

# *AuthorTools: A Package for Document Processing*

**Pavi Sandhu**

*AuthorTools* is an add-on package that simplifies the task of creating publication-quality books and articles from notebooks. This article reviews the features of *AuthorTools* and provides detailed information on using its functions and palettes.

## ■ Introduction

*Mathematica* notebooks are structured documents that integrate text, graphics, code, and mathematical formulas. Notebooks are portable and platform-independent and can be easily transformed to other formats, such as HTML,  $\text{\TeX}$ , XML, or PDF. These features make notebooks a popular format for creating and publishing technical documents.

Although notebooks are typically created using *Mathematica*'s front end, they can also be created and modified using kernel commands. Each notebook is completely specified by a symbolic expression. You can, therefore, select and manipulate parts of a notebook by modifying the underlying expression. This makes it possible to write programs and packages for transforming notebook documents.

The *AuthorTools* package, which was introduced in Version 4.2, provides a good illustration of *Mathematica*'s document-processing abilities. This package adds support for several features that are important in technical publishing, such as creating a table of contents or index. It greatly simplifies the task of creating publication-quality books and articles from notebooks. For example, using *AuthorTools*, you can:

- create a table of contents
- create an index
- create a Browser Categories file for adding information to the Help Browser
- find all differences between any two notebooks
- create bilaterally formatted cells for displaying examples of *Mathematica* calculations

- extract all cells of a particular type and save them in a desired format
- insert objects to display the current values of variables, such as the date, time, and filename
- set printing options, such as headers and footers

Most of these operations can be done either on a single notebook or a set of notebooks that make up a project. For example, you can generate a unified index or table of contents for a book, each chapter of which is in a separate notebook.

The *AuthorTools* package contains about sixty functions and over ten palettes. The palettes provide an easy point-and-click interface for performing most common tasks. However, advanced users might prefer to type and evaluate the functions directly, since this allows for greater control and customization. The rest of this article reviews the various functions and palettes in more detail.

## ■ Functions

### □ Introduction

Most *AuthorTools* functions require you to specify a notebook as the first argument. In general, you can specify a notebook either as a notebook object or a notebook file.

#### Notebook Objects

The *Mathematica* kernel refers to any open notebook via an expression of the form `NotebookObject[fe, id]`, where *fe* specifies the front end in which the notebook is open and *id* is a unique serial number for the notebook.

The command `Notebooks[]` returns a list of notebook objects corresponding to all notebooks currently open in the front end.

```
In[1]:= Notebook[]
```

```
Out[1]= {NotebookObject[<<AuthorToolsGuide.nb>>],
         NotebookObject[<<Messages>>]}
```

This assigns the symbol `nb` to represent the first notebook object in the list.

```
In[2]:= nb = %[[1]]
```

```
Out[2]= NotebookObject[<<AuthorToolsGuide.nb>>]
```

#### Notebook Files

Alternatively, you can identify a notebook by specifying the name and location of the notebook file. You can use the `ToFileName` function to construct a string specifying the full pathname of the file.

```
In[3]:= nb = ToFileName[{$InstallationDirectory, "AddOns", "Applications",
                        "AuthorTools", "Documentation", "English"}, "AuthorToolsGuide.nb"]
```

```
Out[3]= C:\Program Files\Wolfram Research\Mathematica\5.0\AddOns\
         Applications\AuthorTools\Documentation\English\AuthorToolsGuide.nb
```

## Loading the Package

This loads the *AuthorTools* package.

```
In[4]:= << AuthorTools`
```

You can then use the symbol `nb` as an argument to any notebook function.

```
In[5]:= NotebookName[nb]
```

```
Out[5]= AuthorToolsGuide.nb
```

## Processing Multiple Notebooks

You can use *AuthorTools* functions to operate either on a single notebook or multiple notebooks in the directory.

### To process a single notebook:

Specify the notebook as the first argument of the function you want to use.

In general, a function that acts on notebooks can accept either a notebook object or a notebook file as the first argument. The exceptions are those functions that act on the current selection in an open notebook, for example: `HorizontalInsertionPointQ`, `AddIndexEntry`, `SelectionRemoveCellTags`, or `IndexCellOnSelection`. These functions can accept a notebook object as the first argument but not a notebook file.

### To process multiple notebooks:

1. Create a project file that lists all the notebooks you want to process. (See “Creating a Project File” for more information.)
2. Specify the project file as the first argument of the function you want to use. Alternatively, load the project file using the `MakeProject` dialog box, then use the notebook object corresponding to the dialog box as the argument of the function.

In general, a function that accepts a notebook file will also accept a project file. The exceptions are those functions that are only defined for a single notebook and hence cannot be applied to a project, such as: `NotebookName`, `NotebookFolder`, `NotebookCellTags`, or `NotebookFileOptions`. These functions can accept a notebook file as the first argument but not a project file.

## □ General Notebook Manipulation

### Getting Notebook Information

This gives the full pathname of the notebook object.

```
In[6]:= NotebookFilePath[nb]
```

```
Out[6]= C:\Program Files\Wolfram Research\Mathematica\5.0\AddOns\
        Applications\AuthorTools\Documentation\English\AuthorToolsGuide.nb
```

This gives the name of the notebook.

```
In[7]:= NotebookName[nb]
```

```
Out[7]= AuthorToolsGuide.nb
```

This gives the directory in which the notebook is located.

```
In[8]:= NotebookFolder[nb]
```

```
Out[8]= C:\Program Files\Wolfram Research\Mathematica\5.0\
        AddOns\Applications\AuthorTools\Documentation\English\
```

This gives all front end options set in the notebook.

```
In[9]:= NotebookFileOptions[nb]
```

```
Out[9]= {FrontEndVersion → 5.0 for Windows,
        ScreenRectangle → {{0, 1112}, {0, 746}},
        ScreenStyleEnvironment → Brackets,
        PrintingStyleEnvironment → EnhancedPrintout,
        WindowToolbars → {}, WindowSize → {644, 719},
        WindowMargins → {{Automatic, 14}, {Automatic, 1}},
        PrintingCopies → 1, PrintingPageRange → {1, Automatic},
        PrintingOptions → {GraphicsPrintingFormat → RenderInFrontEnd},
        DefaultNewCellStyle → Text, Magnification → 1,
        StyleDefinitions → HelpBrowser.nb}
```

<code>NotebookFilePath[nb]</code>	the full path to the specified notebook
<code>NotebookName[nb]</code>	the filename of the specified notebook
<code>NotebookFolder[nb]</code>	the directory of the specified notebook

Some notebook functions.

### Using the Notebook Cache

Any notebook file created in the front end has a cache that contains information about the notebook's cell structure. The cache appears at the end of the notebook file and can be inspected by opening the notebook in a text editor.

The notebook's cache contains information on the position, style, grouping, and size of each cell. When you open an existing notebook, the front end uses the information in the cache to read into memory only those cells that need to be displayed on the screen. This reduces the memory and time required to read and render a notebook since the entire notebook does not have to be read into memory at once.

The main component of the cache is a notebook file outline. This has the same form as the notebook expression describing the notebook but with each cell expression replaced by a condensed version. Instead of the actual contents of the cell, the condensed expression contains information on the cell's position in the notebook and the number of bytes of data it contains.

