

CalcLink

User Manual and Reference Guide

Introduction

Overview

CalcLink consists of two main components:

- The CalcLink package for *Mathematica*
- The *Mathematica* add-in for Calc

These components work together to provide full two-way connectivity between *Mathematica* and Calc.

- To use the link from *Mathematica*, you load the CalcLink package.
- To use the link from Calc, you call functions of the CalcLink add-in (“extension”) or use the CalcLink control center.

Features

The CalcLink package provides:

A set of *Mathematica* functions that allow you to:

- Read from and write data and formulas to Calc ranges/cells.
- Create, open, modify, save, and close Calc files.
- Access Calc functions from *Mathematica*.
- Print Calc worksheets from *Mathematica*.
- Set cell/range styles in Calc from *Mathematica*.
- Set cell/range properties in Calc from *Mathematica*.
- Access all OpenOffice objects from *Mathematica* with J/Link.

The CalcLink add-in provides:

- A set of cell and cell array functions that allow you to use *Mathematica* functions / expressions and load any packages directly from spreadsheet cells or the CalcPad.
- A set of cell and cell array functions to create new and update existing windows with *Mathematica* graphics (or arbitrary *Mathematica* expressions).
- A symbol browser that allows to search and filter from all standard *Mathematica* (3,600+), package, and all user-defined symbols (functions, options, attributes) along with their evaluated values – anything known to the kernel.
- A “mini front-end” called “CalcPad” (as a “notepad for Calc”) that offers several

powerful front-end features such as syntax color-coding and bracket-matching with much lower resources than the full *Mathematica* front-end.

- A control window to manage the CalcLink session with convenient push-buttons.
- Several functions to create various types of new windows that can be copied or updated with live data
- An interactive expandable/collapsible tree view of any *Mathematica* expression.

About

CalcLink

Version 1.1

Copyright © 2010 Andreas Lauschke Consulting

<http://www.lauschkeconsulting.com>

Table of Contents

Introduction.....	2
Overview	2
Features	2
Installation.....	6
Requirements.....	6
Installation and Configuration.....	6
Look-and-Feel Configuration.....	10
Working in Mathematica	11
Getting Started	11
Loading the Package	11
Assigning and Retrieving Data.....	11
CalcLink Functions: Returning Mathematica Symbols or Performing Tasks.....	17
CalcLink Functions: Returning Java Objects.....	18
Interactive Tree Representation/Inspection of Mathematica Expressions.....	19
Function Reference.....	23
CalcLink functions returning Java objects.....	34
Working in Calc.....	37
Getting Started	37
Loading the Extension.....	37
Function Wizard.....	37
Using the Link.....	38
Preparing the Mathematica kernel for use with Calc.....	41
Using the Eval functions.....	41
Using the CalcLink Graphics Functions.....	46
Window Copy.....	49
Mathematica Symbol Browser.....	51
Interactive Tree Representation/Inspection of Mathematica Expressions.....	56
CalcPad.....	57
Possible kernel-sharing options.....	58
Features of CalcPad:.....	59
Error Messages.....	62
Appendix A.....	65
Document Filters.....	65
Appendix B.....	67
Document Filters: CSV Filter.....	67
Appendix C.....	69
Properties for Printing.....	69

Installation

Requirements

- OpenOffice installation, 3.1 or later. Available from www.openoffice.org.
 - Current version is OpenOffice 3.2.
- *Mathematica* 6 or later. Available from www.wolfram.com.
 - Current version is *Mathematica* 7.0.1.
- Java 6 or later. Available from www.java.com.
 - Current version is Java 6 update 21.
- CalcLink 1.0 or later. Available from www.lauschkeconsulting.com.
 - Current version is CalcLink 1.1.

Mathematica 7 has Java 6 bundled with it, *Mathematica* 6 has Java 5 bundled with it.

Installation and Configuration

To use the *Mathematica* Plug-in (“Extension”) from OpenOffice:

- In your home directory, create a subdirectory called “CalcLink”. Place all files from the CalcLink distribution in that directory (4 files).
- Ensure that OpenOffice uses a Java runtime (this should already be set by default). In Calc, go to Tools->Options->OpenOffice.org->Java and ensure that the check mark for “Use Java runtime” is selected and a Java 6 runtime is selected.
- Add the Java library JLink.jar to the class path of your OpenOffice installation. In Calc, go to Tools->Options->OpenOffice.org->Java->ClassPath->Add Archive. Then add the file JLink.jar which is found in the JLink directory of *Mathematica*. On Windows this is usually <MathematicalInstallDirectory>\SystemFiles\Links\JLink. On Linux/Unix this is usually in <MathematicalInstallDirectory>/SystemFiles/Links/JLink.
- After these last two steps in Tools->Options->OpenOffice.org->Java have been done, OpenOffice should be restarted.

- Import the file CTM.oxt from the CalcLink distribution in OpenOffice. On most operating systems you can simply double-click the file, which should launch the Extension Manager in OpenOffice. You can also add it manually from OpenOffice by going to Tools->Extension Manager->Add in Calc.

To use the *Mathematica* Package from *Mathematica*:

- Ensure you placed all files from the CalcLink distribution (4 files) in the CalcLink directory as in step 1 above.
- Edit the file calclinkconfig.m in the CalcLink directory with a text editor to set the variables urelocation, sofficeolocation, and illocation (see examples below).
 - urelocation is the directory in which the files java_uno.jar, juh.jar, jurt.jar, ridl.jar, and unoloader.jar are.
 - sofficeolocation is the directory in which the file soffice.bin is.
 - illocation is the directory in which the file unoil.jar is.

Note:

On Linux these are usually:

```
urelocation:      /usr/lib/openoffice.org/ure/share/java
sofficeolocation: /usr/lib/openoffice.org3/program
illocation:       /usr/lib/openoffice.org/basis3.1/program/classes
```

On Windows Vista these are usually:

```
urelocation:      <drive letter>:\Program Files\openoffice.org\ure\share\java
sofficeolocation: <drive letter>:\Program Files\openoffice.org3\program
illocation:       <drive letter>:\Program Files\openoffice.org\basis3.1\program\classes
```

- In calclinkconfig.m set the variable launchinitial to False if you don't want the launch of CalcLink to also launch Calc with a new empty spreadsheet. The default is True (i. e. the variable assignment is missing in the file), which means that upon launch of CalcLink Calc will be launched as well with a new empty spreadsheet document.
- If you want to modify the Java runtime arguments used with CalcLink, set the variable commandline in the file calclinkconfig.m to the string representation of the Java runtime arguments that you may want to modify from the default values, e. g. to specify a particular Java runtime you want to use (not the one that is included in the *Mathematica* distribution) or to set additional runtime options (memory settings, JIT-compilation, etc.). Assigning to commandline will automatically reinstall the Java runtime. The default is to not use a special command line and not to reinstall the Java runtime (i. e. the variable assignment is missing to in the file).

- If you did not install CalcLink in the recommended default location (in a subdirectory called “CalcLink” under the user's home directory \$HomeDirectory), you have to add a line in the file calclinkconfig.m which specifies the CalcLink location:

calclinklocation=<full path to the CalcLink directory>;

for example

calclinklocation="E:\MySoftware\Technical\CalcLink\";

on Windows, or on Linux/Unix:

calclinklocation="/home/myusername/technicalsoftware/CalcLink";

Note that you HAVE to use a Java runtime that has a bitness that is compatible to the bitness of your OpenOffice installation as OpenOffice uses JNI (Java Native Interface) to access the local system. The bitness of a JNI library must match the bitness of the JVM that is used. This may require you to use another Java runtime and NOT use the Java runtime that is included in the *Mathematica* distribution. (This will usually only be a problem if you have a 64-bit system and a 64-bit *Mathematica* installation but use a 32-bit OpenOffice installation, as is common on Windows. In that case you have to use a 32-bit JVM for CalcLink because your OpenOffice installation is 32-bit.)

A typical calclinkconfig.m file on 32-bit Windows looks like:

```
urelocation="C:\\Program Files\\OpenOffice.org 3\\URE\\java\\";
soffcelocation="C:\\Program Files\\OpenOffice.org 3\\program\\";
illocation="C:\\Program Files\\OpenOffice.org 3\\Basis\\program\\classes\\";
```

A typical calclinkconfig.m file on 64-bit Windows looks like:

```
commandline="C:\\Program Files (x86)\\java\\jre6\\bin\\javaw.exe";
urelocation="C:\\Program Files (x86)\\OpenOffice.org 3\\URE\\java\\";
soffcelocation="C:\\Program Files (x86)\\OpenOffice.org 3\\program\\";
illocation="C:\\Program Files (x86)\\OpenOffice.org 3\\Basis\\program\\classes\\";
```

A typical calclinkconfig.m file on 32-bit Linux looks like:

```
urelocation="/usr/lib/openoffice.org/ure/share/java/";
illocation="/usr/lib/openoffice.org/basis3.1/program/classes/";
soffcelocation="/usr/lib/openoffice.org/3/program/";
```

A typical calclinkconfig.m file on 64-bit Linux looks like:

```
urelocation="/usr/lib64/openoffice.org/ure/share/java/";
illocation="/usr/lib64/openoffice.org/basis3.1/program/classes/";
soffcelocation="/usr/lib64/openoffice.org/3/program/";
```

Note to Users of SELinux:

When enforcing mode is turned on and policy type is set to “Targeted”, upon first use of CalcLink SELinux will throw a text relocation denial warning:

Summary	
SELinux is preventing uno.bin from loading /usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so which requires text relocation.	
Detailed Description	
The uno.bin application attempted to load /usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so which requires text relocation. This is a potential security problem. Most libraries do not need this permission. Libraries are sometimes coded incorrectly and request this permission. The SELinux Memory Protection Tests web page explains how to remove this requirement. You can configure SELinux temporarily to allow /usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so to use relocation as a workaround, until the library is fixed. Please file a bug report against this package.	
Allowing Access	
If you trust /usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so to run correctly, you can change the file context to textrel_shlib_t. "chcon -t textrel_shlib_t '/usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so'" You must also change the default file context files on the system in order to preserve them even on a full relabel. "semanage fcontext -a -t textrel_shlib_t '/usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so'"	
Fix Command	
chcon -t textrel_shlib_t '/usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so'	
Additional Information	
Source	unconfined_u:unconfined_r:unconfined_execmem_t:s0-s0:c0.c1023
Context:	

This is to be expected as JLink has not been registered with SELinux with a policy module. This text relocation denial warning is not a bug or an error. To allow access to the JLink library execute as root:

```
chcon -t textrel_shlib_t '/usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so'
```

and

```
semanage fcontext -a -t textrel_shlib_t '/usr/local/Wolfram/Mathematica/7.0/SystemFiles/Links/JLink/SystemFiles/Libraries/Linux/libJLinkNativeLibrary.so'
```

You have to do this only once, when you are trying to use CalcLink for the first time.

Look-and-Feel Configuration

CalcLink supports most third-party add-on/plug-in look-and-feels. To install a third-party look-and-feel, download the .jar file from the third-party supplier, place it in the CalcLink directory, identify the name of the entry point class, and place a line

`laf="<full name space class name to entry point class>"`

in a text file lookandfeel.m in the CalcLink directory. Example:

`laf="com.jgoodies.looks.plastic.Plastic3DLookAndFeel"`

This will enable the look-and-feel immediately for the *Mathematica* package. To also use the look-and-feel from Calc (through the OpenOffice plug-in/extension), add the .jar file to the OpenOffice class path, as described in steps 3 and 4 in the previous section.

If no file lookandfeel.m exists in the CalcLink directory or laf is assigned the string "default", CalcLink will use the system's default look-and-feel.

For a good overview of various free and commercial look-and-feels, visit <http://www.javootoo.com>

The following Look-and-Feels have been tested to work with CalcLink:

- JGoodies Plastic3D
- JGoodies PlasticXP
- JGoodies Plastic
- JGoodies Windows
- Office2003 (Windows only)
- OfficeXP (Windows only)
- VisualStudio2005 (Windows only)
- Nimrod
- Fh
- Tiny
- Tonic
- Tonic Slim
- Infonode
- Napkin
- SquareNess
- EaSynth

which can be downloaded from javootoo.

The Alloy look-and-feel has also been tested to work with CalcLink, which can be obtained from <http://lookandfeel.incors.com/>.

Working in *Mathematica*

Getting Started

Loading the Package

To start using the link from *Mathematica*, you must first load the CalcLink package.

With *Mathematica* version 6 and above:

```
Get@ToFileName[{$HomeDirectory, "CalcLink"}, "CalcLink.m"]
```

With *Mathematica* version 7 and above:

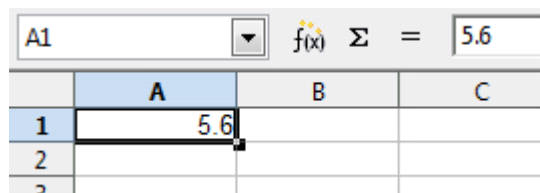
```
Get@FileNameJoin[{$HomeDirectory, "CalcLink", "CalcLink.m"}]
```

Assigning and Retrieving Data

The function Calc[<range>] can be used like a variable in *Mathematica*.

An assignment:

```
Calc["A1"]=5.6
```



The screenshot shows a spreadsheet interface. At the top, a formula bar displays 'A1' in a dropdown menu, followed by icons for functions (f(x)), summation (Σ), and an equals sign (=), with the value '5.6' entered. Below this is a grid of cells. The first row has headers 'A', 'B', and 'C'. The first column has row numbers '1', '2', and '3'. Cell A1 is selected and contains the value '5.6'.

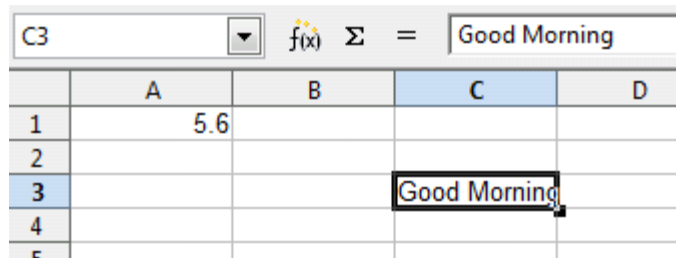
	A	B	C
1	5.6		
2			
3			

A retrieval:

```
Calc["A1"]  
5.6
```

This can be done for data of types integer, real, and string:

Calc["C3"]="Good Morning"



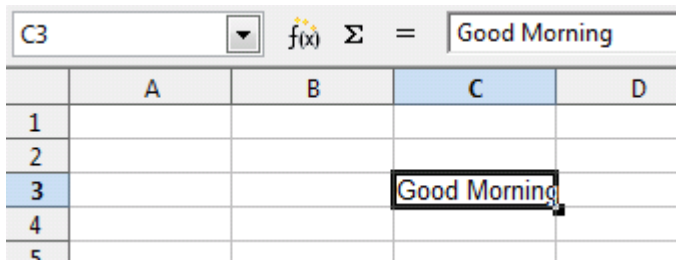
The screenshot shows a spreadsheet interface. At the top, a formula bar displays 'C3' in a dropdown menu, followed by a function icon 'f(x)', a summation symbol 'Σ', an equals sign '=', and the text 'Good Morning'. Below this is a table with columns labeled A, B, C, and D, and rows numbered 1 through 5. Cell C3 is highlighted with a black border and contains the text 'Good Morning'. Cell A1 contains the value '5.6'.

	A	B	C	D
1	5.6			
2				
3			Good Morning	
4				
5				

Calc["C3"]
"Good Morning"

Data in a Calc cell can also be deleted that way:

Calc["A1"]=.



The screenshot shows the same spreadsheet interface as before. The formula bar still displays 'C3' in the dropdown, 'f(x)', 'Σ', '=', and 'Good Morning'. In the table, cell A1 is now empty, indicating that the data has been deleted. Cell C3 remains highlighted and contains 'Good Morning'.

