

Start of Help

This is the main help file for a program named **CSV Editor**. The **CSV Editor** program is a utility program that allows one to edit what are called **CSV** files. The three letters **CSV** stand for Comma Separated Values. A **CSV** file is normally a file associated with a database table or a spreadsheet in which data is arranged into a series of rows and columns. In reality, a **CSV** file is just a text file that contains text that must be formatted according to certain rules. All modern database systems can import and export tables as **CSV** files that are more useful than binary database files with proprietary formats. This help file was last updated 09/18/2014.

How To Read This Document

Everyone should read the first 13 pages of this document. This help file has been designed to represent all requisite elements of the program and is an **Adobe Acrobat PDF** (Portable Document File). The left margin of the **Adobe Reader** program should contain bookmarks with an outline of the help topics in this file. Click the upper-left large symbol  to toggle all bookmarks on/off. The help topics are generally arranged in the same order as the menu items in the program. We encourage you to use bookmarks to jump to topics of interest. Reading the bookmarks alone may give you an idea of the scope of this help document. Just click on any bookmark with your mouse to jump to the corresponding topic page. If any bookmark has a  symbol on its left, click that  symbol to open the bookmark's subtopic list. Then click on a bookmark subtopic.

This help file contains 426 pages of help. There are 167 bookmarks that assist in navigating such a large file. There is a comprehensive index at the end with 489 entries (and 1,243 page references) that expedite finding topics of interest. While this document could be read from beginning to end, we don't suggest this. A better approach is to just choose a menu item and then pick a topic under that menu item.

For the most part all the topics are independent of one another so they can be read in any order. However, we recommend that everyone should read the first 13 pages to get an initial broad overview of the program and to learn a few basic but essential technical facts. There are probably several functions that you will never use. It is a waste of your time to read about those things you won't use. If you ever find yourself reading material that you don't understand, we suggest you pick another topic. In particular, those functions that refer to a program named **ALabel** can be skipped unless you already use and are already familiar with the **ALabel** program.

Each topic has been written by us to remind ourselves how a given feature or function works. If we haven't used a feature for six months, it is worth our time to read about it before we try it. In fact, the real value of this help file is that it gives examples and discusses details of all the features and options in the program. All the major dialog boxes in the program appear somewhere in this help document. If you are running the program and find yourself in a dialog box with a **Help** button, by all means click that **Help** button and then start reading.

The Purpose of the CSV Editor Program

We designed the **CSV Editor** program to help us edit existing database tables, including tables that may have inconsistent or corrupt data. This program can also produce data, with dates, serial numbers, names and addresses, Social Security numbers, and other kinds of information that looks real, but is artificially simulated and randomly generated. This program can create example data sets for testing database software and for use in other projects that require specially constructed data. This program also aids in creating and editing indexes for books and documents. **CSV Editor** is like a Swiss army knife for working with **CSV** files.

All relational database systems are extremely powerful, but they require proper design, maintenance, and testing. They usually require a lot of programming in some form of the **SQL** language, which is why small offices without database programmers avoid using such systems. However, expecting an ordinary Excel user to create and maintain even a very small data set usually leads to unexpected problems and inconsistencies. We cannot emphasize enough that having a few Excel spreadsheets is not the same as having a properly designed and fully functioning and fully tested database system.

Almost all amateur schemes that start out small and simple eventually need to be incorporated into other larger more robust and better designed systems. Sometimes we are given key column data in a format that is inconsistent, if not often unusable for referential integrity purposes. As an example, we once had to convert data in which the letter oh, O, and the digit zero, 0, were used interchangeably and randomly. Apparently the person who entered the data had no need to make a distinction between these two characters. This really happened! Confusing the letter "ell", l, and the digit 1, seems more natural and forgiving, but having character confusion like this in any kind of data is a reason to both chuckle and cry.