Each condensed expression is of the form: `Cell[a, b, c, d, e, rules, string]`. The parameters  $a$  through  $e$  are integers and denote the following:

$a$  = stream position of the `Cell[]` expression, counting all newlines as 1 byte long

$b$  = number of newlines in the file before the `Cell[]` expression

$c$  = length of `Cell[]` in bytes, counting all newlines as 1 byte long

$d$  = number of newlines in `Cell[]`

$e$  = approximate pixel height in cell (used in determining the full height of the notebook for purposes of scaling the scrollbar thumb)

You can view the file outline in a notebook's cache using the `NotebookFileOutline` function.

```
In[10]:= NotebookFileOutline[nb]
Out[10]:= Notebook[{Cell[CellGroupData[{Cell[1727, 52, 22, 0, 108, Title],
Cell[CellGroupData[{Cell[1774, 56, 28, 0, 56, Section],
Cell[CellGroupData[{Cell[1827, 60, 34, 0, 46, Subsection],
Cell[1864, 62, 27, 0, 30, Text}], Open]],
Cell[1906, 65, 34, 0, 46, Subsection}], Open]],
Cell[1955, 68, 28, 0, 56, Section}], Open]]}]
```

The `NotebookLookup` function extracts information from the notebook file outline. With "CellOutline" as the second argument of `NotebookLookup`, you get a list of all the condensed cell expressions occurring in the notebook file outline.

```
In[11]:= NotebookLookup[nb, "CellOutline"]
Out[11]:= {Cell[1727, 52, 22, 0, 108, Title], Cell[1774, 56, 28, 0, 56, Section],
Cell[1805, 58, 36, 0, 46, Subsection],
Cell[1844, 60, 36, 0, 46, Subsection],
Cell[1895, 63, 28, 0, 56, Section], Cell[1926, 65, 28, 0, 56, Section]}
```

With "CellExpression" as the second argument of `NotebookLookup`, you get the complete cell expression for each cell in the notebook.

```
In[12]:= NotebookLookup[nb, "CellExpression"]
Out[12]:= {Cell[Title, Title], Cell[Section 1, Section],
Cell[Subsection 1.1, Subsection], Cell[Subsection 1.2, Subsection],
Cell[Section 2, Section], Cell[Section 3, Section]}
```

With "CellIndex" as the second argument of `NotebookLookup`, you get the serial number of each cell in the notebook. The first cell in the notebook has serial number 1 and so on.

```
In[13]:= NotebookLookup[nb, "CellIndex"]
Out[13]:= {1, 2, 3, 4, 5, 6}
```

The two-argument form of `NotebookLookup` simply generates a list of the form  $\{1, 2, \dots, n\}$  for a notebook containing  $n$  cells. The `NotebookLookup` command

becomes more useful if you use the optional third argument to specify a pattern. This allows you to extract information on all cells that fit the pattern. For example, the following command gives the serial number of all Section cells.

```
In[14]:= NotebookLookup[nb, "CellIndex", Cell[___, "Section", ___]]
```

```
Out[14]= {1, 14, 245, 385}
```

The following command returns the cell expressions of all Section cells.

```
In[15]:= NotebookLookup[nb, "CellExpression", Cell[___, "Section", ___]]
```

```
Out[15]= {Cell[What is AuthorTools?, Section, CellTags → What is AuthorTools?],
          Cell[Packages, Section],
          Cell[Functions List, Section], Cell[Palettes, Section]}
```

<code>NotebookFileOutline[nb]</code>	returns the notebook file outline cache for the specified notebook
<code>NotebookLookup[nb, obj]</code>	returns the objects specified by <i>obj</i> in the specified notebook
<code>NotebookLookup[nb, obj, pat]</code>	returns the objects specified by <i>obj</i> that match the pattern <i>pat</i> in the specified notebook
<code>NotebookFileOptions[nb]</code>	the front end options set in the specified notebook

Some functions that provide information about a notebook's contents.

### Adding and Removing Cell Tags

A cell tag is a string that serves as a tag or marker to identify a specific cell in a notebook. Cell tags are used, for example, to define the targets of hyperlinks or to identify the information that appears when you choose a topic in the Help Browser. *AuthorTools* includes functions for adding or removing cell tags in a notebook.

`NotebookCellTags` returns a list of all cell tags in the specified notebook. The *n*th element of the output list is a list of all cell tags for the *n*th cell in the notebook. Each cell that does not have a cell tag is represented by an empty list.

```
In[16]:= NotebookCellTags[nb]
```

```
Out[16]= {{}, {}, {}, {}, {}, {}}
```

You can add cell tags to cells in a notebook using `AddCellTags`.

You can remove cell tags from cells in a notebook using `RemoveCellTags`.

<code>AddCellTags[nb]</code>	adds the string or list of strings to the cell tags of the currently selected cell(s) in the notebook object <i>nb</i>
<code>RemoveCellTags[nb, pat]</code>	removes all cell tags from the notebook object <i>nb</i> that match the string pattern <i>pat</i>
<code>SelectionRemoveCellTags[nb, tags]</code>	removes the specified tag or list of tags from the currently selected cell(s) in the notebook object <i>nb</i>
<code>NotebookCellTags[nb]</code>	returns a list of all the cell tags present in the notebook object <i>nb</i>

Some functions for adding and removing cell tags.

## □ Processing Multiple Notebooks

### Creating a Project File

The first step in processing multiple notebooks is to create a project file. This is a file that specifies the names and location of all notebooks in a project. You will need to create a different project file for each project you are working on.

A project file is a plain text file with a `.m` suffix. The file must include data in the following format.

```
{"Name" -> name,
 "Directory" -> directory,
 "Files" -> {nb1, nb2, ...}}
```

Here, *name* is the name of the project, *directory* is the full pathname of the project directory, and *{nb1, nb2, ...}* is a list of all notebooks that make up the project.

There are several different ways to create a project file:

- Enter the project data in the text fields of the MakeProject dialog box. (See MakeProject for details.)
- Use the command `WriteProjectData[file, data]`, where *data* specifies the project data as specified in the preceding box.
- Type the project data directly into a text file and save the file with a `.m` suffix.

### Using a Project File

Once you have created a project file, you can operate on all notebooks in the project in one step. To do this you simply specify the project file as the argument to the *AuthorTools* function you want to use.

For example, to generate a table of contents for a project, you can evaluate the command, `MakeContents[projectfile,"Book"]`. The command for generating a table of contents for a single notebook is `MakeContents[nb,"Book"]`.

### Editing a Project File

When you evaluate an *AuthorTools* function with a project file as an argument, *Mathematica* uses `Get` to read in the project file. Any *Mathematica* commands present in the project file are evaluated at this time. Thus, the project file is a convenient place to store commands that you want to apply to a project.

For example, suppose you prepend the following command to the contents of your project file.

```
Needs["AuthorTools`MakeContents`"];
SetOptions[MakeContents,
  SelectedCellStyles -> {"Chapter", "Heading", "Subheading"}];
```

This sets the option `SelectedCellStyles` of the *AuthorTools* function `MakeContents`. If you then use `MakeContents` to create a table of contents for that project, the option setting specified in the project file will be automatically used, overriding the default behavior of `MakeContents`.

**Note:** Any commands you have added to a project file will be overwritten if you use the `WriteProjectData` function or the `MakeProject` dialog box to modify the project file.

### □ Creating a Table of Contents

This generates a table of contents for the notebook. The table of contents is saved in the same directory as the source notebook.

```
MakeContents[nb, "Simple"]
```

The second argument of the function specifies the format of the table of contents. You can choose from three different formats: `"Simple"`, `"Book"`, or `"BookCondensed"`. (See `MakeContents` for more information on these formats.)

**Note:** The `MakeContents` command inserts cell tags into the source notebook and automatically saves the changes. If you do not want your source notebook modified, you should keep a separate copy as a back-up.

If you are going to generate a table of contents in any format other than `Simple`, you should first use `Paginate`. This function calculates the page numbers for the specified notebook and stores them as `TaggingRules` in the notebook.

```
Paginate[nb]
```

The following command generates a table of contents in the `Book` format.

```
MakeContents[nb, "Book"]
```

The option `SelectedCellStyles` determines which cells in the notebook are included in the table of contents. The default setting is `SelectedCellStyles → {"Title", "Section", "Subsection", "Subsubsection"}`. The following command generates a table of contents in the Book format that includes only Title, Section, and Subsection cells.

```
MakeContents[nb, "Book",
  SelectedCellStyles → {"Title", "Section", "Subsection"}]
```

## □ Creating an Index

There are two steps involved in setting up an index.

