Mathematica Link for Excel 3.6

User’s Manual

&

Reference Guide

Copyright © 2014 Wolfram Research, Inc.
# Table of Contents

## Overview

---

### Features

---

### Changes

---

## Working in Mathematica

- **Getting Started** .......................................................... 6
- **General Principles** .................................................. 9
- **Automating Excel** ..................................................... 12
- **Creating Excel Functions** ........................................ 15
- **Creating Excel Macros** ............................................. 21

## Working in Excel

- **Getting Started** .......................................................... 25
- **General Principles** .................................................. 32
- **Working with Functions** ........................................... 34
- **Working with Macros** ............................................... 41
- **Link Management** .................................................... 44
- **Sharing Workbooks** .................................................. 45

## Using the Clipboard

- **Loading the Add-In** .................................................. 47
- **Copying Data from Excel** ......................................... 48
- **Pasting Data to Excel** ............................................... 49
- **Fixing Problematic Data** ......................................... 50

## Excel Reference

- **Excel Worksheet Functions** ....................................... 52
- **Toolbar Commands** ................................................ 55
- **Context Commands** ................................................ 59
- **Keyboard Shortcuts** ................................................ 62
- **Data Types** ............................................................. 64
- **Number Formats** ...................................................... 68

## ExcelLink Reference

- **ExcelLink Function Index** ........................................ 69
Introduction

Overview

Mathematica Link for Excel consists of two main components:

The ExcelLink package.

The MathematicaLink add-in.

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

To use the link from Mathematica, you load the ExcelLink package.

To use the link from Microsoft Excel, you load the MathematicaLink add-in.

Features

The ExcelLink package provides:

A set of Mathematica functions that allow you to:

- Read and write data to Excel ranges.
- Display graphics, typeset equations, and formatted output in Excel.
- Create, open, modify, and close Excel files.
- Develop Excel worksheet functions in Mathematica.
- Develop Excel macros in Mathematica.

The MathematicaLink add-in provides:

A set of worksheet functions that allow you to use Mathematica functions in Excel formulas.

A Mathematica Function Wizard to help you learn about and enter Mathematica functions.

A Mathematica macros window that allows you to turn Mathematica code into Excel macros.

A special Clipboard window that allows you to easily copy and paste data between programs.

A searchable PDF-based manual.
Changes

Version 3.5

New in Version 3.5

Compatibility with Mathematica 8.0.
Compatibility with Excel 2010 (including Excel 2010 64-bit version).

ExcelShare function allows sharing a kernel between Excel and Mathematica.

VBA support routines: MathematicaSet, MathematicaRun, and MathematicaGet.

Support for reading and writing Excel comments (including writing to a range using CellLabel ["my comment"]).

Shift +-click the Mathematica Evaluate button to close link and bring up the Mathematica connection window.

Shift +right-click a range to quickly display the Mathematica Context menu without turning on Mathematica Contexts.

Improved in Version 3.5

Updated toolbar and menu icons.

Improved Mathematica connection management (self-healing link, new connection options window).

Improved handling of Mathematica connection exceptions and evaluation interrupts.

Improved Mathematica messaging (messages now returned in real time).

Improved "Display Message Box" option now only applies to Print [] output.

Improved workbook initialization code evaluation and management.

Improved common multi-workbook initialization now supported using an init.m file in same directory.

Improved support for long-running Mathematica macros.

Improved automatic workbook relinking and addin startup logic.

Other minor improvements and fixes.

Version 3.2

New in Version 3.2

Compatibility with Mathematica 7.0 including Mathematica 7.0-based icons.


ExcelWrite support for writing Grid [{[1, 2, 3], [4, 5], [6]}, opts] f.ex. ANOVATable output.


Support for 64-bit Windows and 64-bit Mathematica connecting to 32-bit Excel.
Improved in Version 3.2

Improved message print format in Mathematica 6 and 7.

Fixed a bug where some workbooks with macro buttons did not relink correctly.


Improved Mathematica Macros dialog method of inserting of code boxes and buttons.

No "kernel connection closed" dialog if initialization code ends with Quit [].

Other minor improvements and fixes.

Version 3.1

New in Version 3.1

Compatibility with Mathematica 6.0.


Keyboard shortcuts for Excel toolbar commands.

Additional Excel message-related options.


Improved in Version 3.1

Improved editing of existing functions using the Mathematica Function Wizard.

Improved compatibility with workbooks originally built with version 2.x of the link.

Improved printable PDF documentation.

Version 3.0

New in Version 3

Display of typesetting and formatted output in Excel.

Creating Mathematica-based macros.

A suite of Mathematica functions to interact with and automate Excel.

Improved in Version 3

Start/End Link button is now an Evaluate button. This button can be used to interrupt current evaluations and re-evaluate the workbook once changes have been made. To end a link, hold down the Shift key and click the Evaluate button.
You can now use the Function Wizard to edit existing formulas, select and assign options more easily, and browse for built-in functions by category.

Standard packages are automatically declared by default and can be browsed directly within the Function Wizard. This functionality replaces the Libraries Dialog.

Data Copy and Paste commands are now available from a Mathematica Context Menu. You can now see the data you are copying and pasting using the Mathematica Clipboard window.

Kernel Dialog has now been incorporated into the Mathematica Clipboard window. You can type Mathematica expressions directly into the Clipboard window, evaluate them, and paste the results somewhere if you choose.

Messages are now displayed and stored to a nonmodal window. Using the window you can scroll through multiple messages, find the source of a message, and even save the messages to a log file.

Formula Activate/Deactivate/Recalculate commands are available from the Mathematica Context Menu as a Comments toggle and Recalculate command.

Increased worksheet function speed. Worksheet functions now calculate up to eight times faster.

Increased worksheet function reliability. Worksheet functions are now robust enough for the most demanding spreadsheet applications. Automated tests have performed billions of continuous evaluations without errors.

Improved worksheet function error handling. Dependent evaluations are now suppressed through the use of native Excel error codes.

To share workbooks with others, you can unlink the workbooks’ formulas and macro buttons using the Unlink button in the Mathematica Options ▶ Workbook tab. If colleagues have the link, they will be automatically prompted to relink formulas when they open the workbook.

Strings and floating-point numbers can now be specified using the DATA worksheet function.

Obsolete in Version 3

The MATH worksheet functions provided in Version 2 have been superseded by a more flexible set of functions in Version 3. The MATH worksheet functions still work in Version 3. However, there may be slight differences in how results are returned.

If you would like to convert your existing formulas, here are examples of how they can be mapped.

<table>
<thead>
<tr>
<th>Version 2</th>
<th>Version 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>=MATHVOLATILE(&quot;Random[]&quot;)</td>
<td>=CALC/EVAL(&quot;Random[]&quot;)</td>
</tr>
<tr>
<td>=MATHDEFINE(&quot;m&quot;, A1:C3)</td>
<td>=EVAL(&quot;Set&quot;, &quot;m&quot;, A1:C3)</td>
</tr>
<tr>
<td>=MATHCODE(B1:B10)</td>
<td>initialization code lives in a code box, no formula required</td>
</tr>
<tr>
<td>MATHFORMULA &amp; CELL functions</td>
<td>experimental functions no longer supported</td>
</tr>
</tbody>
</table>

MATH function mappings.
The ReadExcel function has been superseded by a new suite of Excel functions. Here is an example of how an improved ReadExcel function could be defined using the new Excel functions.

```math
ReadExcel[file_, sheet_, rng_] := Module[{book, data},
If[Not[ExcelCheck[ExcelBook[file]]], book = ExcelOpen[file];
data = ExcelRead[ExcelRange[file, sheet, rng]];]
If[ValueQ[book], ExcelClose[book];
data]
```

This ReadExcel function checks to see if the required file is already open in Excel. If not it opens the file, reads the required data, and closes the book if it was not already open.
Working in *Mathematica*: Getting Started

Loading the Package

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

```
in[1]:= << ExcelLink`
```

The ExcelLink package provides a library of functions and symbols relating to Excel.

```
in[2]:= ?ExcelLink`
```

You can learn more about these functions and symbols by looking up its entry in the *Mathematica* Reference section of this information. You can also access this information within *Mathematica*'s help system by looking under Add-ons.

Assigning and Retrieving Data

```
in[1]:= Needs["ExcelLink"]
```

The top-level Excel function provides an easy way to specify a location in Excel, as if it were a variable, then assigns or retrieves data from it.

```
in[2]:= Excel["A1"] = "hello"
in[3]:= Excel["A1"]
```

```
Out[3]= hello
```

You can also clear data from the Excel location.

```
in[4]:= Excel["A1"] =.
```
When assigning data to Excel ranges, one-dimensional data can be assigned either to single rows or single columns of cells.

In[5]:= Excel["A1:C1"] = {1, 2, 3}
In[6]:= Excel["A1:A3"] = {1, 2, 3}

Two-dimensional data can be assigned to a rectangular range of cells.

In[7]:= Excel["A1:C3"] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}
In[8]:= Excel["A1:C3"] = .

Displaying Graphics

In[1]:= Needs["ExcelLink"]

This displays a graphic in Mathematica.

In[2]:= g = Plot[Sin[alpha^2], {alpha, -4, 4}, PlotStyle -> Hue[.8], Frame -> True]

This displays the same graphic in Excel.

In[3]:= Excel["B3"] = g

As specified, the graphic is displayed at cell B3. This cell will serve as an anchor point for the graphic; however, you can move it anywhere you like. You can also resize the graphic as needed.

Subsequent assignments update and redraw the existing graphic.

In[4]:= g = Plot[Sin[alpha^2], {alpha, -3, 3}, PlotStyle -> Hue[.8], Frame -> True]

This clears the graphic associated with cell B3.

In[6]:= Excel["B3"] = .
Displaying Expressions

\[ \text{In[1]}: \quad \text{Needs["ExcelLink"]} \]

This defines an expression to be displayed.

\[ \text{In[2]}: \quad \text{expr} = \text{Sin[i Pi]}^2 / 2; \]

Here is a typeset form of the expression.

\[ \text{In[5]}: \quad \text{TraditionalForm[expr]} \]

\[ \text{Out[5]} = \text{\frac{1}{2} - \frac{\sin^2(i \pi)}{2}} \]

This displays the typeset form in Excel.

\[ \text{In[6]}: \quad \text{Excel["B3"] = TraditionalForm[expr]} \]

This clears the displayed expression associated with cell B3.

\[ \text{In[7]}: \quad \text{Excel["B3"] = \_}. \]
Working in *Mathematica*: General Principles

Function Overview

Functions in the *ExcelLink* package follow the general convention:

```
ExcelMethod[ExcelObject[...], ...]
```

Four kinds of Excel objects are supported. Here is a list of the objects with some methods that apply to them.

**Book**: New, Open, Refresh, Save, Close.

**Sheet**: Insert, Rename, Delete, Activate.

**Range**: Read, Write, Clear, Resize, Offset, Filter, Select.

**Shape**: Insert, Rename, Delete, Read, Write, Select.

For detailed information on all the objects and methods provided by the *ExcelLink* package, see the *Mathematica* Reference section.

Shorthand Notation

Objects

Most objects in Excel can be referenced directly by a unique identifier. The identifier is typically the name of an object or, in the case of ranges, the address. If an identifier is unique among all object types, you do not need to specify what kind of object it is.

Here a range object is provided as a typed object.

```
ExcelRead[ExcelRange["A1:B10"]]
```

Here the range object is specified only by its address; the identifier implicitly identifies it as a range.

```
ExcelRead["A1:B10"]
```

You can use this type of shorthand referencing in any function that requires an Excel object.

Methods

Read and write operations are so common, a shorthand has also been provided for them, and for clearing a range.

```
In[1]:= Needs["ExcelLink`"]

In[2]:= Excel["A1:B10"] = Table[Random[], {10}, {2}]

In[3]:= Excel["A1:B10"]

Out[3]= {{0.395543, 0.335494}, {0.934515, 0.304868}, {0.495912, 0.426754}, {0.657647, 0.398919}, {0.800834, 0.159929}, {0.216486, 0.109539}, {0.694347, 0.157112}, {0.0578395, 0.273708}, {0.750935, 0.710237}, {0.760348, 0.420566}}

In[4]:= Excel["A1:B10"] =.
```
The above three lines of shorthand code are equivalent to the following.

In[5]:= ExcelWrite[ExcelRange["A1:B10"], Table[Random[], {10}, {2}]]

Out[5]= {{0.779186, 0.105253}, {0.954368, 0.320287}, {0.383643, 0.769759},
{0.0198529, 0.0154194}, {0.887731, 0.343005}, {0.362206, 0.8165},
{0.0868974, 0.183076}, {0.14572, 0.506961}, {0.39255, 0.025964}, {0.0878807, 0.233253}}

In[6]:= ExcelRead[ExcelRange["A1:B10"]]

Out[6]= {"Book1", "Sheet1"}


Notes

- If more than one object has the same identifier (e.g. a shape has the same name as a sheet) the identifier most likely to be used by the calling method is returned.

Object Notation

Using full object notation can be useful when referring to an object by index, or providing context for the object.

Here are a few examples.

ExcelSheet[1]
ExcelSheet["Book1", "Sheet1"]
ExcelSheet["Book1", 1]
ExcelRange["Report.xls", 1, "A1:D100"]

When no context is provided, the active context is assumed.

Excel object references are resolved when they are passed to a method, not before. Until then, they are just Mathematica expressions representing a location in Excel.

In[1]:= Needs["ExcelLink"]

In[2]:= ExcelSheet["My Sheet"]

Out[2]= Sheet: My Sheet

In[3]:= ExcelRead[ExcelSheet["My Sheet"]]

ExcelRead::source: -Sheet: My Sheet- is not a valid range, shape, or sheet.


If you want to see if an object reference is valid, you can do so by using the ExcelCheck function.

In[4]:= ExcelCheck[ExcelSheet["My Sheet"]]


You can return collections of objects as a list by using the plural of an object name.

In[5]:= ExcelSheets[]


You can extract the name of a returned object.

In[6]:= ExcelName[First[ExcelSheets[]]]

Out[6]= Sheet1
You can also return the embedded context information for the objects.

\[\text{In[7]:=} \text{ExcelContext[First[ExcelSheets[]]]}\]
\[\text{Out[7]=} \{\text{Book1}\}\]

Expression Cells

If the Number Format of an Excel cell is set to Text, the contents of the cell are considered to be a Mathematica expression when transferring them to Mathematica via the clipboard or in a macro.

For more information, see Strings in the Excel Reference guide.

Notes

- Cells should be formatted as Text before entering an expression. To convert existing contents to Text, you can reenter them manually or use the provided Expression command from the Mathematica Context menu.
- Expressions such as 1/2 or \(-x\) can only be entered in cells formatted as Text. Otherwise, Excel will attempt to interpret them as something else.
- When working with expression cells, all cells in the range should be formatted as Text. Partial expression ranges are not currently supported.
- From Mathematica, you can use the Expression function to apply or unapply Text format to a range.

Data Cells

If the Number Format of an Excel cell is anything other than Text, the cell is considered a data cell.

When transferring the contents of data cells from Excel to Mathematica:

Data is transferred as it is natively stored in Excel. This means, for example, all numbers will be returned as floating-point doubles. This includes dates that are numbers with special formatting properties.

When transferring Mathematica expressions to Excel data cells:

Nonnative expressions are converted to an equivalent Excel data type whenever possible. Expressions that do not have any possible Excel-equivalent are converted to InputForm strings.

See the Data Types Overview for more details.

Notes

- You can use the ExcelDate function and ExcelForm functions to work with date values once you get them into Mathematica.
- Cell references in Excel-based formulas such as \(\text{EVAL(A1,A2,A3)}\) are an exception to this rule. In this case, you must wrap formula arguments with the provided DATA function to treat them as data cells; \(\text{EVAL("StringJoin", DATA(A2), DATA(A3))}\), for example.
Automating Excel

Opening Excel

By default, the ExcelLink package automatically connects, as needed, to an open instance of Excel. This provides easy, on-demand connectivity. In this mode, you can open or close Excel on your own whenever you wish.

If you write Mathematica code that automates Excel to perform a task, you may want the Mathematica code to initiate opening an instance of Excel. The ExcelInstall function provides a way of doing this.

```
In[1]:= Needs["ExcelLink"]
```

This opens a visible instance of Excel, if one is not already open.

```
In[2]:= ExcelInstall[Visible -> True]
```

Once an Excel automation routine has been developed, you can set Visible -> False instead. This will open a hidden, private instance of Excel to perform the requested tasks.

Importing Workbooks

```
In[1]:= Needs["ExcelLink"]
```

This specifies a file to import data from.

```
In[2]:= f = ToFileName[{ExcelDirectory["Link"], "Examples"}, "Stocks.xls"]
```

This opens the file.

```
In[3]:= ExcelOpen[f]
```

This returns the sheets in the file.

```
In[4]:= ExcelSheets[]
```

This reads in all the data from one of the sheets.

```
In[5]:= data = ExcelRead[ExcelSheet["IBM"]];
```

This previews the first five rows of data from the sheet.

```
In[6]:= Take[data, 5]
```

```
This defines the data in the last column.

```
In[7]:= adjclose = Rest[Part[data, All, -1]]
```

This plots the data.

```
In[8]:= ListPlot[adjclose, Joined -> True, Frame -> True, Axes -> False]
```

This closes the workbook once you are finished importing data from it.

```
In[9]:= ExcelClose[]
```

**Exporting Workbooks**

```
In[1]:= Needs["ExcelLink""]
```

You can provide any format to an exported workbook by referencing an existing template file.

This specifies an existing template file to use for exporting the file.