	A	B	C	D
1				
2				
3			Good Morning	
4				
5				

The string argument to Calc[<range>] is in fact the definition of a range, so

Calc["F3"]=66.7

is equivalent to

Calc["F3:F3"]=66.7

Also lower-case letters or mixed case could be used:

Calc["F3"]=66.7 Calc["f3"]=66.7 Calc["F3:f3"]=66.7 Calc["f3:F3"]=66.7

The same method can be used to assign data to one-dimensional and two-dimensional ranges:


A row-vector:

Calc["A1:D1"] = {1, 2, 6, 7}

D1	f(x)	Σ	=	7
	A	B	C	D
1	1	2	6	7
2				
3				

A column-vector:

Calc["A1:A4"] = {{1}, {5}, {6}, {88}}

A4			$f(x)$	Σ	=	88
	A		B			C
1	1					
2	5					
3	6					
4	88					
5						
6						

A square matrix:

Calc["A1:D4"] = HilbertMatrix[4]

A1:D4	f(x)	Σ	=	0.142857142857143
	A	B	C	D
1	1.000000	0.500000	0.333333	0.250000
2	0.500000	0.333333	0.250000	0.200000
3	0.333333	0.250000	0.200000	0.166667
4	0.250000	0.200000	0.166667	0.142857
5				
6				

An array of elements of various data types (string, integer, real):

Calc["A1:D7"] = {{{"Name", "State", "Sales", "Miles"}, {"Peter", "IL", 347324.55, 456}, {"Mary", "OH", 256789, 321}, {"George", "AZ", 622354.66, 781}, {"Paul", "NV", 0, "N/A"}, {"Melissa", "ID", 166788.45, 489}, {"Kary", "MT", 378956.55, "N/A"}}

D7		f(x)	Σ	=	N/A
	A	B	C	D	
1	Name	State	Sales	Miles	
2	Peter	IL	347324.55	456	
3	Mary	OH	256789	321	
4	George	AZ	622354.66	781	
5	Paul	NV	0	N/A	
6	Melissa	ID	166788.45	489	
7	Kary	MT	378956.55	N/A	
8					
9					

As Calc works only on a machine-number platform, all numeric data returned by Calc is always of type double/real. For sufficiently well-behaved data (roughly speaking, not exceeding the limits of \$MaxNumber, \$MinMachineNumber, and \$MachineEpsilon for numerators, denominators, or exponents) the use of Rationalize is not a problem:

```
(m = Rationalize@CalcGetRange["A1:D4"]) // TableForm
```

1	2	3	4
10	12	14	17
2	5	4	7
12	11	5	7

The following writes the determinant of m back in the spreadsheet:

```
CalcSetRange["C6", Det[m]]
```

C6		f(x)	Σ	=	39
	A	B	C	D	
1	1	2	3	4	
2	10	12	14	17	
3	2	5	4	7	
4	12	11	5	7	
5					
6			39		
7					
8					

This writes the inverse of m back in the spreadsheet:

```
CalcSetRange["A8:D11", Inverse[m]]
```

D11		f(x)	Σ	=	0.82051282051282
	A	B	C	D	
1	1	2	3	4	
2	10	12	14	17	
3	2	5	4	7	
4	12	11	5	7	
5					
6			39		
7					
8	4.64	-0.92	-1.05	0.64	
9	-6.77	1.31	1.46	-0.77	
10	-5.79	1.38	0.74	-0.79	
11	6.82	-1.46	-1.03	0.82	
12					
13					

CalcLink automatically converts numeric data from a symbolic *Mathematica* expression to display it in Calc:

```
CalcSetRange["a1:b2", {{4, Sqrt@5}, {Pi, 64/10}}]
```

B2		f(x)	Σ	=	6.4
	A	B	C		
1	4.00000	2.23607			
2	3.14159	6.40000			
3					
4					

You can also set Calc formulas with CalcLink, not just pass data:

```
CalcSetCell["A2", "=A1/3"]
```

A2		f(x)	Σ	=	=A1/3
	A	B	C		
1	99				
2	33				
3					
4					

You can retrieve the error code from Calc when an error is generated. Here we attempt a division by zero and retrieve the error code (532 is the error code used by OpenOffice Calc to indicate a division by zero error).

CalcSetCell["A15", "=1/0"]

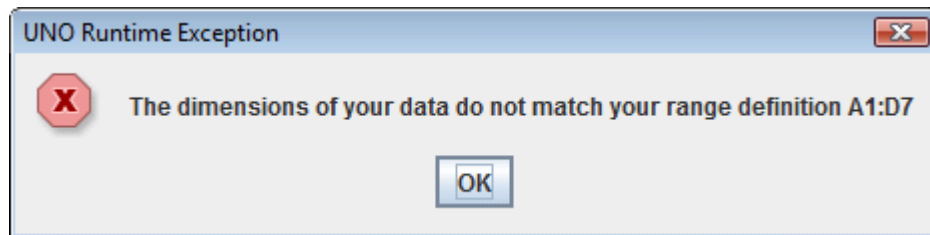
14	
15	#DIV/0!
16	

CalcGetCellError["A15"]

532

CalcLink detects when you are trying to pass data that is incompatible with the dimensions prescribed by the range definition. It is necessary to set the strings defining the ranges compatible with the dimensions of the data to be passed to Calc, otherwise error messages will be thrown:

CalcSetRange["A1:D7", {{1, 2, 3, 4}, {10, 12, 14, 17}, {2, 5, 4, 7}, {12, 11, 5, 7}}]



The tasks described above can also be performed with the more explicit functions

CalcGetRange[<range definition>]
CalcSetRange[<range definition>,<data>]
CalcSetCell[<cell definition>,<data>]
CalcClearRange[<range definition>]

These functions also make it possible to specify the sheet inside the spreadsheet document by using the index of the sheet (0-based), the name of the sheet, or the full Java object representing the sheet. For example, to write the list data {2,4,6,8} into the cell range A1:D4 in the second sheet, one can use

CalcSetRange[1,"A1:D4",{2,4,6,8}]

or

CalcSetRange["Sheet2","A1:D4",{2,4,6,8}]

or

CalcSetRange[<javaobj>,"A1:D4",{2,4,6,8}]

with <javaobj> representing the full Java object for the second sheet (must have been obtained earlier as a Java object representing a Calc document sheet with appropriate CalcLink functions returning Java objects).

Without specifying the sheet in the first argument to CalcSetRange[] CalcLink will use the first worksheet as a default. (This is the case for all other CalcLink functions operating on sheets as well.)

CalcLink Functions: Returning Mathematica Symbols or Performing Tasks

Many more functions are available to write data in a Calc spreadsheet or retrieve information from a Calc spreadsheet or otherwise interact with Calc:

CalcLaunch[] launches a new instance of CalcLink.

CalcConnect[] connects to a running instance of OpenOffice.

CalcGetCellError[] gets the cell error thrown by Calc.

CalcSetDate[] sets a date in date number format.

CalcInsertSheet[] inserts a new Calc spreadsheet.

CalcRemoveSheet[] removes a Calc spreadsheet.

CalcGetSheetNames[] gets all sheet names (as a list of strings)

CalcCreateSpreadsheetDocument[] creates a new Calc spreadsheet document.

CalcChangeSheetName[] changes the sheet name of the specified Calc sheet.

CalcGetSheetName[] gets the sheet name of the specified Calc sheet (as a string).

CalcInsertRange[] inserts a new range of cells.

CalcRemoveRange[] removes a range of cells.

CalcCopyRange[] copies a cell range to a different cell range.

CalcMoveRange[] moves a cell range to a different cell range.

CalcCallFunction[] calls functions that Calc provides itself.

CalcSetCellStyle[] sets a user-defined or pre-defined cell style.

CalcChangeProperty[] changes a user-defined or pre-defined property.

CalcSaveDocument[] saves a specified Calc document as a file.

CalcPrintDocument[] prints a Calc document (standard or with settings/options).

CalcLoadDocument[] loads a Calc document from a file.

CalcClearRange[] clears a range of cells.

CalcCreateSpreadsheetDocument[] creates new Calc spreadsheet document.

CalcCloseDocument[] closes the Calc spreadsheet document.

CalcTree[] generates an interactive tree of the expression using expandable/collapsible tree nodes.

CalcLink Functions: Returning Java Objects

Many functions return Java objects that can be used further in *Mathematica* with J/Link to allow the user to extend the features provided by CalcLink:

CalcGetSheet[] gets a Calc spreadsheet.

CalcGetSheets[] gets all sheets in the Calc document.

CalcGetSpreadsheetDocument[] gets the Calc spreadsheet document.

CalcGetServiceManager[] gets the service manager.

CalcGetContext[] gets the Calc document context.

CalcGetColumns[] gets the specified columns.

CalcGetRows[] gets the specified rows.

CalcShowSymbols[] displays all symbols of the current session.

CalcShowUserSymbols[] displays all user and package symbols of the current session.

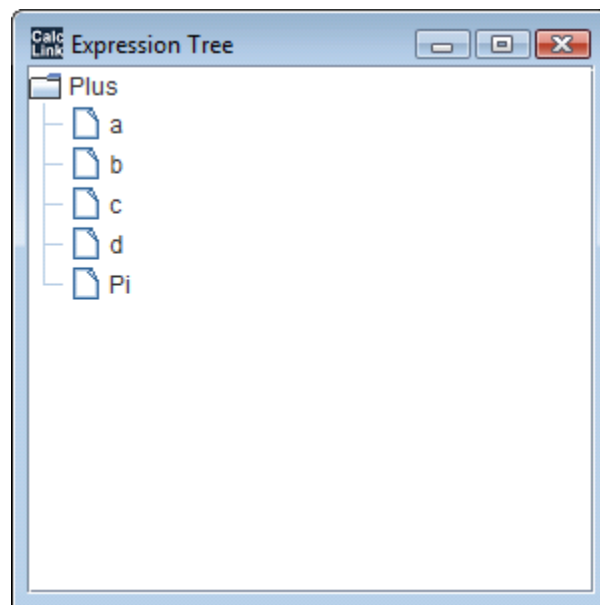
For detailed information on these functions with examples see the section Function Reference on page 24.

Interactive Tree Representation/Inspection of *Mathematica* Expressions

The function `CalcTree[]` creates a new window containing a tree of the expression using expandable/collapsible tree nodes. At every node the name of the head of the expression at that level is shown. The node can be expanded to display all its nodes or leaves. Only leaves (symbols that are atomic, i. e. `AtomQ[]` is `True`) can not be expanded anymore.

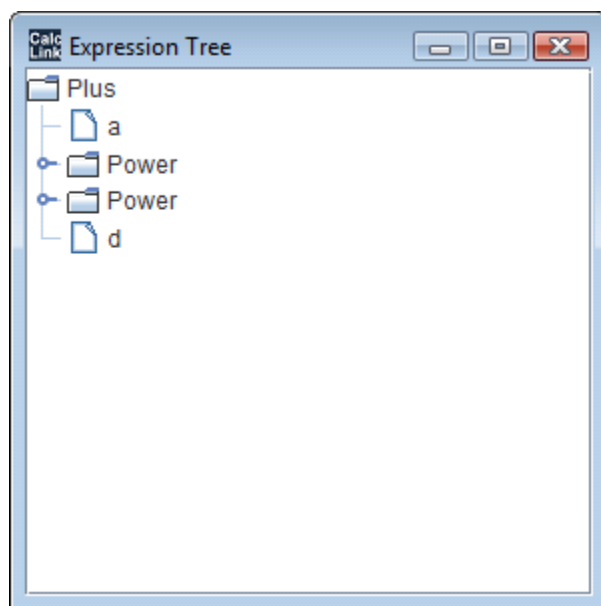
A very simple example involving only leaves under the root node:

`CalcTree[a + b + c + d + Pi]`



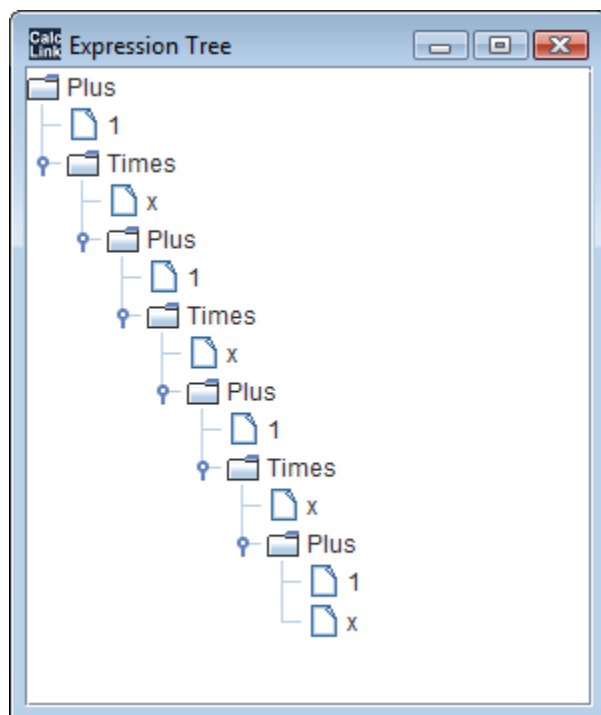
`a` and `d` are atomic, `b^2` and `c^3` are not:

`CalcTree[a + b^2 + c^2 + d]`



HornerForm[] creates an expression that has two nesting levels per order of the polynomial (minus 1).

CalcTree[HornerForm[1 + x + x^2 + x^3, x]]



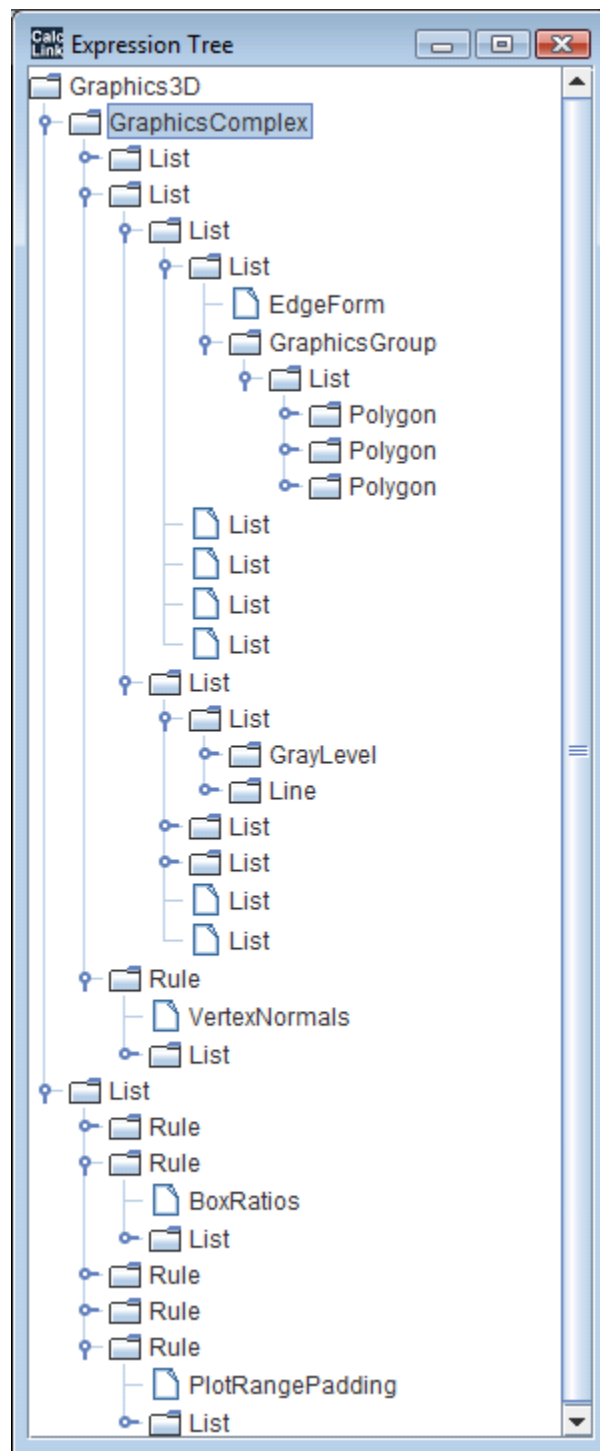
Inspect the coefficients of the Taylor series visually:

CalcTree[Series[Tan@x, {x, 0, 12}]]



The real usefulness of `TREE[]` becomes evident when used on complex, deeply nested expression structures. Sometimes *Mathematica* expressions can be so complex that it is hard to understand the nested symbol structure, so `CalcTree[]` makes it possible to "zoom in" on a branch of interest, while leaving others collapsed.

```
CalcTree[Plot3D[Sin@x Sin@y, {x, -Pi, Pi}, {y, -Pi, Pi}]]
```



While the *Mathematica* function `TreeForm[]` shows the expression in a "top-down" fashion in a possibly more "intuitive" and visually appealing form, the CalcLink function `CalcTree[]` allows for interactive and selective inspection of the branches/leaves of a complex nested expression.

