

The OpenEpi Toolkit

Open Source Input and Output Tools for Developing Statistical Calculators

Written entirely in JavaScript and HTML for Cross-Platform Compatibility

“Go ahead. Write a calculator.”

Developed by
The OpenEpi Project

Supported in part by a grant from the Bill and Melinda Gates Foundation

Andrew G. Dean, Kevin M. Sullivan, and Roger Mir

Atlanta, Georgia

Updated February, 2007

The OpenEpi Toolkit.....	1
Types of Computer Software Calculators.....	2
OpenEpi and Open Source Licensing.....	4
Setting Up a New Application in OpenEpi.....	5
Input Commands	6
newtable	6
title	6
newrow.....	7
Data Types	7
Special Cell Characteristics	8
allowstrata	8
lock	9
unlock.....	10
Color Commands.....	10
The Datarange Command	10
Input Table Example.....	11
Checkcode.....	11
Retrieving and Working With Entered Data.....	12
Producing Output	13
Output Commands.....	13
Example of Output Commands (From Shell.htm or Proportion.htm).....	15
Conclusion	16
Stratified Data	18
Saving and Retrieving Data	18
Analysis of Stratified Data.....	21
Features Not Yet Implemented (the Wish List)	22

Update--February 2007

The supporting programs in the framework, particularly AppHelper.js and Estratum.js, have been changed to implement a tabbed interface and do away with the popup windows

that formerly caused trouble with popup prevention software. Now you can move smoothly among Div's or iFrames within the same window. Help files are still displayed in separate windows, but this should not cause problems. From the developer's point of view, the instructions below are still valid and none of the previous application programs had to be updated (whew!).

Types of Computer Software Calculators

A software calculator is a computer program that produces mathematical or statistical output from data entered on the screen. Many calculators will print results, but most, in contrast to database and statistical programs, do not save or retrieve data. The thousands of calculators available on the Internet and elsewhere are designed for a wide range of purposes—counting calories, construction, engineering, astrology, mathematics, and statistics. Those of most interest in public health and epidemiology are statistical calculators and those for nutrition statistics.

Free calculators available on the Internet vary in quality and level of validation, based on authorship, level of maintenance, and other factors. Most do not have special facilities for translation into multiple languages.

From the software point of view, there are several main types of calculators:

- Windows, Macintosh, Linux, or palmtop calculators developed for a specific desktop, and generally running only on that platform. They are developed in Visual Basic, C++, Delphi (Pascal), and other desktop languages.
- Internet calculators that can be viewed from common browsers (Internet Explorer or IE, Netscape or NS) on any major desktop platform when running on an Internet Server. They are of three major varieties:
 1. Those that run only with the aid of a server, whether the server is on the local machine (Personal Web Server or IIS for Windows) or on the Internet. Languages available for development include Active Server Pages (ASP), ASP.NET, PHP, JSP, and others. Although it may be possible to download the applications and run them from a computer that is not currently connected to the Internet (say, during a field investigation), running and maintaining Personal Web Server or IIS on the local machine represents a significant barrier to convenient access to the calculator when the computer is not connected to the Internet.
 2. Those that run from an Internet server, but also from a local hard disk without requiring a server. These include applications developed in HTML and JavaScript. Since both IE and NS contain JavaScript interpreters they are more or less platform independent. This kind of application might be called “browserware.” Browserware has to contend with differences between browsers, but is not limited to any particular operating system and does not require a server—only a browser—to run. OpenEpi is based on the browserware concept, so that it will run in Windows, Linux, the Macintosh, and future Internet-based palmtops, as long as a modern version of either IE (5+) or NS (7+) is used, and will

run either from a hard disk without a server, or a local or remote Internet server.

3. Some ‘browserware’ applications require additional programs that are accessed from the browser to provide additional features. They use Perl, Java, Python, and other languages to provide additional features, but still function in any operating system that has the relevant run-time module. An Internet server will automatically download the necessary software the first time the application is run or warn that it must be obtained. The well-known Java language and its Java Beans and Java Applets were designed to be the totally cross-platform solution, with a Java run-time module to be included in every browser. However, disputes between Microsoft and Sun Systems have largely removed the advantage of the design, and left JavaScript and HTML as the only practical cross-platform and cross-browser solutions that do not require additional software to be present on the local computer.

The OpenEpi concept is to use HTML and JavaScript to the maximum limit of their capabilities, and to produce applications that run both on the Internet and also from the local hard disk (choice #2 in the list above). In the current implementation, this has the following advantages and disadvantages:

Advantages:

- HTML and JavaScript are both text-based languages and are interpreted, not compiled, so the source code is immediately accessible to users who download the programs and can be read with Notepad or any text-based editor—the ideal situation for an Open Source project.
- No extra programs beyond a browser are required to run the applications.
- Programs are small, on the order of 100 or 1000 times smaller than Windows programs
- Programs run in Windows, Mac, Linux and other systems with mainline browsers
- Can be run with or without a server and with or without an Internet connection
- JavaScript is many useful features, such as associative (‘retrieve-by-name’) arrays, easy to understand object programming, and (not so easy to understand but powerful) pattern matching with regular expressions. A reasonable library of Math functions is included.
- We have been able to develop non-English language translation features that allow the translator to work on a single file listing all the phrases in the programs and the basic documentation.