```
In[2]:= f = ToFileName[{ExcelDirectory["Link"], "Templates"}, "Report.xls"]
```

```
Out[2]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Templates\Report.xls
```

This creates the new workbook based on the template file.

```
In[3]:= ExcelNew[f]
```

```
Out[3]= Book: Report1-
```

The template contains a named range that defines where to put data in the report.

```
In[4]:= ExcelRanges[]
```

```
```

It also contains a named shape to display a graphic.

```
In[5]:= ExcelShapes[]
```

```
```

This simulates rolling two six-sided dice 500 times.

```
In[6]:= roll = Random[Integer, {1, 6}]
trials = Table[roll + roll, {500}];
```

This writes the trial data to the report.

```
In[8]:= ExcelWrite["Trials_Range", trials]"
This displays a histogram to the named shape.

```
In[9]:= If[$VersionNumber < 7, If[$VersionNumber ≥ 6, Needs["Histograms"], Needs["Graphics"]]
        ExcelWrite["Histogram Graphic", Histogram[trials]]
```

From In[9]:=

This defines the file name for the report workbook.

```
In[11]:= f = ToFileName[{ExcelDirectory["Home"], "Autogenerated Report.xls"]
```

This saves the report workbook to disk.

```
In[12]:= ExcelSave[Active, f]
```

This closes the workbook once you have finished exporting data to it.

```
In[13]:= ExcelClose[]
```

This cleans up by deleting the exported workbook file.

```
In[14]:= DeleteFile[f]
```

**Closing Excel**

```
In[1]:= Needs["ExcelLink"]
```

Once your automation routines are completed, you can use the `ExcelUninstall` function to close Excel.

```
In[2]:= ExcelUninstall[]
```

By default, `ExcelUninstall` only closes visible instances of Excel if no workbooks remain open. This avoids accidental data loss. You can force a visible instance of Excel to close, even if workbooks are open, by specifying `Visible -> True` as an option to `ExcelUninstall`. 
Creating Excel Functions

Defining Functions

Here is how to define a *Mathematica* function that adds two numbers.

\[
\text{In}[1]:= \text{addtwo}[x\_, y\_] := x + y
\]

Arguments for the function are specified using `pattern _` indicators. The delayed assignment `:=` operator indicates that the body of the function is evaluated only once the values for the arguments are known.

\[
\text{In}[2]:= \text{addtwo}[2, 2]
\]
\[
\text{Out}[2]= 4
\]

The next example uses a function defined in a standard package, which you must load first.

\[
\text{In}[3]:= \text{If}[\$VersionNumber \geq 6, \text{Needs}["ComputationalGeometry`"], \text{Needs}["DiscreteMath`ComputationalGeometry`"]]
\]

This is an example of a function that generates a graphic.

\[
\text{In}[4]:= \text{triplot}[n\_] := \text{PlanarGraphPlot}[\text{Table}[\text{Random}[], \{n\}, \{2\}]]
\]

The function triangulates a set of \( n \) random coordinates.

\[
\text{In}[5]:= \text{triplot}[10]
\]

From `In[5]:=`

\[
\begin{center}
\includegraphics[width=0.5\textwidth]{triplot_10.png}
\end{center}
\]

Here is a version of the same function, written in two steps.

\[
\text{In}[6]:= \text{triplot}[n\_] := \\
\quad \text{Module}[\{\text{data}\}, \\
\quad \quad \text{data} = \text{Table}[\text{Random}[], \{n\}, \{2\}]; \\
\quad \quad \text{PlanarGraphPlot}[\text{data}]
\]

\[
\text{In}[7]:= \text{triplot}[10]
\]

From `In[7]:=`

\[
\begin{center}
\includegraphics[width=0.5\textwidth]{triplot_10_steps.png}
\end{center}
\]
In this definition, a local variable is defined within the body of the function. The two steps within the body of the function are separated using a semicolon. The final line of code returns the value of the function.

```
In[7]:= triplot[10]
```

```
Out[7]=
```

**Special Considerations**

**Usage**

To help others know how to use your function, you can define a usage message.

```
In[8]:= triplot::"usage" = "triplot[n] plots a random triangulation of n planar points."
```

```
```

The usage message is used by the *Mathematica* Function Wizard to automatically generate argument templates for the function. To be fully compatible with the Function Wizard, you should always use the following convention for your usage messages.

```
In[9]:= f::"usage" = "f[x] does one thing. f[list] does another. f[list, x] does more."
```

```
```

**Options**

You can also define a set of default options for your functions, if needed.

```
In[10]:= Options[triplot] = {Frame -> False, GridLines -> None}
```

```
Out[10]= {Frame -> False, GridLines -> None}
```

This clears the previous definition for the function, then defines it with options.

```
In[11]:= Clear[triplot]
```

```
In[12]:= triplot[n_, opts___Rule] :=
   Module[{data, g, rules},
     data = Table[Random[], {n}, (2)];
     rules = Sequence @@ Join[{opts}, Options[triplot]];
     PlanarGraphPlot[data, rules]
   ]
```
Errors

When you are developing a function that will be used in Excel, you should consider returning the symbol $Failed if something goes wrong in your function. You can do this using the check function.

In[14]:= triplot[n_, opts___Rule] :=
Check[Module[{data, g, rules},
data = Table[Random[], {n}, {2}];
rules = Sequence @@ Join[{opts}, Options[triplot]];_
PlanarGraphPlot[data, rules], $Failed]


The symbol $Failed is converted to a #VALUE! error in Excel that will suppress further dependent calculations.

In[17]:=  triplot["hello"]

Table::iterb : Iterator {hello} does not have appropriate bounds. More...

Out[17]= $Failed

To be complete, you should also create a catch-all function definition that will handle the case where users provide arguments that do not match the pattern you specified. By default, the function returns unevaluated.

In[16]:= triplot[1, 2, 3]

Out[16]= triplot[1, 2, 3]

This traps the error.

In[18]:= triplot[___] := $Failed

In[19]:= triplot[1, 2, 3]

Out[19]= $Failed

You can also create your own error messages to inform the user about what went wrong.

In[20]:= triplot[___] :=
Module[{},
  Message[triplot::"args"];
  $Failed
]

Out[20]= Arguments are incorrect
Here to the

Once Code Out[21]=

Paste

Copy

Create

an

deploy

In[7]=

In[6]=

In[5]=

In[4]=

In[3]=

In[2]=

In[1]=

In[21]:= triplot[1, 2, 3]

   triplot::args : Arguments are incorrect

Out[21]= $Failed

Code Box Deployment

Once you have developed a set of Mathematica functions you would like to use in Excel, you can collect cells that define the functions in one place to make it easier to transfer the code to Excel.

In[1]:= If[$VersionNumber >= 6, Needs["ComputationalGeometry"], Needs["DiscreteMath`ComputationalGeometry"]]

In[2]:= Clear[triplot]

In[3]:= triplot::usage = "triplot[n] plots a random triangulation of n points."

In[4]:= Options[triplot] = {Frame -> False, GridLines -> None};

In[5]:= triplot[n_, opts___Rule] :=
   Check[Module[{data, g, rules},
      data = Table[Random[], (n), {2}];
      rules = Sequence @@ Join[{opts}, Options[triplot]];
      PlanarGraphPlot[data, rules]
   ], $Failed]

In[6]:= triplot[___] :=
   Module[{},
      Message[triplot::"args"];
      $Failed
   ]

In[7]:= triplot::"args" = "Arguments are incorrect";

To deploy this code as an Excel function, you will need to copy the contents of the notebook cells that define the function to an initialization code box in an Excel workbook.

Here is how to do this.

1. Create an initialization code box in Excel:
   - Click Macros on the Mathematica Toolbar.
   - Click New... and name the macro Initialization. This is the default if no other macros exist in your workbook.
   - Select a location for the code box and click OK.

2. Copy the code from Mathematica:
   - Use Kernel ➤ Show In/Out Names to temporarily hide input labels.
   - Select the Input cells to copy. To select noncontiguous cells, hold down the Ctrl key.
   - Press Ctrl + C or choose Edit ➤ Copy.

3. Paste the Mathematica code into the Excel code box:
   - Click and drag inside the code box to select all existing contents.
   - Press Delete to delete the previous contents.
   - Press Ctrl + V or choose Edit ➤ Paste.
You can now use the Mathematica function you created inside Excel.

<table>
<thead>
<tr>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;triplot&quot;,B3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>ExcelGraphic[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes

Using the code box approach, you can create workbooks that have no dependencies on other files.

**Package Deployment**

Mathematica notebooks can automatically generate an associated package file. This provides an easy way for you to export a set of Mathematica function definitions you would like to use in an Excel workbook.

With this in mind, on the Excel side, the MathematicaLink add-in checks for a package file with the same name in the same directory when initializing a workbook. If one is found, the code in the file is considered the initialization code for the workbook.

Here is how to create a package file from a notebook, then use the contents of the package file as initialization code in a workbook:

- Create .nb and .xls files with the same name in the same directory.
- Select the cells that contain code that will be used in the workbook.
- Click Cell ▶ Cell Properties ▶ Initialization Cell to specify the selected cells as initialization cells.
- Save the notebook. When you do this, you will be prompted to create a package file with the contents of the initialization cells.
- Click Create Auto Save Package.
You should now have .nb, .m, and .xls files in the same directory with the same name. In the future, every time you save changes to the notebook, the package file is automatically updated. In turn, the next time you evaluate in Excel, the new set of function definitions will be automatically loaded and used.

Notes

- Using the package approach, you can easily develop and update function definitions for a workbook. However, you must remember to send the package file along with the workbook to enable others to interact with the workbook.

- During development, be sure to save changes to your Mathematica notebook in order to update the package file before using it from the Excel side.
Creating Excel Macros

Developing Macros

Setting Up a New Notebook

When developing Mathematica code, it is best to separate input and output definitions from the main analysis portion of the routine. This way your analysis code can be easily adapted to obtain inputs and send outputs anywhere.

Here is a sequence of Mathematica commands that performs some analysis.

This section defines inputs.

\[
\text{In}[1]:= \text{m} = \{\{1., 2.\}, \{3., 4.\}\};
\]

This section performs your analysis.

\[
\text{In}[2]:= \text{m} = \text{Inverse}[^\text{m}];
\]

This section displays outputs.

\[
\text{In}[3]:= \text{m}
\]
\[
\text{Out}[3]= \{\{-2., 1.\}, \{1.5, -0.5\}\}
\]

To use this code as an Excel macro, you only need to load the ExcelLink package to modify the input and output sections. Before doing this, open Excel and type in the same inputs into the workbook locations indicated in the following.

This section loads required packages.

\[
\text{In}[4]:= \text{Needs}[^\text{"ExcelLink"}]
\]

This section defines inputs from Excel.

\[
\text{In}[5]:= \text{m} = \text{Excel}[^\text{"B3:C4"}];
\]

This section performs your analysis.

\[
\text{In}[6]:= \text{m} = \text{Inverse}[^\text{m}];
\]

This section returns outputs to Excel.

\[
\text{In}[7]:= \text{Excel}[^\text{"B3:C4"}] = ^\text{m}
\]

In this example, the input and output range is the same. This is a way of performing in-place evaluation.

Modifying an Existing Notebook

To convert an existing Mathematica notebook to be used as an Excel macro:

- Locate the cells in your notebook that define inputs to your analysis.
- Modify those cells to use values contained in Excel.
- Likewise, locate the cells in the notebook that display outputs of your analysis.
- Modify those cells to return results to Excel.

For more information, see Code Box Deployment.
Special Considerations

Status

If your analysis takes a while to complete, you may want to provide some feedback to the user on how the analysis is proceeding. You can do this by using the ExcelStatus function.

```mathematica
In[1]:= Needs["ExcelLink`"]
In[2]:= ExcelStatus["Processing data..."]
Pause[3];
ExcelStatus["Analyzing data..."];
Pause[5];
ExcelStatus["Generating report..."];
Pause[1];
ExcelStatus[];
```

This writes status information to the status bar at the bottom left-hand side of the Excel window. In the final line ExcelStatus is called without arguments in order to return the status bar to its default state.

Notes

- Writing status messages makes the analysis section of your notebook Excel specific. However, this may be required for longer routines.
- Writing status messages can also be a good way to see which part of your analysis is taking up the most time.

Dialogs

If you would like to ask the user to select a range or specify a file name during a macro, you can do so using the ExcelDialog function. The symbol $ExcelDialogs gives a list of available dialogs.

```mathematica
In[9]:= $ExcelDialogs
Out[9]= {Range, Open, Save, Files, Folder}
```

This displays the "Range" dialog.

```mathematica
In[10]:= ExcelDialog["Range"]
Out[10]= -Range: B3:C4-
```

Notes

- When running code from Mathematica, you need to activate Excel first to interact with an Excel dialog.

Notebook Deployment

When developing Excel macros, you do not need to transfer code to a workbook. The Mathematica code can remain stored in a notebook file. In this case, any time you want to run a macro on a particular Excel workbook, open the notebook that contains the macro and, with the workbook open in Excel, evaluate the code from the notebook.

Code Box Deployment

To create a standalone workbook interface, you can transfer the Mathematica macros you have developed in a notebook to code boxes in the Excel workbook. Once this is done you can create buttons for the macros.

To deploy Mathematica code as an Excel macro you will need to copy the notebook cells that define the macro to a code box in an Excel workbook.
Here are the notebook cells that contain the code you want to use as a macro.

\[\text{In}[1]:= \text{Needs["ExcelLink"]}\]
\[\text{In}[2]:= \text{m = Excel["B3:C4"]};\]
\[\text{In}[3]:= \text{m = Inverse[m]};\]
\[\text{In}[4]:= \text{Excel["B3:C4"] = m}\]

Here is how to transfer the code to Excel.

1. Create a code box for the macro in Excel:
   - Click Macros on the Mathematica Toolbar.
   - Click New... and name the macro whatever you like. Spaces in the macro name are permitted.
   - Select a location for the code box and click OK.

2. Copy the code from Mathematica:
   - Use Kernel ▶ Show In/Out Names to temporarily hide input labels.
   - Select the cells to copy. To select noncontiguous cells, hold down the Control key.
   - Press Ctrl + C or choose Edit ▶ Copy.

3. Paste the Mathematica code into the Excel code box:
   - Click inside the code box you just created.
   - Click and drag to select all existing contents of the code box.
   - Press Delete to delete the previous contents.
   - Press Ctrl + V or choose Edit ▶ Paste.
To create a button for the Mathematica macro:

- Select the name of the macro from the Available Macros list.
- Click **Button...**.
- Select a location for the button and click **OK**.

For more information on using macros you create in Excel, see **Working with Macros**.

**Notes**

- When running macro code from inside Excel, it is not necessary to load the ExcelLink package. However, you can still include the line in your macro.
- Using the code box approach, you can create workbooks that have no dependencies on other files.

**Package Deployment**

Mathematica notebooks can automatically generate an associated package file. This provides an easy way to export a set of Mathematica commands that can be used as a one-click workbook processing macro.

To create a package file, follow the steps outlined for creating a package file outlined in the "Package Deployment" section of **Creating Excel Functions**. The only difference is, in this case, you will save a sequence of macro commands to the package file instead of a set of function definitions.

You should now have .nb, .m, and .xls files in the same directory with the same name. In the future, every time you save changes to the notebook, the package file is automatically updated. In turn, the next time you click **Evaluate** in Excel, the new workbook processing macro will be used.

**Notes**

- If you would like the kernel to close after workbook processing is complete, include `Quit[]` as the last line of your macro.
Working in Excel: Getting Started

Loading the Add-in

After installing the link, you will see a *Mathematica Link for Excel* folder in your *Start* ➤ *All Programs* menu. The add-in in this folder is required if using the link from the Excel side or when copying and pasting data between programs.

If you did not install the add-in when you installed the software, click *Mathematica Link Add-In* to do so now. The add-in will install itself when it is loaded the first time. This may take a moment depending on your anti-virus/ security settings.

If you are prompted to do so, choose to *Enable Macros* in the security warning dialog. If no dialog appears and nothing works, you may need to adjust your Excel macro security settings to permit the macros in the add-in to run. Refer to the Help menu in your version of Excel regarding how to change your macro security settings. Once you have adjusted your security settings, close Excel and try loading the add-in again.

Once the add-in is loaded, the *Mathematica* toolbar will appear.

- In Excel 2007 or later, the *Mathematica* toolbar can be found under the *Add-Ins* ribbon tab.

If you would like a *Mathematica* menu instead, you can click the *Options* button on the toolbar. All commands in the *Mathematica* menu are identical to those on the *Mathematica* toolbar. The *Mathematica* menu can be useful if you would like to use the `Alt` key to access menu-based commands.

Once you have installed the add-in the first time, you no longer need to use the *Start* menu shortcut. Instead, you can use *Tools* ➤ *Add-ins* dialog to load or unload the *Mathematica Link* add-in. Excel checks the settings in this dialog each time it starts and automatically loads any checked add-ins.

- To access the add-ins manager in Excel 2007 or later, click the *Office* button / *File* menu to the upper-left corner, click *Excel Options*, then under *Add-ins*, next to the *Manage: Excel Add-ins* dropdown box, click *Go*....
Entering a Function

Once the Mathematica Link add-in is loaded, you are ready to perform Mathematica calculations inside Excel. One way of doing this is using the EVAL worksheet function. This worksheet function allows you to call any Mathematica function from within an Excel formula.

Try entering the simple Mathematica function `Prime` in an empty worksheet cell. This is done as shown.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;Prime&quot;,100)</td>
<td>541</td>
</tr>
</tbody>
</table>

The value returned is the 100\(^{th}\) prime number.