At other times we have to deal with corrupt data that is given to us in the form of a plain legacy **ASCII** text file. When that is the case, we may have to use a text editor to manipulate the text until we can extract columns of data. In the worst case scenario, a text editor with the ability to record and playback keyboard macros (we like **TextPad**), and our **CSV Editor** program, and an hour or more of painful manual labor may be required to get a large data set into a useable format. This is why we consider **CSV Editor** to be a utility program. The **CSV Editor** program contains over 175 functions for examining, clearing, comparing, copying, converting, decrypting, de-interlacing, deleting, differencing, duplicating, encoding, encrypting, exchanging, exporting, filling, filtering, formatting, generating, hashing, importing, interlacing, inserting, intersecting, launching, matching, moving, pasting, ranging, randomizing, replacing, replicating, re-shaping, rotating, scrambling, searching, selecting, sorting, splitting, unioning, testing for uniqueness, and validating data. Whew!

What the Program Is NOT

This program is intended to be a useful utility, but it is not a replacement for a spreadsheet nor is it intended to be used as any particular database application. You should not expect to use this program to replace more than just a few single-table applications, even though it may be possible to use it as such. We often create single-table backups as **CSV** files, because such files may make it much easier to recover from disk/computer failures, or to simply migrate data into a new or different database system.

It may be one thing to have database backup files in a proprietary format, or in network attached storage, but it can be quite another thing to actually recover from disk/computer failures using any backup system. Fully recovering from real computer disasters is almost always more complicated than just quickly copying some proprietary backup files. When it comes to recovering or migrating data, **CSV** files are most useful. With a **CSV** file you can at least easily see what your data looks like without having to log into a database system or having to open an application that may be impaired because its own data is somehow corrupt.

We didn't design the **CSV Editor** program for doing extensive routine editing or doing routine maintenance. With the exception of book/document indexes and a few other single-table applications, such as financial schedules and custom reports, the program is primarily used to get your data into a more consistent and useful format. You should plan on taking some time to learn how to use this program. No program is pleasant to learn when you want to quickly accomplish what you think should be a simple and direct task. The functionality you need may exist in several forms. Finding the exact functionality, and learning to use it efficiently, may be another matter.

Beginning Background

When you view a **CSV** file in a simple text editor program, like **NotePad** (or **TextPad**), the data it contains does NOT appear uniform and it certainly does not appear as if it is arranged in columns. As an example of what the view of a raw **CSV** file appears like, consider the following:

```
"Best Actor","Best Actress","Best Director","Best Picture","Year","Tagged"  
"Burt Lancaster","Elizabeth Taylor","Billy Wilder","The Apartment","1960","T"  
"Maxmillian Schell","Sophia Loren","Robert Wise","West Side Story","1961","T"  
"Gregory Peck","Anne Bancroft","David Lean","Lawrence of Arabia","1962","T"  
"Sidney Poitier","Patricia Neal","Tony Richardson","Tom Jones","1963","T"  
"Rex Harrison","Julie Andrews","George Cukor","My Fair Lady","1964","T"  
"Lee Marvin","Julie Christie","Robert Wise","The Sound of Music","1965","T"  
"Paul Schofield","Elizabeth Taylor","Fred Zinnemann","A Man For All Seasons","1966","T"  
"Rod Steiger","Katherine Hepburn","Mike Nichols","In The Heat Of The Night","1967","T"  
"Cliff Robertson","Katherine Hepburn","Sir Carol Reed","Oliver","1968","T"  
"John Wayne","Maggie Smith","John Schlesinger","Midnight Cowboy","1969","T"  
"George C. Scott","Glenda Jackson","Franklin Schaffner","Patton","1970","T"  
"Gene Hackman","Jane Fonda","William Friedkin","The French Connection","1971","T"  
"Marlon Brando","Liza Minelli","Bob Fosse","The Godfather","1972","T"  
"Jack Palance","Glenda Jackson","George Roy Hill","The Sting","1973","T"  
"Art Carney","Ellen Burstyn","Francis Ford Coppola","The Godfather Part II","1974","T"  
"Jack Nicholson","Louise Fletcher","Milos Forman","One Flew Over The Cuckoos Nest","1975","T"  
"Peter Finch","Faye Dunaway","John G. Avildsen","Rocky","1976","T"  
"Richard Dreyfuss","Diane Keaton","Woody Allen","Annie Hall","1977","T"  
"Jon Voight","Jane Fonda","Michael Cimino","The Deer Hunter","1978","T"  
"Dustin Hoffman","Sally Field","Robert Benton","Kramer vs Kramer","1979","T"  
"Robert De Niro","Sissy Spacek","Robert Redford","Ordinary People","1980","T"
```

When this same **CSV** text file gets loaded into the **CSV Editor** program, the data appears nicely arranged into distinct columns as shown in the figure on the next page.