The `CalcTree[]` is also available as a function in the OpenOffice Calc plug-in, called `TREE()`.

Function Reference

CalcLaunch[]

The function `CalcLaunch[]` launches a new instance of `CalcLink`. It instantiates a new connection object (Java) and exposes its methods (and two fields) for use from *Mathematica*.

This can be thought of as a constructor for the connection object. This connection object created by `CalcLaunch` is called “calclink”, and it can be inspected with the J/Link functions `Methods`, `Fields`, `Constructors`. For example, `Methods[calclink]` will show all public methods exposed by `calclink`.

Using `CalcLaunch` without argument or `False` will not launch a new OpenOffice instance, Calc desktop object, document context object, nor service manager object, and accordingly it will not create a new empty spreadsheet document in the desktop. To launch Calc with a new empty spreadsheet document in the desktop (and all related new instantiations) see `CalcLaunch[True]`.

Under normal circumstances, this function will not be used frequently. In most cases it will not be necessary for the user to instantiate new connection objects. About the only situation where `CalcLaunch` would be used is when the OpenOffice session or `CalcLink` was closed, and the user wants to reconnect to a new or existing Calc instance without reloading the `CalcLink` package.

If no instance of OpenOffice or Calc is running, an error will be thrown.

CalcLaunch[False]

same as `CalcLaunch[]`

CalcLaunch[True]

Same as `CalcLaunch[]`, but a new instance of OpenOffice and its desktop object and a new spreadsheet document context and a new service manager object are created, and a new empty spreadsheet document is created in the desktop and connected to from `CalcLink` when the `calclink` object is instantiated.

If the config variable `launchinitial` is set to `True` in `calclinkconfig.m` (which is the default), `CalcLaunch[True]` is executed when the `CalcLink` package is loaded.

CalcConnect[]

This function will connect to a running instance of OpenOffice.

If no instance of OpenOffice is running, an error will be thrown.

CalcCloseDocument[]

The function CalcCloseDocument[] closes the Calc spreadsheet document.

CalcGetCellError[<cell>]

This function retrieves the integer representing the cell error from a cell. The return is an integer indicating the type of error. For example, error code 532 is an integer indicating that a division by zero was attempted by the user. If the Calc desktop is used with CalcLink the corresponding error message is also displayed in the status bar of Calc.

CalcSetDate[]

The function CalcSetDate[] is used to set a date in a specified format in a cell. The supported date formats are:

CalcSetDate[<sheet>,<cell>,<year>,<month>,<day>]

CalcSetDate sets a date expression in the cell <cell> in the sheet <sheet> using <year>, <month>, <day>. Note that <sheet> is a Java sheet object.

CalcSetDate[<index>,<cell>,<year>,<month>,<day>]

CalcSetDate sets a date expression in the cell <cell> in the sheet number <index> using <year>, <month>, <day>.

CalcSetDate[<name>,<cell>,<year>,<month>,<day>]

CalcSetDate sets a date expression in the cell <cell> in the sheet named <name> using <year>, <month>, <day>.

CalcInsertSheet[]

The function CalcInsertSheet[] inserts a new sheet at an indicated position. The possible signatures are:

CalcRemoveSheet[<sheetname>]

This function removes the sheet named <sheetname> from the spreadsheet document.

If there is no sheet with the name <sheetname> an error is thrown.

CalcRemoveSheet[<sheetindex>]

This function removes the sheet numbered <sheetindex> (0-based) from the spreadsheet document.

If there is no sheet numbered <sheetnumber> an error is thrown.

CalcRemoveSheet[]

This function removes the first sheet from the spreadsheet document.

If there is no sheet in the spreadsheet document an error is thrown.

CalcInsertSheet[<sheetname>,<position>]

CalcInsertSheet will insert a new empty sheet with the name <sheetname> at position <position> in the current spreadsheet document. Note that in line with OpenOffice, this is zero-based, so to insert a new sheet as the first sheet you would use

CalcInsertSheet[0,<sheetname>].

CalcInsertSheet[<sheetname>]

CalcInsertSheet will insert a new empty sheet with the name <sheetname> at position 0 (i. e. as the first sheet) in the current spreadsheet document.

CalcGetSheetNames[]

The function CalcGetSheetNames[] returns a list of the sheet names of the current spreadsheet document. The return type is a list of strings.

CalcCreateSpreadsheetDocument[]

The function CalcCreateSpreadsheetDocument[] creates a new Calc spreadsheet document and returns a reference to the Java object representing the new spreadsheet document. This function will merely create a new spreadsheet document Java object but NOT display it in the desktop, it is merely a new Java object. The usefulness of this function is limited to those users who want to manage OpenOffice Java objects themselves with J/Link further downstream with *Mathematica*. If a new empty spreadsheet document is to be created and made available through the OpenOffice desktop, one should use CalcLaunch[] or CalcLaunch[True] or reload the CalcLink package (with the option launchinitial set to True).

CalcChangeSheetName[]

The function CalcChangeSheetName[] changes the name of the indicated sheet to a new name. The following signatures are available:

CalcChangeSheetName[<new name>,<index>]

This will change the name of the sheet identified by <index> to the new name <new name>. Note that in line with OpenOffice the index of the sheet is zero-based, so to change the name of the second sheet to a new name one would use CalcChangeSheetName[3,<new name>].

If there is no sheet for the index <index> an error is thrown.

CalcChangeSheetName[<current name>,<new name>]

This will change the name of the sheet identified by <current name> to the new name <new name>.

If there is no sheet with the name <current name> an error is thrown.

CalcChangeSheetName[<new name>]

This will change the name of the sheet identified by <current name> to the new name <new name>.

If there is no sheet with the name <current name> an error is thrown.

CalcGetSheetName[]

This function returns a string with the name of the sheet identified by <index>. Note that in line with OpenOffice this index is zero-based.

If there is no sheet for the index <index> an error is thrown.

CalcInsertRange[]

This function inserts a new range of cells. The following signatures are available:

CalcInsertRange[<index>,<range>,<type>]

The function CalcInsertRange[<index>,<range>] inserts new empty columns between the indices used in <range> in sheet numbered <index> using <type> to specify the insertion type (NONE, DOWN, RIGHT, ROWS, COLUMNS).

If <range> or <type> is not available in the sheet or is not plausible, an error is thrown.

CalcInsertRange[<sheetname>,<range>]

The function CalcInsertRange[<sheetname>,<range>] inserts new empty columns between the indices used in <range> in sheet named <sheetname> using <type> to specify the insertion type.

If <range> or <sheetname> is not available in the sheet or is not plausible, an error is thrown.

CalcInsertRange[<sheet>,<range>]

The function CalcInsertRange[<sheet>,<range>] inserts new empty columns between the indices used in <range> in sheet <sheet>. Note that <sheet> is a Java sheet object (which must have been obtained earlier) using <type> to specify the insertion type.

If <range> or <sheet> is not available in the sheet or is not plausible, an error is thrown.

CalcInsertRange[<range>]

The function CalcInsertRange[<range>] inserts new empty columns between the indices used in <range> using <type> to specify the insertion type.

If <range> is not available in the sheet or is not plausible, an error is thrown.

CalcRemoveRange[]

The function CalcRemoveRange[] removes the cell range identified by <range>. The cells of the range themselves are completely removed (not just their contents cleared).

The following signatures are available:

CalcRemoveRange[<sheetobject>,<range>,<type>]

This function completely removes the cell range <range> in the sheet <sheetobject> (which is a Java sheet object that must be obtained prior to calling this function) using <type> to specify row or column deletion mode (NONE, UP, LEFT, ROWS, COLUMNS).

If the <sheetobject> does not exist or <range> is not available in the sheet or is not plausible, appropriate errors are thrown.

CalcRemoveRange[<sheetname>,<range>,<type>]

This function completely removes the cell range <range> in the sheet named <sheetname> using <type> to specify row or column deletion mode.

If there is no sheet named <sheetname> or <range> is not available in the sheet or is not plausible, appropriate errors are thrown.

CalcRemoveRange[<index>,<range>,<type>]

This function completely removes the cell range <range> in the sheet numbered <index> using <type> to specify row or column deletion mode.

If there is no sheet numbered <index> or <range> is not available in the sheet or is not plausible, appropriate errors are thrown.

CalcRemoveRange[<range>,<type>]

This function completely removes the cell range <range> in the first sheet using <type> to specify row or column deletion mode.

If there is no sheet in the spreadsheet document or <range> is not available in the sheet or is not plausible, appropriate errors are thrown.

CalcCopyRange[]

The function CalcCopyRange[] copies the contents of a cell range to another cell range. The contents of the target cell range are completely overwritten without warning.

The following signatures are available:

CalcCopyRange[<sheetobject>,<range>,<target>]

This function copies the contents of the cell range <range> in the sheet identified by the Java sheet object <sheetobj> (must be obtained prior to calling this function) to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or <sheetobject> does not exist, appropriate errors are thrown.

CalcCopyRange[<sheetname><range>,<target>]

This function copies the contents of the cell range <range> in the sheet name <sheetname> to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet named <sheetname>, appropriate errors are thrown.

CalcCopyRange[<index>,<range>,<target>]

This function copies the contents of the cell range <range> in the sheet named <index> to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet numbered <index>, appropriate errors are thrown.

CalcCopyCellRange[<range>,<target>]

This function copies the contents of the cell range <range> in the first sheet to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet in the Calc spreadsheet document, appropriate errors are thrown.

CalcMoveRange[]

The function CalcMoveRange[] moves the contents of a cell range to another cell range. The contents of the target cell range are completely overwritten without warning.

The following signatures are available:

CalcMoveRange[<sheetobject>,<range>,<target>]

This function moves the contents of the cell range <range> in the sheet identified by the Java sheet object <sheetobj> (must be obtained prior to calling this function) to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or <sheetobject> does not exist, appropriate errors are thrown.

CalcMoveRange[<sheetname><range>,<target>]

This function moves the contents of the cell range <range> in the sheet name <sheetname> to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet named <sheetname>, appropriate errors are thrown.

CalcMoveCellRange[<index>,<range>,<target>]

This function moves the contents of the cell range <range> in the sheet named <index> to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet numbered <index>, appropriate errors are thrown.

CalcMoveCellRange[<range>,<target>]

This function moves the contents of the cell range <range> in the first sheet to the cell range that has <target> at its top-left cell. The contents of the target cell range are completely overwritten without warning.

If <range> is not available in the sheet or is not plausible or <target> is invalid or there is no sheet in the Calc spreadsheet document, appropriate errors are thrown.

CalcCallFunction[]

This function calls functions that Calc provides itself.

CalcCallFunction[<range>,<function>]

This function calls the Calc function <function> on the data in <range>. The return is a single value. Possible values for <function> are: "AVERAGE", "MIN", "MAX", "STDEV", "STDEVP", "NONE", "COUNT", "COUNTNUMS", "AUTO", "PRODUCT", "SUM", "VAR", "VARP".

CalcCallFunction[<index>,<range>,<functionobject>]

This function calls the Calc function <functionobject> on the data in <range> in sheet <index>. The <functionobject> is specified as com`sun`star`sheet`GeneralFunction`<name>, and as the class members in com`sun`star`sheet`GeneralFunction are static members, J/Link requires to load the class before any static members are used with

LoadJavaClass[“com.sun.star.sheet.GeneralFunction”].

Example:

LoadJavaClass[“com.sun.star.sheet.GeneralFunction”];
CalcCallFunction[<index>,<range>,com`sun`star`sheet`GeneralFunction`MAX]

Errors are thrown if the sheet does not exist, the requested Calc function does not exist, or a general math error occurred (e. g. division by zero or number overflow encountered).

CalcCallFunction[<sheetname>,<range>,<functionobject>]

same as CalcCallFunction[<index>,<range>,<functionobject>], but the sheet named <sheetname> is used.

CalcCallFunction[<sheetobject>,<range>,<functionobject>]

same as CalcCallFunction[<index>,<range>,<functionobject>], but the sheet identified by the sheet object <sheetobject> is used (must be obtained prior).

CalcCallFunction[<range>,<functionobject>]

same as CalcCallFunction[<index>,<range>,<functionobject>], but as a default the first sheet is used.

CalcClearRange[<sheetobject>,<range>]

This function completely clears the cells in the range <range> in sheet <sheetobject> where <sheetobject> is a Java object that must be obtained prior to using this signature.

An error is thrown if the sheet doesn't exist or the range definition is not plausible.

CalcClearRange[<index>,<range>]

This function completely clears the cells in the range <range> in sheet numbered <index> (0-based).

An error is thrown if the sheet doesn't exist or the range definition is not plausible.

CalcClearRange[<sheetname>,<range>]

This function completely clears the cells in the range <range> in sheet named <sheetname>.

An error is thrown if the sheet doesn't exist or the range definition is not plausible.

CalcClearRange[<range>]

This function completely clears the cells in the range <range> in the first sheet.

An error is thrown if the sheet doesn't exist or the range definition is not plausible.

CalcSetCellStyle[]

The function CalcSetCellStyle[] applies a user-defined or pre-defined cell style. The following signatures are available.

CalcSetCellStyle[<stylename>,<color>]

The function CalcSetCellStyle[<stylename>,<color>] sets the style named <stylename> to have the color value <color>. The style name <stylename> is a string, and the color value <color> is an integer.

If the style named <stylename> or the color value <color> does not exist or is not defined, an appropriate error is thrown.

CalcSetCellStyle[<stylename>,<transparency>]

The function CalcSetCellStyle[<stylename>,<transparency>] sets the style named <stylename> to have the transparency setting <transparency>. The style name <stylename> is a string, and the transparency setting <transparency> is a boolean value.

If the style named <stylename> or the transparency setting <transparency> does not exist or is not defined, an appropriate error is thrown.

CalcChangeProperty[]

This function changes a user-defined or pre-defined property. The following signatures are available:

final String myrangename,final String propertyvalue,final int color)

CalcChangeProperty[<sheetobject>,<range>,<propertyvalue>,<value>]

This function changes the property named <propertyvalue> to the value <value> in the range <range> in the sheet identified by the Java spreadsheet object <sheetobject> (which must have been obtained prior to calling this function).

If the spreadsheet object <sheetobject> does not exist or <range> does not exist or is not plausible or <propertyvalue> does not exist or <value> is an invalid value for <propertyvalue>, appropriate error messages are thrown.

CalcChangeProperty[<sheetname>,<range>,<propertyvalue>,<value>]

This function changes the property named <propertyvalue> to the value <value> in the range <range> in the sheet named <sheetname>.

If the sheet named <sheetname> does not exist or <range> does not exist or is not plausible or <propertyvalue> does not exist or <value> is an invalid value for <propertyvalue>, appropriate error messages are thrown.

CalcChangeProperty[<index>,<range>,<propertyvalue>,<value>]

This function changes the property named <propertyvalue> to the value <value> in the range <range> in the sheet numbered <index> (0-based).

If the sheet numbered <index> does not exist or <range> does not exist or is not plausible or <propertyvalue> does not exist or <value> is an invalid value for <propertyvalue>, appropriate error messages are thrown.

CalcChangeProperty[<range>,<propertyvalue>,<value>]

This function changes the property named <propertyvalue> to the value <value> in the range <range> in the first sheet.