1. Associate index entries with specific cells in your source notebook.

The simplest way to insert index entries is using the Edit Notebook Index dialog box, accessed from the MakeIndex palette. Alternatively, you can type and evaluate the `AddIndexEntry` function. The following command associates the specified main entry and subentry with the currently selected cell(s) in the notebook.

```
AddIndexEntry[nb, {main, sub}]
```

2. Generate the index using the `MakeIndex` function.

This generates an index for the notebook.

```
MakeIndex[nb, "Simple"]
```

The second argument of `MakeIndex` specifies the format of the index. You can choose from four different formats: "Simple", "Book", "Two-Column", or "BrowserIndex". (See `MakeIndex` for more information on these formats.)

**Note:** The `MakeIndex` command inserts cell tags into the source notebook and automatically saves the changes. If you do not want your source notebook modified, you should keep a separate copy as a back-up.

## □ Finding Differences

This generates a notebook that lists the differences between the two notebooks, *nb1* and *nb2*.

```
NotebookDiff[nb1,nb2]
```

`NotebookDiff` will also find the differences between two projects, directories, or lists of files. This generates a notebook that summarizes the differences in the two sets of notebooks.

```
NotebookDiff[project1, project2]
```

By default, `NotebookDiff` finds all possible differences between the notebooks, including differences in cell styles or options. However, you can narrow the scope of differences reported by `NotebookDiff` by specifying options. For example, this excludes cells in the Input and Output style from the diffing operation.

```
NotebookDiff[nb1,nb2, ExcludeCellsOfStyles→ {"Input", "Output"}]
```

There are several other options to `NotebookDiff`, for example, to ignore cells that have the same content but differ only in their cell style or only in their options.

To view the differences within two different cells, use `CellDiff`. This highlights the differences in content by enclosing them in colored brackets of the form: (< and >). Style and option differences are also listed.

```
CellDiff[cell1, cell2]
```

If the notebooks you are comparing are style sheets, you can either use `NotebookDiff` or a more specialized version called `StyleSheetDiff`.

```
StyleSheetDiff[nb1,nb2]
```

`NotebookDiff` and `StyleSheetDiff` are both implemented in terms of a lower-level function called `DiffReport`. This compares two generic lists, creating a report of all the insertions, deletions, and updates between the two lists.

```
DiffReport[list1, list2]
```

## □ Restoring a Notebook

### *NotebookRestore*

`NotebookRestore` takes a notebook containing one or more syntax errors and creates a new notebook containing all the cells that did not have a syntax error. For example, suppose `nb1` represents a notebook file that has been corrupted by

removing one quote from a cell style name. If you try to open this file, the front end will report a syntax error and suggest you cancel the open operation.

```
In[17]:= nbObj = NotebookOpen[nbFile]
```

```
Out[17]= $Failed
```

Using `NotebookRestore`, you can at least access those cells in the notebook that are not corrupted. `NotebookRestore` will open a new notebook window containing all the good cells from the given notebook file and insert an indicator at each place where notebook data has been deleted.

```
In[18]:= NotebookRestore[nbFile]
```

```
Out[18]= NotebookObject[<<Untitled-1>>]
```

Instead of deleting the corrupt data, you can display it verbatim within the indicator by setting the option `DeleteCorruptCells`  $\rightarrow$  `False`.

```
In[19]:= nbObj = NotebookRestore[nbFile, DeleteCorruptCells  $\rightarrow$  False]
```

```
Out[19]= NotebookObject[<<Untitled-2>>]
```

All the corrupt cell indicators have the cell tag "Corrupt", so you can easily highlight them by choosing `Corrupt` from the list of cell tags displayed under the `Find`  $\triangleright$  `Cell Tags` menu. Alternatively, you can walk through each indicator one by one using the `NextCorruptCell` function.

```
In[20]:= NextCorruptCell[nbObj]
```

```
Out[20]= NotebookSelection[NotebookObject[<<Untitled-3>>]]
```

If your notebook contains large graphics cells or large blocks of typesetting, it may take a long time for *Mathematica* to read in the expression and try to determine if it contains any syntax errors. For that reason, there are options that allow you to skip over graphics and/or typeset cells without processing them.

```
In[21]:= Options[NotebookRestore]
```

```
Out[21]= {DeleteCorruptCells  $\rightarrow$  $DeleteCorruptCells,
          DeleteGraphicsCells  $\rightarrow$  $DeleteGraphicsCells,
          DeleteTypesetCells  $\rightarrow$  $DeleteTypesetCells}
```

```
In[22]:= NotebookRestore[nbFile, DeleteTypesetCells  $\rightarrow$  True]
```

```
Out[22]= NotebookObject[<<Untitled-4>>]
```

### SalvageCells

`SalvageCells` is a noninteractive way to extract the list of cells from a corrupt notebook file. This function is intended for users who want to manipulate the cell listing directly, as part of a program. `SalvageCells` takes the same options, and returns the same information, as `NotebookRestore`.

This salvages all the cells from a notebook file.

```
In[23]:= cellList = SalvageCells[nbFile, DeleteCorruptCells  $\rightarrow$  False];
```

This gives the total number of cells.

```
In[24]:= Length[cellList]
```

```
Out[24]= 16
```

This gives the total number of blocks removed from the notebook.

```
In[25]:= Length[
    badCells = Cases[cellList, Cell[___, CellTags -> "Corrupt", ___]]]
```

```
Out[25]= 2
```

You can view just the bad data in a separate notebook.

```
In[26]:= NotebookPut[Notebook[badCells]]
```

```
Out[26]= NotebookObject[<<Untitled-5>>]
```

## □ **Pagination**

This paginates the specified notebook. If the argument is a project file, the command paginates all notebooks in the project.

```
Paginate[nb]
```

This paginates all notebooks in a project. The setting `StartingPages -> "Next"` means that the starting page for each notebook is the next page following the last page of the previous notebook.

```
Paginate[projectfile, StartingPages -> "Next"]
```

Other possible settings for `StartingPages` are "Even", "Odd", `Inherited`, or a list of integers. The following command paginates all notebooks in a project and sets the starting page for the first four notebooks to be 1, 25, 57, and 83.

```
Paginate[projectfile, StartingPages -> {1,25,57,83}]
```

The following paginates all notebooks in a project. The starting page number for each notebook in the project is inherited from the setting of the option `StartingPageNumber` for that notebook.

```
Paginate[projectfile, StartingPages -> Inherited]
```

## □ **Creating a Browser Categories File**

This generates a simple browser categories file for the notebook.

```
MakeCategories[nb]
```

The second argument of `MakeCategories` specifies the format of the `Browser-Categories.m` file. You can choose from three different formats: "Simple", "Full", or "FullNoTags".

The following command generates a full browser categories file for the notebook.

```
MakeCategories[nb, "Full"]
```

**Note:** The `MakeCategories` command inserts cell tags into the source notebook and automatically saves the changes. If you do not want your source notebook modified, you should keep a separate copy as a back-up.

### □ Creating Bilateral Cells

You can use the `MakeBilateral` function to create bilateral cells for displaying sample *Mathematica* calculations. Each bilateral cell consists of two columns with expository text to the left and input and output cells to the right. To create a bilateral cell, you must have a pair of input and output cells preceded by a cell containing text.

This is the integral of the sine function.

```
In[27]:= Integrate[Sin[x], x]
```

```
Out[27]= -Cos[x]
```

This converts the text cell and input-output pair into a bilateral cell.

```
In[28]:= MakeBilateral[nb]
```

Here is the resulting bilateral cell.

This is the integral of the

```
In[1]:= Integrate[Sin[x], x]
```

sine function.

```
Out[1]= -Cos[x]
```

The cell styles that appear in the left column of the bilateral cell are determined by `FirstBilateralStyles`, an option to `MakeBilateral`. The default setting of this option includes only one style, `MathCaption`.

The cell styles that appear in the right column of the bilateral cell are determined by `RestBilateralStyles`. The default setting includes `Input`, `Output`, and `Graphics` cell styles.