Disadvantages

- There is no completely cross-platform solution to saving data on a local computer without running a server, although we do so in Windows systems by using HTA files.
- JavaScript is fussy about a number of things, like upper and lower case function names, and the use of “==” for testing equality, and very relaxed about other things, like data types. Passing data between browser windows is tricky and

dependent on timing, the alignment of the planets, and good luck. (Like making an information “drop” in the park in a cold-war spy novel.)

- Calculation-intensive operations may be less efficient because of the interpreted nature of JavaScript, although there are some impressive applications on the net. This has not been a problem so far.
- Reading and writing databases is currently limited to server-side operations, but, as mentioned above, we have developed methods to overcome this in particular operating systems so that future calculators can save and retrieve data locally on the hard disk.

OpenEpi and Open Source Licensing

OpenEpi is an Open Source project, supported in part by a grant from the Bill and Melinda Gates Foundation to Emory University to enhance the statistical capabilities of Epi Info, a public domain software package produced by the Centers for Disease Control and Prevention (CDC), Atlanta, Georgia, USA.

The Open Source movement is based on providing free access to source code so that it can be understood, criticized, and improved by a wide audience. We would prefer to develop our own license along the lines of:

“OpenEpi source code, documentation, and concepts are available to all to use as they wish. Please help improve the code by sending suggestions or new code back to the authors. We suggest citation to acknowledge contributing authors’ works as with any scientific publication.

The user assumes responsibility for determining appropriate use of the software, for consequences of its use, and for checking results against other reliable sources. No warrantee of any kind is given.”

Since the wording above does not constitute an approved Open Source license, we have also chosen to apply the “MIT License” chosen from the www.opensource.org site.

The MIT License (from www.opensource.org)

Copyright (C), 2003, 2004 by Andrew G. Dean, Kevin Sullivan, and Roger Mir, Atlanta, Georgia

"Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.”

"The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. in no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software."

Setting Up a New Application in OpenEpi

The OpenEpi Toolkit provides methods for configuring a data entry page and for constructing HTML output to display statistical results. This leaves the author of a new program free to do statistical programming in one or more javascript (.js) files that are linked to the shell program provided to make an application. If programming is done in JavaScript and tested in both Netscape and Internet Explorer, the application should run in Windows, the Mac, and Linux, and in either browser.

The OpenEpi Toolkit provides a sample HTML file (Proportion.htm) and a javascript file (Proportion.js) that contains a sample of statistical programming that can be altered to produce a new program. The rest of the Toolkit consists of Etable.htm and its associated directory of ETable_Files. These files provide the input and output functions that can be configured using a set of commands provided for this purpose.

To begin programming a new application, you set up a new subdirectory, named for your application, and at the same level in the directory tree as the ETable subdirectory. In the discussion that follows, the new application is referred to as "NEWAPP," for which you would substitute your own name to indicate the kind of statistics being offered.

"Proportion" and "TwobyTwo" are examples.

NewApp is developed by copying the Toolkit directory at the same level in the directory hierarchy (Copying and then Pasting the directory will result in "Copy of Toolkit" which can be renamed to match your intended application; let's call it NewApp. Inside the new directory, change the sample files, Proportion.htm and Proportion.js, to NewApp.htm and NewApp.js (actually to the name you choose for your new application).

If you now run NewApp.htm, it should behave exactly as does Proportion.htm in the toolkit directory.

Then the Input and Output functions can be used from within the configureInput() and calculateOEResults() functions in NewApp.htm. The statistical functions that you write are contained in a file called NewApp.js, although other configurations are possible. Since a shell program completely without code would not function or be very informative, Proportion.htm, Proportion.js, and now, NewApp.htm and NewApp.js, contain the code for statistics for a simple binary proportion, to be replaced by your own code. The images are those for the Proportion application, to be replaced with images appropriate for your application.

The directory structure should be as follows:

ETable

- ETable_Files directory (DataStore.js, dynchart.js, dynlayer.js, EStratum.js, OECommands.js, ReadCookie.js, StatFunctions1.js)
- Image directory (images used by Etable: e.g., shim.gif, redpin.gif, greenpin.gif)

NewApp

- NewApp.htm, NewApp.js
- In the SCREENS folder, images of input and output screens, named NewAppIn.gif and NewAppOut.gif.

Input Commands

Etable.htm, the data input page, is opened in a popup window from NewApp.htm. It is configured by the commands in the “configureInput” function. ETable input tables offer input features normally associated with desktop applications, such as the use of either Tab or Enter keys to move from field to field, the up and down arrow keys to control the direction of cursor movement, and the possibility of writing JavaScript “check code” to validate or transform input in any specified cell of the input table.