If there is no sheet in the Calc spreadsheet document or <range> does not exist or is not plausible or <propertyvalue> does not exist or <value> is an invalid value for <propertyvalue>, appropriate error messages are thrown.

CalcSaveDocument[]

The function CalcSaveDocument[] allows the user to save the current Calc spreadsheet document as a file. Note that for file formats other than OpenOffice spreadsheet document format appropriate filter files have to be included in the OpenOffice installation. For more information and a table of the file formats supported by OpenOffice through filter files see Appendix A. The following signatures are available:

CalcSaveDocument[<filename>]

This function saves the current spreadsheet document as file name <filename>. This signature uses the default file type “ods” for “OpenOffice spreadsheet document”. For other formats, e. g. “Excel 97”, see CalcSaveDocument[<filename>,<format type>].

CalcSaveDocument[<filename>,<format type>]

The function CalcSaveDocument[<filename>,<format type>] saves the current spreadsheet document as file name <filename> using the document format type <format type>. A document format type has to be set up in the document filters (e. g. “Excel 97”, “Star Writer”, etc.). For more information and a table of the file formats supported by OpenOffice through filter files, see Appendix A. For the CSV filter see Appendix B.

CalcPrintDocument[]

The function CalcPrintDocument[] prints the current spreadsheet document using standard printer settings. For printing spreadsheet documents with settings, see CalcPrintDocument[<settings>].

If there is no standard printer available, an error is thrown.

CalcPrintDocument[<settings>]

This function prints the current spreadsheet document using printer settings specified by <settings>. Printer settings are set up in properties passed as arguments in <settings>. For more information on printer settings, see Appendix C.

If there is no printer available or the settings in the filter file are invalid an error is thrown.

CalcLoadDocument[]

The function CalcLoadDocument[<filename>] loads an OpenOffice spreadsheet document from file and displays it in the desktop of OpenOffice. For spreadsheet documents of other format types see CalcLoadDocument[<filename>,<format type>].

If the file <filename> cannot be found or is not a valid OpenOffice Calc spreadsheet document an error is thrown.

CalcLoadDocument[<filename>,<format type>]

The function CalcLoadDocument[<filename>,<format type>] loads a spreadsheet document using filter settings specified by <format type> from file and displays it in the OpenOffice desktop.

If the file <filename> cannot be found or is not a valid spreadsheet document as specified by the filter settings from <format type> an error is thrown.

CalcClearRange[<range>]

The function CalcClearRange[<range>] clears the contents of the cell range specified by <range>].

If <range> is not available in the sheet or is not plausible, an error is thrown.

CalcCreateSpreadsheetDocument[]

The function CalcCreateSpreadsheetDocument[] creates new Calc spreadsheet document.

If no instance of OpenOffice is running, an error is thrown.

CalcLink functions returning Java objects

Many CalcLink functions return Java objects that can be used further in *Mathematica* with J/Link to allow the user to extend the features provided by CalcLink:

CalcGetSheet[<index>]

The function CalcGetSheet[<index>] returns a reference to the Java object representing the document spreadsheet identified by <index>.

If there is no sheet for index <index> an error is thrown.

CalcGetSheet[<sheet name>]

The function CalcGetSheet[<sheet name>] returns a reference to the Java object representing the document spreadsheet identified by <sheet index>.

If there is no sheet for index <sheet index> an error is thrown.

CalcGetSheets[]

The function CalcGetSheets[] returns a reference to the Java collection object representing the sheets contained in the document spreadsheet.

If there is no Calc spreadsheet document in Calc or OpenOffice is closed an error is thrown.

CalcGetSpreadsheetDocument[]

The function CalcGetSpreadsheetDocument[] returns a reference to the Java object representing the spreadsheet document.

If there is no Calc spreadsheet document in Calc or OpenOffice is closed an error is thrown.

CalcGetServiceManager[]

The function CalcGetServiceManager[] returns a reference to the Java object representing the OpenOffice service manager.

The service manager is one of the only two public fields exposed by the calclink object.

If there is no Calc spreadsheet document in Calc or OpenOffice is closed an error is thrown.

CalcGetContext[]

The function CalcGetContext[] returns a reference to the Java object representing the OpenOffice document context of the current spreadsheet document.

The spreadsheet document context is one of the only two public fields exposed by the calclink object.

If there is no Calc spreadsheet document in Calc or OpenOffice is closed an error is thrown.

CalcGetColumns[<columns>]

The function CalcGetColumns[<columns>] returns a reference to the Java collection object representing the columns specified by <columns>.

If the columns specified by <columns> don't exist in the current spreadsheet document or the definition of the columns is not plausible, appropriate errors are thrown.

CalcGetColumns[<rows>]

The function CalcGetColumns[<rows>] returns a reference to the Java collection object representing the rows specified by <rows>.

If the rows specified by <rows> don't exist in the current spreadsheet document or the definition of the rows is not plausible, appropriate errors are thrown.

CalcTree[<expression>]

The CalcTree[<expression>] function creates a new window containing a tree of the expression using expandable/collapsible tree nodes. At every node the name of the head of the expression at that level is shown. The node can be expanded to display all its nodes or leaves. Only leaves (symbols that are atomic, i. e. AtomQ[] is True) can not be expanded anymore.

CalcShowSymbols[]

The function CalcShowSymbols[] creates a new window that shows all symbols of the current session (the output of Names[]). The filter field can be used to filter out the symbols that match the filter text. The matching uses Perl 5 regular expression syntax. For more details see page 52.

CalcShowUserSymbols[]

The function `CalcShowUserSymbols[]` creates a new window that shows all user and package symbols of the current session (the output of `Names["Global`*"]`). The filter field can be used to filter out the symbols that match the filter text. The matching uses Perl 5 regular expression syntax. For more details see page 52.

Working in Calc

Getting Started

Loading the Extension

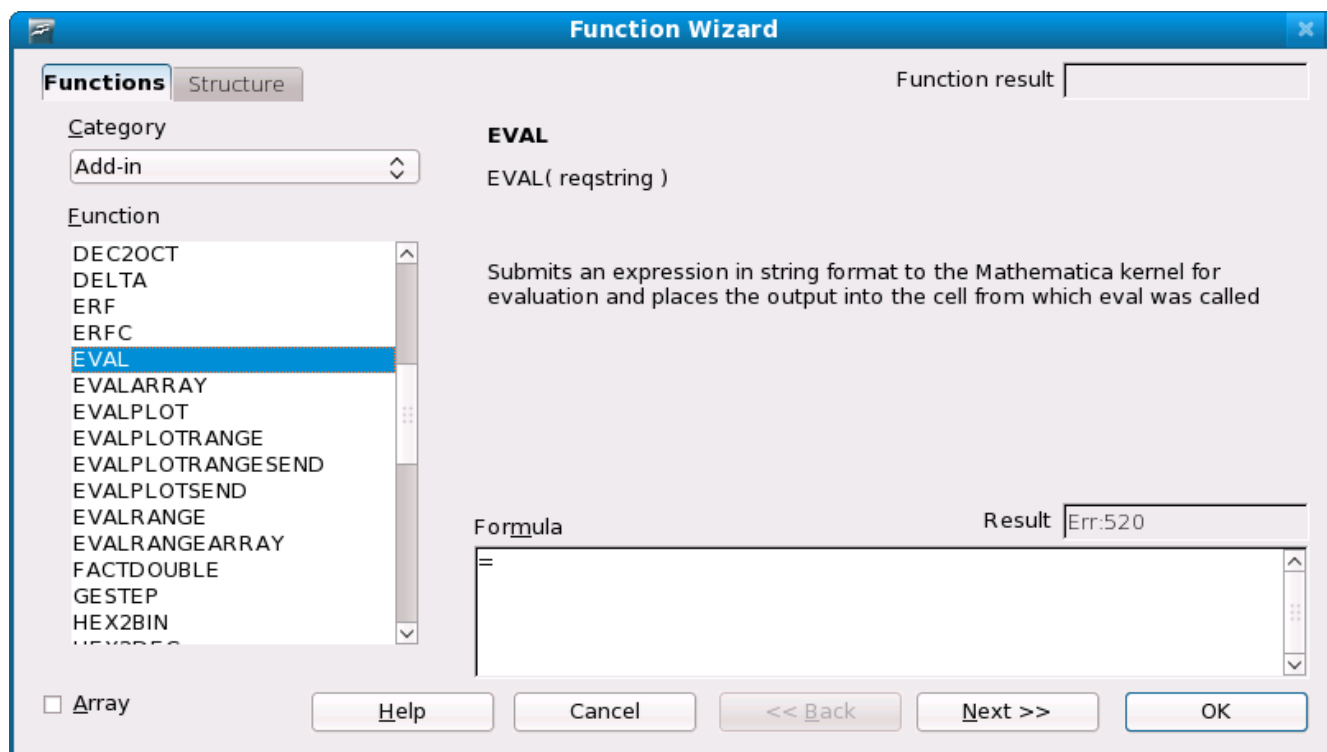
The CalcLink distribution contains a file CTM.oxt (“oxt” stands for “OpenOffice extension”). This file must be imported into OpenOffice with the Extension Manager. On most operating systems a simple double-click on that file should bring up the extension manager and allow the user to import the file. If not, the extension manager can be found in OpenOffice in Tools -> Extension Manager, and the file can be imported manually from there.

Function Wizard

All CalcLink functions can be browsed using the Calc function wizard. With the mouse cursor in a cell, click the function wizard icon in the symbol pane:



A new window will appear with the Calc function wizard. When Category is set to “All” or “Add-In” all CalcLink functions are included in the list of browsable functions, along with definitions/explanations of the functions and their arguments and their respective types.



Using the Link

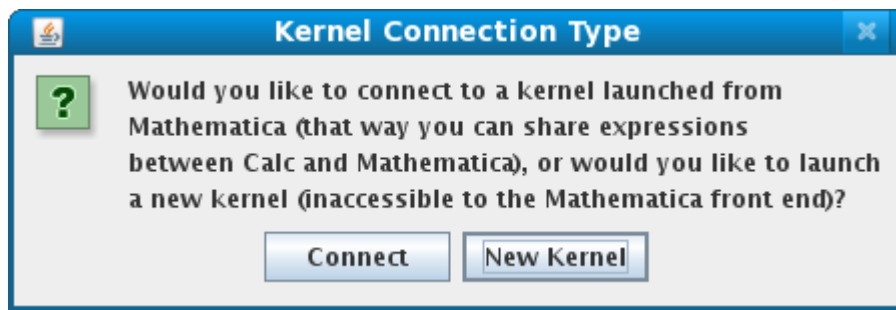
You establish a connection to a *Mathematica* kernel that is already running or launch a new one with the function `CONNECT()`. Simply type

`=CONNECT()`

or

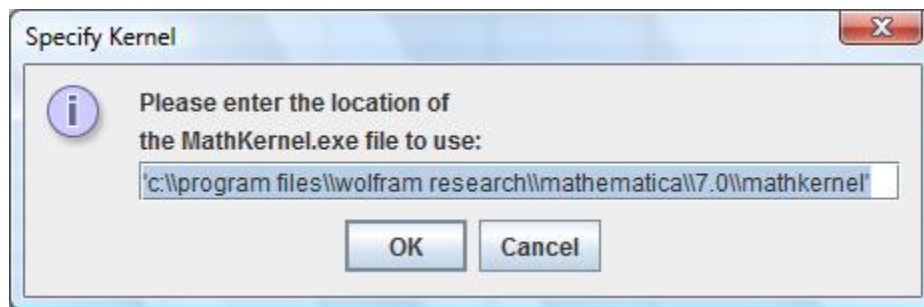
`=connect()`

in a Calc cell. You will get a new window that allows you to specify if you want to use a link that was previously created from the *Mathematica* front-end (see “Preparing the *Mathematica* kernel for use from Calc” below, or if you want to launch a new kernel to be used from Calc, which has no connection/link to any kernel that was launched from *Mathematica*. (These two options make it possible to share one kernel – and thus its data and symbols – between the Calc spreadsheet and a *Mathematica* session/front-end that was started before, or the CalcPad.)

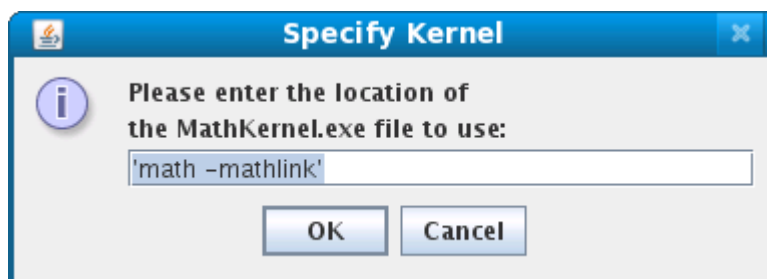


Next, if you chose to launch a new kernel, you will get a window dialog that allows you to specify the kernel file you want to use. You can either accept the default or select a different one, keeping in mind that path names and file locations are very different on different operating systems and that backspace characters and quote characters have to be escaped. For example, Windows uses the backslash character to separate directory names, and Linux/Unix use the forward slash character for that (which does not have to be escaped). CalcLink provides defaults that correspond to the operating system used, but in a few cases it may be necessary to make adjustments.

On Windows Vista:



On Linux/Unix:

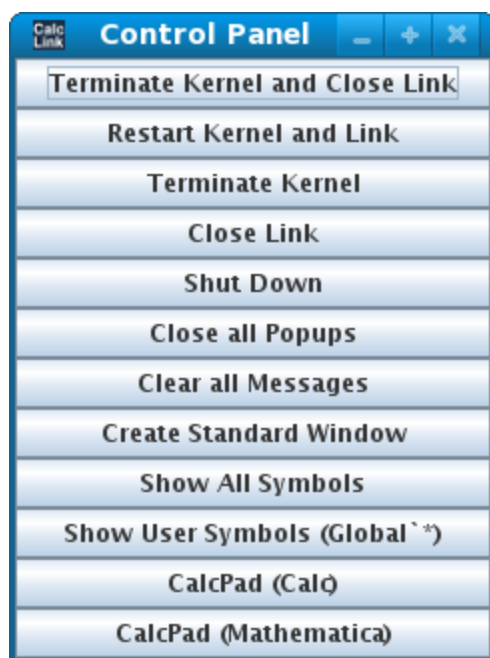
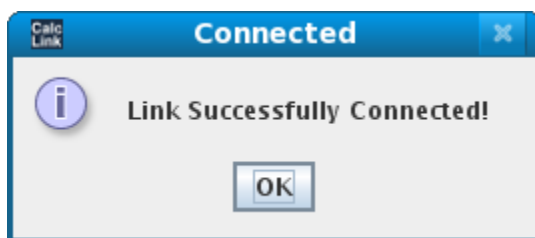


On Windows the kernel file is usually found in
<MathematicaInstallDirectory>\MathKernel.exe
while on Linux/Unix it is usually under

<MathematicaInstallDirectory>/Executables/MathKernel

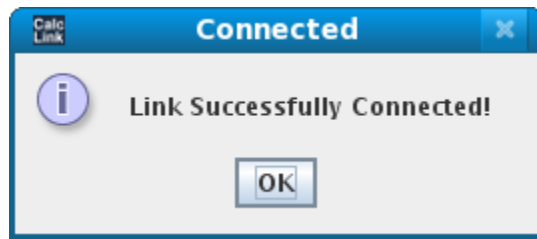
In almost all cases you would want to use the kernel file in the default location, which is the location where the *Mathematica* installer has placed the file (accept the default and click “OK”).

When the connection was successful, you will get a confirmation dialog box, and a new master control window will appear with additional functions for the management of the CalcLink session.



If the file was not found or you have used up all your kernels for other *Mathematica* sessions, you will get an appropriate error message.

If you chose to connect to a kernel that was launched previously from a *Mathematica* front end, CalcLink will connect to that kernel provided it is still running (otherwise an error message will be thrown indicating that no such kernel is available to connect to).