Once you have created a bilateral cell, you can reverse the process and get back the constituent cells using the following command.

```
In[29]:= DivideBilateral[nb]
```

### □ Extracting Cells from a Notebook

This exports all Section style cells to a notebook.

```
In[30]:= ExportNotebook[nb, "Section", "Notebook"]
Out[30]:= {/Applications/Mathematica 5.0.app/AddOns/
Applications/AuthorTools/Documentation/English/Sample11.nb}
```

You can also extract all Section cells by specifying a list of two elements as the second argument. `ExportNotebook[nb1, {"CellStyle", "Section"}, "Notebook"]` is equivalent to `ExportNotebook[nb1, "Section" "Notebook"]`.

```
In[31]:= ExportNotebook[nb, {"CellStyle", "Section"}, "Notebook"]
Out[31]:= {/Applications/Mathematica 5.0.app/AddOns/
Applications/AuthorTools/Documentation/English/Sample11.nb}
```

This command exports all cell groups in which the heading cell is an input cell. This enables you, for example, to extract all input-output pairs in a notebook.

```
In[32]:= ExportNotebook[nb, {"CellGroup", "Input"}, "Notebook"]
Out[32]:= {/Applications/Mathematica 5.0.app/AddOns/
Applications/AuthorTools/Documentation/English/Sample11.nb}
```

This command exports all cells having the cell tag “Reference1”.

```
In[33]:= ExportNotebook[nb, {"CellTag", "Reference1"}, "Notebook"]
Out[33]:= {/Applications/Mathematica 5.0.app/AddOns/
Applications/AuthorTools/Documentation/English/Sample11.nb}
```

This command exports all graphics cells as separate GIF files.

```
In[34]:= ExportNotebook[nb, {"CellTag", "reference1"}, "GIF"]
Out[34]:= {/Applications/Mathematica 5.0.app/AddOns/
Applications/AuthorTools/Documentation/English/Sample11.nb}
```

## □ Printing

This sets the starting page number of the specified notebook to 15.

```
SetOptions[nb, StartingPageNumber → 15]
```

This sets the printing margins in the specified notebook. The option value is a list of four numbers,  $\{l, r, b, t\}$ , which specify the value of the left, right, bottom, and top margins in printer’s points.

```
SetOptions[nb, PrintingMargins → {72,72,108,108}]
```

This modifies the notebook so the cell brackets are invisible when the notebook is printed.

```
SetOptions[nb, CellBrackets → False]
```

## ■ Palettes

### □ Introduction

The *AuthorTools* palettes provide an easy point-and-click interface to the package functions. You can use the palettes to process either a single notebook or multiple notebooks in a single directory.

#### To process a single notebook:

1. Open the notebook. If it is already open, click the notebook to select it.
2. Click a button on the palette you want to use.

#### To process multiple notebooks:

1. Open the MakeProject dialog box.
2. Click `load` and choose a project file. (If you have not already created a project file, see [Creating a Project File](#) to learn how.)
3. With the MakeProject dialog box as the input notebook, click a button on the palette you want to use. The palette functions will apply to all notebooks that belong to the specified project.

### □ OpenAuthorTools

The OpenAuthorTools palette provides a convenient way to access the other palettes in the package. Choose any topic to open the corresponding palette. Click the top button of this or any other palette to get help.

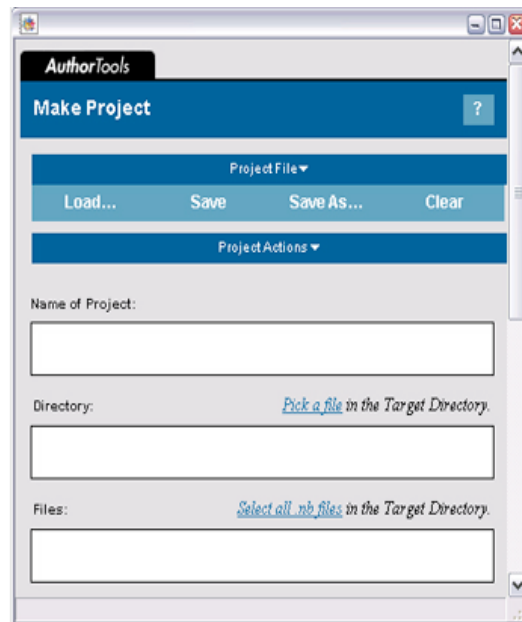


- **MakeProject**—set up and handle projects involving multiple notebooks
- **MakeIndex**—create an index
- **MakeContents**—create a table of contents
- **MakeCategories**—create a browser categories file to customize the contents of the Help Browser
- **MakeBilateralCells**—create a bilateral cell to display example calculations
- **NotebookDiff**—find differences between notebooks
- **NotebookRestore**—retrieve content from a notebook that contains a syntax error
- **Paginate**—set page numbers
- **ExportCells**—extract all cells of a specified type, such as graphics cells
- **InsertValue**—insert the value of variables such as the current date and time
- **SetPrintingOptions**—specify properties of printed pages, such as headers and footers

## □ **MakeProject**

### *Introduction*

The MakeProject dialog box makes it easy to set up and manage projects involving multiple notebooks. For example, you can use this dialog box to generate a unified table of contents or index for a set of notebooks.



### Creating a Project File

The first step in processing multiple notebooks is to create a project file. This is a file that specifies the names and location of all notebooks in a project. You can create any number of project files, one for each project you are working on. Each project file has a `.m` suffix.

#### To create a project file:

1. Open the MakeProject dialog box.
2. Type in a name for your project in the Name of Project text box.
3. Choose Target Directory. A file browse dialog box appears.
4. Select a file in the directory containing your source notebooks. The full pathname of the directory is pasted into the Directory text box.
5. Choose Select All. A list of all notebook files in the directory you specified appears in the Files text box.
6. Edit the list to delete any notebooks that do not belong in the project.
7. Choose Save under the Project File tab to save the project file.

### Handling a Project

#### To process all notebooks in a project:

1. Open the MakeProject dialog box.
2. Click Load and choose the project file you need. (If you have not already created a project file, see Creating a Project File to learn how.)
3. With the MakeProject dialog box as the input notebook, click a button on the palette you want to use. The palette functions will apply to all notebooks in the specified project.

Once you have loaded a project, you can use the buttons under the Project Actions tab to perform a variety of tasks.

- To create a table of contents in a specific style, click the corresponding button under Contents.
- To create an index in a specific style, click the corresponding button under Index.
- To create a BrowserCategories.m file, choose BrowserCategories under Other.
- To create a BrowserIndex.nb file, choose BrowserIndex under Other.
- To create page numbers to all notebooks in the project, choose Paginate under Other.

## □ **MakeContents**

### *Introduction*

The MakeContents palette enables you to generate a table of contents for any notebook or group of notebooks.



You can format your table of contents in any of three predefined styles:

- **Simple**—lists all the entries without page numbers or any special formatting. A Simple table of contents is like an outline. It is useful for checking that your topics are in the right order before generating a table of contents in one of the other two styles.
- **Book**—resembles a table of contents in a typical book, with each topic and subtopic on a separate line.
- **Condensed**—combines multiple subtopics on the same line to achieve a more compact appearance.

### **Creating a Table of Contents**

**To create a table of contents:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the MakeContents palette.
3. Choose **Paginate** to assign page numbers to the source notebooks. You can skip this step if you have paginated the notebooks previously or are creating a Simple table of contents, which does not contain any page numbers.
4. Choose either **Make Simple Contents**, **Make Book Contents**, or **Make Condensed Contents**, depending on which style of table of contents you want to create.

It may take a few moments for the evaluation to be completed, depending on the size of the source notebooks. The newly generated table of contents appears on the screen and is saved in the same directory as the source notebook.

**Note:** *Mathematica* automatically adds cell tags to the source notebooks and saves all changes. It is therefore advisable to make a back-up copy of your notebooks before generating a table of contents.