While there are rules for creating **CSV** files, those rules are sometimes loosely applied. We give a list of 15 rules at the end of this help file, but you don't need to know those rules to use the **CSV Editor** program. Trying to learn or understand those rules now would only confuse most users. Our 15 rules are excellent guidelines, but not everyone consistently applies the rules. The world would be a simpler and much better place if everyone creating **CSV** files followed these rules!

In the figure shown on the next page, the raw data appears uniform because the columns have been separated into distinct vertically-aligned cells. In the next figure, the first yellow row is used to number the columns. The second yellow row contains header titles for the columns. The data row numbers are the numbers in the yellow column on the left. You cannot edit any yellow-background cells. Only the white-background cells of the grid are available for editing.

The screenshot shows a CSV editor window titled "CSV Editor - AcademyAwardsFirstImage.csv". The menu bar includes File, Blocks, Columns, Rows, Conversions, Utilities, Validation, Uniqueness, View, Options, and Help. The toolbar features various icons for file operations and data manipulation. The main area displays a table with 21 rows of data. The columns are labeled "Column 1" through "Column 6". The first row contains header-like text: "Best Actor", "Best Actress", "Best Director", "Best Picture", "Year", and "Tagged". The subsequent rows list individual winners and their films, such as Burt Lancaster for Best Actor in 1960 for "The Apartment" and Elizabeth Taylor for Best Actress in 1960 for "The Apartment". The last row lists Robert De Niro and Sissy Spacek for Best Picture in 1980 for "Ordinary People".

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Best Actor	Best Actress	Best Director	Best Picture	Year	Tagged
1	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	1960	T
2	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	1961	T
3	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	1962	T
4	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	1963	T
5	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	1964	T
6	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	1965	T
7	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	1966	T
8	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	1967	T
9	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	1968	T
10	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	1969	T
11	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	1970	T
12	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	1971	T
13	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	1972	T
14	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	1973	T
15	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	1974	T
16	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	1975	T
17	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	1976	T
18	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	1977	T
19	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	1978	T
20	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	1979	T
21	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	1980	T

It is an option for any **CSV** file to have its first row treated as a header row or not. This option is easily and immediately controlled by clicking either the **H+** or **H-** button near the far right of the toolbar. The caption on this button reflects the current state of this option as being turned on or turned off.

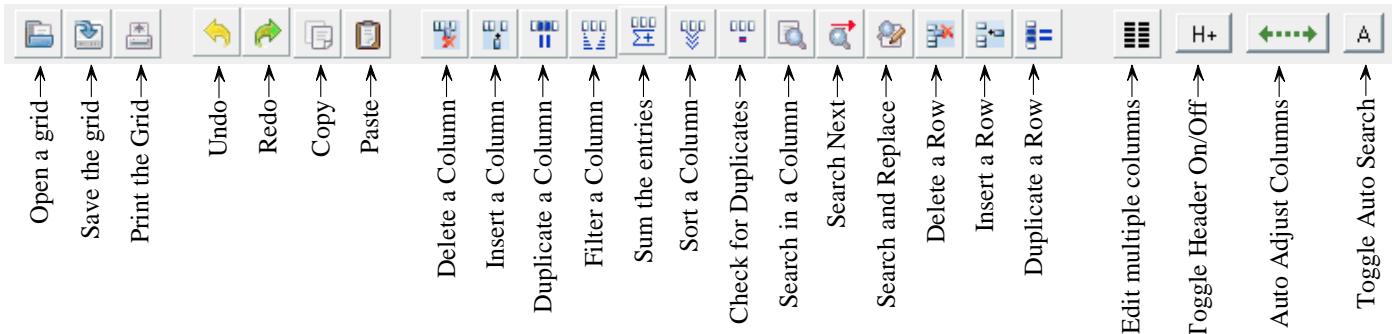
There are two special characters that appear as in the very upper-left corner of the above example grid.