When the Calculate button is clicked by the user, the items entered are placed in a standard array, from which they can be retrieved by your application to perform the calculations. ETable then calls the OECalculate function in its “opener” window—that is, your application, known here as NewApp.htm. The code you have placed in OECalculate and in a statistical function called from it in NewApp.js retrieves the appropriate items from the input array, does the calculations, and then uses the Output object commands to deliver HTML output to the user. The results pop up in a separate window and can be cut and pasted to Microsoft Word or other programs for editing and saving.

After creating an Input object with a variable assignment, as in the NewApp.htm function configureInput(), the methods of the Input object become available to be used as commands. The commands currently implemented are (note lower case must be used, or they will not work; JavaScript is case sensitive):

newtable(numRows, numCols, widthinpixels, heightinpixels, leftmargininpixels, topmargininpixels)

This command constructs a new table object in ETable.htm of the dimensions specified. Further commands are used to better define the table, although newtable is sufficient to construct a blank table.

title(“textoftitle”)

The specified text will be placed in the header or title row at the top of the input table. HTML tags can be included with the text as long as they are inside the quotation marks.

newrow(<list of items>)

This allows specifying an entire row of cells. Specifications for cells are separated by commas. Each one can contain a cell type and any number of special characteristics ending in a colon ":". A text item or number that does NOT end in a colon can also be included. (To include a colon as part of the text, use a double colon "::".) The entire cell specification must be enclosed in double quotes (or, if it contains double quotes internally, in single quotes).

Data Types

The data type for the cell should be specified as one of the following:

data:

The cell will receive data from the user and place it in the standard output data matrix for statistical calculations.

rowtot:

The cell will automatically maintain row totals for any data cells in the same horizontal row. A value can be entered by clicking on the cell and affirming the desire to enter a value. If the total is entered, and only one other cell is blank, the value of the missing cell is calculated automatically. Thus, if the user knows that 379 of 1280 persons had a disease, entering 374 under Disease=(+) and 1280 for the row total is sufficient to complete the row. ETable will place 901 in the column for Disease=(-) in a 2 by 2 table.

coltot:

The cell will automatically maintain column totals for any data cells in the same vertical column. As with row totals, a single empty cell will be filled in automatically if the column total and other cells are entered.

grandtot:

The cell will automatically maintain the grand total by adding the values of the column totals.

varname:

The cell can receive text that is intended to represent a variable name, such as "Ill?" At present the data type has no special function other than to require confirmation when the user wants to enter text, but, in future versions, it is anticipated that identifying a variable name as such will be useful.

valname:

The cell can receive text that represents a value of a variable, the most common being "yes" and "no" or (+) and (-). Future versions may make

use of this identification, although the current version treats cells of type varname:, valname:, and label: identically.

label:

The cell contains text for display or instructions. The user can edit or enter text after confirmation.

Special Cell Characteristics

The special characteristics that can be included in a cell item are:

- Spanx: where x is the number of columns to be occupied by this cell
- hx: where x is a number from 1 to 6, indicating the HTML header style to be applied
- bold: for bold type
- b: left justify
- c: center
- r: right justify
- t: vertically align to Top
- m: vertically align to Middle
- b: vertically align to Bottom
- color#rrggbb: where #rrggbb is a color indicator in hexadecimal. #000000 is black. #ffffff (or #FFFFFF) is white, and #FF0000 is pure red. Charts of colors are available on the Internet.
- nowrap: text will be kept on the same line, without wrapping
- (Not Yet Implemented) pattern##### where ##### can be any “regular expression” or mask specifying what the user is allowed to enter. If the data item entered does not match the pattern, a message appears and the item must be corrected or cleared. Useful regular expressions are:

A study of the examples below should clarify how cell specifications are built up within the newrow command function.

usetablesettings(true) and usetablesettings(false)

If true, “Exposure” (or its equivalent) is assumed to be on the left of the table and “Disease” at the top. The settings for table configuration in the Settings.htm (Options/Settings on the menu) page do not affect table orientation.

allowstrata(true) and allowstrata(false)

If true, presents an AddStratum button at the top of the table. The user can create additional strata with the button. After the first stratum, a button also appears to delete the current stratum. A dropdown selection box allows moving to another stratum. Data

entered in strata are stored in the variable dataMatrix[n] where n is the number of a stratum, beginning with 1 (not 0).

settingslink(true) and settingslink(false)

If true, a link to the settings page for two by two tables is provided to the left of the table. Below the link is a text item indicating the current setting for Confidence Level.

tablecmd("myTable.")

This command provides access to a number of commands contained in the myTable object. To see what these are, examine the file called EStratum.js. All the properties and methods listed in the Stratum object can be reached as myTable objects. For example, the following default values for the table are listed.

```
this.cellFont = 'arial'  
this.cellFontSize = '16px'
```

In the input object in your application, the following commands may be used to set different properties for the font and font size.

```
tablecmd("myTable.cellFont='times new roman'")  
tablecmd("myTable.cellFontSize='10px'")
```

Note that every detail of case and single and double quotation marks must be exactly right. You can swap single and double quotation marks, but not "nest" two sets of double or single marks. Nesting a set of singles inside a pair of doubles is OK, and vice versa. The name of the object is "myTable" with a small "m" and capital "T". These commands are made available for special tinkering, and will not be used often. It is safest to put these and other commands after the newtable command that creates the myTable object.

lock(what)

By default, cells in a table of type, "data," are available for data entry. The cursor visits each data cell in turn or as a mouseclick places the cursor in a data cell. Cells of other data types are available for entry by mouse click after the user responds affirmatively to a question about wanting to enter data in this type of cell. The lock command locks the designated cell(s) so that the color does not change when the mouse cursor hovers over the cell, and entry is prevented. Values of "what" must be in quotation marks, and can include:

"labels"

Locks all cells except "data", "rowtot", and "coltot." ("grandtot" is always locked).