### Customizing a Table of Contents

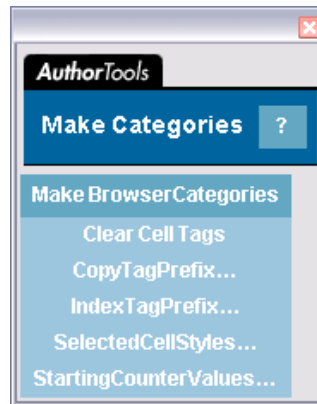
The buttons at the bottom of the palette let you specify options to customize various features of the table of contents. To set the value of an option, click the button bearing the name of that option. This brings up a dialog box displaying the current value of the option. You can edit the text and click **Apply** for the changes to take effect. Click **OK** to close the dialog box.

- **Paginate** calculates page numbers for the source notebooks. The page numbers are saved as `TaggingRules` in the notebook. By default, the first notebook has starting page number 1.
- **Clear Cell Tags** removes the automatically generated cell tags in the source notebooks. If you remove the cell tags, the hyperlinks in the table of contents will take you to the start of the notebook instead of to a specific topic.
- **CellTagPrefix** lets you specify a prefix to the automatically generated cell tags.
- **ContentsFileName** lets you specify the name of the notebook containing the table of contents.
- **SelectedCellStyles** lets you specify the cell styles that should be included in the table of contents. The default setting is `SelectedCellStyles` → `{"Title", "Section", "Subsection", "Subsubsection"}`.

## □ **MakeCategories**

### Introduction

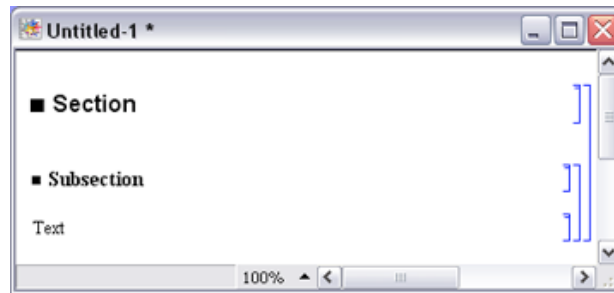
A `BrowserCategories.m` file determines how information is organized and displayed in the *Mathematica* Help Browser. The **MakeCategories** palette enables you to generate a `BrowserCategories.m` file for your notebook or project. By creating such a file, you can add your own content to the online help so it is accessible from the Help Browser.



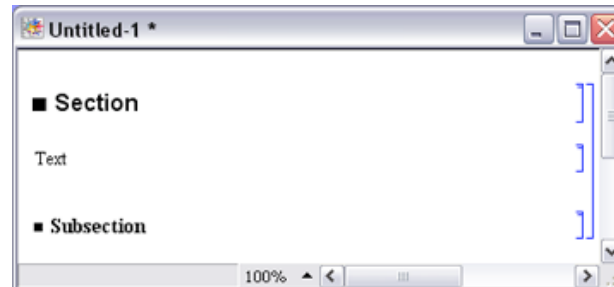
### Outline Format

When generating a browser categories file, *Mathematica* assumes the source notebook is in strict outline format. This means all cells in the notebook are arranged in a consistent hierarchical order.

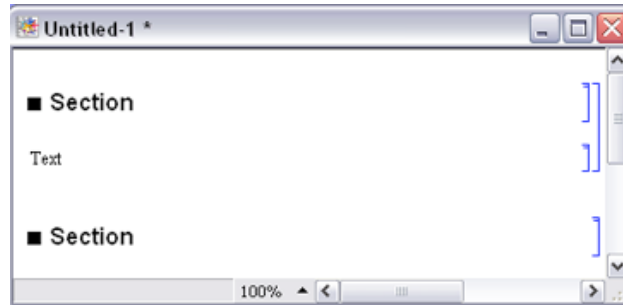
Consider a notebook composed entirely of Section, Subsection, and Text cells. If every Text cell is contained in a Subsection cell and every Subsection cell is contained in a Section cell, as shown here, the notebook is said to be in strict outline format.



However, if the following sequence of cells occurs anywhere in the notebook, the hierarchical structure breaks down. This is because in the resulting browser categories file, the Section cell will have to serve both as an item (to display the Text cell) and a category (to contain the Subsection cell), which is not possible.



Note that if a Section cell contains only Text cells and no Subsection cells as shown here, the strict outline format is again restored, since now each Section cell can serve as an item.



**Note:** If you attempt to generate a browser categories file from a notebook that is not in strict outline format, you will get a warning message. The resulting browser categories file will ignore the presence of any cells that stray from the strict outline format, so those cells will not be viewable in the Help Browser.

### Creating Browser Categories

To generate a browser categories file:

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the MakeCategories palette.
3. Choose Make Browser Categories.

A browser categories file is created and saved in the same directory as your source notebook. Cell tags are automatically added to the input notebook.

**Note:** When you generate a browser categories file, your source notebooks are modified and all changes are automatically saved. It is therefore advisable to make a back-up copy of your notebook before using this palette.

The BrowserCategories palette automatically inserts cell tags into the target notebook. To remove the automatically generated cell tags in the source notebook(s), click Clear Cell Tags. If you remove the cell tags, the hyperlinks in the table of contents will take you to the start of the notebook instead of to a specific topic.

### Customizing the Browser Categories

The three buttons at the bottom of the palette enable you to specify options to customize various features of the BrowserCategories.m file. To set the value of an option, click the button bearing the name of that option. This brings up a dialog box displaying the current value of the option. You can edit the text and click Apply for the changes to take effect. Click OK to close the dialog box.

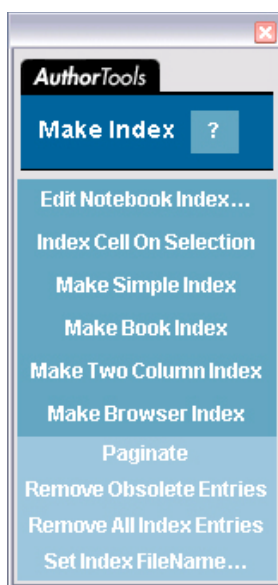
Here are the options:

- `CopyTagPrefix` lets you specify a string that is prepended to all copy tags in the `BrowserCategories.m` file. The default setting for this option is `CopyTagPrefix` → "b: ".
- `IndexTagPrefix` lets you specify a string that is prepended to all index tags in the `BrowserCategories.m` file. The purpose of this option is to make the index tags unique among all applications installed in the Help Browser. If set to `Automatic`, `IndexTagPrefix` defaults to the setting for `ProjectName`.
- `SelectedCellStyles` lets you specify the cell styles that are used in creating the browser categories file. Possible settings are `Automatic` or a list of cell styles. The default setting for this option is `SelectedCellStyles` → {`Title`, `Section`, `Subsection`, `Subsubsection`}.
- `StartingCounterValues` lets you specify the autonumbering of chapters, sections, subsections, and so on. A setting of `StartingCounterValues` → {1,0,1,0}, for example, will number the chapters starting at 1, the sections starting at 0, the subsections starting at 1, and so on. The default setting for the option, `StartingCounterValues` → {0,0,0,0}, numbers everything starting at 0.

## □ **MakeIndex**

### *Introduction*

The `MakeIndex` palette enables you to generate an index for your notebook or project. *Mathematica* calculates page numbers for all the index entries you specify and lists them in alphabetical order. Each entry in the index is hyperlinked to the related material in the source notebooks.



To create an index for a document, you must assign an index entry to each cell in the notebook that will be referenced in the index. This is done using the Edit Notebook Index dialog box of the MakeIndex palette.

Once all the index entries have been defined, you can use the palette to generate the index. For any document, you can generate several different types of indexes.

### Simple Index

A Simple Index is used to check that all the index entries you specified were entered correctly. It lists all index entries, including duplicates and subentries, on separate lines. The text of the hyperlink for each entry consists of the cell tag of the target material and the name of the notebook in which it occurs.

The Simple Index is saved in the same directory as the source notebooks. By default the file is called SimpleIndex.nb, but you can specify a different name using the Set Index FileName button at the bottom of the palette.