If you click on the first of these, , the cursor will be landed in the very upper-left corner of the entire grid. If you click on the second, , the cursor will be landed on the very last row of the entire grid and in the last column on the right. Thus clicking these two special symbols provides a quick way to move to the very beginning or the very end of all your data.

When the first data row is not displayed as a header row, the lower of these two special symbols will appear as: . You can still click the symbol to quickly move to the very end of all the data. If you hold down the **CTRL** key on your keyboard while you click either of these special characters, the program will select the block of cells that stretches from your current position in the grid to either the upper-left or lower-right corner of the grid. A quick method to select all cells in the grid is to just click the symbol and then hold down the **CTRL** key on your keyboard and finally click either the symbol or the symbol.

The Toolbar Buttons

The **CSV Editor** program has a 24-button toolbar that appears as:



Using a toolbar button is always faster than pulling down a menu item. If you slowly move your mouse across these buttons (without clicking any button) you will see a little tool-tip popup hint that tells the purpose of each button. The status line at the bottom-left of the program window will show the same hint.

The first three buttons on the left of the toolbar are used to Open or Save or Print a grid. The next two toolbar buttons are the Undo and Redo buttons which only show when there is something to be undone or redone. When you first open a file you won't see either of these two buttons. Otherwise, at most one of these two buttons is ever shown at any time. The next two buttons you see are the Copy and Paste buttons.

The next ten toolbar buttons are all related to column operations. In order, these buttons are used to: (1) Delete a column or (2) Insert a column or (3) Duplicate a column or (4) Filter on a column or to (5) Sum and Average the entries in a column or to (6) Sort a column or to check for (7) Duplicates in a column or to (8) Search or (9) Search Next in a column or to do (10) Search and Replace in a column.

The next three buttons are the only toolbar buttons concerned with rows. These three buttons allow you to Delete a row or Insert a row or Duplicate a row.

The fourth-to-last button is used to edit multiple columns at once.

The third-to-last button is used to toggle whether the first row in the grid is displayed as a Header row or not. This will make that row's background color either yellow (on) or white (off). This button performs the same operation as a menu item under the **View** menu. Having this button in the toolbar saves you from having to open the **View** menu.

Just remember that when the header row is turned on and its background is colored yellow, then that row is not editable, and it is not a numbered row. When the header row is turned off and it has a normal white background, then that row is numbered as row 1 and then you can edit any cell in that row and that numbered row can participate in all editing operations that automatically operate using a range of rows, assuming row 1 is in the selected range. Regardless of the state of the header row, that row will be saved as the first row in the saved file whenever the grid contents get saved to a file.

The second-to-last button  is used to automatically adjust the display widths of all columns for display purposes. This button also performs an operation that is duplicated by a menu item under the **View** menu. You will find using this toolbar button is much faster than opening the **View** menu. There is more information about setting column display widths near the end of this help file where the **View** menu is discussed.

The rightmost or last button  is used to automatically toggle the **Auto Search Window** on and off. When turning this special control tool on, this button performs an operation that is duplicated by a menu item under the **View** menu. The **Auto Search Window** is used to quickly find an item in a column by locating the first cell whose first character matches the button you pressed. The search direction can be bi-directional, as explained near the end of this help file in the topic that fully discusses how to use the **Auto Search Window**.

Quickly Selecting A Row Or A Column

You can quickly select any row or any column by holding down the **CTRL** key on your keyboard and then just left click your mouse on any yellow column number label or any yellow row number label. After that you may choose any operation that operates on a row or column or any block of cells. As an example, you might next select the **Blocks** menu **Block Formatting...** to further perform particular formatting on a selected block.

We highly recommend you select a column or a block or a row before you try to perform any kind of an operation that requires selecting a column or a block or a row. When you select a column or a block or a row before you perform an operation, you normally should not have to set any row or column ranges after that because the corresponding dialogs should automatically fill those parameters for you. So pre-selecting either columns or blocks or rows saves time and effort.

Clipboard Functions & Special Keystrokes & Selecting Cell Blocks

The **CSV Editor** program has copy and paste functions related to blocks of cells. These special functions are accessible with menu items under the **Blocks** menu and they are also hidden features of your keyboard.