- International versions of Excel may require semicolon-separated formulas such as =EVAL("Prime",100).
- During your first calculation, a Mathematica kernel will be launched as a computation server for Excel. This new process may appear in your Windows task bar.

Now, try creating an interactive prime number calculator by specifying a cell reference as an argument.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;Prime&quot;,A1)</td>
<td>#N/A</td>
</tr>
</tbody>
</table>

Unless you have already entered a value in cell A1, the function returns unevaluated and displays an error code. The #N/A error code indicates that one or more inputs to the formula are not available.

Type any integer you choose in cell A1, and a prime number will be calculated for you.

Here are some other examples.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;Factorial&quot;,10)</td>
<td>3628800</td>
</tr>
<tr>
<td>=EVAL(&quot;Expand&quot;,&quot;(x+1)^3&quot;)</td>
<td>1 + 3 * x + 3 * x^2 + x^3</td>
</tr>
<tr>
<td>=EVAL(&quot;Plot&quot;,&quot;x^2&quot;,&quot;(x,0,5)&quot;)</td>
<td>ExcelGraphic [1]</td>
</tr>
</tbody>
</table>

Mathematica contains thousands of functions. If you know precisely which Mathematica function you wish to use and the arguments it takes, you can type it directly into your spreadsheet, as shown.

If you are unsure of the name of a Mathematica function or how to use it, or want to explore the functions Mathematica has to offer, click Functions on the Mathematica toolbar. This will launch the Mathematica Function Wizard.
The first step of the wizard is designed to help you find, and learn about, Mathematica functions.

The Mathematica Function Wizard—step 1.

The next step is designed to help you interactively specify arguments to the function.

The Mathematica Function Wizard—step 2.
An optional third step is available to select and specify options for functions such as Plot that have a defined set of options.

The Mathematica Function Wizard—step 3.

Notes

- Unlike Excel, Mathematica uses case-sensitive syntax. Therefore, be sure to capitalize Mathematica function names like Prime.

- If you do not include a * character in your name search, the wizard defaults to a non-case-sensitive search for *name*.

- You cannot interact with a Mathematica front end that is in server mode. If you would like to work in the Mathematica front end, you can launch another standard instance of the front end.

Creating a Macro

To call Mathematica code as an Excel macro, the Mathematica code must be contained in a named code box inside Excel. Once you have created the named code box, you can then create a button that will call the code in the box. The Mathematica Macros manager can help you do this.

To create a macro code box:

- Click Macros on the Mathematica toolbar.

- Click New... and specify a name. In this case, name the macro "Example".

- Select where to place the code box.

- Click OK.
The Example macro is added to the list Available Macros and a code box for the Example macro is inserted at the location you specified.

The initial contents of the code box are specified by a customizable template. In this case, you can leave the default code as is. Before running the code, however, you should type values into the cells referenced in the macro.

To run a macro:

- Select the macro from the available macros list.
- Click Run.

To make it more convenient to run the macro, you can create a button for the macro.

To create a macro button:

- Select the macro from the available macros list.
- Click Button...
- Select where to place the button.
- Click OK.
Creating a button for a Mathematica macro.

Notes

- The default macro name, Initialization, is reserved for code you want to run automatically every time you connect to a kernel or click Evaluate.

- To move a button or a code box, hold down the Ctrl key before selecting it.

- Code boxes can be located anywhere in a workbook. They do not need to be on the same sheet as the button calling the code.

Evaluating an Expression

You can use the Mathematica Clipboard window to evaluate Mathematica expressions the same way you would in a Mathematica notebook. This may be useful, for example, to quickly check the state of a variable or experiment with a function you are using for the first time.

To evaluate an expression in the Mathematica Clipboard window:

- Click Clipboard on the Mathematica toolbar.

- Type the expression into the Mathematica Clipboard window.

- Click Evaluate.
Once the Mathematica kernel has evaluated the expression, the answer replaces the contents of the Clipboard window.

- To restore the original input expression, click Restore.

**Notes**

- To paste evaluation results to a location in Excel, click Paste. You will be prompted to specify where to paste the results.

- To paste evaluation results to another program, click Put. This puts the evaluation results onto the global Clipboard. You can then paste them into another program.

**Unloading the Add-in**

If you are finished using the link, you can uncheck Mathematica Link in the Add-Ins manager and click OK. This unloads the add-in from Excel.

When you unload the link, you will be prompted to delete your personal settings.

Click No to preserve your settings until the next time you use the link. Click Yes to restore default settings the next time you use the link.

**Notes**

- If you do not uncheck Mathematica Link in the Add-Ins manager, the link is automatically loaded the next time you start Excel.
Working in Excel: General Principles

Non-Modal Dialogs

All Mathematica Link dialogs are non-modal. You can leave dialogs open and still work in Excel.

There are two ways of closing a link dialog:

- Click Close or press the Esc key. The dialog remembers its current location and state the next time it is displayed.
- Click the X in the upper right-hand corner. The dialog returns to a default location and state the next time it is displayed.

Notes

- Occasionally, changes you make in Excel may not be reflected in the contents of an open dialog. If this happens, redisplay the dialog to refresh its contents.

The Context Menu

Mathematica Link commands that operate on the current selection are available from a Mathematica context menu. To display this context menu, right-click a range when Mathematica contexts are enabled.

To enable or disable Mathematica contexts, click Contexts on the Mathematica toolbar.

Notes

- All commands on the Mathematica context menu also have a keyboard shortcut. See Keyboard Shortcuts for a listing.
**Expression Cells**

If the Number Format of an Excel cell is set to Text, the contents of the cell are considered to be a *Mathematica* expression when transferring them via the clipboard or in a macro.

For more information, see *Strings* in the Excel Reference guide.

---

**Notes**

- The number format of a range can also be changed using Excel's built-in Format ▶ Cells ▶ Number tab. Cells should be formatted as Text before entering an expression. To convert existing contents to Text, you can reenter them manually or use the provided Expression command from the *Mathematica* Context menu. Expressions such as $1/2$ or $-x$ can only be entered in cells formatted as Text. Otherwise, Excel will attempt to interpret them as something else. When working with expression cells, all cells in the range should be formatted as Text. Partial expression ranges are not currently supported. From *Mathematica*, you can use the `ExcelFormat` function to apply or unapply Text format to a range.

**Data Cells**

If the Number Format of an Excel cell is *anything other than* Text, the cell is considered a data cell.

When transferring the contents of data cells from Excel to *Mathematica*:

- Data is transferred as it is natively stored in Excel. This means, for example, all numbers will be returned as floating point doubles. This includes dates that are numbers with special formatting properties.

When transferring *Mathematica* expression to Excel data cells:

- Non-native expressions are converted to an equivalent Excel data type whenever possible. Expressions that do not have any possible Excel-equivalent are converted to `InputForm` strings.

See the *Data Types overview* for more details.

---

**Notes**

- You can use the `ExcelDate` function and `ExcelForm` functions to work with date values once you get them into *Mathematica*. Cell references in Excel-based formulas such as `=EVAL(A1,A2,A3)` are an exception to this rule. In this case, you must wrap formula arguments with the provided `DATA` function to treat them as data cells; `=EVAL("StringJoin", DATA(A2), DATA(A3))`, for example.
Working with Functions

The Link Functions

There are five worksheet functions provided by the Mathematica Link add-in.

<table>
<thead>
<tr>
<th>Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL</td>
<td>perform Mathematica evaluations</td>
</tr>
<tr>
<td>EXPR</td>
<td>build Mathematica expressions</td>
</tr>
<tr>
<td>DATA</td>
<td>specify an argument as native Excel data</td>
</tr>
<tr>
<td>RULE</td>
<td>build a Mathematica rule</td>
</tr>
<tr>
<td>CALC</td>
<td>force a function to respond to the Excel calculate command (F9)</td>
</tr>
</tbody>
</table>

The Link functions.

Together, these functions can be used to build up expressions and perform evaluations in Mathematica in very flexible ways. For more detailed information on each worksheet function, see Excel Worksheet Functions.

Using the EVAL worksheet function, you can call any function defined in Mathematica. This immediately extends the number of functions available inside Excel from a few hundred to several thousand.

<table>
<thead>
<tr>
<th>Excel syntax</th>
<th>Mathematica syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;f&quot;,&quot;arg1&quot;,&quot;arg2&quot;,...)</td>
<td>f[arg1, arg2,...]</td>
</tr>
</tbody>
</table>

The EVAL function.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;Simplify&quot;,&quot;x^2+2x+1&quot;)</td>
<td>(1 + x)^2</td>
</tr>
<tr>
<td>=EVAL(&quot;Random&quot;,&quot;Integer&quot;,&quot;{1,6}&quot;))</td>
<td>an integer between 1 and 6</td>
</tr>
</tbody>
</table>

EVAL examples.

As shown, Mathematica syntax must be wrapped in quotes when directly typed into an Excel formula. If this is not done, Excel's formula parser will try to interpret these as Excel syntax. To avoid having to do this, you can create references to arguments in Excel cells. This is discussed in more detail in Specifying Arguments.

In its single-argument form, the EVAL function can also be used to evaluate Mathematica expressions such as symbols or operator forms of expressions.

<table>
<thead>
<tr>
<th>Excel syntax</th>
<th>Mathematica syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;expression&quot;)</td>
<td>expression</td>
</tr>
</tbody>
</table>

The single-argument form of EVAL.
<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=EVAL(&quot;$Version&quot;)</code></td>
<td>the version of the kernel you are running</td>
</tr>
<tr>
<td><code>=EVAL(&quot;7! + 5!&quot;)</code></td>
<td>5160</td>
</tr>
</tbody>
</table>

`EVAL` single-argument examples.

**Specifying Function Heads**

When specifying a *Mathematica* function, the function does not necessarily have to be a named *Mathematica* function. The function can also be specified as a nameless pure function. Using pure functions, you can use multiple functions to create a new function on the fly.

The syntax of *Mathematica* pure functions is as follows:

Slots for individual arguments are specified as `#1, #2, ...`.

All arguments can collectively be inserted at one point using `##`.

Here, a pure function with two arguments is created in a step-by-step way.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=EVAL(&quot;Sum[1/x^3,{x,10}]&quot;)</code></td>
<td>this sums the first 10 terms in the series</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Sum[1/x^3,{x,#1}],[&quot;10&quot;]</code>)`</td>
<td>number of terms is now specified as a pure function argument</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Sum[1/x^#2,({x,#1}],[&quot;10&quot;,&quot;3&quot;]</code>)`</td>
<td>exponent of <code>x</code> is now specified as a second pure function argument</td>
</tr>
</tbody>
</table>

Creating a pure function.

Note that, as shown, arguments do not have to appear in sequential order inside the function. The index specifies which argument goes where.

Once values that may be edited have been specified as arguments, a reference to the cells containing the values can be made. This is discussed in the next section.

**Notes**

- If you are familiar with pure functions, you will notice the pure function indicator (`) is not used here. There is no need because, in this context, it is clear that if `#` signs are present, a pure function is being specified.
  
  For more information on creating and using pure functions, refer to the tutorial Pure Functions.
Specifying Arguments

While arguments can be typed directly into a formula, it is generally more convenient to specify arguments as the contents of a cell or a range of cells. Editing values contained in cells is much easier than editing values embedded in a formula. And, if entered in a cell, Mathematica syntax does not need to be wrapped in quotes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=EVAL(&quot;Random[Integer,\{1,6\}]&quot;)</code></td>
<td>function and its arguments entered as a single expression</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Random&quot;,&quot;Integer&quot;,\{1,6\})</code></td>
<td>function arguments passed individually</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Random&quot;,C2,\{1,6\})</code></td>
<td>symbol Integer now provided by cell C2</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Random&quot;,C2,D2:E2)</code></td>
<td>list {1, 6} now provided by range D2:E2</td>
</tr>
</tbody>
</table>

Specifying range arguments.

Using the last form of the example, you can easily change the upper and lower bound of the random number by changing cells D2 and E2. Also, by typing Real in cell C2 you can change the type of random number returned.

A single column or single row of cells is interpreted as a one-dimensional Mathematica list; if a range has multiple rows and multiple columns, it is returned as a 2D list of lists.

Specifying String Arguments

To specify a string in Mathematica, you can wrap a text argument with the DATA function. The following methods return a list of Mathematica functions that end in Solve.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=EVAL(&quot;Print&quot;,DATA(&quot;Hello&quot;))</code></td>
<td>evaluates Print [Hello]</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Print&quot;,DATA(A1))</code></td>
<td>string now provided in cell A1</td>
</tr>
</tbody>
</table>

Specifying string arguments.

Specifying Numeric Data

Mathematica evaluations are performed at the precision of the inputs provided. If you would like the kernel to perform evaluations numerically at floating-point precision, wrap your inputs with the DATA function.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=EVAL(&quot;Eigenvalues&quot;,A1:C3)</code></td>
<td>performs evaluation using symbolic or numeric methods depending on the inputs provided in A1:C3</td>
</tr>
<tr>
<td><code>=EVAL(&quot;Eigenvalues&quot;,DATA(A1:C3))</code></td>
<td>always performs evaluation using numeric methods</td>
</tr>
</tbody>
</table>

Specifying numeric arguments.
Specifying Subexpressions

Using the \texttt{EXPR} function you can build up multi-function expressions for a single evaluation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;f&quot;,EXPR(&quot;g&quot;,1),EXPR(&quot;g&quot;,2))</td>
<td>evaluates $f[g[1],g[2]]$</td>
</tr>
</tbody>
</table>

Specifying subexpressions.

Specifying Options

\textit{Mathematica} functions may have defined options associated with them. To specify an optional argument you can use the \texttt{RULE} worksheet function.

<table>
<thead>
<tr>
<th>Method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;ListPlot&quot;,A1:B100,RULE(&quot;PlotJoined&quot;,D2))</td>
<td>specifies an option using the value in cell D2</td>
</tr>
</tbody>
</table>

Specifying an option.

Generating Graphics

The \texttt{EVAL} worksheet functions can also be used to return \textit{Mathematica} graphics.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;ListPlot&quot;,&quot;Table[Random[],{50}]&quot;)</td>
<td>ExcelGraphic [1]</td>
</tr>
<tr>
<td>=EVAL(&quot;Graphics&quot;,&quot;Polygon[{{0,0},{9,3},{3,5}}]&quot;&quot;)</td>
<td>ExcelGraphic [2]</td>
</tr>
<tr>
<td>=EVAL(&quot;Plot3D&quot;,&quot;Sin[x+Cos[x*y] y]&quot;,&quot;{x,0,4},&quot;{y,0,4}&quot;&quot;&quot;)</td>
<td>ExcelGraphic [3]</td>
</tr>
</tbody>
</table>

\texttt{EVAL} graphics examples.

\textit{Mathematica} graphics are displayed as picture objects in Excel. By default, these pictures are rendered as Windows metafiles. This format scales reasonably well since it is a vector format. Font sizes, however, may become a bit distorted when a graphic is resized. If this happens, you can force the graphic to be re-rendered at its new size by recalculating its formula. Re-rendering the graphic will reapply any font size you have specified as a graphics option.

Notes

- It is the name of a graphic that associates it with a particular Excel cell. For example, if a formula in cell E10 returns graphics, the picture named "Graphic E10" will be updated on the same sheet. If no picture named "Graphic E10" exists on that sheet, a new graphic will be created. This is the extent to which the picture is "linked". When cutting and pasting a \textit{Mathematica}-generated picture to another application or worksheet, this name-based "link" is not maintained. The copied graphic is simply a static picture. By modifying the name of a picture, you can change which cell updates that graphic. To modify the name of a picture, you can select the picture and click in the Name box on the left-hand side of the formula bar.
Generating Messages

If a message is generated during a kernel evaluation, it is sent to the Mathematica messages window. Here are some evaluations that generate kernel messages.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;7&quot;)</td>
<td>$\text{Failed or } \text{HVALUE!} $</td>
</tr>
<tr>
<td>=EVAL(&quot;1/0&quot;)</td>
<td>ComplexInfinity</td>
</tr>
<tr>
<td>=EVAL(&quot;Inverse&quot;,{(1,2),(2,4)})</td>
<td>Inverse[{{1, 2}, {2, 4}}]</td>
</tr>
</tbody>
</table>

**EVAL single argument examples.**

When a Mathematica evaluation returns $\text{Failed}$, an Excel #VALUE! is returned. This suppresses further evaluations that depend on this result. However, often when messages are generated, the original calculation request may be returned unevaluated, as shown in the last example.

**Notes**

- Options that control how the link responds to a kernel message can be specified under the Mathematica Options $\rightarrow$ Message tab.

Using Array Formulas

To return lists of values to multiple cells, Mathematica functions can be entered as an array formula. Some functions in Mathematica naturally return lists, others can be mapped over lists. Here are some examples.

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=EVAL(&quot;Range&quot;,&quot;10&quot;)</td>
<td>{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</td>
</tr>
<tr>
<td>=EVAL(&quot;Map&quot;,&quot;Factorial&quot;,A1:A10)</td>
<td>a list containing the factorial of each number in A1:A10</td>
</tr>
</tbody>
</table>

Formulas that can be entered as an array.

To enter a formula as an array:

Select all cells that will be in the array.

Enter the formula.

Press $\text{Ctrl} + \text{Shift} + \text{Enter}$. Once entered, array formulas appear surrounded by $\{ \}$ in the formula bar. All cells in an array formula range share a single formula, each element of the array returned to a separate cell.