### Book Index

A Book Index contains all the index entries formatted with the following features.

- Duplicate index entries to the same page are removed. Duplicate index entries on consecutive pages in the same notebook are combined into a single page range.
- Subindex entries are formatted on a separate line and indented as shown here.
  - element, 2
  - Fuzzy Logic Pack, 1
    - loading, 1-2
  - Fuzzy Sets, 1, 3
    - as functions, 1
    - as models, 4
- The text of each hyperlink is the page number(s) on which the index entry appears.

If the Book index builds without error, it is saved in the same directory as the source notebooks. By default the file is called BookIndex.nb, but you can specify a different name using the Set Index FileName button at the bottom of the palette.

### Two-Column Index

A two-column index has the same features as a Book Index, except that the entries are arranged in two columns.

If the Two-Column Index builds without error, it is saved in the same directory as the source notebooks. By default the file is called TwoColumnIndex.nb, but you can specify a different name using the Set Index FileName button at the bottom of the palette.

## Browser Index

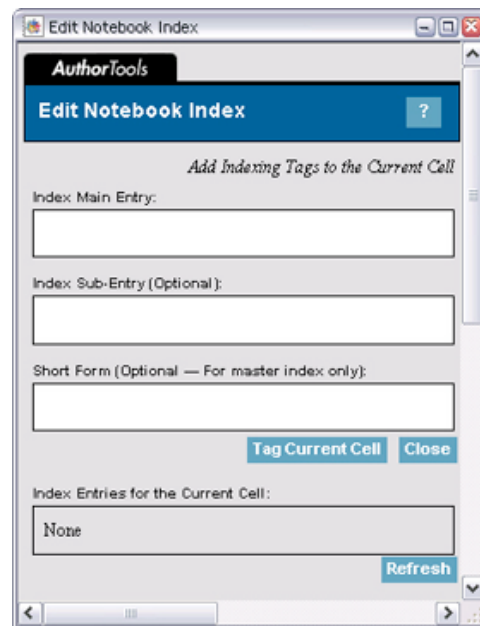
A browser index file defines a list of keywords to be included in the Master Index of the Help Browser. By creating a browser index for your notebooks, you can integrate your own reference material into the existing online help.

For the `BrowserIndex.nb` file to be compiled without error, an appropriate browser categories file must exist in the same directory as the source notebooks. To create a browser categories file, you can use the `MakeCategories` palette included with *AuthorTools*.

## Setting Up an Index

### Setting Up Index Entries

To create an index, you must associate index entries with specific cells in your source notebooks. This is done using the Edit Notebook Index dialog box, which allows you to add, edit, and remove index entries.



This dialog box contains three text boxes:

**Index Main Entry**—This specifies a term in the index.

**Index Sub-Entry**—This is useful for distinguishing two cells that fall under the same main entry.

**Short Form**—This defines a short word or phrase to represent the entry in the Master Index of the Help Browser.

### To associate an index entry with a cell (or group of cells):

1. Select the cell(s).

2. Specify the entry by typing it into the text fields of the dialog box. The main entry and subentry can be either a string or a two-dimensional expression, but the short form must be a string.
3. Choose **Tag Current Cell** to assign the entry to the selected cell(s).

Any index entries associated with a cell are displayed in the box at the bottom of the dialog box, if you select that cell. After you add a new entry, you can choose **Update Tag List** to check that the entry you specified has been assigned.

**Note:** Alternatively, you can select all or part of a cell's contents and click the **Index Cell on Selection** button. This makes the index entry for the cell the same as the selected text.

**To edit or remove an index entry:**

1. Select the cell. A list of the currently assigned entries associated with that cell is displayed at the bottom of the dialog box. Each entry also has hyperlinks marked **Edit** and **Remove** next to it.
2. Click **Edit** to edit the entry or **Remove** to remove it.
3. Edit the entries displayed in the text fields of the dialog box.
4. Choose **Tag Current Cell**. This replaces the entry you edited with the new entry.

**To generate an index:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the **MakeProject** dialog box and load the project file for your project. (If you have not already created a project file, see **Creating a Project File** to learn how.)
2. Open the **MakeIndex** palette.
3. Click **Edit Notebook Index**. This brings up a dialog box.
4. Use this dialog box to associate index entries with all cells in the notebook that are to be referenced in the index.
5. Choose **Paginate** to assign page numbers to the source notebooks. You can skip this step if you have paginated the notebooks previously or if you are creating a **Simple** index, which does not contain any page numbers.
6. Click the button corresponding to the type of index you want to create. For example, to build a **Simple** index choose **Make Simple Index**. The other formats available are **Book Index**, **Two Column Index**, and **Browser Index**. (See **MakeIndex** to learn about the differences among these four types of index.)

**Note:** *Mathematica* automatically creates an index based on all the index entries that you defined. The index is saved as a separate notebook in the same directory as the source notebook.

## Browser Index

When you choose an entry in a browser index, the target cells are displayed in the Help Browser. For the index entries to work properly, an appropriate browser categories file must be present in the same directory as the source notebooks.

If you associate an index entry with a cell that cannot be displayed in the Help Browser, the index entry will bear the label “match not found” to indicate that the entry does not have a well-defined target.

For a cell to be displayed in the Help Browser, the browser categories file must contain an item corresponding to that cell. If no such item is present, the cells are not the target of a terminal category in the columns of the Help Browser and so the index entry cannot link to those cells.

This can happen if the source notebook is not in strict outline format and the target cells fall outside the outline. Even if the source notebook is in outline format but the target cell is a heading cell that does not correspond to an item in the browser categories file, the index entry will not work.

### To fix an index entry that does not have a match:

1. Edit the browser categories file by hand to insert an item corresponding to the target cells for the index entry. (For details of how browser categories files work, see *Creating a Browser Categories File*.)
2. Use the MakeIndex palette to regenerate the browser index.

## □ Paginate

### Introduction

The Paginate palette enables you to assign page numbers to one or more notebooks. You must paginate your source notebooks at least once before you generate a table of contents or index. Otherwise, the table of contents or index will not contain any page numbers.



## Paginating Notebooks

### To paginate one or more notebooks:

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the Paginate palette.
3. Choose Paginate.

### Options for Pagination

The three buttons at the bottom of the palette enable you to specify options that control how the page numbers are assigned. To set the value of an option, click the button bearing the name of that option. This brings up a dialog box displaying the current value of the option. You can edit the text and click Apply for the changes to take effect. Click OK to close the dialog box.

Here are the options:

- `StartingPages` determines the starting page number for all notebooks in a project. The option is specified as a list,  $\{p_1, p_2, \dots, p_n\}$ , where  $p_k$  specifies the starting page number of the  $k$ th notebook in the project. Each element of the list can be set to an integer, "Next", "Odd", "Even", or `Inherited`. The default setting is `StartingPages`  $\rightarrow$   $\{1, \text{"Next"}\}$ .

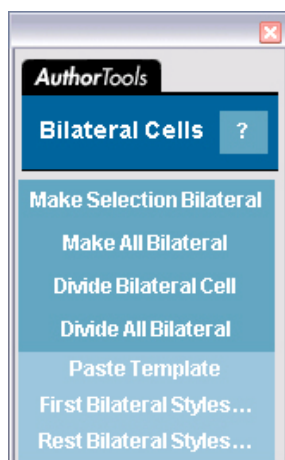
If  $p_k$  is "Next", "Odd", or "Even", the starting page for the  $k$ th notebook is the next page, next odd page, or next even page following the last page of the previous notebook. If  $p_k$  is `Inherited`, the starting page number for the  $k$ th notebook is inherited from the setting of the option `StartingPageNumber` for that notebook.

- `OpenAllCellGroups` determines whether to open all cell groups before calculating page breaks. The default setting is `OpenAllCellGroups`  $\rightarrow$  `True`.
- `PaginationFunctions` specifies a function (*func*) to apply to each notebook after page numbers are calculated but before the notebook is closed. If this option is set to something other than `Null`, the return values from `Paginate` will have *func*[*nb*] appended to each sublist.