The Windows operating system has a clipboard that can be used when editing inside any single grid cell. You can use the special keyboard keystrokes **CTRL+C** and **CTRL+V** to do copy and paste operations when editing inside a single cell. When not editing inside a single cell, you can copy a selected block of cells to the Windows clipboard. Later, that block can be pasted into another Windows application such as Microsoft Excel or Word. That same block could also be pasted into another position within the same **CSV Editor** grid.

If you wish to copy and paste a block of cells then we suggest you first uncheck the **Options** menu item with the caption **Editor is always active**. Then to first select a block of cells, it is easiest to just left-click your mouse on the cell in the upper left of where you want to select, then drag the mouse to the right and down to extend your selection. If you extend too much, while still holding the left mouse button down, you can shrink the selection by dragging the mouse back up and/or to the left.

Some users may prefer to use the keyboard to select a block of cells. You can select a first cell by moving the selection position to the upper left corner where you want to start the selection. Then with the **Shift** key held down on the keyboard, use the down ▼, and right ► keyboard arrow keys to select more cells for the block. You will see all the selected cells change their background color as you select them.

After selecting all the cells for the block, press **CTRL+C** to copy those cells to the Windows clipboard. In fact, when using the keyboard the program will normally flash a message indicating the copy operation was successful. After copying, click on the cell for the upper-left corner where you want to paste the selected block. That single cell will change its background color.

Then you can press **CTRL+V** on your keyboard and the selected block of cells will be pasted into a new block. You will normally see another message flash that indicates the program has successfully done the paste operation. Note that these copy and paste functions execute when you let up on these keys on your keyboard. They do not execute when you first press these keys down. An option under the **Options** menu allows you to turn off the display of the automatic messages related to copying and pasting.

Mouse users can just click the paste button  in the toolbar to perform the paste operation.

When a block of cells are copied to the Windows clipboard, then on a cell by cell basis, only the first 2048 characters in any one cell get copied for that cell. In other words, the program limits the cell sizes when copying to the Windows clipboard. If any cell contains more than 2048 characters you will see a warning message identifying the cell whose data was deliberately truncated by the copy process. This should be rare, because having more than 2048 characters in one cell would be very unusual.

You should also know that if you select a large block of cells for pasting within the same grid, then when it comes time to paste, it is possible that parts of the block to be pasted will extend beyond the boundaries of the current grid. If that is the case, then the program will automatically expand the grid as needed to accommodate the new cells. However, the program will never allow more than 200 columns and it won't ask for permission to go beyond the 50,000 row limit. These grid expansion limits will only be a potential issue when the new paste position is near a right-most column or near a bottom-most row of an already nearly maximum sized grid.

The two buttons in the toolbar that appear as   can also be used to do copy and paste operations with the Windows clipboard, after selecting blocks of cells in the grid. These buttons are primarily for mouse users.

As already mentioned, if you hold down the **CTRL** key on your keyboard while you click either of the special symbols  or , the program will select the block of cells that stretches from your current position in the grid to either the upper-left or lower-right corner of the grid. A quick method to select all cells in the entire grid is to just click the  symbol and then hold down **CTRL** on your keyboard and click the  symbol.

There is another special keyboard/keystroke function you may find useful. When you select a block of cells (as described above) and you are not editing within a single cell, you can press the **DELETE** key on your keyboard to make empty strings in all those selected cells. This provides a quick way to clear a block of all character data while not actually removing the block of cells from the grid. Of course if you accidentally hit the **DELETE** key, you can click the undo button  or use the **Blocks** menu **Undo** command to correct your error.

If you **CTRL** click any yellow header column entry with the mouse (in the 2nd yellow row when you have the header row option turned on) the program will automatically move the grid selection to that column, but in the first data row, and the entire grid view will shift to show the first data row.

When using the keyboard, if you press **CTRL**+**▲** or **CTRL**+**▼** then the selection will move to the top or the bottom of the current column. If you press **CTRL**+**◀** or you press **CTRL**+**▶** you will move the selection to the start or end in the current row. The entire grid view will shift to show the selected cell. So these keystrokes allow you to quickly jump to the extreme top or bottom of a column or to the extreme left or right in a row.