The Mathematica Link add-in provides two tools to help make using array formulas easier:

The Mathematica Function Wizard can help you automatically enter and edit array formulas.
The **Array** command on the *Mathematica* context menu provides a way to easily toggle between single-cell and array formulas.

The **Array** toggle command on the *Mathematica* context menu.

**Notes**

- If you manually edit an array formula but forget to select all cells in the array beforehand, an error message will appear when you try to reenter the formula. If this happens, press Esc to cancel any changes you made. Then, select all cells in the array before making the changes again.

- Array formulas can significantly reduce calculation times if the same operation is being performed on a large number of cells. If you have filled a range with the same formula by dragging the formula across the range, it may be much faster to perform these calculations using an array formula. There is a transaction time associated with each call to *Mathematica*. If a single array formula is used to return values for 100 cells, recalculation could be up to 100 times faster than calling 100 individual formulas.

### Controlling Recalculation

#### Forcing Recalculation

Excel typically recalculates formulas only when the inputs to a formula have been changed. However, when you change the definition of a function or the value of a symbol in *Mathematica*, there is no way for Excel to automatically know about this change. In this case, you may want to force formulas to recalculate.

To force repeated recalculation of a specific formula:

Wrap the formula with the **CALC** function. You can then trigger recalculation of that formula by pressing F9.

**Example**

```
=CALC(EVAL("Random[]"))
```

**Description**

Gives a new random number each time F9 is pressed.

Marking a formula for repeated calculation using the **CALC** function.

To force one-time recalculation of the current selection:

Press Ctrl + Shift + = or choose **Recalculate** from the *Mathematica* context menu.

To force recalculation of all link functions in the current workbook:

Click the **Evaluate** button. This is an option specified under *Mathematica Options ➤ Workbook*. 
Disabling Recalculation

By temporarily storing formulas as comments, you can disable recalculation on a formula-by-formula basis. You may want to do this to preserve currently calculated values or to suppress unwanted recalculation.

To comment link formulas in the current selection:

Press \texttt{Ctrl} + \texttt{Shift} + \texttt{F} or check \texttt{Comment} from the Mathematica context menu.

To uncomment link formulas in the current selection:

Press \texttt{Ctrl} + \texttt{Shift} + \texttt{F} again or uncheck \texttt{Comment} on the Mathematica context menu.

Commenting a formula essentially freezes the formula in its currently calculated state. Commenting formulas for an entire workbook is discussed as a way of sharing a workbook with others in \textit{Sharing Workbooks}.

Notes

- To select an entire sheet, click the upper left-hand corner of the header row, between A and 1. You can then apply selection commands to the sheet.
- The \texttt{CALC} function can be used to generate random numbers for Excel-based simulations.
Working with Macros

Macro Types

There are three types of Mathematica macros that you can create.

<table>
<thead>
<tr>
<th>Macro Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitions Macro</td>
<td>defines Mathematica functions used in a workbook</td>
</tr>
<tr>
<td>Processing Macro</td>
<td>processes a workbook when the <strong>Evaluate</strong> button is clicked</td>
</tr>
<tr>
<td>Task Macro</td>
<td>performs a task when an associated button on a spreadsheet is clicked</td>
</tr>
</tbody>
</table>

Types of Mathematica macros.

The difference between the first two is that a workbook definitions macro does not perform actions on a workbook, a workbook processing macro does.

All Mathematica macros can be deployed by creating a code box to store the Mathematica code inside a workbook. For workbook-level macros, however, you can also store the Mathematica code in a package file.

<table>
<thead>
<tr>
<th>Macro Type</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitions Macro</td>
<td>initialization code box or workbook package file</td>
</tr>
<tr>
<td>Processing Macro</td>
<td>initialization code box or workbook package file</td>
</tr>
<tr>
<td>Task Macro</td>
<td>macro code box</td>
</tr>
</tbody>
</table>

Deploying Mathematica macros.

A workbook package file should be in the same directory and have the same name as a workbook with the .m extension replacing .xls.

Working with Code Boxes

Code boxes are simply text boxes that have been formatted for writing Mathematica code and given a special name. They are, in every other way, ordinary text boxes.

There are two ways of selecting a text box: selecting the box itself or selecting text inside the box. The selection border of the text box will be different in each case. To do anything other than edit the text inside the box you will need to make sure the box itself is selected. Once the box itself is selected, you can, for example, move it or resize it, copy and paste it to another location, or delete it.
When typing in a text box, be aware that Excel goes into a different mode. Most toolbar buttons, for example, are no longer available.

As with many toolbar buttons, Mathematica macros do not work while Excel is in this mode. To get out of this edit mode, press the Esc key or click out of the box.

**Working with Buttons**

By default, the caption of the button will be the same as the macro name. You can change this. You can also change the size of the button or move it. To do this, you will need to select the button.

To select a button:

- Hold down the Ctrl key and then click the button.

Once you have a button selected, you can work with it as you would any other shape. You can, for example, move it or resize it, change its caption, copy and paste it to another location, or delete it.

Right-clicking a button will also select it and work with it in various ways.
Developing on the Clipboard

The Mathematica Clipboard window can evaluate multiple lines of code at a time. When multiple lines are evaluated on the Clipboard, only the result from the last line is returned. You can use this behavior to progressively build up code that can be used as a macro by alternatively clicking Evaluate and Restore.

[Image of Mathematica Clipboard]

Developing code on the Clipboard.

When used this way, the Mathematica Clipboard window becomes a floating macro code box. The Clipboard Evaluate button becomes a Run button for the macro during development. When you are done developing, you can transfer the code to a code box.

To transfer code from the Mathematica Clipboard to a code box:

- Click Put on the Mathematica Clipboard.
- Select where to put the code in the code box.
- Press Ctrl + V or choose Edit ▶ Paste.

Developing in Mathematica

If you are familiar with the Mathematica notebook front end, you can develop code in that environment, then transfer it to a code box in Excel.

For more information on developing macros in Mathematica, see Creating Excel Macros.

Protecting Your Code

Once you have developed a macro, you may want to protect your code. This can be done by hiding or protecting the sheet containing the code.

To hide a sheet in Excel:

- Click Format ▶ Sheet ▶ Hide.

To protect a sheet in Excel:

- Click Tools ▶ Protection ▶ Protect Sheet.

See Excel's Help menu for more information on these commands.
Link Management

Opening a Link
The first time you request a Mathematica evaluation in Excel, a link to a Mathematica kernel will open automatically. If you would like to force a link to open, even if there is nothing to evaluate, you can:

- Click the Evaluate icon on the Mathematica toolbar.

You can manually specify a kernel to connect to using the Mathematica Options ➤ Kernel tab. Settings you specify will be used automatically the next time you connect.

Notes
- If you do not have a typical Mathematica installation, you may be required to specify a kernel the first time you use the link.

Interrupting Evaluations
To interrupt a Mathematica evaluation:

- Click Evaluate on the Mathematica toolbar.

Although most Mathematica evaluations can be interrupted, some cannot. If the kernel does not respond to an abort request after some time, you can choose to close the kernel to end the evaluation.

To close a kernel during an evaluation and return control to Excel:

- Hold down Shift and click Evaluate on the Mathematica toolbar, or
- Right-click the Mathematica kernel in the task bar and choose Close.

Notes
- Pressing the Esc key does not interrupt Mathematica evaluations. The Esc key is used by Excel to send interrupts to local Visual Basic code.
- Aborted formulas return #NULL!, thereby suppressing evaluation of dependent formulas.

Closing a Link
To close a link with a Mathematica kernel, you can:

- Hold down the Shift key while clicking Evaluate.
- Click Close under Mathematica Options ➤ Kernel.
- Evaluate Quit [] in the Clipboard window, a function, or a macro.

Notes
- The kernel is automatically closed whenever Excel is closed or the MathematicaLink add-in is uninstalled.
Sharing Workbooks

Unlinking a Workbook

Before sharing a workbook containing Mathematica formulas or macros, you may first want to unlink the workbook. Unlinking a workbook will allow others to take a look at the results in the workbook without launching a kernel, encountering pathnames from your hard drive, or inadvertently replacing cell values with #NAME? errors.

To unlink a workbook:

- Click Unlink on the Mathematica Options ➤ Workbook tab.

When unlinked, formulas are stored as cell comments, and macro buttons display a "macro cannot be found" error when clicked.

Once a workbook is unlinked, it can easily be viewed without problems on other machines, even machines where Mathematica Link is not installed. Those who do have the link will have the option of relinking the workbook and interacting with it.

Notes

- In Excel, cell comments are typically indicated by a small red triangle in the upper-right corner of a cell. This is an option that can be specified under Tools ➤ Options ➤ View. To view a comment, place your cursor over the cell.
Relinking a Workbook

When opening an unlinked workbook, link users are automatically prompted to relink formulas in the workbook.

- Click **Yes** to relink all formulas contained in the workbook.
- Click **No** to leave the workbook as is. You can relink formulas later, at any time, by clicking **Relink** on the **Mathematica Options ▶ Workbook** tab.

Notes

- Relinking formulas recalculates all Mathematica formulas in the workbook, overwriting the existing set of results.
  - If a workbook contains only Mathematica macros, the relink prompt will not appear. Macro buttons are relinked without prompting since this has no other impact on the workbook.

Fixing Broken Links

If a workbook was not unlinked prior to opening it on another machine, broken path-based formula links can be fixed automatically by clicking **Relink** under **Mathematica Options ▶ Workbook**.

This is possible, of course, only if the Mathematica Link add-in is available on that machine.

Notes

- Fixing broken links recalculates all Mathematica formulas in the workbook, overwriting the existing set of results.
Loading the Add-In

Before you can transfer data between *Mathematica* notebooks and Excel workbooks via the clipboard, you first need to load the *Mathematica Link* add-in in Excel. To do this, see *Working in Excel: Getting Started*.

Once installed, the *Mathematica Link* add-in adds special copy and paste commands to Excel. These commands convert Excel data to and from *Mathematica* lists.
Copying Data from Excel

To copy the contents of an Excel range to a Mathematica notebook:

- In Excel, select the range you want to copy.
- Press \( \text{Ctrl} + \text{Shift} + \text{C} \) or right-click and choose **Copy** if *Mathematica* Contexts are enabled. This will copy the contents of the range onto the Clipboard as a *Mathematica* list.
- Switch to your *Mathematica* notebook and place the cursor where you would like to insert the list.
- Press \( \text{Ctrl} + \text{V} \) or choose **Paste** from the **Edit** menu.

Notes

- If the contents of the range were not converted into a *Mathematica* list, verify *Mathematica* keyboard shortcuts are enabled in Excel. You can do this under *Mathematica Options > Interface*.
- By default, text cells are copied as *Mathematica* strings. Only cells formatted as Text are considered to contain *Mathematica* expressions. See Strings. If needed, you can convert strings to expressions using \( \text{ToExpression} \ [\text{data}] \).
- Empty cells are copied as the symbol \( \text{Empty} \). If needed, you can convert these to a desired default value, such as \( 0 \) or \( " \), using \( \text{ReplaceAll} \ [\text{data}, \text{Empty} \rightarrow \text{val}] \).
Pasting Data to Excel

To paste the contents of a Mathematica list to an Excel range:

- In a Mathematica notebook, select the list you wish to copy.
- Press $\text{Ctrl} + \text{C}$ or choose \textbf{Copy} from the Edit menu.
- Switch to Microsoft Excel and select where to insert the contents of the list. If a single cell is selected, data will be pasted below and to the right of that cell.
- Press $\text{Ctrl} + \text{Shift} + \text{V}$ or right-click and choose \textbf{Paste} if Mathematica contexts are enabled.

Notes

- If nothing was pasted, verify Mathematica keyboard shortcuts are enabled in Excel. You can do this under Mathematica Options $\rightarrow$ Interface.

- For best results, convert your data to Mathematica InputForm before pasting the data into Excel. You can do this in Mathematica or using the Mathematica Clipboard window in Excel. See Fixing Problematic Data.

  Be sure the opening and closing brackets of the Mathematica list are selected. Excel will not recognize the Clipboard contents as a Mathematica list unless these brackets are present.

- A convenient way to select an entire Mathematica list is to place the cursor at the beginning of the list and triple-click.
Fixing Problematic Data

If you run into problems pasting data into Excel, it is likely the copied data is not in the proper form. The Paste command requires data to be in Mathematica InputForm. Depending on how you copied the data and from what kind of notebook cell you copied it, you may end up with data in a format other than InputForm.

The Paste command can fix certain common problems with the data automatically; however, in some cases you may still need to convert to InputForm on your own.

You can do this in two ways:

- In Mathematica before copying, use Cell ▶ Convert ▶ InputForm.

- In Excel, after copying, use the Evaluate button in the Mathematica Clipboard window.

If you do not want to modify your source notebook, the second approach is the most convenient.

Here is an example of some data to paste.

```
In[1]:= data = Table[Random[]*10^-8, {10}]
```

```
Out[1]= {6.92556×10^-9, 5.23638×10^-9, 4.21124×10^-9, 4.61886×10^-9, 9.65829×10^-10,
        5.8988×10^-9, 4.26737×10^-9, 1.64441×10^-9, 1.50418×10^-9, 7.90577×10^-9}
```

To view this data in the Mathematica Clipboard window, copy the output, then in Excel, open the Mathematica Clipboard window and click Get.

The Mathematica Clipboard window before clicking Evaluate.
You can see that additional formatting characters are present in the data. In this case, the fixes are minor, so pasting directly into Excel does, in fact, still work. However, you can also convert the contents of the Clipboard to InputForm by clicking Evaluate.

Once the data has been standardized to InputForm, you can continue to paste it into Excel using the standard Paste command or the Paste button on the Clipboard window.

Notes

- An alternative to copying and pasting is to assign and retrieve Excel data programmatically from Mathematica. This can be significantly faster if you are dealing with large datasets. For more information, see Getting Started.
Excel Worksheet Functions

EVAL

=EVAL(expr) evaluates expr in a Mathematica kernel.

=EVAL(head, arg1, arg2, ...) builds an expression from the provided head and args and evaluates it.

- =EVAL(head, arg1, arg2, ...) is equivalent to =EVAL(EXPR(head, arg1, arg2, ...)).
- You can use =EXPR(head, arg1, arg2, ...) to see exactly what will be evaluated in Mathematica.
- =EVAL(head, arg1, arg2, ...) evaluates only once head and all arguments have been provided. If head or any argument refers to an empty cell or an error, evaluation is suppressed and #N/A is returned.
- See notes for EXPR.
- See also: DATA, RULE.

EXPR

=EVAL(expr) evaluates expr in a Mathematica kernel.

=EVAL(head, arg1, arg2, ...) builds an expression from the provided head and args and evaluates it.

- =EXPR(head, arg1, arg2, ...) builds the Mathematica expression head[arg1, arg2, ...].
- Example: =EXPR("f","x","y") returns f[x,y].
- You can build any expression using nested EXPR calls.
- If head contains #, head is automatically assumed to be a pure function.
- Example: =EXPR("Sqrt[#1^2+#2^2"]",A1,A2).
- =EXPR(head, arg1, arg2, ...) returns #N/A if head or any arguments refer to an empty cell or an error.
- =EXPR(range) trims empty cells and errors from the end of range.
- See also: EVAL, DATA, RULE.
DATA

- `EVAL(expr)` evaluates `expr` in a Mathematica kernel.
- `EVAL(head, arg1, arg2, ...)` builds an expression from the provided `head` and `args` and evaluates it.

- The DATA function is typically used inside EVAL, EXPR, or RULE calls.
- Example: `EVAL("FileNames", DATA("*"), DATA("C:\"))`.
- You can wrap numeric data in the DATA function to ensure numeric calculation methods are used instead of symbolic methods. The former is often much faster.
- Unlike EXPR ranges, DATA ranges can contain empty cells or errors.
- `DATA(range)` trims empty cells and errors from the end of `range`.
- See also: EVAL, EXPR, RULE.

RULE

- `EVAL(expr)` evaluates `expr` in a Mathematica kernel.
- `EVAL(head, arg1, arg2, ...)` builds an expression from the provided `head` and `args` and evaluates it.

- `RULE(lhs, rhs)` builds the Mathematica expression `lhs->rhs`.
- Example: `RULE("x", 1)` returns `x->1`.
- The RULE function can be used to specify options for the Mathematica function in EVAL or EXPR calls.
- If `lhs` and `rhs` are references to multicell ranges of the same size, a list of rules is returned.
- See also: EVAL, EXPR, DATA.
### CALC

=**EVAL**(expr) evaluates expr in a Mathematica kernel.

=**EVAL**(head, arg1, arg2, ...) builds an expression from the provided head and args and evaluates it.

- Volatile functions recalculate when any cell changes or when the F9 key is pressed.
- Nonvolatile formulas recalculate only when a referenced input cell changes.
- Excel functions RAND() and NOW() are examples of built-in volatile functions.
- Volatile functions are typically used to run simulations in Excel.
- You can use the **CALC** worksheet function to use Mathematica-generated random numbers within Excel-based simulations.
- Example: =CALC(EVAL("Random[Integer,[1,6]]")).
- You can use the **CALC** worksheet function to "tag" specific functions to respond when F9 is pressed or when ExcelCalculate is called from Mathematica code.
- Example: =CALC(EVAL("Set", "stockprice", G10)).
- Note: volatile functions recalculate frequently and can, therefore, significantly slow down recalculation and editing operations in your spreadsheet. To avoid any potential problems, use volatile functions sparingly and, if needed, disable them when making modifications to your spreadsheets.
- See also: **EVAL**.
Toolbar Commands

Evaluate

Establishes a connection with a Mathematica kernel.
Reinitializes and/or recalculates the active workbook (based on workbook options).
Closes or resets the connection with a Mathematica kernel (depending on modifier key).
Interrupts Mathematica evaluation (if an evaluation is currently running).