## □ MakeBilateralCells

### Introduction

The `MakeBilateralCells` palette allows you to display *Mathematica* calculations in a compact, bilateral format. Each bilateral cell consists of expository text to the left and input and output cells to the right. Bilateral cells are used extensively in *The Mathematica Book* to display examples.



### Creating Bilateral Cells

To convert a *Mathematica* example into bilateral form:

1. Create the example you want to convert to bilateral form. By default, each example must consist of a single MathCaption cell followed by any number of input and output cells. However, you can change the cell styles used to construct a bilateral cell, as explained in Specifying Cell Styles.
2. Open the MakeBilateralCell palette.
3. Select the cells you want to format.
4. Choose Make Selection Bilateral.

The selected cells are automatically converted into bilateral cells. The text from each MathCaption cell appears in the left column while the corresponding input and output cells appear in the right column.

For example, consider the following sequence of cells:

Here is the integral  $\int x^n dx$  in *Mathematica*.

```
Integrate[x^n, x]
```

$$\frac{x^{1+n}}{1+n}$$

If you select the cells and choose Make Selection Bilateral, all three cells are combined into a bilateral cell.

Here is the integral  $\int x^n dx$  in *Mathematica*.

```
In[1]:= Integrate[x^n, x]
Out[1]=  $\frac{x^{1+n}}{1+n}$ 
```

If you have a notebook containing multiple *Mathematica* examples, you can convert all the examples into bilateral form in one step. Each example must consist of a single MathCaption cell followed by any number of input and output cells.

**To convert all examples in a notebook or project into bilateral form:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the MakeBilateralCell palette.
3. Choose Make All Bilateral.

*Mathematica* automatically finds all cases in the notebook where input and output cells follow a MathCaption cell and converts each such example into a single bilateral cell.

### ***Dividing Bilateral Cells***

Once you have created a bilateral cell, you can reverse the process and convert the cell back into its constituent Input, Output, and MathCaption cells. To do this, select the bilateral cell and choose Divide Bilateral Cell. To divide all bilateral cells in a notebook in one step, select the notebook and choose Divide All Bilateral.

### ***Updating Bilateral Cells***

Once you have created a bilateral cell, you cannot evaluate its input cells. If you want to modify the input and re-evaluate, you must divide the bilateral cell, evaluate the raw input cells, and then change your cells back into bilateral form.

**To modify input and output in a bilateral cell:**

1. Select the bilateral cell and decompose it by choosing Divide Bilateral Cell.
2. Edit the cells you want to modify.
3. Select any Input cell you edited and re-evaluate it (by pressing `SHIFT+ENTER`). A new Output cell is generated.
4. Reformat the group of cells by selecting it and choosing Make Selection Bilateral.

### ***Pasting Bilateral Templates***

Another way to create a bilateral cell is to first create a template and then add the content you need in the appropriate text fields.

**To create a template for a bilateral cell:**

1. Open the MakeBilateralCell palette.

2. Place the cursor at the point where you want the bilateral cell to be created.
3. Choose Paste Template. A template is pasted into the notebook, containing some sample text.
4. Replace the sample text with material appropriate for your example.

### Specifying Cell Styles

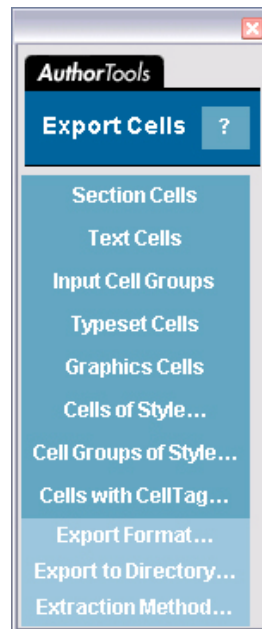
The two buttons at the bottom of the palette allow you to specify what cell styles should be used in creating a bilateral cell. To set the value of either option, click the appropriate button to bring up a dialog box. You can edit the text and click Apply for the changes to take effect. Click OK to close the dialog box.

- First Bilateral Styles specifies the cell styles that should be used for the caption of a bilateral cell. The default setting is `$FirstBilateral`Styles`. This typically has the value `{"MathCaption"}`.
- Rest Bilateral Styles determines what cell styles should be used for the content of a bilateral cell. The default setting is `$RestBilateral`Styles`. This typically includes Input, Output, and Graphics cells.

## □ ExportCells

### Introduction

The ExportCells palette enables you to extract content of a specified type from a given notebook or project and save the content in a desired format. For example, you can extract all the graphics cells from a notebook and save each graphic as a separate GIF file.



You can extract content of the following types:

- all cells of a particular style, such as all Section or Text cells
- all cell groups with a heading cell of a particular style, such as all input cell groups
- all cells with a specified cell tag

By default, the content is saved in a new notebook. However, you can choose from any of the other export formats supported by *Mathematica*: plain text, HTML,  $\text{\TeX}$ , GIF, JPEG, BMP, EPS, TIFF, PICT, and so on. To view a complete list of possible formats, choose `ExportFormat` at the bottom of the palette.

### **Exporting Cells**

**To export all cells of a specified style:**

1. For a single notebook, open it and make it the currently selected notebook. For multiple notebooks, open the `MakeProject` dialog box and load the project file for your project. (If you have not already created a project file, see `Creating a Project File` to learn how.)
2. Open the `ExportCells` palette.
3. Choose `Cells of Style`.
4. In the dialog box that appears, enter the style of the cells you want to export.
5. Click `OK` to close the dialog box and apply the changes to your notebook. Click `Cancel` if you do not want the changes to be saved. Click `Apply` to apply the changes without closing the dialog box. Then click `OK`.

The cells you specified are extracted from the source notebooks and saved in the same directory. By default, the exported cells are saved in a new notebook. If you want to export cells in a different format, you must set the value of the option `ExportFormat` as explained in `Specifying the Export Format`.

**To export all cell groups of a specified style:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the `MakeProject` dialog box and load the project file for your project.
2. Open the `ExportCells` palette.
3. Choose `Cell Groups of Style`.
4. In the dialog box that appears, enter the style of the heading cells for the cell groups you want to export. Then click `OK`.

The cells you specified are extracted from the source notebooks and saved in the same directory. By default, the exported cells are saved in a new notebook. If you

want to export cells in a different format, you must set the value of the option `ExportFormat`, as explained in *Specifying the Export Format*.

**To export all cells having a specified cell tag:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the `MakeProject` dialog box and load the project file for your project.
2. Open the `ExportCells` palette.
3. Choose `Cells with CellTag`.
4. In the dialog box that appears, enter the cell tag of the cells you want to export. Then click `OK`.

All cells with the cell tag you specified are extracted from the source notebooks and saved in the same directory. By default, the exported cells are saved in a new notebook. To export cells in a different format, you must set the value of the option `ExportFormat` as explained in *Specifying the Export Format*.

### ***Specifying the Export Format***

**To specify the export format:**

1. Choose `ExportFormat`. This brings up a dialog box showing the current setting of the `ExportFormat` option.
2. Specify the format you want to save in by typing it in the text box. You can also click a button bearing the name of a specific format to paste that format into the text box.
3. Click `Apply` for the new setting to take effect. Click `OK` or `Close` to close the dialog box.

### ***Specifying the Export Directory***

**To specify the directory to export to:**

1. Choose `Export To Directory`. This brings up a dialog box showing the current setting of the `ExportDirectory` option.
2. In the text box, enter the full pathname of the directory to which you want to save. You can also use the `Browse` button to bring up a file dialog box, then locate a file in the target directory and click `Open`.
3. Click `Apply` for the option to take effect. Click `OK` to close the dialog box.