Names of cell types. All cells of the designated type will be locked.

"label"

"data"

"varname"

“valname”
“rowtot”
“coltot”

More than one value can be given, separated by commas, as in:

lock(“rowtot”, “coltot”)

Individual cells can be locked by using cell designators such as “E0D0” for the cell with minimum values for both Exposure and Disease (absence of both, for example). A little experimentation should identify whether you have named the intended cell for locking, as locked cells do not change color when the mouse cursor hovers over them.

unlock(what)

The unlock function accepts the same parameters as the lock function, but serves to unlock cells that have been locked. Hence, if lock(“labels”) has been issued, unlock(“varname”) will insure that the user can enter in cells of type, “varname”, but not in those with type “label” or “valname.”

Color Commands

Color commands are cell, row, or column specific, as follows:

cellcolor(row, column, “#rrggbb”)
rowcolor(row, “#rrggbb”)
colcolor(col, “#rrggbb”)

Column and row numbers start with 0 at the upper left and top of the table. The last row and column numbers are therefore rows-1 and cols-1

The Datarange Command

The datarange command sets up default cell types in a table automatically merely by giving the coordinates of the upper left and bottom right cells in a rectangular area. If the area does not include the last row and column, these are automatically set up as row and column totals and a grand total is calculated in the lower rightmost cell. If columns and rows are available to the left and top of the data range, these are automatically designated as varname and valname cells from the outside in. Other cells are assigned to the “label:” type. For an R x C table, for example, this command allows setting up the entire table with very few commands.

datarange(rowmin, rowmax, colmin, colmax)

Example:

In a table of 8 columns and 13 rows, the following command within the configureInput function will set up 5 columns and 10 rows of data cells, with row,

column, and grand totals, 2 rows above, and 2 columns to the left for variable names and values:

```
datarange(2,2,rows-2, cols-2)
```

The upper left corner of the table is row 0, column 0, (not 1,1).

Input Table Example (from the Proportion application)

```
newtable(crows,ccols,w,h,x,y);
```

In the program values have previously been assigned to the parameters for greater clarity. You might choose to put the actual numbers in the newtable command, as in newtable(2,3,200,100,400,100).

```
title("<b>Simple Proportion</b>");
```

Place the words “Simple Proportion” in the title or header at the top of the table. The “b” tags are standard HTML tags for “bold”. These or other tags can be included within the quoted string.

```
newrow("varname:<b>Proportion</b>", "valname:b:Numerator", "data:");
```

Constructs a row of three cells. The first is of type “varname:” and contains the word “Proportion” with an Internet tag for “bold”. The second cell is given the type “valname:” and the text “Numerator”. The third cell is of type “data:”, for entry of the numerator.

```
newrow("label:","valname:b:Denominator", "data:");
```

The second row has three cells also, since it must match the first row, and these are of types “label:”, “valname:”, and “data:”, with the word “Denominator in the second cell.

Checkcode

A function called checkcode in the Shell.htm program can be used to evaluate each entry the user makes and to take appropriate action if necessary. In the current Shell.htm, the function is set up to guarantee that only numbers are entered in cells of type “data:”, but the row and column numbers of the current cell are passed to the function so that conditions can be set up only for certain rows, columns, or cells. Values of other cells can be obtained using functions of the myTable object within ETable.htm, and their values can also be set with the myTable insert function.

The Checkcode function places a great deal of power in the hands of the statistical programmer to control and manipulate data during entry, but current programming may require quite a bit of study of the Javascript in the myTable object and trial-and-error.

Future development can extend and simplify how checkcode works if this is a popular feature.

Retrieving and Working With Entered Data

When the user clicks the Calculate button in ETable, values in the cells of type “data” are placed in a one-dimensional array, with each item identified by a text string as follows:

“E0D0” –The baseline levels for both Exposure(E) and Disease(D), i.e., the absence of both

“E0D1”—No exposure and the next level of Disease (“yes” or (+) in a 2x2 table

“E1D0”—Exposure present but no disease

“E1D1”—Both Exposure and Disease present, or next level up from zero E and D, if the table is larger than 2 x 2.

Tables can be as large as the screen will accommodate, for example “E10D5”. If the concepts of Disease and Exposure are not important, the E and D can be taken as arbitrary identifiers.