Preparing the *Mathematica* kernel for use with Calc

Simply type

```
CalcEnableKernelSharing[]
```

or

```
CalcEnableKernelSharing[True]
```

after you have loaded the CalcLink package in *Mathematica*. `CalcEnableKernelSharing` prepares the *Mathematica* kernel in an existing *Mathematica* session so that CalcLink can later attach to that kernel. The option `True` further prepares this *Mathematica* kernel to make its computations interruptible from any new link that attaches to it. The default is `False`.

Using the Eval functions

The simplest CalcLink function is `eval[<string>]`. You submit an expression to be evaluated by the *Mathematica* kernel by submitting the string form of the expression through `eval[<string>]` from a Calc spreadsheet cell.

The following example just adds two integers in the *Mathematica* kernel:

A1 f(x) Σ = =EVAL("44+33")				
	A	B	C	D
1	77			
2				
3				

This example computes the determinant with the *Mathematica* determinant function:

A1	f(x) Σ =	=EVAL("Det@{{3,4},{6,7}}")			
	A	B	C	D	E
1	-3				
2					
3					

CalcLink makes it possible to bring symbolic computations to Calc:

A1	f(x) Σ =	=EVAL("First[x/.Solve[3 x-7==4,x]]")			
	A	B	C	D	E
1	3.666667				
2					
3					

You can make assignments in the *Mathematica* kernel that way using Set:

A1	f(x) Σ =	=EVAL("c=88")		
	A	B	C	D
1	88			
2				
3				

This will show the value previously assigned:

A1	f(x) Σ =	=EVAL("c")		
	A	B	C	D
1	88			
2				
3				

If you have a *Mathematica* session in parallel that has its own kernel and when launching CalcLink you didn't launch a new kernel but attached to the one that is already running, you can now also share expressions between the two. For example, the value of the variable *c* would now be known from the *Mathematica* front-end, and any assignments made there could be retrieved that way from Calc.

When submitting CalcLink functions from a Calc spreadsheet cell you can use any string expression you want, including strings that are built with the string concatenation functions from Calc (This would allow you to embed data from your spreadsheet in the expression submitted to the kernel for evaluation, however, for this task the CalcLink functions

EVALRANGE[], EVALRANGEARRAY[], EVALPLOT RANGE[], and EVALPLOT RANGESEND[] are much more convenient, see below).

Another very useful function is EVALRANGE[<expression string>,<range string>]. Using the determinant example above and with the data {{3, 4}, {6, 7}} in cells A1 through B2 we get:

A4		f(x) Σ =	=EVALRANGE("Det";A1:B2)		
	A	B	C	D	
1	3	4			
2	6	7			
3					
4	-3				
5					
6					

The string representing the *Mathematica* expression can be any valid *Mathematica* expression, however, pure functions are used in the expression submitted to the kernel for evaluation, so in many cases # and & have to be used to build an expression. In the following example, B1 contains

=EVALRANGE("Sum[k,{k,#}]&";A1")

and is copied downward to B10.

B10		f(x) Σ =	=EVALRANGE("Sum[k,{k,#}]&";A10)		
	A	B	C	D	E
1	1	1			
2	2	3			
3	3	6			
4	4	10			
5	5	15			
6	6	21			
7	7	28			
8	8	36			
9	9	45			
10	10	55			
11					
12					

With the numbers 1 through 10 in cells A1 through A10 column B now contains the sums of the first k integers, by row. (Also notice that the cells A1 through A10 were used individually when the second argument of EVALRANGE[] provides for a full cell range. This is true for all CalcLink functions that take cell ranges as arguments: it's always possible to apply a range formula to a single cell as well.)

Of course the same can be obtained by splicing in the cell values with & in the Calc function line:

B10		f(x)	Σ	=	=EVAL("Binomial["&A10&"+1,2])"
	A	B	C	D	E
1	1	1			
2	2	3			
3	3	6			
4	4	10			
5	5	15			
6	6	21			
7	7	28			
8	8	36			
9	9	45			
10	10	55			
11					

CalcLink fully supports array formulas. You can compute one- and two-dimensional data in *Mathematica* and return it over an array in Calc with the functions EVALARRAY[] and EVALRANGEARRAY[]. You first have to select the cells in which you want to place the output, e. g. A1:D4. Then you type the equals sign (=), this will allow you to type the CalcLink function in the lower right cell of the selected cell range. You then have to press Cntrl+Shift+Return to indicate to Calc that you want the formula to be applied as an array formula (and not as a cell formula).

Assuming A1:D4 are selected/highlighted, typing

=EVALARRAY("Table[i j,{i,4},{j,4}]")

and then pressing Ctrl+Shift+Return will place the table output in the cells A1:D4.

A1:D4		f(x)	Σ	=	{=EVALARRAY("Table[i j,{i,4},{j,4}]")}
	A	B	C	D	E
1	1	2	3	4	
2	2	4	6	8	
3	3	6	9	12	
4	4	8	12	16	
5					
6					

The function EVALRANGEARRAY[] works similar to EVALARRAY[], but it also allows to use a range definition for the computation (similar to EVALRANGE[], but returning an array).

In the following example we display the Hilbert matrix with 8 rows and columns in Calc. All we need to do is highlight the cells A1:H8 and enter the formula

=evalarray("HilbertMatrix[8]")

and hit Ctrl-Shift+Enter.

A1:H8		f(x)	Σ	=	{=EVALARRAY("HilbertMatrix@8")}				
	A	B	C	D	E	F	G	H	I
1	1.0000000	0.5000000	0.3333333	0.2500000	0.2000000	0.1666667	0.1428571	0.1250000	
2	0.5000000	0.3333333	0.2500000	0.2000000	0.1666667	0.1428571	0.1250000	0.1111111	
3	0.3333333	0.2500000	0.2000000	0.1666667	0.1428571	0.1250000	0.1111111	0.1000000	
4	0.2500000	0.2000000	0.1666667	0.1428571	0.1250000	0.1111111	0.1000000	0.0909091	
5	0.2000000	0.1666667	0.1428571	0.1250000	0.1111111	0.1000000	0.0909091	0.0833333	
6	0.1666667	0.1428571	0.1250000	0.1111111	0.1000000	0.0909091	0.0833333	0.0769231	
7	0.1428571	0.1250000	0.1111111	0.1000000	0.0909091	0.0833333	0.0769231	0.0714286	
8	0.1250000	0.1111111	0.1000000	0.0909091	0.0833333	0.0769231	0.0714286	0.0666667	
9									
10									

The CalcLink function EVALARRAY[] is used when the input is a *Mathematica* expression and the output is two-dimensional (including one-dimensional, but not a simple cell). The use of "...ARRAY" in the CalcLink function name denotes that the output is a Calc cell range.


When the input consists of a *Mathematica* expression applied to data as well as the data, and the output is for a single cell, the CalcLink function EVALRANGE[] is used. The use of "...RANGE" in the CalcLink function name denotes that the input is a Calc cell range.

The following computes the determinant of the matrix in the cell range A1:D4 and places it in the cell A6:

A6		f(x)	Σ	=	{=EVALRANGE("Det";A1:D4)}				
	A	B	C	D	E				
1	11	8	15	-17					
2	2	-5	12	-7					
3	-3	5	11	-2					
4	14	4	-19	-7					
5									
6	3480								
7									
8									

When the input consists of a *Mathematica* expression applied to data as well as the data, and the output is two-dimensional (including one-dimensional) the CalcLink function EVALRANGEARRAY[] is used. The use of "...RANGEARRAY" in the CalcLink function name denotes that the input is a Calc cell range and that the output is a Calc cell range.

The following computes the inverse of the matrix in the cell range A1:D4 and places it in the cell range A6:D9:

A6:D9			$f(x)$	Σ	=	{=EVALRANGEARRAY("Inverse";A1:D4)}	
	A	B	C	D	E		
1	11	8	15	-17			
2	2	-5	12	-7			
3	-3	5	11	-2			
4	14	4	-19	-7			
5							
6	0.537931	-0.531034	-1.000000	-0.489655			
7	0.039943	-0.100862	0.041667	-0.008046			
8	0.219828	-0.202586	-0.375000	-0.224138			
9	0.502011	-0.569828	-0.958333	-0.518391			
10							
11							

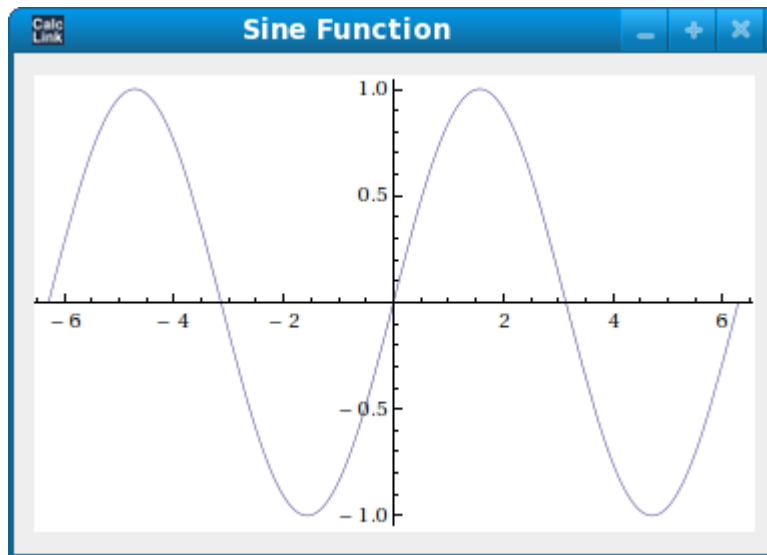
Using the CalcLink Graphics Functions

CalcLink provides functions to create new windows with *Mathematica* graphics from a Calc spreadsheet cell or the CalcLink control center. The CalcLink function EVALPLOT[] generates *Mathematica* graphics output that results from the submitted expression, creates a new window on the screen, and in the spreadsheet cell from which it was submitted it returns an integer that represents that particular window instance (so that it can be referred to later). The title string allows to set a title for the window, and the third and fourth arguments specify width and height of the new window, with 1 and 1 indicating that defaults should be used (appropriate for the widths and heights of the resulting graphics output).

For example,

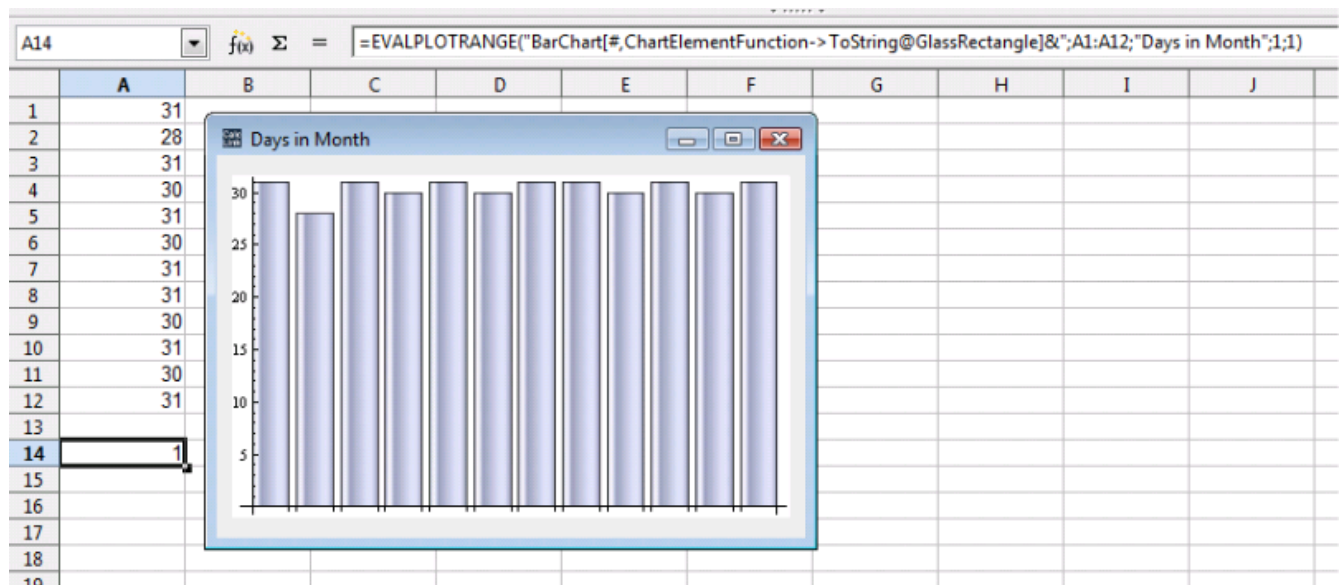
```
=EVALPLOT("Plot[Sin[x],{x,-2Pi,2Pi}];" "Sine Function";1;1)
```

will pop up a new window with default width and height with a plot of the sine function and window title "Sine Function", and CalcLink will return a 1, indicating that this can now be referred to as "window handle 1" (see EVALPLOTSEND[] and EVALPLOTTRANGESEND[] below).






Any CalcLink function creating a new window would now return a 2 as the window identifier, but return a 2, so the window can subsequently be referred to by using 2 as a window reference.



The following example displays a simple bar chart with the number of days of the months. The data is read from the cell range A1:A12, the title of the window is set to "Days in Month", and window sizing (height and width) is done automatically when the values for height and width are set to 1.



The cell range can be filled in by dragging the mouse over the range, a red rectangle indicates the selected range:

		  	=EVALPLOT(RANGE("BarChart[#,ChartElementFunction->ToString@GlassRectangle]&"A1:A12"Days in Month";1;1)							
	A	B	C	D	E	F	G	H	I	J
1		31								
2		28								
3		31								
4		30								
5		31								
6		30								
7		31								
8		31								
9		30								
10		31								
11		30								
12		31								
13										
14	=EVALPLOT(RANGE("BarChart[#,ChartElementFunction->ToString@GlassRectangle]&"A1:A12"Days in Month";1;1)									
15										
16										

The contents displayed can be any valid expression returning something that can be displayed (roughly, everything except for Null, \$Failed, \$Aborted, or the various Hold[] functions):

A1		 	=EVALPLOT("TableForm@Table[Hypergeometric1F1[k,l,z],[k,0,5],[l,1,3]]";"Hypergeometric Functions";500;200)							
	A	B	C	D	E	F	G	H	I	J
1		2								
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										

Hypergeometric Functions		
1	$\frac{1}{s}$	$\frac{1}{s^2} \frac{(-1+e^s-s)}{s^2}$
e^s	$\frac{-1+e^s}{s}$	$\frac{2}{s^2} \frac{(1-e^s+e^s s)}{s^2}$
$e^s (1+z)$	$e^s \left(1 + \frac{s}{2}\right)$	$e^s \left(1 + \frac{s}{2}\right)$
$e^s \left(1 + 2z + \frac{s^2}{2}\right)$	$e^s \left(1 + z + \frac{s^2}{6}\right)$	$e^s \left(1 + \frac{s}{3}\right)$
$e^s \left(1 + 3z + \frac{3s^2}{2} + \frac{s^3}{6}\right)$	$e^s \left(1 + \frac{3s}{2} + \frac{s^2}{2} + \frac{s^3}{24}\right)$	$e^s \left(1 + \frac{2s}{3} + \frac{s^2}{12}\right)$
$e^s \left(1 + 4z + 3z^2 + \frac{2s^3}{3} + \frac{s^4}{24}\right)$		

The CalcLink functions of the ...SEND type make it possible to update graphics or any other displayed material (tables, formulae) live as the cells providing the input for the *Mathematica* computation change. For example, one could use live streaming data from a financial network and update the display live, as the prices tick in the Calc spreadsheet.