- Workbook options can be specified under the Mathematica Options ▶ Workbook tab.
  - Shift + click: closes the kernel.
  - Ctrl + click: clears the kernel’s Global context. (Soft kernel reset)
  - Ctrl + Shift + click: closes and restarts the kernel. (Hard kernel reset)
- Location: Mathematica toolbar, Mathematica menu
- Keyboard shortcut: Ctrl + Alt + E

Functions

Establishes a connection with a Mathematica kernel.
Reinitializes and/or recalculates the active workbook (based on workbook options).
Closes or resets the connection with a Mathematica kernel (depending on modifier key).
Interrupts Mathematica evaluation (if an evaluation is currently running).

- If no link formula is selected, the Wizard starts at the Mathematica Function Browser step.
- If an existing link formula is selected, the Wizard starts at the Mathematica Function Arguments step, and the Wizard is populated with the existing formula.
- Using the Wizard you can search for, learn about, and interactively build and edit Mathematica function calls.
- Note: the Wizard is nonmodal. You can leave it open and still work in Excel.
- Location: Mathematica toolbar, Mathematica menu
- Keyboard shortcut: Ctrl + Alt + F
# Macros

Establishes a connection with a Mathematica kernel.

Reinitializes and/or recalculates the active workbook (based on workbook options).

Closes or resets the connection with a Mathematica kernel (depending on modifier key).

Interrupts Mathematica evaluation (if an evaluation is currently running).

- Using this window, you can create code boxes and corresponding buttons that allow you to run Mathematica code as if it were an Excel macro. Using this window, you can also create initialization code boxes for workbooks that contain code that is run automatically when you connect to a kernel or reevaluate the workbook.

- Note: the window is nonmodal. You can leave it open and still work in Excel.

- Location: Mathematica toolbar, Mathematica menu

- Keyboard shortcut: Ctrl + Alt + M

# Clipboard

Establishes a connection with a Mathematica kernel.

Reinitializes and/or recalculates the active workbook (based on workbook options).

Closes or resets the connection with a Mathematica kernel (depending on modifier key).

Interrupts Mathematica evaluation (if an evaluation is currently running).

- Using this window, you can interactively copy and paste data to Excel ranges.

- You can also type code directly into the Mathematica Clipboard window, evaluate it, and view the results.

- Note: the window is nonmodal. You can leave it open and still work in Excel.

- Location: Mathematica toolbar, Mathematica menu

- Keyboard shortcut: Ctrl + Alt + C

# Contexts

Establishes a connection with a Mathematica kernel.

Reinitializes and/or recalculates the active workbook (based on workbook options).

Closes or resets the connection with a Mathematica kernel (depending on modifier key).

Interrupts Mathematica evaluation (if an evaluation is currently running).

- When menus are enabled, if you right-click a range, a custom Mathematica Context menu will appear.
When menus are disabled, if you right-click a range, the default Excel Context menu will appear.

Location: Mathematica toolbar, Mathematica menu

Keyboard shortcut: Ctrl + Alt + X

Messages

Establishes a connection with a Mathematica kernel.
Reinitializes and/or recalculates the active workbook (based on workbook options).
Closes or resets the connection with a Mathematica kernel (depending on modifier key).
Interrupts Mathematica evaluation (if an evaluation is currently running).

Using this window, you can use view messages generated by the Mathematica kernel.
You can also locate the source of messages, browse through past messages, and save messages to a log file.
Note: the window is nonmodal. You can leave it open and still work in Excel.

Location: Mathematica toolbar, Mathematica menu

Keyboard shortcut: Ctrl + Alt + G

Options

Establishes a connection with a Mathematica kernel.
Reinitializes and/or recalculates the active workbook (based on workbook options).
Closes or resets the connection with a Mathematica kernel (depending on modifier key).
Interrupts Mathematica evaluation (if an evaluation is currently running).

You can browse this window to explore link options available to you. If you change an option, but want to leave the window open, click Apply to apply the changes.

Note: the window is nonmodal. You can leave it open and still work in Excel.

Location: Mathematica toolbar, Mathematica menu

Keyboard shortcut: Ctrl + Alt + O
Help

Establishes a connection with a *Mathematica* kernel.

Reinitializes and/or recalculates the active workbook (based on workbook options).

Closes or resets the connection with a *Mathematica* kernel (depending on modifier key).

Interrupts *Mathematica* evaluation (if an evaluation is currently running).

Adobe Reader is required to view this version of the manual.

Contents of this manual can also be viewed in the *Mathematica* Documentation Center under Add-Ons & Links » *Mathematica Link for Excel*.

Location: *Mathematica* toolbar, *Mathematica* menu

Keyboard shortcut: **Ctrl + Alt + H**
## Context Commands

### Expression

<table>
<thead>
<tr>
<th>Toggle command. Applies or removes the Text number format from the selected range.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contents of the range are automatically reentered after the format change.</strong></td>
</tr>
<tr>
<td><strong>If the number format for all cells is Text, the format for all cells is changed to General.</strong></td>
</tr>
<tr>
<td><strong>If the number format for all cells is not Text, the format for all cells is changed to Text.</strong></td>
</tr>
<tr>
<td><strong>Location: Context Menu</strong></td>
</tr>
<tr>
<td><strong>Keyboard shortcut:</strong> Ctrl + Shift + E</td>
</tr>
</tbody>
</table>

### Copy

<table>
<thead>
<tr>
<th>Toggle command. Applies or removes the Text number format from the selected range.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>If the number format for all cells is Text, strings are copied as Mathematica text expressions.</strong></td>
</tr>
<tr>
<td><strong>If the number format for all cells is not Text, strings are copied as Mathematica strings.</strong></td>
</tr>
<tr>
<td><strong>The copy command uses Mathematica InputForm to represent the contents of the range.</strong></td>
</tr>
<tr>
<td><strong>If the Mathematica Clipboard window is open, the window is automatically updated, allowing you to view what has been placed on the Clipboard.</strong></td>
</tr>
<tr>
<td><strong>Location: Context Menu</strong></td>
</tr>
<tr>
<td><strong>Keyboard shortcut:</strong> Ctrl + Shift + C</td>
</tr>
</tbody>
</table>

### Paste

<table>
<thead>
<tr>
<th>Toggle command. Applies or removes the Text number format from the selected range.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>If the number format for all cells in the target range is Text, the contents of the Clipboard will be pasted as Mathematica text expressions.</strong></td>
</tr>
<tr>
<td><strong>If the number format for all cells is not Text, the contents of the Clipboard will be converted to native Excel data types wherever possible.</strong></td>
</tr>
<tr>
<td><strong>If the Mathematica Clipboard window is open, you can view the text expression before it is pasted. The Paste command requires the contents of the Clipboard to be in Mathematica InputForm.</strong></td>
</tr>
<tr>
<td><strong>Note:</strong> If the expression on the Clipboard is not in InputForm, you can use the Mathematica Clipboard window to convert the expression to InputForm by clicking Evaluate.</td>
</tr>
</tbody>
</table>
Clear

Toggle command. Applies or removes the Text number format from the selected range.

When using this command, the number format for the range is reset to General. This is the only difference between this command and the Excel standard Edit ▶ Clear Contents command.

Function

Toggle command. Applies or removes the Text number format from the selected range.

If the selected cell does not contain a link formula, the Wizard goes directly to the Mathematica Function Arguments step. This is the only difference between this command and the Toolbar Command ▶ Functions command.

Use this command if you know what function you would like to enter, and you do not need to browse for it.

See also: Toolbar Command ▶ Functions

Array

Toggle command. Applies or removes the Text number format from the selected range.

When converting an array formula to a single-cell formula, you can select the top-left cell or the entire array.

When converting a single-cell formula to an array formula, the array formula is automatically sized correctly, or you can preselect the cells you would like to be included in the array.
Comments

Toggle command. Applies or removes the Text number format from the selected range.

- Commenting formulas allows you to temporarily disable evaluation of the selected formulas.
- To comment all Mathematica formulas on a sheet, click the top-left corner of the header rows between A and 1 to select the entire sheet.
- To comment all Mathematica formulas in a workbook, click Unlink under Mathematica Options ▶ Workbook.
- Location: Context Menu
- Keyboard shortcut: Ctrl + Shift + "

Recalculate

Toggle command. Applies or removes the Text number format from the selected range.

- To force recalculation of all Mathematica formulas on a sheet, click the top-left corner of the header rows, between A and 1 to select the entire sheet.
- To force recalculation of all Mathematica formulas in the workbook, you can click Evaluate if your options are set to do so.
- Location: Context Menu
- Keyboard shortcut: Ctrl + Shift + =
# Keyboard Shortcuts

## Toolbar Shortcuts

Keyboard shortcuts are available for all commands on the *Mathematica* toolbar. These toolbar shortcuts all include the **Alt** key.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate</td>
<td>Ctrl + Alt + E</td>
</tr>
<tr>
<td>Functions</td>
<td>Ctrl + Alt + F</td>
</tr>
<tr>
<td>Macros</td>
<td>Ctrl + Alt + M</td>
</tr>
<tr>
<td>Clipboard</td>
<td>Ctrl + Alt + C</td>
</tr>
<tr>
<td>Contexts</td>
<td>Ctrl + Alt + X</td>
</tr>
<tr>
<td>Messages</td>
<td>Ctrl + Alt + G</td>
</tr>
<tr>
<td>Options</td>
<td>Ctrl + Alt + O</td>
</tr>
<tr>
<td>Help</td>
<td>Ctrl + Alt + H</td>
</tr>
</tbody>
</table>

For descriptions of these commands, see the [Toolbar Commands](#) section in this reference guide.

## Context Shortcuts

Keyboard shortcuts are available for all commands in the *Mathematica* context menu. Context shortcuts all include the **Shift** key. They all operate on the *current selection* in Excel.

<table>
<thead>
<tr>
<th>Command</th>
<th>Shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Ctrl + Shift + E</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl + Shift + C</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl + Shift + V</td>
</tr>
<tr>
<td>Clear</td>
<td>Ctrl + Shift + Delete</td>
</tr>
<tr>
<td>Function</td>
<td>Ctrl + Shift + F</td>
</tr>
<tr>
<td>Array</td>
<td>Ctrl + Shift + A</td>
</tr>
<tr>
<td>Comments</td>
<td>Ctrl + Shift + '</td>
</tr>
<tr>
<td>Recalculate</td>
<td>Ctrl + Shift + =</td>
</tr>
</tbody>
</table>

For descriptions of these commands, see the [Context Commands](#) section in this reference guide.
Workbook Shortcuts

Keyboard shortcuts are available for the following workbook-level operations.

- **Reevaluate initialization code**: Ctrl + Enter
- **Recalculate link formulas**: Ctrl + =
- **Unlink / Relink workbook**: Ctrl + ' 

Keyboard shortcuts for workbook operations.

The functionality provided by some of these commands can optionally occur automatically as part of the Evaluate toolbar command. Related options and buttons can be found on the Mathematica Options ▶ Workbook tab.

Notes

- Keyboard shortcuts can be disabled and re-enabled using the Mathematica Options ▶ Interface tab.
Data Types

Overview

Excel cells can contain six different types of data.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>machine-precision double</td>
</tr>
<tr>
<td>Dates</td>
<td>machine-precision double with special formatting</td>
</tr>
<tr>
<td>Strings</td>
<td>unicode text</td>
</tr>
<tr>
<td>Booleans</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Errors</td>
<td>#VALUE!, #REF!, ...</td>
</tr>
<tr>
<td>Empty</td>
<td>empty cell</td>
</tr>
</tbody>
</table>

Native Excel data types.

Mathematica data types not natively supported by Excel can be stored as strings in a range formatted as Text.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>Rational</td>
<td>1/2</td>
</tr>
<tr>
<td>Complex</td>
<td>1 + 2 I</td>
</tr>
<tr>
<td>Symbol</td>
<td>x</td>
</tr>
<tr>
<td>Expression</td>
<td>x + y</td>
</tr>
</tbody>
</table>

Mathematica data types that can be stored in text cells.

To specify the format of a range, you can use the ExcelFormat function provided by the ExcelLink package. Or, from within Excel, you can select a format in the Format ➤ Cell... ➤ Number pane or use the Expression toggle command provided by the MathematicaLink add-in.

Notes

- For worksheet functions, such as EVAL or EXPR, inputs are always assumed to contain text expressions, even if Text formatting has not been applied to input ranges. This assumption makes it easier to build Mathematica expressions. To use data as it is natively stored in Excel, wrap the reference to the data range with the DATA worksheet function.

- When a number is stored as text in Excel, an error flag may appear on the cell. If you find these flags distracting, you can turn this type of error checking off in the Tools ➤ Options... ➤ Error Checking pane.
Numbers

Excel stores all numbers as machine-precision doubles. A number may appear to be an integer in Excel when no decimal point is displayed, however internally the number is stored as a floating point. Whether a decimal point is displayed is a matter of formatting. See Excel Number Formats for more information.

Maximum and minimum numbers that can be natively stored in Excel are defined by machine-precision limits.

<table>
<thead>
<tr>
<th>$\text{MaxMachineNumber}$</th>
<th>$\text{MinMachineNumber}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7976931348623157\times10^{308}</td>
<td>2.2250738585072014\times-308</td>
</tr>
</tbody>
</table>

Excel number limits.

Outside of this range, numbers can be stored as Mathematica text expressions.

Dates

Excel dates are stored as numbers where the integer part represents the day and the fractional part represents the time of day.

| 0.0 | December 30, 1899 (beginning of day) |
| 2.0 | January 1, 1900 (beginning of day) |
| 2.5 | January 1, 1900 (midday) |
| 36526.0 | January 1, 2000 (beginning of day) |
| 36526.5 | January 1, 2000 (midday) |

How Excel stores date and time information.

What makes a number appear as a date in Excel is a matter of formatting. See Excel Number Formats for more information.

Notes

- Excel cells only format dates properly after March 1, 1900. Excel erroneously considers 1900 a leap year, and negative numbers cannot be formatted as dates in Excel.
- The ExcelForm function does not suffer from the same restrictions as Excel cells and can therefore be used to print out dates before March 1, 1900.

Strings

In Excel, textual data can be entered into cells specifically formatted as Text. It can also be entered into cells with other formats, such as General. MathematicaLink for Excel uses this formatting difference to distinguish between text expressions and strings. If a cell is formatted as Text, the content of the cell is considered to be a Mathematica expression; otherwise, the contents are considered to be a Mathematica string.

<table>
<thead>
<tr>
<th>xyz (Text format)</th>
<th>xyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>xyz (other format)</td>
<td>&quot;xyz&quot;</td>
</tr>
</tbody>
</table>

Excel string mapping.
In Excel, the two versions of the string appear identical. However, there are significant behavioral differences between text cells and other cells:

Strings assigned to nontext cells pass through an Excel interpreter. The interpreter checks to see if the string being assigned appears to be a data type it knows about. If so, the string is converted to the identified data type and an appropriate cell format is automatically applied if needed.

Strings assigned to text cells do not pass through the Excel interpreter. They can therefore contain exact numbers, rationals, or anything else that can be stored as a Mathematica expression.

As an example, if 1/2 is assigned to a cell with the General format, the rational is interpreted as January 2 of the current year, and a date format is automatically assigned to the cell. When 1/2 is assigned to a text cell, the fraction remains as entered.

Notes

- For worksheet functions, such as EVAL or EXPR, input ranges are always assumed to contain text expressions, even if the input range has not been formatted as Text. This assumption makes it easy to build Mathematica expressions. You can wrap data ranges with the DATA worksheet function to treat text contained in the range as Mathematica strings.

Booleans

Excel Booleans are mapped to corresponding Mathematica symbols.

<table>
<thead>
<tr>
<th>TRUE</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>False</td>
</tr>
</tbody>
</table>

Excel Boolean mapping.

Errors

Excel errors are converted to Mathematica strings.

<table>
<thead>
<tr>
<th>#N/A</th>
<th>&quot;#N/A&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>#REF!</td>
<td>&quot;#REF!&quot;</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>&quot;#VALUE!&quot;</td>
</tr>
<tr>
<td>#NULL!</td>
<td>&quot;#NULL!&quot;</td>
</tr>
<tr>
<td>#NAME?</td>
<td>&quot;#NAME?&quot;</td>
</tr>
<tr>
<td>#NUM!</td>
<td>&quot;#NUM!&quot;</td>
</tr>
<tr>
<td>#DIV/0!</td>
<td>&quot;#DIV/0!&quot;</td>
</tr>
</tbody>
</table>

Excel error mapping.
Of these errors, four may be returned by Mathematica worksheet functions. In this case, the errors have the following meanings.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#N/A</td>
<td>error or empty found in inputs</td>
</tr>
<tr>
<td>#REF!</td>
<td>connection error or invalid kernel path</td>
</tr>
<tr>
<td>#VALUE!</td>
<td>evaluation returned $Failed</td>
</tr>
<tr>
<td>#NULL!</td>
<td>evaluation returned $Aborted or link was closed</td>
</tr>
</tbody>
</table>

Mathematica worksheet function errors.

Notes

- For worksheet functions, such as EVAL or EXPR, the #N/A error suppresses further evaluation. To force a range containing empty cells or errors to be evaluated in Mathematica, you can use the DATA function as an argument wrapper. DATA(range) is a data range that may contain empty cells or errors. You can then handle the empty values and errors in your Mathematica code.