The columns of any grid have a variable width, usually depending on the most-wide cell data in each column. Under the **View** menu there is a submenu item that allows the program to automatically determine the displayed width of all your columns. You can also just grab and drag a column header dividing line to re-size the display width of any column. This is the easiest way to expand or shrink the displayed width of a column.

It is also possible to manually set the display width of any column and not allow the program to automatically determine the width of that column as you make editing changes. You can learn about all the column-width display related functions in this help file by reading all the topics under the **View** menu.

About Our Supported Data Types

The **CSV Editor** program uses strings to represent all data in grid cells. This means the program uses strings as its native data type. However, this does not mean everything that looks like a string can only be used as a string. It just means every data object you need must be representable in a format that includes the ability to be converted to and from strings that fit in one grid cell. (2,048 maximum characters per cell) At the same time, this program provides a lot of functionality that goes beyond simple strings. Our block formatting, validation, and conversion functions assume most data is represented by one of the following five standard data types, all of which have multiple string representations. Cell data can always be converted to a different representation.

1. String data

We define a string to be a finite sequence of the printable ASCII characters that represents an entity that is usually something like a name or a description or a category. For example **Lincoln Blvd.** represents the name of a street while **magnetic tape** is more of a description. **The Sound of Music** is a string that could be both the name of a movie and a possible description of a movie's content.

2. Numeric data

Numeric data represents some form of a number. Numbers are used to measure and to quantify entities and can be added, subtracted, multiplied and divided. The most ubiquitous example of using numbers as a data type would be money. Although there are many different kinds of numeric data, numbers are normally thought of as representing floating point values or real numbers. Most numeric data will be in a simple decimal form, as in the string **3141.5926**, but other types of representations for numeric values are possible (e.g. scientific notation: **3.1415926E+03**).

3. Date data

A date is normally thought of as a way of representing either a particular day in history or an elapsed time period. Like numbers, dates can have multiple formats where **MM/DD/YYYY** and **YYYYMMDD** are two of the most popular and the two simplest. For example, **07/04/1776** is a string that could represent the original Independence Day for the United States. Every person in the modern world has a birth date, that is also sufficiently described by giving a year, and a month and a day. In some situations giving the year alone (think fiscal year) is sufficient to make a date time period and in other situations only a month may be sufficient to represent a period of time, as opposed to a particular day date.

4. Boolean data

A Boolean value represents entities that can have only one of two values. Usually those values are represented by the words **true** and **false**, but other representations like **yes** and **no** or like **0** and **1**, or **T** and **F** are also possible. Answers to questions like "**Are you married?**" or "**Did this check clear the bank?**" are normally encoded using some representation of Boolean data. Consistent representations are essential.

5. Time data

Time data is usually used to represent a specific moment in time as might be recorded by an ordinary clock, or an elapsed time. We represent times in the string format **hh:mm:ss** where **hh** represents hours and **mm** represents minutes and **ss** represents seconds. The number of hours may be more than 2 digits when the time is an elapsed time, not a clock time. Both minutes and seconds may be restricted to the range 0-59. Time data in the format **hh:mm:ss** really represents three whole numbers at once.

Each column in a **CSV** grid is expected to hold only one and the same type of data. Generally speaking, you never want to mix two different data types in a single column, unless you are prepared to manage such rows in either temporary or special ways.

Entire books have been written about data and data types and how to represent various entities in a computer. One entity or data type that cannot be stored in a **CSV** grid cell would a *.JPG picture image. But this does not mean the program can't display pictures, because it can. It just can't display a picture in a cell because cells only contain strings. To display a picture you can place the filename in a cell (a filename/filepath is a string of characters) and then select the menu item to **View Images Using Grid Cell Filenames** under the **Utilities** menu to display the corresponding picture.

An MP3 music file is another entity that the **CSV Editor** program does not handle natively, but like picture files, you can place the filepath of an MP3 file in a grid cell and then you can play that music file by using the **Launch a File From the Current Column** function that is under the **Utilities** menu. Using this same function you should be able to launch any other type of file that your computer can associate with any particular known file type or application. For example, video files in the *.MP4 format should be playable, similar to MP3 files.