“Why use these complicated identifiers, and what happened to a, b, c, and d?” you ask. The advantage of this scheme is that ETable keeps track of changes in settings for the 2 x 2 table (mainly), and will always deliver the same numbers for E0D0 no matter where on the screen the cell for minimal exposure and minimal disease is configured by the user. Thus it is possible to set up the data entry table rotated 90 degrees or with minimal values on the left instead of the right and top instead of bottom, and the statistical programmer does not have to worry about how the table is configured. Of course the cells must be properly labeled on the screen and the user must read the labels when entering data, but, within these limits, the system is independent of screen layout. If `usetablesettings(false)` appears in the `configureInput` function, E will always be on the left, representing rows and D will be at the top, with values in columns.

So much for a single table. Each of the one-dimensional arrays describing a table is placed inside the “data” array passed to the `OECalculate` function. Stratum number 1 becomes `data[1]` and stratum n will be `data[n]`. Hence to retrieve the value for E0D0 from the first stratum and set it equal to “A” for your calculations, the JavaScript command assignment is:

```
A=parseFloat(data[1][“E0D0”])
```

The standard JavaScript function, `parseFloat()`, is used to convert the cell contents to a number and also to remove any stray text that might be present. It yields a floating point number, or, if no valid number is found, “NaN” for “Not a Number”. Your statistical routine should check for this eventuality and do something appropriate (give a friendly alert box, assign a zero, etc.).

If you have configured an input table you can see the array it produces by uncommenting (removing the “//”) from the line:

```
//EntryWin.writeTable(EntryWin.dataMatrix[1])
```

in the OECalculate function. Then run the application and enter some distinctive numbers (like 1,2,3...) in the input table before clicking the Calculate button in ETable. You should see the data array displayed with appropriate labels so that you can figure out how to convert it to a format useful for your calculations. If the table is large or there are many strata, you may have to use the .length properties of the arrays to see how big they are and write “for” loops to retrieve all the data. Remember that JavaScript arrays are 0 based, although we have chosen to leave data[0] empty for future use and so that data[1] corresponds to Stratum 1, or to the unstratified condition. In any case, when you have hooked up the assignments correctly, perhaps with the aid of some temporary alert boxes, the application should give the same results no matter how the table is configured by the user in the Settings.htm page.

Producing Output

Output is best produced by passing the output data object (in NewApp, known as cmdObj) to the statistical routine. The elements of the array called cmdObj.data[1] contain the data in Stratum 1 as described above, and those of cmdObj.data[0], are named array elements (e.g., "numD", "numE", "cols", "rows", "conflevel") that are the “metadata” or description of the data. Of course, this is all rather ponderous for managing a single numerator and denominator, as in Proportion.js. You can see a more complex example by examining the TwobyTwo files, which implement single and stratified two by two tables that can be rotated or flipped through user preference and still maintain the same array labels to indicate disease and exposure variables in cmdObj.data[1] and, if present, other strata.

The NewApp.js statistical function (we recommend calling it “doStatistics”) does its calculations and then uses the output table commands to configure one or more tables for the output page. The html page is constructed automatically and placed in the cmdObj object as cmdObj.s (s for “string”). The page is configured, displayed, and possibly saved as a file, by the writeResults function in NewApp.htm.

Saving pages occurs if the special program OpenEpiSave.hta is run on a local disk (after gritting your teeth and telling the security system that you really want to run it). OpenEpiSave.hta runs the OpenEpi menu, from which you can run the other programs. When this is the case, you will see a message after pressing the Calculate button, indicating that the output has been saved and showing the name of the saved file.

Output Commands

After creating an Output object with a variable assignment, as in:

```
Var Out = new Output;
```

The following methods and properties are available:

newtable(cols, colwidth)

Starts a new HTML table for output, specifying the number of columns and their width in pixels. Although all the columns are the same width, extra columns can be allotted for formatting purposes. For example, if you want a space to the left of the table and need the width of two columns for the first item, followed by two normal-width data coluns, you could specify 5 columns. To make horizontal lines that extend to the right of the last column, you might prefer to specify 6 columns. Since a lower resolution screen is 800 pixels wide, 6 columns of 100 pixels each would be practical.

newrow()

Inserts a blank row, to skip a line.

newrow(<list of items>)

Items in the list must be valid strings separated by commas. Thus, “Enter population size” is a valid cell item, as are “2003” and Fmt(x), if Fmt is a function in the local environment. The plus sign can be used to concatenate strings so that “OR=”+Fmt(OR) becomes a single string. Except for the image item, the order of items does not matter.

Each item can have one or more of the following commands, followed by a colon, and then the item to place in the row. *Note that there is no Data Type in output tables.*

- image = “imagepathname”: An image to be placed in the cell. The path should be relative to the location of the application, as in image/myimage.gif, for example. If the image and pathname, followed by a colon, appear to the left of the text, the image is placed on the left side of the cell. If the text appears first, then the image will be placed to the right of the text. Thus you can place a red pin image to the left of a number to point out statistically interesting results, by using “image=redpin.gif:”+myNumber. Swapping the order of the two items will place the pin on the right.
- Spanx: where x is the number of columns to be occupied by this cell
- hx: where x is a number from 1 to 6, indicating the HTML header style to be applied
- bold: for bold type
- b: left justify
- c: center
- r: right justify
- t: vertically align to Top
- m: vertically align to Middle
- b: vertically align to Bottom
- color#rrggbb: where #rrggbb is a color indicator in hexadecimal. #000000 is black. #ffffff (or #FFFFFF) is white, and #FF0000 is pure red. Charts of colors are available on the Internet.