Empty

Mathematica Link for Excel handles empty cells differently depending on where the cells are located.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-trailing empty</td>
<td>Empty</td>
</tr>
<tr>
<td>trailing empty</td>
<td>trimmed from range</td>
</tr>
</tbody>
</table>

Excel empty cell mapping.

For empty cells to be trimmed, an entire trailing row or column within the range must contain nothing but empty cells.

Notes

- The trimming of trailing empty cells allows you to easily work with entire rows and columns of data.
Number Formats

Excel number formats allow you to define how a number will appear in your spreadsheet.

<table>
<thead>
<tr>
<th>Number Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>General</td>
</tr>
<tr>
<td>1.0E+03</td>
<td>0.0E+00</td>
</tr>
<tr>
<td>$ 1,000.00</td>
<td>$ 1,000.00</td>
</tr>
<tr>
<td>100000%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Examples of Excel number formats.

Excel number formats are also used to format dates and time information that has been stored as a number.

<table>
<thead>
<tr>
<th>Date/Time Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 1, 2000</td>
<td>mmmm d, yyyy</td>
</tr>
<tr>
<td>12:00</td>
<td>hh:mm</td>
</tr>
<tr>
<td>01-Jan-00 12:00:00</td>
<td>dd-mmm-yy hh:mm:ss</td>
</tr>
</tbody>
</table>

Examples of Excel date formats.

For more examples of Excel number formats, browse the Custom category in the Format ▶ Cells… ▶ Number pane in Excel.
**ExcelLink Functions**

**ExcelActivate**

```plaintext
ExcelActivate[book]  
makes book the active book.
```

```plaintext
ExcelActivate[sheet]  
makes sheet the active sheet.
```

**Examples**

```plaintext
In[1]:=<ExcelLink`

In[2]:= ExcelActivate[ExcelSheet["Book1","Sheet1"]]

Out[3]= {Book1, Sheet1}

In[4]:= ExcelActivate["Sheet3"]

Out[5]= {Book1, Sheet3}

In[6]:= ExcelActivate["Sheet1"]

Out[7]= {Book1, Sheet1}
```

**ExcelAddress**

```plaintext
ExcelAddress[range]  
returns the A1-style address of the specified range.
```

**Examples**

```plaintext
In[1]:=<ExcelLink`

In[2]:= ExcelAddress[{{1, 1}, {10, 3}}]

```
ExcelBook

ExcelBook[id]
represents a workbook identified by id. The id can be Active, a name, a full path, or an index.

Details

- The id can be one of the following:

<table>
<thead>
<tr>
<th>id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>currently active workbook</td>
</tr>
<tr>
<td>&quot;Book1&quot; or &quot;Report.xls&quot;</td>
<td>the name of the workbook</td>
</tr>
<tr>
<td>&quot;C:\Reports\Report.xls&quot;</td>
<td>the full path of the workbook</td>
</tr>
<tr>
<td>1 or -1</td>
<td>a positive or negative position index</td>
</tr>
</tbody>
</table>

- The first workbook opened has position 1. The last workbook opened has position -1.

- Once a workbook is saved, the .xls suffix is required to identify the workbook by name or path.

Examples

```
In[1]:= <<ExcelLink

In[2]:= ExcelBooks[]
Out[2]= {Book: Book1}

In[3]:= ExcelName[ExcelBook[Active]]
Out[3]= Book1

In[4]:= ExcelNew[]

In[5]:= ExcelBooks[]

In[6]:= ExcelName[ExcelBook[1]]
Out[6]= Book1

In[7]:= ExcelName[ExcelBook[-1]]

In[8]:= ExcelClose[]

In[9]:= ExcelBooks[]
Out[9]= {Book: Book1}

In[10]:= ExcelCheck[ExcelBook[2]]
Out[10]= False
```
ExcelBooks

ExcelBooks[]
gives a list of workbooks currently open in Excel.

Examples

\[
\begin{align*}
\text{In}[1]:= & \quad \text{<<ExcelLink```}
\text{In}[2]:= & \quad \text{ExcelBooks[]}
\text{Out}[2]= & \quad \{-\text{Book: Book1}\}\n\text{In}[3]:= & \quad \text{ExcelNew[]}
\text{Out}[3]= & \quad -\text{Book: Book2}
\text{In}[4]:= & \quad \text{ExcelBooks[]}
\text{Out}[4]= & \quad \{-\text{Book: Book1}, -\text{Book: Book2}\}
\text{In}[5]:= & \quad \text{ExcelClose[]}
\end{align*}
\]

ExcelCalculate

ExcelCalculate[]
causes all formula-based calculations in Excel to update.

Details

- This function can be useful if Excel has been set to manual calculation mode or if volatile functions, such as random number generators, are being used.

Examples

\[
\begin{align*}
\text{In}[1]:= & \quad \text{<<ExcelLink```}
\text{In}[2]:= & \quad \text{ExcelWrite["A1", "=RAND()"]}
\text{In}[3]:= & \quad \text{ExcelRead["A1"]}
\text{Out}[3]= & \quad 0.0145436
\text{In}[4]:= & \quad \text{ExcelCalculate[]}
\text{In}[5]:= & \quad \text{ExcelRead["A1"]}
\text{Out}[5]= & \quad 0.0438649
\text{In}[6]:= & \quad \text{ExcelClear["A1"]}
\end{align*}
\]

ExcelCall

ExcelCall
is an internal function called by all functions that need to communicate with Excel.
ExcelCheck

ExcelCheck[object]
   checks if object is valid.

ExcelCheck[object, format " ]
   checks if object has the specified format.

Details

- ExcelCheck always returns True or False.
- If object is a shape, the format can be "Image", "Text", or "Other".
- If object is a sheet, the format can be "Work" or "Chart".
- If object is a range, the format can be "General", "Text", a number format, or "Mixed".

Examples

In[1]:= <<ExcelLink` 
In[2]:= object=ExcelRange["Sheet1","A1"] 

In[3]:= ExcelCheck[object]

In[4]:= object=ExcelRange["Bogus","A1"] 

In[5]:= ExcelCheck[object] 

In[6]:= object=ExcelSheet[1] 
Out[6]= Sheet: 1

In[7]:= ExcelCheck[object, "Work"] 
Out[7]= True

In[8]:= ExcelCheck[object, "Chart"] 
Out[8]= False
ExcelClear

ExcelClear[range]
clears data from the specified range in Excel.

ExcelClear[sheet]
clears all data contained in the specified sheet.

Details

* ExcelClear[All] and ExcelClear[Active] both clear all data on the active sheet.

Examples

In[1]:= <<ExcelLink`
In[2]:= ExcelWrite["A:A", $Path]
In[3]:= ExcelClear["A:A"]
In[4]:= ExcelWrite["Sheet1", Table[Random[],{1000},{10}]]
In[5]:= ExcelClear["Sheet1"]

ExcelClose

ExcelClose[]
closes the active workbook.

ExcelClose[book]
closes the specified book.

Examples

In[1]:= <<ExcelLink`
In[2]:= book = ExcelNew[]
Out[2]= -Book: Book2-

In[3]:= ExcelBooks[]

In[4]:= ExcelClose[book]
In[5]:= ExcelBooks[]
ExcelContext

ExcelContext[]
returns a list identifying the current active context.

ExcelContext[object]
returns a list identifying the context of the specified object.

Examples

\[\text{In[1]}:= \text{``ExcelLink``}\]
\[\text{In[2]}:= \text{ExcelActivate["Book1"]}\]
\[\text{In[3]}:= \text{ExcelActivate["Sheet1"]}\]
\[\text{In[4]}:= \text{ExcelContext[]}\]
\[\text{Out[4]}= \{\text{Book1, Sheet1}\}\]
\[\text{In[5]}:= \text{object=ExcelRange["Book1","Sheet2","A1:C10"]}\]
\[\text{Out[5]}= \text{Range: A1:C10}\]
\[\text{In[6]}:= \text{ExcelContext[object]}\]
\[\text{Out[6]}= \{\text{Book1, Sheet2}\}\]

ExcelDate

ExcelDate[value]
converts an Excel date value to a date list \(\{y, m, d, h, n, s\}\).

ExcelDate[{y, m, d, h, n, s}]
converts a date list to an Excel date value.

Details

- Excel dates count days from December 30, 1899.
- Time of day information is provided by the fractional part of the date, where 0.5 represents noon.
- Excel dates can be printed in various formats using the ExcelForm function.

Examples

\[\text{In[1]}:= \text{``ExcelLink``}\]
\[\text{In[2]}:= \text{d = Date[]}\]
\[\text{Out[2]}= \{2006, 7, 11, 13, 8, 48.6900112\}\]
\[\text{In[3]}:= \text{n = ExcelDate[d]}\]
\[\text{Out[3]}= 38909.5\]
In[4]:= n // InputForm
Out[4]/InputForm= 38909.54778576389

In[5]:= ExcelForm[n, "ddd, mmm d, yyyy"]

In[6]:= ExcelForm[n, "h:mm AM/PM"]
Out[6]= 1:08 PM

In[7]:= ExcelDate[n]

ExcelDelete

ExcelDelete[sheet]
deletes the specified sheet.

ExcelDelete[shape]
deletes the specified shape.

Examples

In[1]:= <<ExcelLink

In[2]:= ExcelSheets[]

In[3]:= ExcelInsert["Sheet"]
Out[3]= -Sheet: Sheet4-

In[4]:= ExcelSheets[]

In[5]:= ExcelDelete[ExcelSheet[-1]]

In[6]:= ExcelSheets[]

ExcelDialog

ExcelDialog[type]
displays a dialog specified by type.

ExcelDialog[type, title]
displays a dialog with a custom title.
Details

- Supported dialog types are given by $\$ExcelDialogs$.

Examples

\[
\begin{align*}
\text{In[1]:=} & \quad \text{\textless ExcelLink\textgreater} \\
\text{In[2]:=} & \quad \$\text{ExcelDialogs} \\
\text{Out[2]=} & \quad \{\text{Range, Open, Save, Files, Folder}\} \\
\text{In[3]:=} & \quad \text{ExcelDialog["Range"]} \\
\text{Out[3]=} & \quad \text{Range: B2:C6} \\
\text{In[4]:=} & \quad \text{ExcelDialog["Open"]} \\
\text{Out[4]=} & \quad \text{C:\Documents and Settings\Anton\Desktop\Excel Data Types.xls} \\
\text{In[5]:=} & \quad \text{ExcelDialog["Save"]} \\
\text{Out[5]=} & \quad \text{C:\Documents and Settings\Anton\Desktop\New File.xls} \\
\text{In[6]:=} & \quad \text{ExcelDialog["Files"]} \\
\text{Out[6]=} & \quad \{\text{C:\Documents and Settings\Anton\Desktop\Manager.xls,} \\
& \quad \quad \text{C:\Documents and Settings\Anton\Desktop\Temp.xls,} \\
& \quad \quad \text{C:\Documents and Settings\Anton\Desktop\Test.xls}\} \\
\text{In[7]:=} & \quad \text{ExcelDialog["Folder"]} \\
\text{Out[7]=} & \quad \text{C:\Documents and Settings\Anton\My Documents}
\end{align*}
\]

ExcelDirectory

\[
\begin{align*}
\text{ExcelDirectory[\textit{name}]}
& \quad \text{gives a directory specified by name.} \\
\text{ExcelDirectory[\textit{book}]}
& \quad \text{gives the directory of the specified book.}
\end{align*}
\]

Details

- Supported named directories are given by $\$ExcelDirectories$.

- If \textit{book} is unsaved, its directory is given as None.

Examples

\[
\begin{align*}
\text{In[1]:=} & \quad \text{\textless ExcelLink\textgreater} \\
\text{In[2]:=} & \quad \text{ExcelDirectory["Installation"]} \\
\text{Out[2]=} & \quad \text{C:\Program Files\Microsoft Office\Office10} \\
\text{In[3]:=} & \quad \text{s = ToFileName[{ExcelDirectory["Link"], "Examples"}, "Stocks.xls"]} \\
\text{Out[3]=} & \quad \text{C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Examples\Stocks.xls}
\end{align*}
\]
In[4]:= book = ExcelOpen[s]

In[5]:= ExcelDirectory[book]
Out[5]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Examples

In[6]:= ExcelClose[book]

ExcelFilter

ExcelFilter[range, {filter1, filter2, ...}]
applies filters to header fields in range.

ExcelFilter[range, None]
turns off all filters for the specified range.

Details

- The first filter is applied to the first field, the second filter to the second field, and so on.
- Filters can be value strings such as "Q4", or comparison strings such as ">100".
- If a filter is All, no filter is applied to the field at that position.

Examples

In[1]:= <<ExcelLink

In[2]:= s = ToFileName[{ExcelDirectory["Link"], "Examples"}, "Cities.xls"]
Out[2]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Examples\Cities.xls

In[3]:= ExcelOpen[s]

In[4]:= rng = ExcelRange["USA", All]
Out[4]= Range: All

In[5]:= ExcelFilter[rng, {All, "Nevada"}]

In[6]:= ExcelRead[rng]

In[7]:= ExcelFilter[rng, None]

In[8]:= ExcelClose[s]
**ExcelFormat**

<table>
<thead>
<tr>
<th>ExcelFormat[target]</th>
<th>returns the format of target.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExcelFormat[target, format]</td>
<td>sets the format of the target.</td>
</tr>
</tbody>
</table>

**Details**

- The target can be a range or a sheet.
- The format can be "General", "Text", or a number format such as "$ #,###.00", "0.0%", or "yyyy-mm-dd hh:mm:ss".
- The format can also be "AutoFit" to adjust column widths in target to display all contents.
- If multiple formats exist in target, ExcelFormat[target] returns "Mixed".

**Examples**

```plaintext
In[1]:= <<ExcelLink`
In[2]:= ExcelFormat["A1"]
In[3]:= ExcelWrite["A1", .5]
In[4]:= ExcelFormat["A1","0.0%"]
In[5]:= ExcelFormat["A1"]
Out[5]= 0.0%
In[6]:= ExcelFormat["A1", "hh:mm"]
In[7]:= ExcelFormat["A1"]
Out[7]= hh:mm
In[8]:= ExcelFormat["A1", "General"]
In[9]:= ExcelFormat["A1"]
In[10]:= ExcelClear["A1"]
```
**ExcelForm**

ExcelForm[expr]

prints expr as it would appear in Excel in "General" format.

ExcelForm[expr, format]

prints expr as it would appear in the specified format.

**Details**

- The format can be specified using special characters such as "$ #,###.00", "0.0%", or "yyyy-mm-dd hh:mm:ss".

**Examples**

In[1]:=<ExcelLink`

In[2]:= v = {3., 1.5*^-15, 1/2, 3 + 5 I, "Hello", Empty, True, False}

Out[2]= {3., 1.5 \times 10^{-15}, \frac{1}{2}, 3 + 5 i, Hello, Empty, True, False}

In[3]:= Map[ExcelForm, v]

Out[3]= {3, 1.5E-15, 0.5, 3 + 5I, Hello, , TRUE, FALSE}

In[4]:= n = ExcelDate[Date[]]

Out[4]= 38909.6

In[5]:= ExcelForm[n, "dd-mm-yyyy hh:mm AM/PM"]

Out[5]= 11-Jul-06 01:36 PM

In[6]:= ExcelForm[5/7, "0.0%"]

Out[6]= 71.4%

In[7]:= ExcelForm[12345, "n,#### e"]

Out[7]= 12,345 e

**ExcelGraphic**

ExcelGraphic[graphic, opts, ...]

specifies options for how graphic should be displayed in Excel.