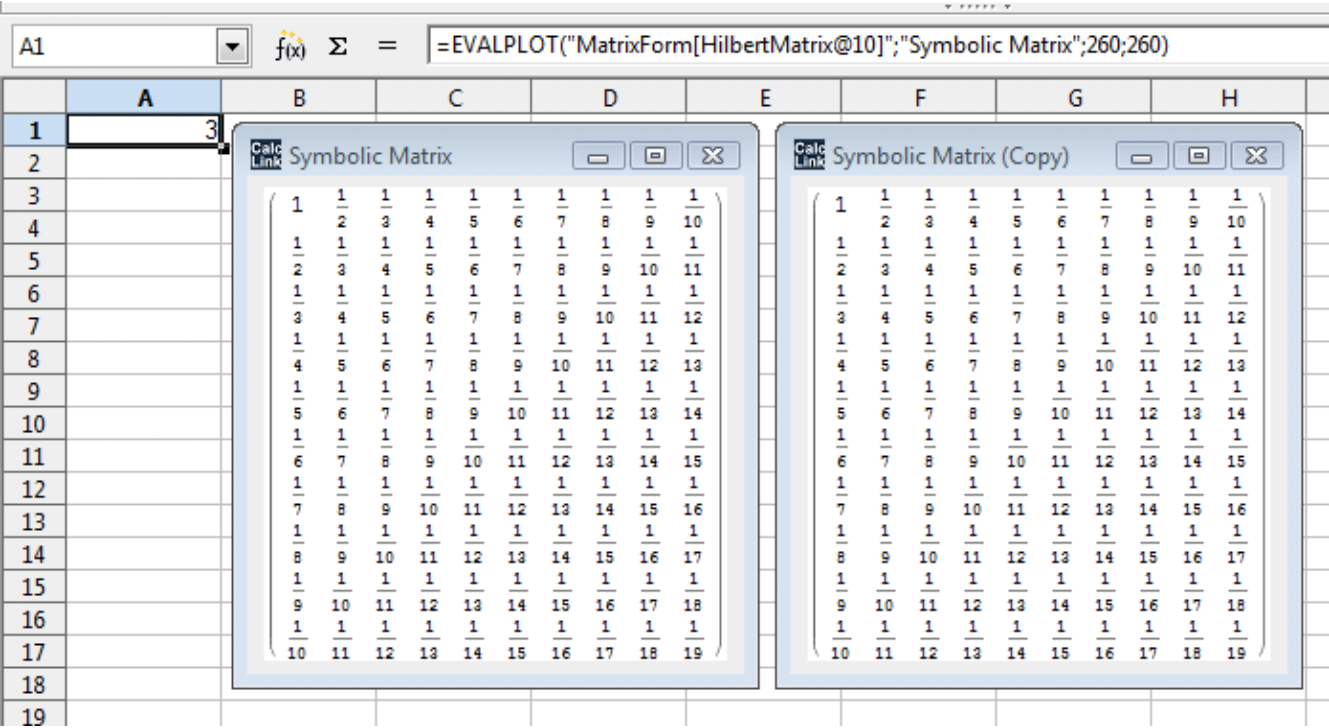
The CalcLink function EVALPLOTSEND[] takes a "window identifier" as generated by all CalcLink plot functions as the first argument, and generates a new *Mathematica* graphics expression based on the the evaluation of the second argument string, and REPLACES the graphics in the window identified by the "window identifier", instead of creating a new window. With the function EVALPLOTSEND[] it is possible to replace the contents of an existing window with new graphics every time a recomputation is triggered in Calc.

The CalcLink function EVALRANGEPLOTSEND[] works similar to EVALPLOTSEND[], but it

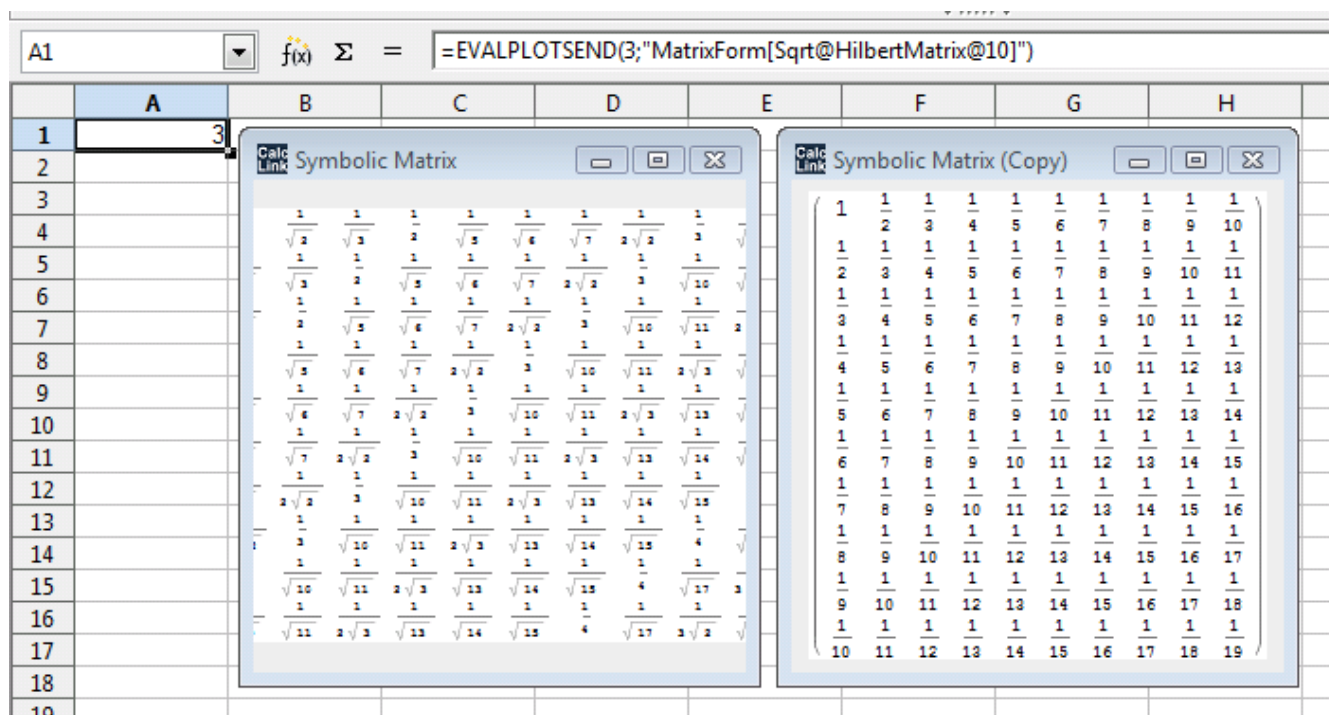
allows for the use of cell ranges to include data in the submission to the kernel.

Window Copy

CalcLink provides a "window copy" feature that allows the user to create an identical copy of a graphics window with a simple right-click (or center-click, including depressable mouse-wheels) anywhere in the window area. A new window will be created displaying the contents of the original window and adding "(COPY)" in the title. The copy can be used to inspect how the graphics output changes as the window contents in the original window are modified/updated.



Here we replace the contents of the original window with new content:



This feature also makes it possible to keep "snapshots" of dynamically changing windows. Any new or updated window can be copied any number of times, and thus the user can "save" the contents of an original window before it is updated (otherwise all previous content in an updated window is lost), or closed.

You can create as many identical window copies from any original window as you want, and they are all included in the list of windows to be closed when the "Close all Popups" button in the control center is pressed. Copies are not themselves copyable, only the original window can be copied. Any updates to a window identifier go to the original window, never to a copy.

NOTE:

It is important to understand that all CalcLink functions resubmit their evaluation strings to the *Mathematica* kernel every time the cell contents that are used by CalcLink functions changes.

This can be

- data that CalcLink range functions (e. g. EVALRANGE[]) refer to, or
- the contents of cells that contain CalcLink functions (e. g. by copying such cells).

If nothing in the inputs to CalcLink functions changes, CalcLink will not resubmit for evaluation as there is nothing to recompute.

In the first case this means that if CalcLink functions are used in many spreadsheet cells that refer to a lot of data that changes a lot, several computations will be triggered. In the second case these computations can be triggered by copying cells with CalcLink functions either

manually or by macro.

In both cases, if many CalcLink functions are used that generate new windows with *Mathematica* graphics, many new windows would pop up. Therefore it is important to ensure that CalcLink functions that create NEW graphics windows are used judiciously when data in spreadsheet cells changes frequently or macros are used.

If properly used, this is a strong advantage. Having both CalcLink functions that create new windows as well as CalcLink functions that replace the graphics contents of existing windows with new contents makes it possible to:

- generate a few new windows with contents that shouldn't change anymore afterwards
- update existing windows with new contents when recomputations are triggered in Calc when input data has changed, thus providing “live” updates in the cells or in the windows with *Mathematica* graphics. Quite often the cell contents in a spreadsheet change frequently, e. g. through streaming financial data from financial networks or web services (or, in fact, a *Mathematica* program using the CalcLink package to update spreadsheet cells!), or by manual recomputations (F9), and this design enables the user to get instant graphics updates based on recomputations without having to manually trigger the updates of the window contents. It does, however, make it important to use CalcLink functions that create new windows with care!

Mathematica Symbol Browser

CalcLink provides two *Mathematica* symbol browsers. If you type =SHOWSYMBOLS(0) in any Calc spreadsheet cell or press the button “Show All Symbols” in the command center pane, you will get a new window that shows a list of all *Mathematica* symbols in a scrollable grid, along with their evaluated values.

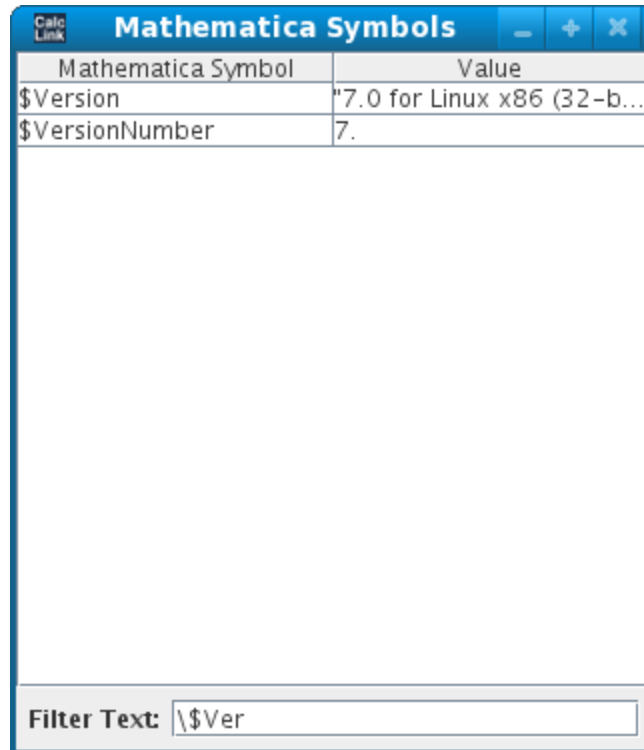
Mathematica Symbols	
Mathematica Symbol	Value
Abort	Abort
AbortKernels	AbortKernels
AbortProtect	AbortProtect
Above	Above
Abs	Abs
AbsoluteCurrentValue	AbsoluteCurrentValue
AbsoluteDashing	AbsoluteDashing
AbsoluteFileName	AbsoluteFileName
AbsoluteOptions	AbsoluteOptions
AbsolutePointSize	AbsolutePointSize
AbsoluteThickness	AbsoluteThickness
AbsoluteTime	AbsoluteTime
AbsoluteTiming	AbsoluteTiming
AccountingForm	AccountingForm
Accumulate	Accumulate
Accuracy	Accuracy
AccuracyGoal	AccuracyGoal
ActionDelay	ActionDelay
ActionMenu	ActionMenu
Filter Text: <input type="text"/>	

At the bottom of the window is a filter field, where you can enter the names of *Mathematica* symbols (or parts thereof), and the window will automatically filter out the *Mathematica* symbols that contain the strings or substrings you have entered. This shows a list of all *Mathematica* symbols containing “Pi”.

Mathematica Symbols	
Mathematica Symbol	Value
EllipticPi	EllipticPi
HeavisidePi	HeavisidePi
LightPink	RGBColor[1, 0.925, 0.9...
Pi	3.141592653589793`
Pick	Pick
Piecewise	Piecewise
PiecewiseExpand	PiecewiseExpand
PieChart	PieChart
PieChart3D	PieChart3D
Pink	RGBColor[1, 0.5, 0.5]
Pivoting	Pivoting
PixelConstrained	PixelConstrained
PrimePi	PrimePi
\$MaxPiecewiseCases	100.`
\$PipeSupported	True
Filter Text: <input type="text" value="Pi"/>	

The *Mathematica* Symbol Browser uses Perl 5 regular expression pattern matching, so you can use \ (backspace character) as an escape character (e. g. “\...” to see all *Mathematica* symbols beginning with a dollar sign).

The following shows all symbols that begin with “\$Version”:



The image shows a window titled "Mathematica Symbols" with a blue header bar. Inside the window is a table with two columns: "Mathematica Symbol" and "Value". The table contains two rows of data. Below the table is a "Filter Text:" label followed by a text input field containing the text "\\$Ver".

Mathematica Symbol	Value
\$Version	"7.0 for Linux x86 (32-b...
\$VersionNumber	7.

Filter Text: \\$Ver

The symbols window also displays its contents on the tooltip, so longer strings in the output can be further inspected with the tooltip (or the window resized).

Mathematica Symbol	Value
\$MachineAddresses	{fe80::9c03:7d36:5db7:684...
\$MachineDomain	—
\$MachineDomains	{}
\$MachineEpsilon	2.220446049250313 ⁻¹⁶
\$MachineID	"6102-23786-95542"
\$MachineName	"mooniac-pc"
\$MachinePrecision	15.954589770191003
\$MachineType	"PC"
\$MaxExtraPrecision	50.
\$MaxLicenseProcesses	2.
\$MaxLicenseSubprocesses	4.
\$MaxMachineNumber	1.7976931348623157 ³⁰⁸
\$MaxNumber	5.2975574590400395044...
\$MaxPiecewiseCases	100.
\$MaxPrecision	Infinity
\$MaxRootDegree	1000.

Filter Text: \$Ma

The following shows all symbols that end with “Plot”:

Mathematica Symbol	Value
ArrayPlot	ArrayPlot
ContourPlot	ContourPlot
DateListLogPlot	DateListLogPlot
DateListPlot	DateListPlot
DensityPlot	DensityPlot
DiscretePlot	DiscretePlot
GraphPlot	GraphPlot
LayeredGraphPlot	LayeredGraphPlot
LineIntegralConvolution...	LineIntegralConvolution...
ListContourPlot	ListContourPlot
ListCurvePathPlot	ListCurvePathPlot
ListDensityPlot	ListDensityPlot
ListLineIntegralConvoluti...	ListLineIntegralConvoluti...
ListLinePlot	ListLinePlot
ListLogLinearPlot	ListLogLinearPlot
ListLogLogPlot	ListLogLogPlot
ListLogPlot	ListLogPlot
ListPlot	ListPlot
ListPolarPlot	ListPolarPlot

Filter Text: Plot\$

The following shows all symbols that contain “Plot” but don't begin with “L”:

Mathematica Symbol	Value
ArrayPlot	ArrayPlot
ContourPlot	ContourPlot
ContourPlot3D	ContourPlot3D
DensityPlot	DensityPlot
DiscretePlot	DiscretePlot
GraphPlot	GraphPlot
GraphPlot3D	GraphPlot3D
MatrixPlot	MatrixPlot
MaxPlotPoints	MaxPlotPoints
ParametricPlot	ParametricPlot
ParametricPlot3D	ParametricPlot3D
Plot	Plot
Plot3D	Plot3D
Plot3Matrix	Plot3Matrix
PlotDivision	PlotDivision
PlotJoined	PlotJoined
PlotLabel	PlotLabel
PlotMarkers	PlotMarkers
PlotPoints	PlotPoints

Filter Text:

The symbol browser includes your own symbols as well as the symbols of *Mathematica* packages you have loaded. For example, if you have made a symbol definition `mysymbol=232`, it will be included in the symbol list. (The list displayed contains everything included in `Names["*"]`, which includes the symbols from loaded packages.)