- nowrap: text will be kept on the same line, without wrapping

Example of Output Commands (From Proportion.htm)

Within the OECalculate Function

```
//6 columns and 100 pixels per column
newrow()
//blank line
newrow()
//blank line
title("<h2>" + "Results" + "</h2>");
//Title goes across all cells. Text can have HTML tags that make it
//bold or give it particular styles. These are optional. Be sure to
//include a closing tag for each one.
line(6);
title("<h2>Confidence Limits for Proportion "+ vx + "/" + vN + "</h2>");
line(6);
//Here is the call to the javascript routine that does the calculations
//Note that it also writes results to the output.s object, and is therefore not
//a cleanly defined module, although this could be fixed by putting the results
//in an array and then writing to the res object from here after retrieving the
//array from the calculation module.
```

CalcBin();

Within the CalcBin() function in Proportion.js or Shell.js.

The statistics are performed and the results accumulated in global variables like vP and vPU. They are then collected into an HTML output table using the following commands:

```
with (outTable)
{
  newrow("span3:", "bold:c:Lower CL", "bold:c:Proportion", "bold:c:Upper CL")
  newrow("span3:Exact", DL, Fmt(vP), DU)
  newrow("span3:Wald (Normal Approximation)", Fmt(vPL), "", Fmt(vPU))
  newrow("span3:Modified Wald", Fmt(vWL), "", Fmt(vWU))
  newrow("span3:Score (Wilson)", Fmt(vSL), "", Fmt(vSU));
  newrow("span3:Score with Continuity Correction (Fleiss
Quadratic)", Fmt(vSCL), "", Fmt(vSCU));
```

```
//The following is a conditional statement that puts a footnote at the bottom of the table
only if npq is less than 5.
```

```

        if (npq < 5)
        {
            newrow();
            newrow("span6:npq=" + Fmt(npq) + ". The Wald method (Normal
Approximation) is not recommended when npq < 5.");
        }
    }
}

```

Back in the OECalculate function the HTML string constructed by the commands above in the output object is retrieved and written to a new popup window for the user to see.

```

//Tell output that we are done
        endtable(); //This puts closing tags for the last row and the table.

//Retrieve the HTML string, "outTable.s", and make it valid HTML by adding a header
//and ending tags. In most cases you would also include a replica of the input
//table, but Proportion is so simple that the input figures are just included
//in the title.
        combinedresults = htmlheader+ outTable.s+"</body></html>"
    }
//Now open a window and use document.write to display the contents of the HTML
//output from the statistical module, in this case, CalcBin();

var resultwin=window.open("result.htm","resultwin");
resultwin.document.open();
resultwin.document.write(combinedresults);
resultwin.document.close();
resultwin.resizeTo(800,600);

```

Making HTML the Easy Way

If you copied the Proportion.htm and Proportion.js files to start constructing your application, you may have noticed the variables at the top of Proportion.js, and perhaps even experimented with changing the text to fit your own needs. The variables are:

Title
 Authors
 Description

Demo
 Exercises

The first three are all that are needed to construct the main HTML page for the application itself, and the next two contain instructions for a Demo and one or more Exercises that will show users how to use the application. If you stick with the

prescribed variable names and use care to avoid embedded double quotes, to be sure that every line has a beginning and ending quote, and that continuations of lines either use the + sign or repeat the variable name on the next line as in Authors+=” Anonymous”.

Conclusion

That’s all there is to constructing the Input, data retrieval, and Output for a new module. Additions to the commands will, no doubt, arise with future versions. Meanwhile, it is quite possible to use additional JavaScript programming to achieve the look and feel you want by studying the code and experimenting with modifications. If you have ideas for improvements or have made modifications to Etable or the shell yourself, please send them to us with documentation so that all can benefit.

Andy Dean agdean9@hotmail.com

Stratified Data

There is an input command to allow entering stratified data. See “allowStrata” under Input Commands for more details.

If allowStrata(true) appears in the “with input” section of the application, clicking the “Add Stratum” button on the screen adds another stratum and also saves data from the current stratum to memory.

Saving and Retrieving Data

The data manipulation functions in ETable and its submodule DataStore.js save entered data to memory when the “Calculate” or “Add Stratum” buttons are clicked on the input screen. The resulting global array, called “dataMatrix”, can be further processed in the jsStringFromArray function to produce a string of JavaScript that can be read in directly from a file or evaluated with the “eval” function to recreate the dataMatrix array. The array is named “A” rather than “dataMatrix” for economy in storage and so that it will not overwrite an existing dataMatrix unless the programmer intends that it should.