**Details**

- Excel [id] = ExcelGraphic[graphic, opts, ...] displays graphic with specified options.
- Excel [id] = graphic displays graphic using default options.
- Options ExcelGraphic gives a list of available options.
- SetOptions ExcelGraphic, opts, ... sets the defaults.
Examples

\[\text{In}[1]:= \text{<<ExcelLink}}\]

\[\text{In}[2]:= g = \text{Plot}[\sin(x), \{x, -\pi, \pi\}]\]

\[\text{Out}[2]=\]

\[\text{In}[3]:= \text{Excel["B3"] = g}\]

\[\text{In}[4]:= \text{Options[ExcelGraphic]}\]

\[\text{Out}[4]=\{\text{ImageSize \to Automatic, ImageMargins \to Automatic, ImageFormat \to Automatic, ImageResolution \to Automatic, TextStyle \to Automatic}\}\]

\[\text{In}[5]:= \text{Excel["B3"] = ExcelGraphic[g, ImageFormat \to \text{"GIF"}, ImageSize \to \{400, 300\}, ImageMargins \to 20, TextStyle \to \{\text{FontFamily \to \text{"Arial"}, FontSize \to 9}\}]}\]

\[\text{In}[6]:= \text{Excel["B3"] = .}\]

**ExcelInsert**

<table>
<thead>
<tr>
<th>ExcelInsert[]</th>
<th>inserts a sheet in the active book.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExcelInsert[name]</td>
<td>inserts a sheet with the specified name.</td>
</tr>
<tr>
<td>ExcelInsert[ExcelSheet [id]]</td>
<td>inserts the specified sheet.</td>
</tr>
<tr>
<td>ExcelInsert[ExcelShape [id]]</td>
<td>inserts the specified shape.</td>
</tr>
</tbody>
</table>

**Details**

- The id can be a "name" or Automatic.
- Option Position can be used to specify the position where the new object is to be inserted.
- For sheets, Position can be an existing sheet name or index.
- For shapes, Position can be a \{top, left\} coordinate pair.
- Option Format can be used to specify the format of the inserted object.
- For sheets, Format can be "Work", "Chart", or a reference to an existing sheet to use as a template.
- For shapes, Format can be "Image", "Text", or a reference to an existing shape to use as a template.
Examples

\begin{itemize}
  \item In[1]:= \texttt{\textless ExcelLink\textgreater }
  \item In[2]:= \texttt{ExcelSheets[]}
    \item Out[2]= \{\texttt{Sheet: Sheet1-}, \texttt{Sheet: Sheet2-}, \texttt{Sheet: Sheet3-}\}
  \item In[3]:= \texttt{ExcelInsert[]}
    \item Out[3]= \{\texttt{Sheet: Sheet4-}\}
  \item In[4]:= \texttt{ExcelSheets[]}
    \item Out[4]= \{\texttt{Sheet: Sheet1-}, \texttt{Sheet: Sheet2-}, \texttt{Sheet: Sheet3-}, \texttt{Sheet: Sheet4-}\}
  \item In[5]:= \texttt{ExcelDelete[ExcelSheet[-1]]}
    \item In[6]:= \texttt{ExcelSheets[]}
      \item Out[6]= \{\texttt{Sheet: Sheet1-}, \texttt{Sheet: Sheet2-}, \texttt{Sheet: Sheet3-}\}
  \item In[7]:= \texttt{ExcelInsert["New", Position \rightarrow 1]}
    \item Out[7]= \{\texttt{Sheet: New-}\}
  \item In[8]:= \texttt{ExcelSheets[]}
    \item Out[8]= \{\texttt{Sheet: New-}, \texttt{Sheet: Sheet1-}, \texttt{Sheet: Sheet2-}, \texttt{Sheet: Sheet3-}\}
  \item In[9]:= \texttt{ExcelDelete[ExcelSheet[1]]}
    \item In[10]:= \texttt{ExcelSheets[]}
      \item Out[10]= \{\texttt{Sheet: Sheet1-}, \texttt{Sheet: Sheet2-}, \texttt{Sheet: Sheet3-}\}
\end{itemize}

\textbf{ExcelInstall}

\begin{itemize}
  \item ExcelInstall[] \textbf{starts communications with Excel.}
\end{itemize}

Details

- Option \texttt{Visible} specifies if a visible instance of Excel is used. The default is \texttt{Visible \rightarrow Automatic}.  
- If \texttt{Visible} is \texttt{Automatic}, a visible instance of Excel must already be open to start communications.  
- If \texttt{Visible} is \texttt{False}, a hidden instance of Excel is launched for private use by \texttt{Mathematica}.  
- By default, \texttt{ExcelInstall[]} is called the first time communication with Excel is required.

Examples

\begin{itemize}
  \item In[1]:= \texttt{\textless ExcelLink\textgreater }
  \item In[2]:= \texttt{ExcelInstall[Visible \rightarrow True]}
    \item Out[2]= \texttt{LinkObject[C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 2, 2]}
\end{itemize}
ExcelNew[]

Book: Book1

ExcelBooks[]

{Book: Book1}

ExcelClose[]

ExcelUninstall[]

ExcelInstall[Visible -> False]

LinkObject[C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 3, 2]

ExcelNew[]

Book: Book1

ExcelBooks[]

{Book: Book1}

ExcelUninstall[]

ExcelName

ExcelName[object]

returns the name of the specified object.

Examples

<<ExcelLink`

objects = ExcelSheets[]

{Sheet: Sheet1, Sheet: Sheet2, Sheet: Sheet3}

ExcelName /@ objects

{Sheet1, Sheet2, Sheet3}
Excel

\textbf{Excel[\textit{id}]} identifies a location to read or write from in Excel.

\textbf{Excel[\textit{id}]} reads the contents of \textit{id}.

\textbf{Excel[\textit{id}]} = \textit{expr} writes \textit{expr} to \textit{id}.

\textbf{Excel[\textit{id}]} = . clears the contents of \textit{id}.

Details

- The location identified by \textit{id} can be a range, shape, or sheet.
- The \textit{id} can be a name, A1-style address, All, Selection, or an Excel object reference.
- Assigning graphics, typeset equations, or formatted output to a location displays a graphic.
- \textbf{Excel[\textit{id}]} is equivalent to \textbf{ExcelRead [\textit{id}]}.
- \textbf{Excel[\textit{id}]} = \textit{expr} is equivalent to \textbf{ExcelWrite [\textit{id}, \textit{expr}]}.
- \textbf{Excel[\textit{id}]} = . is equivalent to \textbf{ExcelClear [\textit{id}]}.
- \textbf{ExcelRange}, \textbf{ExcelShape}, or \textbf{ExcelSheet} can be used to create an Excel object reference.
- Excel object references can be used to provide a specific context for \textit{id} or to specify a different type of \textit{id}.

Examples

\begin{verbatim}
In[1]:= <<ExcelLink`  
In[2]:= Excel["A1:C3"] = IdentityMatrix[3]  
In[3]:= Excel["A1:C3"]  
Out[3]= {{1., 0., 0.}, {0., 1., 0.}, {0., 0., 1.}}  
In[4]:= Excel["A1:C3"] =.  
In[5]:= data = Table[Random[], {10000}, {10}];  
In[6]:= Excel["Sheet1"] = data  
In[7]:= data = Excel["Sheet1"];  
In[8]:= Dimensions[data]  
Out[8]= {10000, 10}  
In[9]:= Excel["Sheet1"] =.  
\end{verbatim}
### ExcelNew

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExcelNew[]</td>
<td>creates a new workbook with the default number of sheets.</td>
</tr>
<tr>
<td>ExcelNew[i]</td>
<td>creates a book with ( i ) sheets.</td>
</tr>
<tr>
<td>ExcelNew[{name_1, name_2, ...}]</td>
<td>creates a book with the specified named sheets.</td>
</tr>
</tbody>
</table>

#### Details
- Template names must be specified using a full path.

#### Examples
```
In[1]:= <<ExcelLink`

In[2]:= ExcelBooks[]

In[3]:= ExcelNew[]

In[4]:= ExcelClose[]

In[5]:= v = Map[ToString, Range [2000, 2010]]

In[6]:= ExcelNew[v]

In[7]:= ExcelClose[]

In[8]:= s = ToFileName[{ExcelDirectory["Link"], "Templates"}, "Report.xls"]
Out[8]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Templates\Report.xls

In[9]:= ExcelNew[s]

In[10]:= ExcelClose[]
```
ExcelObject

```
ExcelObject[type, id]
   represents an object of the specified type identified by id in the active context.
```

```
ExcelObject[type, context] @ ExcelObject[type, id]
   provides a context for the object id.
```

Details

- All Excel objects are represented internally in this form.

Examples

```
In[1]:= <<ExcelLink`
In[2]:= object=ExcelRange["Sheet1", "A1"]
In[3]:= InputForm[object]
Out[3]/InputForm= ExcelObject["Sheet", "Sheet1"][ExcelObject["Range", "A1"]]
In[4]:= object=ExcelRange["Book1", "Sheet1", "A1"]
In[5]:= InputForm[object]
Out[5]/InputForm= ExcelObject["Book", "Book1"][ExcelObject["Sheet", "Sheet1"]][
   ExcelObject["Range", "A1"]]
```

ExcelOffset

```
ExcelOffset[range, {rows, cols}]
   offsets range by the specified number of rows and columns.
```

Details

- All specifies that range should be offset to the end of contiguous data in that dimension.
- None specifies that range should not be offset in that dimension.

Examples

```
In[1]:= <<ExcelLink`
In[2]:= ExcelOffset["A1", {5, None}] 
Out[2]= Range: A6-
In[3]:= ExcelOffset["A:A", {None, 5}] 
Out[3]= Range: F:F-
```
$$\text{In[4]:=} \text{ExcelOffset["A1:C3", \{1, 1\}]}$$


$$\text{In[5]:=} \text{ExcelWrite["A:A", Range[10]]}$$

$$\text{In[6]:=} \text{ExcelOffset["A1", \{All, None\}]}$$


$$\text{In[7]:=} \text{ExcelClear["A:A"]}$$

**ExcelOpen**


**Details**

- File name book.xls must be specified using a full path.
- If it is already open in Excel, the workbook is activated.
- Newer Excel file formats such as .xlsx, .xlsm, and .xslb files are also supported.

**Examples**

$$\text{In[1]:=} \text{\{ExcelLink\}}$$

$$\text{In[2]:=} \text{dir = ToFileName[\{ExcelDirectory["Link"], "Examples\}]}$$

$$\text{Out[2]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Examples\}$$

$$\text{In[3]:=} \text{files = FileNames["*.xls", dir];}$$

$$\text{In[4]:=} \text{books = ExcelOpen /@ files}$$


$$\text{In[5]:=} \text{ExcelClose /@ books}$$


**ExcelOutput**

ExcelOutput[expr, opts] specifies options for how expr should be output in Excel.

**Details**

- Excel [id] = ExcelOutput[expr, opts, ...] displays expr with specified options.
- Options[ExcelOutput] gives a list of available options.


Examples

In[1]:= <<ExcelLink`
In[2]:= expr = Expand[(x + y)^5]
Out[2]= x^5 + 5 x^4 y + 10 x^3 y^2 + 10 x^2 y^3 + 5 x y^4 + y^5
In[3]:= Excel["B8"] = OutputForm[expr]
In[4]:= Options[ExcelOutput]
In[5]:= Excel["B8"] = ExcelOutput[expr, ImageMargins \[\rightarrow\] 10, TextStyle \[\rightarrow\] {FontSize \[\rightarrow\] 16}]
In[6]:= Excel["B8"] =.

ExcelPosition


ExcelPosition[range]
returns the position of the top–left cell of range as a \{row, col\} index pair.

Examples

In[1]:= <<ExcelLink`
In[2]:= ExcelPosition["C10"]
Out[2]= \{10, 3\}
In[3]:= ExcelPosition["A1:C10"]
Out[3]= \{1, 1\}

ExcelRange


ExcelRange[id]
represents a range identified by id in the active context.

ExcelRange[sheet, id]
represents a range in the specified sheet.

ExcelRange[book, sheet, id]
represents a range in the specified book.

Details

- The id can be one of the following:

<table>
<thead>
<tr>
<th>Selection</th>
<th>the currently selected range</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>all cells in the sheet</td>
</tr>
<tr>
<td>&quot;Data_Range&quot;</td>
<td>a defined range name</td>
</tr>
</tbody>
</table>
"A1:C10" an A1-style address

\{1,1\} a \{row, col\} index pair

\{(1,1),(10,3)\} a list of two index pairs \{top, left\}, \{bottom, right\} defining a rectangular range

- See ExcelSheet for a listing of valid id values for sheet.
- See ExcelBook for a listing of valid id values for book.

**Examples**

```
In[1]:= <<ExcelLink`

In[2]:= ExcelSelect[ExcelRange["A1:C5"]]

In[3]:= ExcelAddress[ExcelRange[Selection]]


In[4]:= ExcelAddress[ExcelRange[{1,1}]]


In[5]:= ExcelAddress[ExcelRange[{{1,1},{5,3}}]]


In[6]:= ExcelSize[ExcelRange[All]]

Out[6]= \{65536, 256\}
```

**ExcelRanges**

```
ExcelRanges[ ]
gives a list of named ranges in the active sheet.

ExcelRanges[sheet]
gives a list for the specified sheet.

ExcelRanges[book]
gives a list of all named ranges in book.
```

**Examples**

```
In[1]:= <<ExcelLink`

In[2]:= s = ToFileName[{ExcelDirectory["Link"], "Examples"}, "Stocks.xls"]

Out[2]= C:\Documents and Settings\Anton\Application Data\Mathematics\Applications\ExcelLink\Examples\Stocks.xls

In[3]:= book = ExcelOpen[s]

Out[3]= Book: Stocks.xls-

In[4]:= ExcelRanges[]

```
In[5]:= \texttt{ranges = ExcelRanges[book]}


In[6]:= \texttt{ExcelContext /\@ ranges}


In[7]:= \texttt{ExcelClose[book]}

ExcelRead

\begin{tabular}{|l|}
\hline
\textbf{ExcelRead[range]} \\
\hspace{1cm} reads data from the specified range in Excel. \\
\hline
\textbf{ExcelRead[sheet]} \\
\hspace{1cm} reads all data contained in the specified sheet. \\
\hline
\textbf{ExcelRead[shape]} \\
\hspace{1cm} reads the contents of the specified shape. \\
\hline
\end{tabular}

Details

- If \texttt{range} includes trailing empty cells, they are ignored.

Examples

\begin{itemize}
\item \texttt{in[1]:= <<ExcelLink\`}
\item \texttt{in[2]:= ExcelWrite["A1:C3", IdentityMatrix[3]]}
\item \texttt{in[3]:= ExcelRead["A1:C3"]}
\hspace{1cm} \texttt{Out[3]= \{(1., 0., 0.), (0., 1., 0.), (0., 0., 1.)\}}
\item \texttt{in[4]:= ExcelRead["A:C"]}
\hspace{1cm} \texttt{Out[4]= \{(1., 0., 0.), (0., 1., 0.), (0., 0., 1.)\}}
\item \texttt{in[5]:= ExcelRead["Sheet1"]}
\hspace{1cm} \texttt{Out[5]= \{(1., 0., 0.), (0., 1., 0.), (0., 0., 1.)\}}
\item \texttt{in[6]:= ExcelClear["A1:C3"]}
\end{itemize}
**ExcelRefresh**

`ExcelRefresh[]`
refreshes all external data queries and pivot tables in the active book.

`ExcelRefresh[book]`
refreshes the specified book.

**Details**

- External data sources can include text files, web pages, or databases.
- To set up an external data query in Excel, use the Data ▶ Import External Data commands.

**Examples**

```
In[1]:= <<ExcelLink`
In[2]:= s = ToFileName[{ExcelDirectory["Link"], "Examples"}, "Stocks.xls"]
Out[2]= C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Examples\Stocks.xls

In[3]:= ExcelOpen[s]
Out[3]= -Book: Stocks.xls-

In[4]:= ExcelRefresh[]

In[5]:= data = ExcelRead /* ExcelSheets[];
In[6]:= ExcelClose[s]
```

**ExcelRename**

`ExcelRename[sheet, name]`
renames the specified sheet.

`ExcelRename[shape, name]`
renames the specified shape.

**Examples**

```
In[1]:= <<ExcelLink`
In[2]:= sheets = ExcelSheets[]

In[3]:= ExcelRename["Sheet1", "Summary"]
Out[3]= -Sheet: Summary-

In[4]:= sheets = ExcelSheets[]
```
In[5]:= ExcelRename["Summary", "Sheet1"]
Out[5]= -Sheet: Sheet1-

In[6]:= sheets = ExcelSheets[]

**ExcelResize**

ExcelResize[range, {rows, cols}]
resizes range to the specified number of rows and columns.

ExcelResize[range, All]
resizes range to include all contiguous data.

**Details**

- None specifies that range should not be resized in that dimension.
- All specifies that range should be extended to include all contiguous data in that dimension.

**Examples**

In[1]:= <<ExcelLink`
In[2]:= ExcelResize["A1", {10, None}]

In[3]:= ExcelResize["A1", {10, 10}]

In[4]:= data = Table[Random[], {10}, {3}];
In[5]:= ExcelWrite["A1:C10", data]
In[6]:= ExcelResize["A1", All]

In[7]:= ExcelResize["A1", {All, None}]

In[8]:= ExcelClear["A1:C10"]
ExcelResult

ExcelResult
is an internal function used to return lengthy results to Excel.

Details

- Options [ExcelResult] gives a list of options that affect how results are returned.
- SetOptions [ExcelResult, opts, ...] can be used to specify these options.

Examples

In[1]:= <<ExcelLink``
In[2]:= Options[ExcelResult]
Out[2]= {MaxCharacters -> 255, NumberMarks -> False}

In[3]:= expr = Expand[(x + .8)^9]
Out[3]= 0.134218 + 1.50995 x + 7.54975 x^2 + 22.0201 x^3 + 41.2877 x^4 + 43.008 x^5 + 23.04 x^7 + 7.2 x^8 + x^9

In[4]:= StringLength[ToString[InputForm[expr]]]
Out[4]= 205

In[5]:= ExcelForm[expr]

In[6]:= SetOptions[ExcelResult, MaxCharacters -> 100]
Out[6]= {MaxCharacters -> 100, NumberMarks -> False}

In[7]:= ExcelForm[expr]

In[8]:= SetOptions[ExcelResult, MaxCharacters -> 255]
Out[8]= {MaxCharacters -> 255, NumberMarks -> False}

In[9]:= SetOptions[ExcelResult, NumberMarks -> True]
Out[9]= {MaxCharacters -> 255, NumberMarks -> True}

In[10]:= ExcelForm[expr]
Out[10]= 0.1342177280600000006` + 1.5099494400000009` x + 7.549747200000004` x^2 + 22.02009600000001` x^3 + 41.287680000000016` x^4 + 51.609600000000015` x^5 + 43.00800000000001` x^6 + 23.040000000000006` x^7 + 7.2` x^8 + x^9

In[11]:= SetOptions[ExcelResult, NumberMarks -> False]
ExcelRun

\texttt{ExcelRun[\textit{macro}]}

runs the specified VBA \textit{macro} in Excel.

\texttt{ExcelRun[\textit{macro}, \textit{arg}_1, \textit{arg}_2, ...]}

runs the \textit{macro} with the specified arguments.

Examples

If the ExcelLink add-in is currently loaded in Excel, this displays the Mathematica Clipboard window.

\begin{verbatim}
In[1]:= <<ExcelLink`
In[2]:= ExcelRun["MathematicaClipboard"]
\end{verbatim}

ExcelSave

\texttt{ExcelSave[]} saves changes to the active workbook.