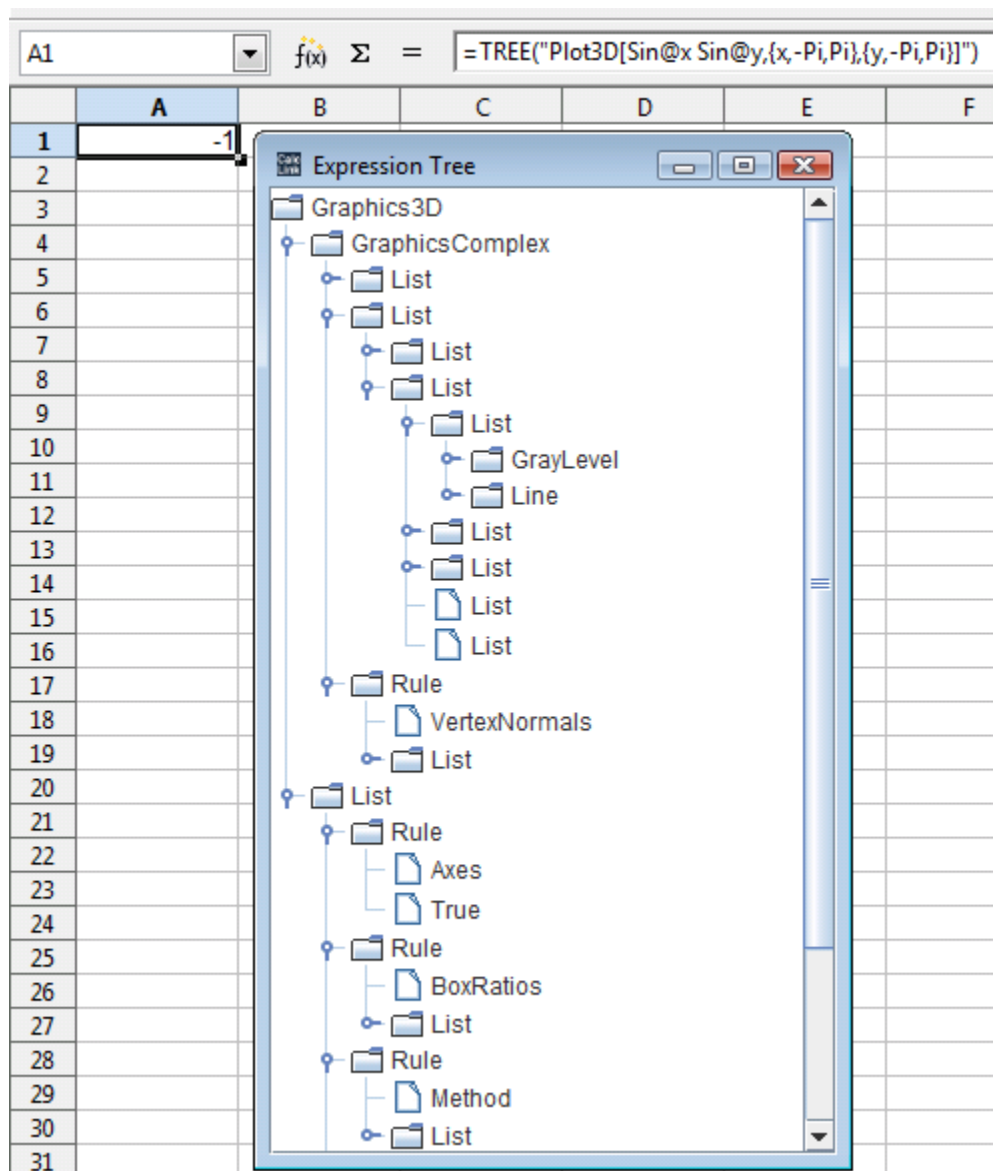
The other *Mathematica* symbol browser only displays the contents of the `Global`` context. This includes all user-defined symbols (unless they were put in their own contexts).

You can launch as many *Mathematica* symbol browsers as you want and filter for symbols in them independently of one another. New symbols browsers HAVE to be launched after new symbols have been created (user-defined or from a *Mathematica* package that was loaded) to make them appear in the displayed list.

Once a symbol browser is displayed the symbol list contained therein can not be changed/updated anymore. This is not considered a drawback because when the user is done with a task in the symbol browser the user will most likely close the window (which also releases the memory space held for the list of over 3600 symbols along with their values in the Java heap), and a new one with an updated symbol list can always be popped up again with a simple click on the push-button “Show All Symbols” on the CalcLink control center pane.

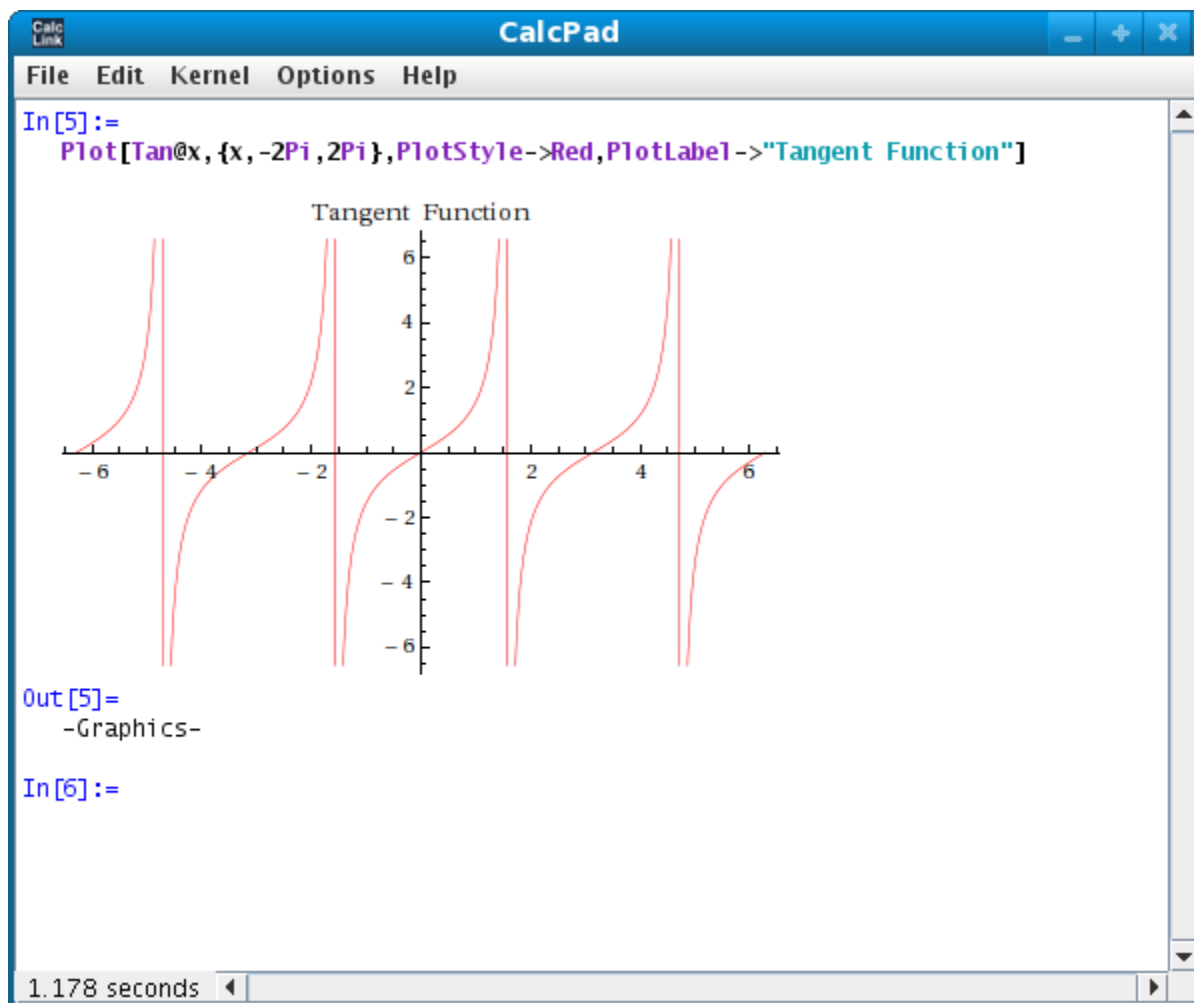
Interactive Tree Representation/Inspection of *Mathematica* Expressions

The function `TREE()` creates a new window containing an interactive tree of the expression using expandable/collapsible tree nodes. It is the same as the CalcLink function `CalcTree[]` of the *Mathematica* plug-in (but requires a string as input, not an expression).



CalcPad

CalcLink contains a “mini front-end” called “CalcPad” (“Scratchpad for CalcLink”). It is a much simpler front-end than the full *Mathematica* front-end, but it is quite useful for tasks where the full *Mathematica* front-end is not needed, runtime memory is critical, and merely a “scratchpad” type of front-end is desired. The *Mathematica* front-end has many more features, but also requires a much greater overhead in terms of memory and complexity.



There are two ways to launch the CalcPad: Attaching to the kernel that is used from Calc (which itself can be using the kernel that was launched from an outside *Mathematica* session, or can have its own kernel), or by attaching to a *Mathematica* kernel that was launched from another *Mathematica* session.

If you launch the CalcPad with =CALCPAD(0) in a spreadsheet cell the CalcPad will use the kernel that is used by Calc. This may be its own kernel, or one launched from another *Mathematica* session previously. If you launch the CalcPad with =CALCPAD(1) in a spreadsheet cell, CalcPad will be launched and attach to a kernel that was launched from another *Mathematica* session previously (but not attach to the kernel used by Calc). This provides for a very flexible setup of expression sharing between the three front-ends (see

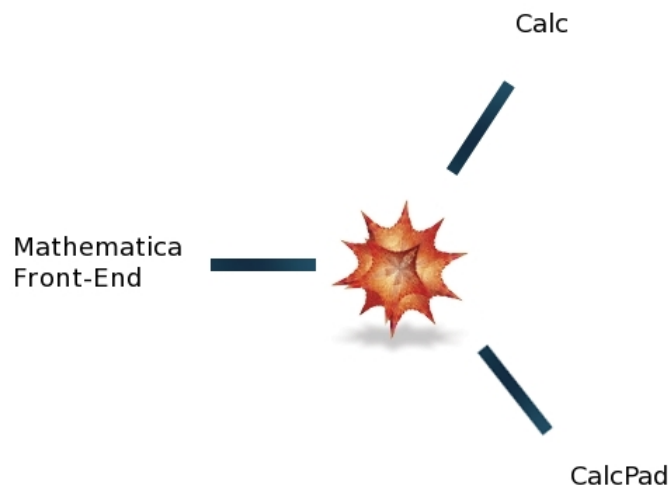
below).

Many more areas of application can be thought of. On a .Net platform NETLink can be launched, and the user can program in NETLink in the CalcPad from Calc! Or a new Java runtime can be launched with J/Link, and additional Java or J/Link programming can be done from the CalcPad from Calc!

Possible kernel-sharing options

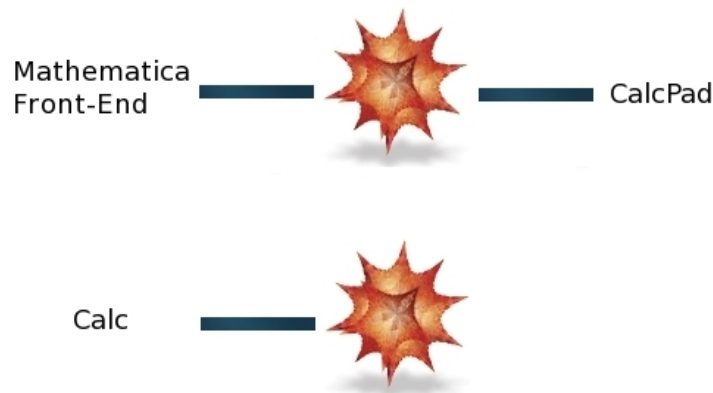
All three front-ends share the same kernel:

In a *Mathematica* session use `CalcEnableKernelSharing[]` or `CalcEnableKernelSharing[True]` BEFORE using CalcLink from Calc. Then launch CalcLink from Calc by choosing the “Connect” option (left button) in the initial dialog. Then type `=CALCPAD(1)` in a spreadsheet cell. Now any symbols from either one of the three front ends will be known in the respective other two, because they are all in the same kernel.



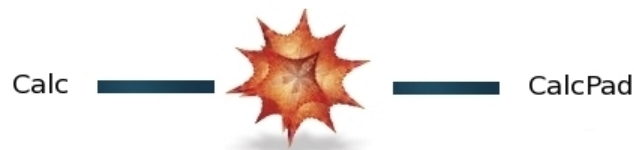
Mathematica and CalcPad share the same kernel, Calc has its own:

In a *Mathematica* session use `CalcEnableKernelSharing[]` or `CalcEnableKernelSharing[True]` BEFORE using CalcLink from Calc. Then launch CalcLink from Calc by choosing the “New Kernel” option (right button) in the initial dialog. Then type `=CALCPAD(1)` in a spreadsheet cell. Now the *Mathematica* front-end and CalcPad share symbols, while the kernel used by Calc is separate.



Calc and CalcPad share the same kernel (no *Mathematica* front-end used):

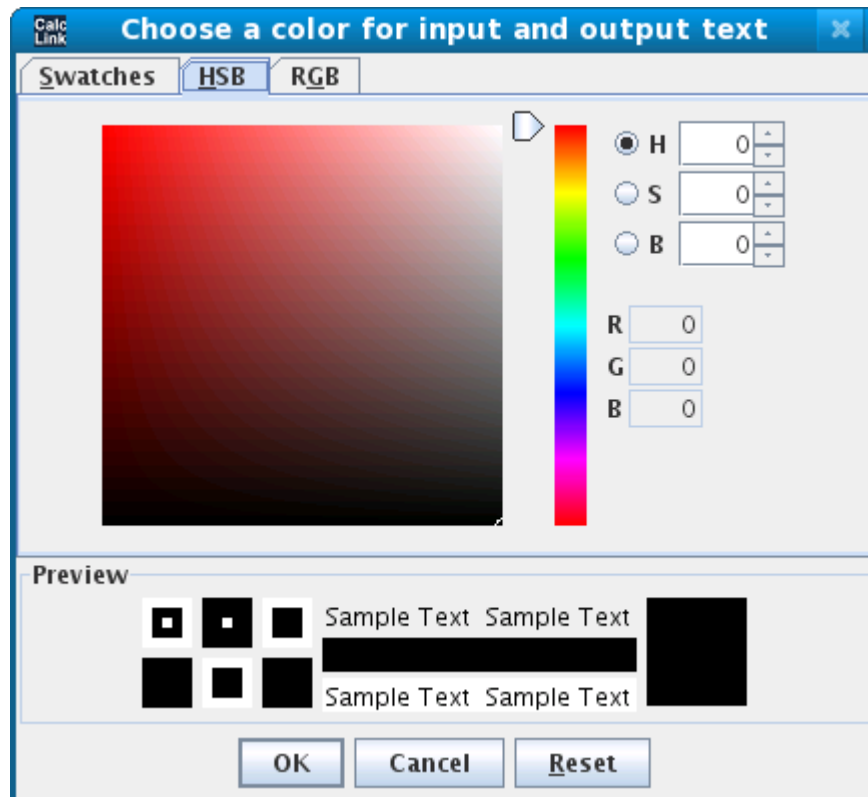
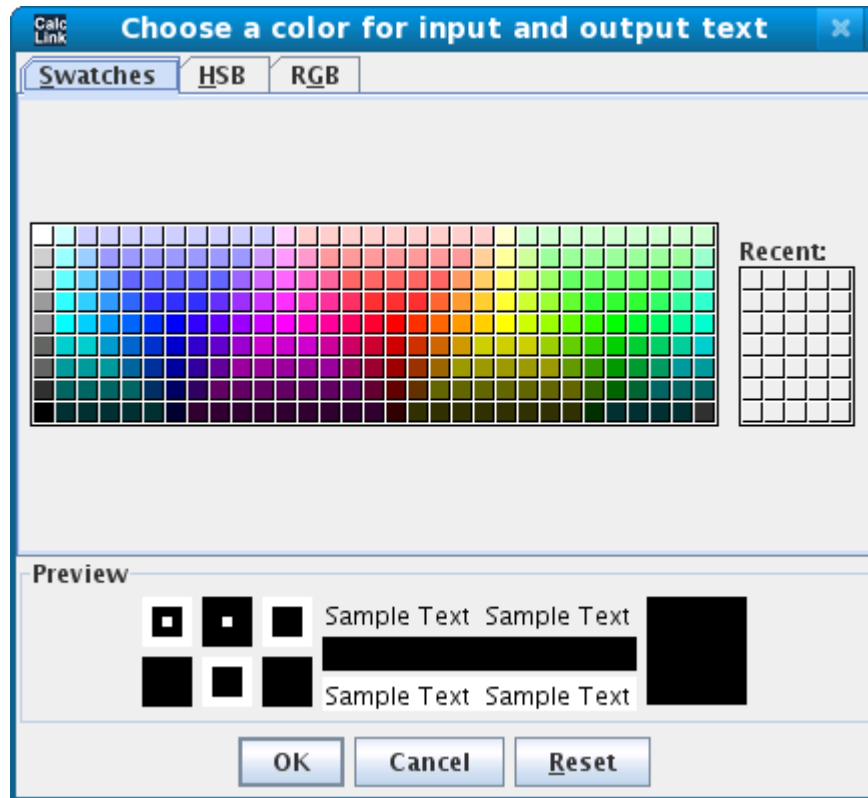
When you launch CalcLink from Calc, you choose the “New Kernel” option (right push-button) in the initial dialog. Then type either =CALCPAD(0) or =CALCPAD(1) in a spreadsheet cell. Now Calc and CalcPad share the same kernel.

