Into 138
Short Date 71
Shortest Edit Script 277
Show 196
Show Addins Dialog 277
Show Commands 278
Show Globals 197
Show Namespaces 197
Show Preferences 278
Show Properties 278
Show Symbols 197
Simplify Expr 167
Sin 256
Sine 256
SinH 257
Slider Box 90
slider box messages 326
SINv 253
SLN, Microsoft Excel 118
SITrans 253
Sobol Quasi Random Sequence 278
Socket 278
socket messages 376
Solve 161
Sort Ascending 161
Sort Descending 161
Sort List 138
Sort List Into 139
Spacer Box 91
Speak 278
Spline Coef 161
Spline Eval 161
Spline Smooth 162
SQL functions 235
SQL messages 379
Sqrt 253
Squash 254
Squish 254
SSQ 247
SSS import 113
Starts With 39
statistical functions 238–248
Status Msg 279
Std Dev 247
Step 63
Stop 64
stored process (SAS) messages 367
Straight Line Depreciation 118
String Col Box 91
string col box messages 328
String Col Edit Box 91
Students t Density 185
Students t Distribution 185
Students t Quantile 186
Subscript 215
Substitute 139
Substitute Into 139
Substr 39
Subtract 279
Subtract To 29
SuInv 254
Sum 247
Sum Of Years Digits Depreciation 118
Summarize 247
Summarize YByX 248
Summation 248
Suppress Formula Eval 215
SuTrans 254
SVD 162
Sweep 162
SYD, Microsoft Excel 118

T

t Density 185
t Distribution 185
t Log CDistribution 185
t Log Density 185
t Log Distribution 185
T Noncentrality 185
t Quantile 186
Tab Box 91
tab box messages 328
Tab Page Box 91
Table Box 92
table box messages 329
Tabulate messages 352
Tan 257
Tangent 257
TanH 257
Text 132
Text Box 92
Text Color 132
Text Edit Box 92
text import wizard 109
Text Size 132
Throw 197
Tick Seconds 71
Time of Day 71
Titlecase 39
Today 71
Tolerance Limit 248
toolbars, show or hide 277
tooltips 332
Trace 162
Tranparency 133
transcendental functions 248–254
Transpose 162
Tree Box 92
tree box 332
tree box messages 332
Tree Node 93
tree node 332
tree node messages 332
Triangulation 93
triangulation messages 334
Trigamma 254
trigonometric functions 254–257
Trim 40
Trim Whitespace 40
TripleS Import 113
Try 197
Tukey HSD P Value 186
Tukey HSD Quantile 186>Type 198

U
Unlock Symbols 198
Unregister Addin 279
Uppercase 40
utility functions 257–280

V
V Center Box 93
V Concat 163
V Line 133
V List Box 93
V Max 163
V Mean 163
V Min 163
V Sheet Box 93
V Size 133
V Splitter Box 94
V Standardize 163
V Std 163
V Sum 164
Vec Diag 164
Vec Quadratic 164

W
Wait 198
Watch 198
Web 279
Web Browser 94
Week of Year 71
Weibull Density 186
Weibull Distribution 186
Weibull Quantile 186
While 64
Wild 198
Wild List 199
Window 94
window messages 335
Word 40
Words 140
Write 199

X
X Function 133
X Origin 133
X Range 133
X Scale 133
XML Attr 280
XML Decode 280
XML Encode 280
XML Text 280
XML, parse document 41
XPath Query 41
XY Function 134

Y
Y Function 134
Y Origin 134
Y Range 134
Y Scale 134
Year 72

Z
Zero Or Missing 64
zip archive messages 382