\texttt{ExcelSave[\textit{book}]} saves changes to the specified \textit{book}.


Details

- If \textit{book} has never been saved, a file name must be specified.
- File name \textit{book}.xls must be specified using a full path.
- Newer Excel file formats such as .xlsx, .xlsm, and .xlsb files are also supported.

Examples

\begin{verbatim}
In[1]:= <<ExcelLink`
In[2]:= book = ExcelNew[\{"Data"\}]
Out[2]= Book: Book2-
In[3]:= ExcelWrite[\"Data\", Table[Random[], \{100\}, \{10\}]]
In[4]:= s = ToFileName[ExcelDirectory[\"User\", \"Temporary.xls\"]]
Out[4]= C:\Documents and Settings\Anton\Desktop\Temporary.xls
In[5]:= book = ExcelSave[book, s]
Out[5]= Book: Temporary.xls-
In[6]:= ExcelClose[book]
In[7]:= DeleteFile[s]
\end{verbatim}
**ExcelSelect**

`ExcelSelect[range]`
- selects the specified `range`.

`ExcelSelect[shape]`
- selects the specified `shape`.

**Examples**

```
In[1]:= <<ExcelLink

In[2]:= ExcelContext[]
Out[2]= {Book1, Sheet1}

In[3]:= ExcelSelect[ExcelRange["Sheet3", "B2:D5"]]
Out[4]= {Book1, Sheet3}

In[5]:= ExcelAddress[Selection]

In[6]:= ExcelSelect[ExcelRange["Sheet1", "A1"]]
```

**ExcelShape**

`ExcelShape[id]`
- represents a shape identified by `id` in the active context.

`ExcelShape[sheet, id]`
- represents a shape in the specified `sheet`.

`ExcelShape[book, sheet, id]`
- represents a shape in the specified `book`.

**Details**

- The `id` can be one of the following:

<table>
<thead>
<tr>
<th>Selection</th>
<th>the currently selected shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Rectangle 1&quot;</td>
<td>the name of the shape</td>
</tr>
<tr>
<td>1 or -1</td>
<td>a positive or negative position index</td>
</tr>
</tbody>
</table>

- The first shape inserted on a sheet has position 1. The last shape inserted on a sheet has position -1.

- See `ExcelSheet` for a listing of valid `id` values for `sheet`.

- See `ExcelBook` for a listing of valid `id` values for `book`. 

Examples

\[
\begin{align*}
\text{In[1]:= } & \text{ <<ExcelLink`}
\text{In[2]:= } & \text{ shape1 = ExcelInsert[ExcelShape["Rectangle 1"]]};
\text{In[3]:= } & \text{ shape2 = ExcelInsert[ExcelShape["Rectangle 2"]];}
\text{In[4]:= } & \text{ ExcelShapes[]}
\text{Out[4]= } & \{-\text{Shape: Rectangle 1}, -\text{Shape: Rectangle 2}\}
\text{In[5]:= } & \text{ ExcelName[ExcelShape[1]]}
\text{Out[5]= } & \text{Rectangle 1}
\text{In[6]:= } & \text{ ExcelName[ExcelShape[-1]]}
\text{Out[6]= } & \text{Rectangle 2}
\text{In[7]:= } & \text{ ExcelSelect[ExcelShape[1]]}
\text{In[8]:= } & \text{ ExcelName[ExcelShape[Selection]]}
\text{Out[8]= } & \text{Rectangle 1}
\text{In[9]:= } & \text{ ExcelCheck[ExcelShape[5]]}
\text{Out[9]= } & \text{False}
\text{In[10]:= } & \text{ ExcelDelete[shape1]}
\text{In[11]:= } & \text{ ExcelDelete[shape2]}
\end{align*}
\]

\textbf{ExcelShapes}

\begin{itemize}
\item \textbf{ExcelShapes[ ]} gives a list of supported shapes in the active sheet.
\item \textbf{ExcelShapes[\textit{sheet}]} gives a list for the specified \textit{sheet}.
\item \textbf{ExcelShapes[\textit{book}]} gives a list of all supported shapes in \textit{book}.
\end{itemize}

Examples

\[
\begin{align*}
\text{In[1]:= } & \text{ <<ExcelLink`}
\text{In[2]:= } & \text{ ExcelShapes[]}
\text{Out[2]= } & \{}
\text{In[3]:= } & \text{ shape = ExcelInsert[ExcelShape["Rectangle 1"]]}\]
\text{Out[3]= } & \{-\text{Shape: Rectangle 1}\}
\text{In[4]:= } & \text{ ExcelShapes[]}
\text{Out[4]= } & \{-\text{Shape: Rectangle 1}\}
\text{In[5]:= } & \text{ ExcelDelete[shape]}
\end{align*}
\]
ExcelShare

\[ \text{ExcelShare[] \text{ starts kernel sharing with Excel on a link named "ExcelShare".}} \]

Details

- You must have the Mathematica Link add-in loaded in Excel to share a kernel with Excel.
- \text{ExcelShare[]} must be called \textit{before} connecting from Excel.
- In Excel, if your \textit{Mathematica Connection} is set to "Automatic", the Mathematica Link add-in will first try to connect to a link named "ExcelShare" and then launch its own kernel if a link with that name is not available.
- In Excel, if your \textit{Mathematica Connection} is set to "Shared", the Mathematica Link add-in will try to connect to a link "ExcelShare" and fail if a link with that name is not available.
- Anytime you close a connection from the Excel side, you need to call \text{ExcelShare[]} again in \textit{Mathematica} before starting a new shared kernel session.
- \text{ExcelShare[]} only works in \textit{Mathematica} 6.0 or later.

Examples

\begin{verbatim}
In[1]:= <<ExcelLink

In[2]:= ExcelShare[

Once you have connected from the Excel side, you can share definitions of data between Excel and Mathematica

\text{In[3]:= x}
\text{Out[3]= 100}
\end{verbatim}

The above assumes you defined \textit{x} as 100 in Excel using a formula, macro, or the clipboard.

ExcelSheet

\[ \text{ExcelSheet[,id] \text{ represents a sheet identified by id in the active context.}} \]

\[ \text{ExcelSheet[book, id] \text{ represents a sheet in the specified book.}} \]

Details

- The \textit{id} can be one of the following:

\begin{verbatim}
Active                      the current active sheet
"Sheet1"                   the name of the sheet
1 or -1                     a positive or negative position index
\end{verbatim}

- See \texttt{ExcelBook} for a listing of valid \textit{id} values for \textit{book}.
- The first sheet in the workbook has position 1. The last sheet in the workbook has position -1.
Examples

In[1]: = <<ExcelLink`

In[2]: = ExcelSheets[]
Out[2]= {−Sheet: Sheet1−, −Sheet: Sheet2−, −Sheet: Sheet3−}

In[3]: = ExcelName[ExcelSheet[1]]
Out[3]= Sheet1

In[4]: = ExcelName[ExcelSheet[−1]]
Out[4]= Sheet3

In[5]: = ExcelName[ExcelSheet[Active]]
Out[5]= Sheet1

In[6]: = ExcelCheck[ExcelSheet[5]]

ExcelSheets

ExcelSheets[ ]
gives a list of sheets in the active workbook.

ExcelSheets[book]
gives a list for the specified book.

Examples

In[1]: = <<ExcelLink`

In[2]: = ExcelSheets[]
Out[2]= {−Sheet: Sheet1−, −Sheet: Sheet2−, −Sheet: Sheet3−}

In[3]: = sheet = ExcelInsert["Sheet"]
Out[3]= −Sheet: Sheet4−

In[4]: = ExcelSheets[]
Out[4]= {−Sheet: Sheet1−, −Sheet: Sheet2−, −Sheet: Sheet3−, −Sheet: Sheet4−}

In[5]: = ExcelDelete[sheet]
ExcelSize

\[ \text{ExcelSize}[\text{range}] \]
returns the size of range as \( \{\text{rows}, \text{cols}\} \).

Examples

In[1] := <<ExcelLink`
Out[2] := \{10, 3\}

ExcelStatus

\[ \text{ExcelStatus}[\text{text}] \]
displays text in the status bar in Excel.

\[ \text{ExcelStatus}[] \]
restores the status bar to its default state.

Examples

In[1] := <<ExcelLink`
In[2] := ExcelStatus["Performing analysis..."];
Pause[2];
ExcelStatus["Generating report..."];
Pause[1];
ExcelStatus[];

ExcelTypeset

\[ \text{ExcelTypeset}[\text{expr}, \text{opts}, \ldots] \]
specifies options for how expr should be typeset in Excel.

Details

\* Excel [id] = ExcelTypeset[expr, opts, ...] displays expr with specified options.
\* Excel [id] = form[expr] displays expr in the specified form using default options.
\* Options [ExcelTypeset] gives a list of available options.
\* SetOptions [ExcelTypeset, opts, ...] sets the defaults.

Examples

In[1] := <<ExcelLink`
In[2] := expr = Expand[(\(\alpha + \beta\) \(\) \(4\)])
Out[2] := \(\alpha^4 + 4 \alpha^3 \beta + 6 \alpha^2 \beta^2 + 4 \alpha \beta^3 + \beta^4\)
ExcelUninstall

```
ExcelUninstall[]
```

ends communications with Excel.

Details

- Option Visible specifies if visible instances of Excel should be closed when the link is uninstalled. The default is Visible → Automatic.
- If Visible is Automatic, visible instances of Excel are closed if no workbooks are open.
- If Visible is True, visible instances are always closed and unsaved changes to open workbooks are discarded.
- If Visible is False, visible instances are never closed.

Examples

```
In[1]:= <<ExcelLink`
In[2]:= ExcelInstall[Visible→False]
Out[2]= LinkObject[C:\Documents and Settings\Anton\Application
Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 2, 2]
In[3]:= $ExcelLink
Out[3]= LinkObject[C:\Documents and Settings\Anton\Application
Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 2, 2]
In[4]:= ExcelUninstall[]
In[5]:= $ExcelLink
```

ExcelUnshare

```
ExcelUnshare[]
```

ends kernel sharing with Excel.

Details

- Calling ExcelUnshare[] from a Mathematica session is not required. There is no harm in leaving the shared link open once it is established.
- ExcelUnshare[] is automatically called when Excel closes a shared kernel session.
Examples

```
In[1]:= <<ExcelLink`
In[2]:= ExcelShare[]
Out[2]= LinkObject[ExcelShare, 2, 2]

In[3]:= $ExcelShare
Out[3]= LinkObject[ExcelShare, 2, 2]

In[4]:= ExcelUninstall[]
In[5]:= $ExcelShare
```

**ExcelWrite**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ExcelWrite[range, data]</code></td>
<td>writes data to the specified range in Excel.</td>
</tr>
<tr>
<td><code>ExcelWrite[sheet, data]</code></td>
<td>fills sheet with data.</td>
</tr>
<tr>
<td><code>ExcelWrite[range, graphic]</code></td>
<td>displays a graphic at the specified range location.</td>
</tr>
<tr>
<td><code>ExcelWrite[range, form]</code></td>
<td>displays the expression form at the specified range location.</td>
</tr>
<tr>
<td><code>ExcelWrite[shape, graphic]</code></td>
<td>displays a graphic to the specified shape.</td>
</tr>
<tr>
<td><code>ExcelWrite[shape, form]</code></td>
<td>displays the expression form to the specified shape.</td>
</tr>
<tr>
<td><code>ExcelWrite[shape, text]</code></td>
<td>writes text to the specified shape.</td>
</tr>
</tbody>
</table>

**Details**

- If data does not completely fill the range, remaining cells are cleared.

**Examples**

```
In[1]:= <<ExcelLink`
In[2]:= ExcelWrite["A1", 1]
In[3]:= ExcelWrite["A1:A3", {1, 2, 3}]
In[4]:= ExcelWrite["A1:C1", {1, 2, 3}]
In[5]:= ExcelWrite["A1:C3", {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}]
In[6]:= ExcelWrite["A1:C3", {{1, 2}, {3, 4}}]
```
\[\text{In}[7]:=\text{data} = \text{Table}[\text{Random[]}, \{30\}, \{30\}]\]

\[\text{In}[8]:=\text{ExcelWrite["Sheet1", data]}\]

\[\text{In}[9]:=\text{ExcelClear["Sheet1"]}\]

\[\text{In}[10]:=\text{ExcelWrite["A1", \text{ListContourPlot[data, ContourLines} \to \text{False}]}\]

\[\text{ImageFormat}\]

\[\text{MaxCharacters}\]

\[\text{ToExcel}\]

\[\text{$ExcelDialogs$}\]

\[\text{Examples}\]

\[\text{In}[1]:= <<\text{ExcelLink}`\]
In[2]:= $ExcelDialogs
Out[2]= {Range, Open, Save, Files, Folder}

In[3]:= ExcelDialog["Folder","Select a folder"]
Out[3]= C:\Documents and Settings\Anton\My Documents

$ExcelDirectories

$ExcelDirectories gives a list of supported special directories that can be used with the ExcelDirectory function.

Details

- "Installation" is the directory where Excel is installed.
- "Home" is the default file location in Excel. This is specified in Excel under Tools ▶ Options... ▶ General.
- "User" is the directory where Excel user settings are stored.
- "Link" is the directory where Mathematica Link for Excel is installed.

Examples

In[1]:= <<ExcelLink`
In[2]:= $ExcelDirectories
Out[2]= {Installation, Home, User, Link}

In[3]:= ExcelDirectory /@ $ExcelDirectories
Out[3]= {C:\Program Files\Microsoft Office\Office10, C:\Documents and Settings\Anton\My Documents, C:\Documents and Settings\Anton\Application Data\Microsoft, C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink}

$ExcelGraphic

$ExcelGraphic is an internal counter used to display graphics in Excel.

Examples

In[1]:= <<ExcelLink`
In[2]:= $ExcelGraphic
Out[2]= 0
In[3]:= \text{g} = \text{Plot}[\sin(x), \{x, \pi, -\pi\}]

Out[3]=

In[4]:= \text{ExcelGraphic}[g]

Out[4]= \text{ExcelGraphic}[1]

In[5]:= $\text{ExcelGraphic}$

Out[5]= 1

$\text{ExcelLink}$

$\text{ExcelLink}$
gives the \text{LinkObject} being used to communicate with Excel.

\textbf{Examples}

In[1]:= <<\text{ExcelLink}`

In[2]:= $\text{ExcelLink}$

In[3]:= $\text{ExcelInstall}[]$

Out[3]= \text{LinkObject}[C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 2, 2]

In[4]:= $\text{ExcelUninstall}[]$

Out[4]= \text{LinkObject}[C:\Documents and Settings\Anton\Application Data\Mathematica\Applications\ExcelLink\Binaries\ExcelLink.exe, 2, 2]

In[5]:= $\text{ExcelUninstall}[]$

In[6]:= $\text{ExcelLink}$

$\text{ExcelOutput}$

$\text{ExcelOutput}$
is an internal counter used to display formatted output in Excel.

\textbf{Examples}

In[1]:= <<\text{ExcelLink}`

In[2]:= $\text{ExcelOutput}$

Out[2]= 0
\textbf{In[3]}:= \texttt{expr = BaseForm[10, 2]}
\texttt{Out[3]/BaseForm} = 1010_2

\textbf{In[4]}:= \texttt{ExcelForm[expr]}
\texttt{Out[4]}= \texttt{ExcelOutput[1]}

\textbf{In[5]}:= \texttt{\$ExcelOutput}
\texttt{Out[5]}= 1

\textbf{$\$ExcelResult$}

\texttt{$\$ExcelResult$}
\texttt{is an internal counter used to return lengthy results to Excel.}

\textbf{Examples}
\texttt{In[1]}:= \texttt{<<ExcelLink`}
\texttt{In[2]}:= \texttt{\$ExcelResult}
\texttt{Out[2]}= 0

\texttt{In[3]}:= \texttt{expr = Expand[(x + 2) ^ 100];}
\texttt{In[4]}:= \texttt{ExcelForm[expr]}
\texttt{Out[4]}= \texttt{ExcelResult[1]}

\texttt{In[5]}:= \texttt{\$ExcelResult}
\texttt{Out[5]}= 1

\textbf{$\$ExcelShare$}

\texttt{$\$ExcelShare$}
\texttt{gives the LinkObject being used to share a kernel session with Excel.}

\textbf{Examples}
\texttt{In[1]}:= \texttt{<<ExcelLink`}
\texttt{In[2]}= \texttt{ExcelShare[]}
\texttt{Out[2]}= \texttt{LinkObject[ExcelShare, 2, 2]}

\texttt{In[3]}= \texttt{\$ExcelShare}
\texttt{Out[3]}= \texttt{LinkObject[ExcelShare, 2, 2]}
$ExcelTypeset

$ExcelTypeset is an internal counter used to display typeset equations in Excel.

Examples

\textbf{In[1]:= } \texttt{\textbackslash \textbackslash ExcelLink`

\textbf{In[2]:= } $ExcelTypeset

\textbf{Out[2]= } 0

\textbf{In[3]:= } expr = \texttt{TraditionalForm[Expand[(x + 1/2)^5]]}

\textbf{Out[3]//TraditionalForm=}

\begin{align*}
\frac{5 x^4}{2} + \frac{5 x^3}{4} + \frac{5 x^2}{16} + \frac{5 x}{32} + 1
\end{align*}

\textbf{In[4]:= } ExcelForm[expr]


\textbf{In[5]:= } $ExcelTypeset

\textbf{Out[5]= } 1