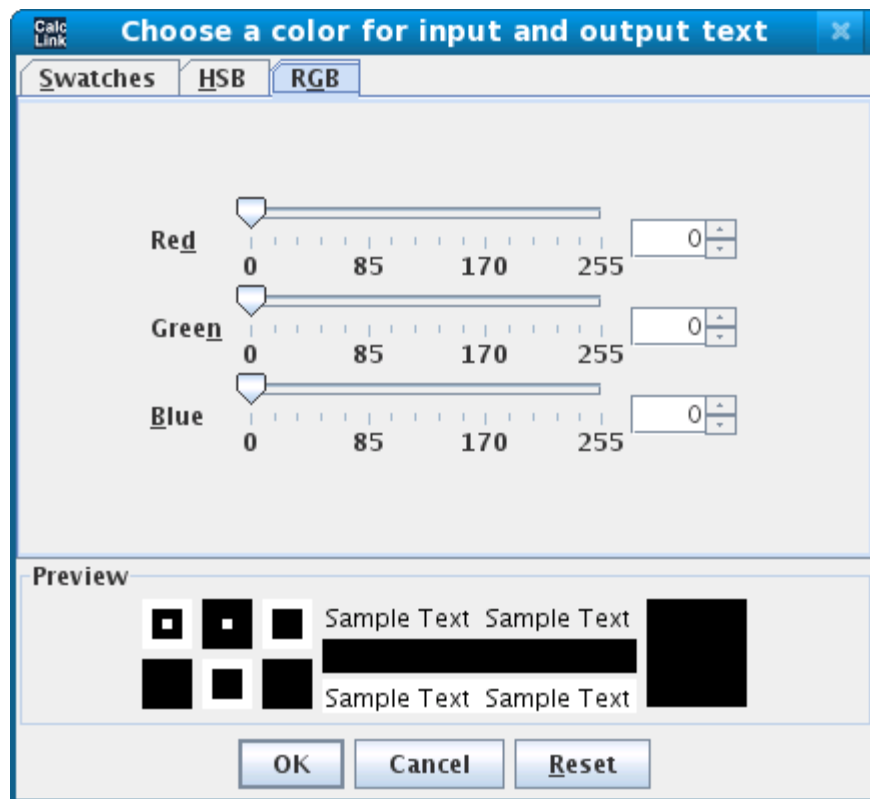


Features of CalcPad:

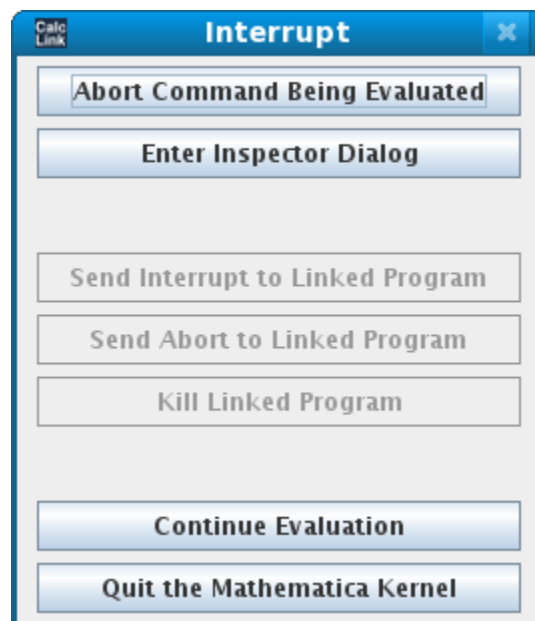
CalcPad offers powerful features that make it a very convenient “scratchpad” for Calc interaction. CalcPad provides bracket-matching for {} and () and [] type brackets, syntax coloring, various color, font size, font style selections, unlimited Undo / Redo functionality, a dialog for interrupting/aborting running computations in the kernel, as well as copy/cut/paste functionality (with the clipboard, e. g. for *Mathematica* or Calc or any other application), to name a few. Those features can be found in the menus of CalcPad.

Various ways to specify font styles:





A dialog to control the active kernel evaluation:

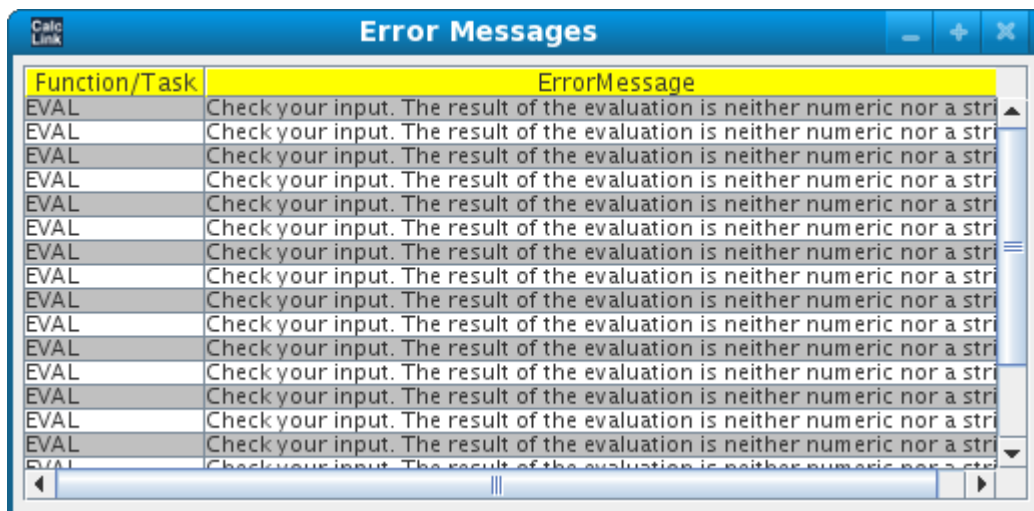


CalcPad uses the *Mathematica* front-end as a service. Therefore, all graphics have the features provided by the *Mathematica* front-end for graphics, e. g. anti-aliasing, adaptive plot refinement, etc.

You can launch as many CalcPads as you want and use them independently of one another, however, they all share the same kernel, and that kernel is always shared with Calc, or the *Mathematica* front end, or both.

Error Messages

If CalcLink encounters errors/exceptions they are displayed in an error messages window. The window contains a table with two columns, one showing the name of the function or task during which the exception occurred (e. g. during “EVAL” or during the “Attempt to Restart the Kernel”), and the other showing the actual textual message of the exception.



Function/Task	ErrorMessage
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string
EVAL	Check your input. The result of the evaluation is neither numeric nor a string

Every exception is included in the error messages window, critical exceptions as well as non-critical exceptions. No matter what part of CalcLink encountered the exception, it is listed in the error messages window. Non-critical exceptions can for example be the result of submitting an expression to the kernel that doesn't return a *Mathematica* expression at all, such as =EVAL(“Hold[5]”). The *Mathematica* function Hold does not return anything, therefore CalcLink can not display its output as numeric or string.

Note that all symbols that are returned as a valid *Mathematica* symbol other than numeric or string are converted to string, for example =EVAL(“Head@{1,2,3}”) is returned as the string “List” although the return type of the requested expression is List, not string! Any valid return that is not numeric is automatically converted to string by CalcLink, however, there MUST BE a valid return (unlike the return of Hold)!

Mistakes resulting in non-critical exceptions can easily happen, for example by copying cells with data or formulas that all cause non-critical exceptions to be thrown. Such mistakes should not cause new windows with error messages to appear, as the mistake that was made is fairly obvious, and the user does not want to get a series of modal dialog boxes which all

but the first one carry no meaningful information for the user. Therefore these exceptions are displayed in a blotter-style error messages window, where new ones are simply appended line by line.

Critical exceptions, however, have to interrupt the flow of interaction of the user as immediate attention is needed, and therefore modal dialog boxes describing the critical exception appear in addition to appending the exception messages in the error messages window. The two most important cases for critical exceptions are J/Link and MathLink exceptions that relate to the availability of the link. In some cases it is possible to recover from a link error in J/Link, but in almost all cases of MathLink errors it is impossible to recover. If CalcLink was closed or the connection object `calclink` is no longer available (for example because it was closed by the user with the function `=CLOSELINK()` or pressing the “Close Link” button in the CalcLink control center pane), the user can simply launch a new CalcLink (by typing `=CONNECT()` in a cell or by pressing one of the buttons in the CalcLink control center pane to start a new CalcLink and/or kernel). However, if the user closed Calc and/or OpenOffice and/or closed the *Mathematica* kernel and subsequently attempts to submit an expression to the *Mathematica* kernel for evaluation, CalcLink errors, J/Link errors, and MathLink errors are thrown, and it is not possible to recover from this situation. It is necessary to restart Calc and CalcLink.



Error Messages	
Function/Task	ErrorMessage
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Check your input. The result of the evaluation is neither numeric nor a stri...
EVAL	Error trying to return a result of numeric or string type
EVAL	Link is not open.
EVAL	MathLinkException was unrecoverable; closing link.

In general, if CalcLink is still running, it is possible to recover from critical exceptions, but if CalcLink itself is closed or disconnected, it is not possible to recover.

Appendix A

Document Filters

The following table shows the filter names available in OpenOffice along with their options. For the CSV filter see Appendix B.

Filter name	Description	Import	Export
StarOffice XML (Calc)	Standard XML filter	•	•
calc_StarOffice_XML_Calc_Template	XML filter for templates	•	•
StarCalc 5.0	The binary format of StarOffice Calc 5.x	•	•
StarCalc 5.0 Vorlage/Template	StarOffice Calc 5.x templates	•	•
StarCalc 4.0	The binary format of StarCalc 4.x	•	•
StarCalc 4.0 Vorlage/Template	StarCalc 4.x templates	•	•
StarCalc 3.0	The binary format of StarCalc 3.x	•	•
StarCalc 3.0 Vorlage/Template	StarCalc 3.x templates	•	•
HTML (StarCalc)	HTML filter	•	•
calc_HTML_WebQuery	HTML filter for external data queries	•	
MS Excel 97	Microsoft Excel 97/2000/XP	•	•
MS Excel 97 Vorlage/Template	Microsoft Excel 97/2000/XP templates	•	•
MS Excel 95	Microsoft Excel 5.0/95	•	•
MS Excel 5.0/95	Different name for the same filter	•	•
MS Excel 95 Vorlage/Template	Microsoft Excel 5.0/95 templates	•	•

MS Excel 5.0/95 Vorlage/Template	Different name for the same filter	•	•
MS Excel 4.0	Microsoft Excel 2.1/3.0/4.0	•	
MS Excel 4.0 Vorlage/Template	Microsoft Excel 2.1/3.0/4.0 templates	•	
Lotus	Lotus 1-2-3	•	
Text - txt - csv (StarCalc)	Comma separated values	•	•
Rich Text Format (StarCalc)		•	•
dBase		•	•
SYLK	Symbolic Link	•	•
DIF	Data Interchange Format	•	•

Appendix B

Document Filters: CSV Filter

The following table shows the filter names available in OpenOffice for the CSV filter.

This filter accepts an option string containing five tokens, separated by commas. The following table shows an example string for a file with four columns of type date - number - number - number. In the table the tokens are numbered from (1) to (5). Each token is explained below.

Example Filter Options String	Field Separator (1)	Text Delimiter (2)	Character Set (3)	Number of First Line (4)	Cell Format Codes for the four Columns (5)	
					Column	Code
File Format: Four columns date-num- num-num	,	"	System	line no. 1	1 2 3 4	YY/MM/DD = 5 Standard = 1 Standard = 1 Standard = 1
Token	44	34	0	1	1/5/2/1/3/1/4/1	

For the filter options above, set the PropertyValue FilterOptions in the load arguments to "44,34,0,1,1/5/2/1/3/1/4/1". There are a number of possible settings for the five tokens.

1. Field separator(s) as ASCII values. Multiple values are separated by the slash sign ("/"), that is, if the values are separated by semicolons and horizontal tabulators, the token would be 59/9. To treat several consecutive separators as one, the four letters /MRG have to be appended to the token. If the file contains fixed width fields, the three letters FIX are used.
2. The text delimiter as ASCII value, that is, 34 for double quotes and 39 for single quotes.
3. The character set used in the file as described above.
4. Number of the first line to convert. The first line in the file has the number 1.
5. Cell format of the columns. The content of this token depends on the value of the first token.

- If value separators are used, the form of this token is `column/format[/column/format/...]` where column is the number of the column, with 1 being the leftmost column. The format is explained below.
- If the first token is FIX it has the form `start/format[/start/format/...]`, where start is the number of the first character for this field, with 0 being the leftmost character in a line. The format is explained below.

Format specifies which cell format should be used for a field during import:

Format Code	Meaning
1	Standard
2	Text
3	MM/DD/YY
4	DD/MM/YY
5	YY/MM/DD
6	-
7	-
8	-
9	ignore field (do not import)
10	US-English

The type code 10 indicates that the content of a field is US-English. This is useful if a field contains decimal numbers that are formatted according to the US system (using "." as decimal separator and "," as thousands separator). Using 10 as a format specifier for this field tells OpenOffice.org API to correctly interpret its numerical content, even if the decimal and thousands separator in the current language are different.

Appendix C

Properties for Printing

The following table shows the properties that have to be set for a print job that does not use standard settings (for standard settings just use CalcPrintDocument[], i. e. without any arguments). The first group of properties is used to control the printer, the second is used to control the print job.

Properties for Printer	
Name	string - Specifies the name of the printer queue to be used.
PaperOrientation	PaperOrientation - Specifies the orientation of the paper.
PaperFormat	PaperFormat - Specifies a predefined paper size or if the paper size is a user-defined size.
PaperSize	Size - Specifies the size of the paper in 100th mm.
IsBusy	boolean - Indicates if the printer is busy.
CanSetPaperOrientation	boolean - Indicates if the printer allows changes to PaperOrientation.
CanSetPaperFormat	boolean - Indicates if the printer allows changes to PaperFormat.
CanSetPaperSize	boolean - Indicates if the printer allows changes to PaperSize.

Properties for Print Job	
CopyCount	short - Specifies the number of copies to print.
FileName	string - If set, specifies the name of the file to print to.
Collate	boolean - Advises the printer to collate the pages of the copies. If true, a whole document is printed prior to the next copy, otherwise the page copies are completed together.
Sort	boolean - Advises the printer to sort the pages of the copies.
Pages	string - Specifies the pages to print with the same format as in the print dialog of the GUI, for example, "1, 3, 4-7, 9-".