After entering a single stratum in a TwobyTwo table, the javascript string version of the data looks like this (displayed with a temporary alert box):

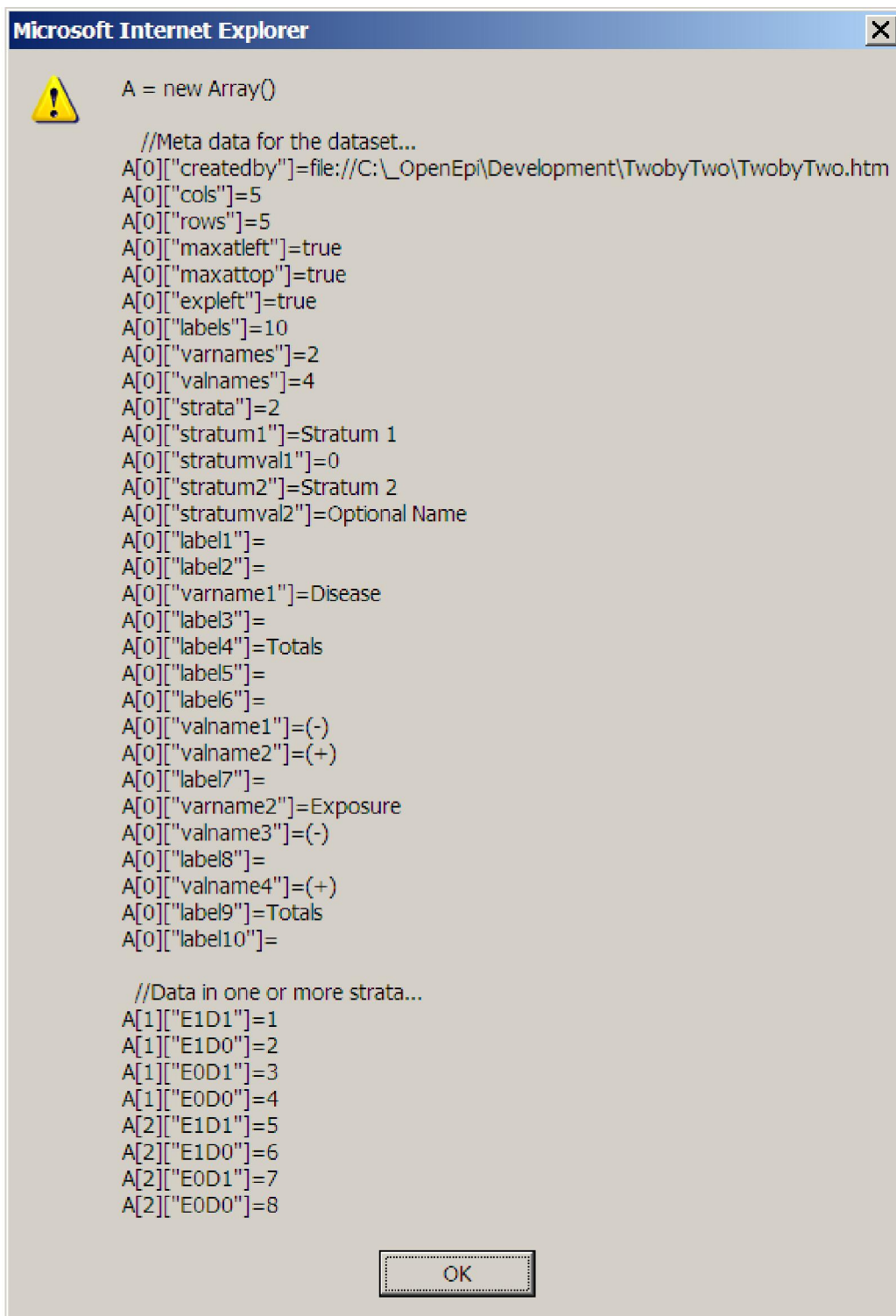
The screenshot shows the 'Open Epi Input Table' application in a Microsoft Internet Explorer window. The application has a 'Calculate' button, a 'Clear' button, and a 'Settings' link (Conf. level=95%). The main table is titled 'Open Epi 2 x 2 Table' and has columns for 'Exposure', 'Disease', and 'Totals'. The 'Disease' column has sub-columns for '(+)' and '(-)'. The 'Exposure' column has sub-columns for '(+)' and '(-)'. The table contains the following data:

		Disease		Totals
		(+)	(-)	
Exposure	(+)	1	2	3
	(-)	3	4	7
Totals		4	6	10

A JavaScript alert box is displayed over the table, showing the following code:

```
A = new Array()
A[0]["createdby"] = file:///C:/_OpenEpi/D
A[0]["cols"] = 5
A[0]["rows"] = 5
A[0]["maxatleft"] = true
A[0]["maxatop"] = true
A[0]["expleft"] = true
A[0]["labels"] = 10
A[0]["varnames"] = 2
A[0]["valnames"] = 4
A[0]["strata"] = 1
A[0]["label1"] =
A[0]["label2"] =
A[0]["varname1"] = Disease
A[0]["label3"] =
A[0]["label4"] = Totals
A[0]["label5"] =
A[0]["label6"] =
A[0]["valname1"] = (-)
A[0]["valname2"] = (+)
A[0]["label7"] =
A[0]["varname2"] = Exposure
A[0]["valname3"] = (-)
A[0]["label8"] =
A[0]["valname4"] = (+)
A[0]["label9"] = Totals
A[0]["label10"] =
A[1]["E1D1"] = 1
A[1]["E1D0"] = 2
A[1]["E0D1"] = 3
A[1]["E0D0"] = 4
```