### ***Specifying the Extraction Method***

**To specify the extraction method:**

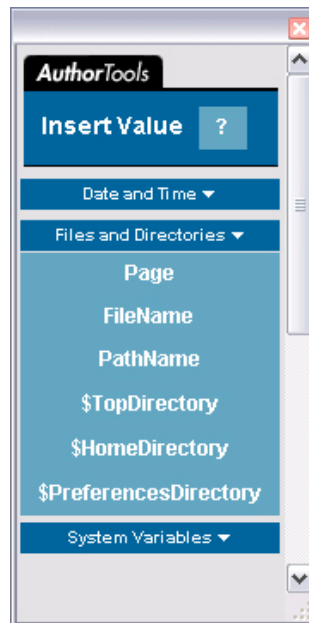
1. Choose `ExtractionMethod`. This brings up a dialog box showing the current setting of the `ExtractionMethod` option.
2. Enter the extraction method you want to use in the text box. Possible settings are:

- NotebookGet, which is faster but uses more memory
  - NotebookRead, which is slower but uses less memory
3. Click Apply for the new setting to take effect. Click OK or Close to close the dialog box.

## □ InsertValue

### Introduction

Using the InsertValue palette, you can insert an object to display the current value of variables such as the names of the home directory or the preferences directory, as well as the current filename, pathname, date, or time. The display object, once inserted, is dynamically updated so it always reflects the current value of the variable.



For example, the text in the following cell contains display objects that refer to the current date and time. This ensures that each time the notebook is opened, the cell always shows the current date and time.

```
Today's date is 5/11/04.
The current time is 12:54:19.
```

You can choose from variables divided into three categories.

- Date and Time—Enter the current date or time in various formats.
- Files and Directories—Enter the current filename, pathname, page number, or the values of selected directories.

- **System Variables**—Enter system parameters, such as the operating system or version number of *Mathematica*.

### Inserting a Value

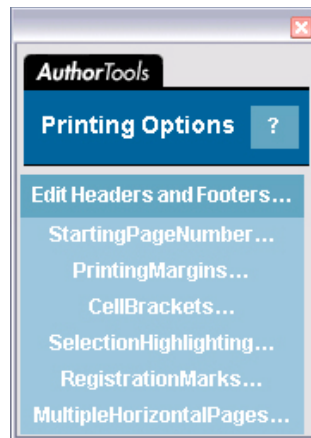
To insert the value of a variable:

1. Open the InsertValue palette.
2. Choose any of the three categories: Date and Time, Files and Directories, or System Variables to view the buttons in that category.
3. Click the button with the name of the variable whose value you want to enter. The value of the selected variable is pasted into the notebook at the position of the cursor.

## □ SetPrintingOptions

### Introduction

The SetPrintingOptions palette allows you to set options that control how a notebook is printed. You can use this palette, for example, to specify headers and footers, set the starting page number, or control the size of margins.



### Headers and Footers

You can specify different headers/footers for left and right pages. In addition, for each left or right page, you can specify three different types of headers/footers: left-aligned, right-aligned, and centered.

To specify headers/footers for a document:

1. Open the SetPrintingOptions palette.
2. Choose Edit Headers and Footers. This brings up a dialog box.
3. Choose one of the horizontal tabs bearing the name of the header/footer you want to specify. A set of text boxes is displayed.

4. Enter the header/footer you want in the appropriate text box. You can either type text directly or choose *Insert Value*. This opens the *InsertVariables* palette for entering variables such as the filename or date.
5. To enter a running head, choose *Insert Running Head*, type a cell style in the dialog box that appears, and click *OK*.
6. When you have finished editing, click *OK* to close the *Headers and Footers* dialog box and apply the changes to your notebook. Click *Cancel* if you do not want the changes to be saved.

**To specify the facing of pages:**

Choose the appropriate setting for the *Set Facing Pages* option at the top of the dialog box. This option can be set to one of three different settings.

- First page facing left—This is indicated by two boxes to the right of the button with the number 1 in the left box.
- First page facing right—This is indicated by two boxes to the right of the button with the number 1 in the right box.
- Left and right pages are equivalent—This is indicated by a single box to the right of the button with the number 1 inside it.

For the first two cases, you can specify separate headers/footers for left and right pages. For the third case, you can specify headers/footers for one type of page only. The number of text boxes available for displaying headers/footers change automatically to reflect this difference.

**To control whether headers/footers are printed on the first page:**

Click one of the radio buttons marked *Yes* or *No* after the “Header/Footer on First Page?” statement.

**To insert a line below/above all headers/footers:**

Click one of the radio buttons marked *Yes* or *No* after the “Left/Right Header/Footer Lines?” statement.

**Starting Page Number****To specify the starting page number for the printed notebooks:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the *MakeProject* dialog box and load the project file for your project. (If you have not already created a project file, see *Creating a Project File* to learn how.)
2. Open the *SetPrintingOptions* palette.
3. Choose *StartingPageNumber*. This brings up a dialog box.
4. Enter the starting page number.
5. When you have finished editing, click *OK* to close the dialog box and apply the changes to your notebook. Click *Cancel* if you do not want the changes to be saved. Click *Apply* to apply the changes without closing the dialog box.

### **Printing Margins**

**To set the margins for the printed notebooks:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the SetPrintingOptions palette.
3. Choose PrintingMargins. This brings up a dialog box.
4. Enter the left, right, bottom, and top margins in printer's points (72 points equal one inch).
5. When you have finished editing, click OK to close the dialog box and apply the changes to your notebook. Click Cancel if you do not want the changes to be saved. Click Apply to apply the changes without closing the dialog box.

### **Cell Brackets**

**To specify whether cell brackets should be printed:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project. (If you have not already created a project file, see Creating a Project File to learn how.)
2. Open the SetPrintingOptions palette.
3. Choose PrintCellBrackets. This brings up an editing dialog box.
4. Click True if you want cell brackets to be printed and False otherwise.
5. When you have finished editing, click OK to close the dialog box and apply the changes to your notebook. Click Cancel if you do not want the changes to be saved. Click Apply to apply the changes without closing the dialog box.

### **Selection Highlighting**

**To specify whether the current selection should appear highlighted when a notebook is printed:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the MakeProject dialog box and load the project file for your project.
2. Open the SetPrintingOptions palette.
3. Choose SelectionHighlighting. This brings up an editing dialog box.
4. Click True if you want the notebook selection to appear highlighted in print and False otherwise.
5. When you have finished editing, click OK to close the dialog box and apply the changes to your notebook. Click Cancel if you do not want the

changes to be saved. Click **Apply** to apply the changes without closing the dialog box.

### **Registration Marks**

**To specify whether trim marks should be added to indicate the corners of a printed page:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the **MakeProject** dialog box and load the project file for your project.
2. Open the **SetPrintingOptions** palette.
3. Choose **RegistrationMarks**. This brings up an editing dialog box.
4. Click **True** if you want trim marks to be printed and **False** otherwise.
5. When you have finished editing, click **OK** to close the dialog box and apply the changes to your notebook. Click **Cancel** if you do not want the changes to be saved. Click **Apply** to apply the changes without closing the dialog box.

### **Multiple Horizontal Pages**

**To specify whether cells that extend beyond the width of the page should be printed on additional pages:**

1. For a single notebook, open the notebook and make it the currently selected notebook. For multiple notebooks, open the **MakeProject** dialog box and load the project file for your project. (If you have not already created a project file, see **Creating a Project File** to learn how.)
2. Open the **SetPrintingOptions** palette.
3. Choose **MultipleHorizontalPages**. This brings up an editing dialog box.
4. Click **True** if you want the cells to continue printing on another page. Click **False** if you want the cells to be chopped off at the edge of the page.
5. When you have finished editing, click **OK** to close the dialog box and apply the changes to your notebook. Click **Cancel** if you do not want the changes to be saved. Click **Apply** to apply the changes without closing the dialog box.

### **About the Author**

Pavi Sandhu has a Ph.D. in physics from Boston University. He is the author of *The MathML Handbook*, published by Charles River Media in November 2002.

#### **Pavi Sandhu**

Senior Technical Writer  
Oracle Corporation  
Redwood Shores, CA