After entering two strata, the result is:



```
A = new Array()

//Meta data for the dataset...
A[0]["createdby"]=file:///C:/_OpenEpi/Development/TwoobyTwo/TwoobyTwo.htm
A[0]["cols"]=5
A[0]["rows"]=5
A[0]["maxatleft"]=true
A[0]["maxattop"]=true
A[0]["expleft"]=true
A[0]["labels"]=10
A[0]["varnames"]=2
A[0]["valnames"]=4
A[0]["strata"]=2
A[0]["stratum1"]=Stratum 1
A[0]["stratumval1"]=0
A[0]["stratum2"]=Stratum 2
A[0]["stratumval2"]=Optional Name
A[0]["label1"]=
A[0]["label2"]=
A[0]["varname1"]=Disease
A[0]["label3"]=
A[0]["label4"]=Totals
A[0]["label5"]=
A[0]["label6"]=
A[0]["valname1"]=(-)
A[0]["valname2"]=(+)
A[0]["label7"]=
A[0]["varname2"]=Exposure
A[0]["valname3"]=(-)
A[0]["label8"]=
A[0]["valname4"]=(+)
A[0]["label9"]=Totals
A[0]["label10"]=

//Data in one or more strata...
A[1]["E1D1"]=1
A[1]["E1D0"]=2
A[1]["E0D1"]=3
A[1]["E0D0"]=4
A[2]["E1D1"]=5
A[2]["E1D0"]=6
A[2]["E0D1"]=7
A[2]["E0D0"]=8
```

OK

Note that level 0 of the array always contains the meta data. The items within a particular stratum are in no particular order, but they are retrieved by their string keys, not their

numeric indices, and JavaScript takes care of the retrieval. Those expressed as plurals of field types, (“valnames”, “varnames”, “labels”) and “cols” and “rows” are numbers recording how many items are contained in the metadata for that type. Thus, we find out there are two “varnames” since `A[0][“varnames”]=2` (well, actually “==2” because of the dark side of JavaScript, but you know what I mean.). Examining the list, you can find that `varname1` is “Disease” and `varname2` is “Exposure”. This will be true no matter how the user has decided to orient the table. Similarly, for a TwobyTwo table, if the order of values of the variables is reversed on the screen (+ and + on the upper left, as usual), they will appear in the array string in the standard order, with “-“ coming before “+”.

The JavaScript string is automatically saved in the output file on the disk when OpenEpi is being run from the OpenEpiSave.hta on a Windows computer.

Analysis of Stratified Data

The data entry table can be configured to enter more than a single stratum by giving the command `allowstrata(true)` to the `inputTable` object. This places an “Add Stratum” button on the screen, and the user can then add as many strata as desired. The data are delivered to the application in the form of the data array, with each stratum represented by a single value of the array index. Hence `data[1]` is the first stratum. `Data[0]` contains metadata for the entire dataset.

`TwobyTwo`, `PersonTime2`, and `MatchCC` all make use of the `MartinStats.js` module for exact statistics as well as performing their own calculations. In order to have flexibility in presenting the results from both sources, these three programs use the device of constructing commands for output and storing these in memory as arrays until they are needed for output, after which they are retrieved with the “eval” JavaScript function. To see how this works, you might follow the use of the array called “oddsBased” through the various parts of `TwobyTwo.js`. The headings for columns presenting odds based data are placed in a `newrow` command in `oddsBased[0]`, with results for the first stratum in `oddsBased[1]`. If there are n strata, `oddsBased[n+1]` contains a `newrow` command showing the data for the crude table and `oddsBased[n+2]` contains adjusted results for all strata. Presenting data for all strata, including the crude and adjusted results then simply requires using the `eval` command successively with `oddsBased[0]` through `oddsBased[n+2]`. This sends a series of `newrow` commands to the `cmdObj` that build the HTML string, `cmdObj.s`, that is later retrieved and sent to the `writeResults` function.

In order to allow a choice between full display of tables for each stratum or merely showing summary results for all strata, the output for `TwobyTwo`, `PersonTime`, and `MatchCC` is presented in the order:

Stratum 1 table
Stratum 1 analysis

Stratum n table
Stratum n analysis

Summary stratified analysis

Crude (Total) table
Crude table analysis

Features Not Yet Implemented (the Wish List)

In the Input module, it would be nice to be able to specify column widths. This need is somewhat met by the fact that the “span” specification now works within newrow. It removes the lines between cells so that they appear to span, but, since every cell is a separate table (for now), text items do not flow into the next cell as in an HTML table.

Anyway, for now, let’s leave the following input command on the wishlist:

columnwidths(list of columnwidths in pixels, separated by commas)

A graphing module is partially complete and is used in the Diagnostic Test Evaluation module. It produces line graphs, but we hope to have it doing bars and grouped bars in a later release.