To gain an appreciation of the broader range of data that the **CSV Editor** can sometimes use, we suggest you later look at the functionality under the **Conversions** menu and the **Validation** menu.

On the previous page we briefly introduced the numeric data type and suggested numbers will normally be stored in cells as decimal values. However, the **CSV Editor** program can convert between decimals and fractions, and it can convert between integers and Roman Numerals, and it can convert with integers in binary, octal, decimal, and hexadecimal formats. It can convert between ASCII characters and integers. It can convert times and dates between decimals and Julian Day Numbers. It can verify both Vehicle Identification Numbers and International Standard Book Numbers. So the program sometimes uses numbers for special and very specific purposes other than purposes associated with ordinary numeric decimal values. As another example of performing non-traditional numerical work, this program can sum a series of elapsed time values that are all in the **hh:mm:ss** format and it will return the final answer, with all three integers normalized, but in this same time format where the hours may be larger than 99. This special computation is not easy for a human to do!

This program also has an extensive set of mathematical functions found on scientific calculators that can be applied using what we call Math Expressions. In addition to Math Expressions, this program can perform Linear Regression and computes all the standard statistics related to a set of X,Y data pairs. In other words it can fit equations that determine the best mathematical relationship between any two numeric variables. It can also convert money values between any two types of currency. So when we speak about working with numeric data, this program accommodates much more than just ordinary decimal numbers. Roman Numerals and fractions are not one of the five standard data types, but they can be represented by simple strings. We can convert data to many other forms that are not one of our five standard data types. We can even convert data to binary Huffman codes. Cells can contain un-typed data because they hold strings of characters. A given string of characters (like a binary Huffman code value) can represent anything you want it to represent!

Whenever you edit any entry in any grid cell, whatever you type is what stays in the cell. In other words, there is no automatic formatting or validation performed on what you just typed when you press the **Enter** key. What you see is what you get. The program has no idea if you are trying enter a number, or a date, or a Boolean value, or anything else. So if you accidentally enter an invalid digit in a number or if you inadvertently enter an invalid date that looks legitimate, neither you nor the program will know that something might be wrong.

Formatting and Validating and Converting Data

We should briefly introduce and explain the uses and distinctions of three terms related to how we check and manipulate data. Full details are contained under menus given elsewhere in this document. This page just generally describes the purpose of each of the three functions: Formatting, Validating, and Converting.

Formatting data is a process we apply where we know what the input data type should be, and where we specify how we want to format that data. Our main formatting function is under the **Blocks** menu and has the menu title **Block Formatting....** As a couple of simple explanatory examples, we might format the string **THIS IS A TEST** as Camel Case so it becomes **This Is A Test**. We might format the number **2718281.828** so it represents the currency value **\$2,718,281.83**, accurate to the nearest penny. We might format the date **07/04/1776** so it looks like **1776-July-04** or **Thursday July 04, 1776**. We could format a Boolean value like **0** so it becomes **F**. Finally we could format the 24-hour military time value **15:36:42** so it looks like **03:36:42 PM** which is in a 12-hour format with an **AM/PM** indicator. Formatting data generally assumes the input data is one of our five standard data types, and formatting gives you the option to slightly change the format of the string, usually (but not always) without changing the interpretation of the actual value. Formatting is often used to insure all the data elements have exactly the same consistent formatting. All details related to formatting are under the **Block Formatting...** menu function and the **Block Formatting** dialog.

Validating data is a process where we also need to choose a specific input data type. However, different from formatting, validating data never changes the data. The purpose of validating data is to report whether the data is correct or not. Correct in this case means the data can be correctly interpreted. For numbers and dates and times, validating may also involve checking that all values lie within a given range. Usually we recommend using a **Report Column** when validating data. Unlike formatting, validating data is usually only applied to a single column at a time. When you validate data, the only outputs you see are messages either telling you all the data is Ok, or you will see individual error messages that sometimes try to suggest a way of correcting the invalid data in an individual cell. The **Uniqueness** menu also includes functions to check uniqueness of rows or columns or cells within a block. All validation functions and details are under the **Validation** menu.

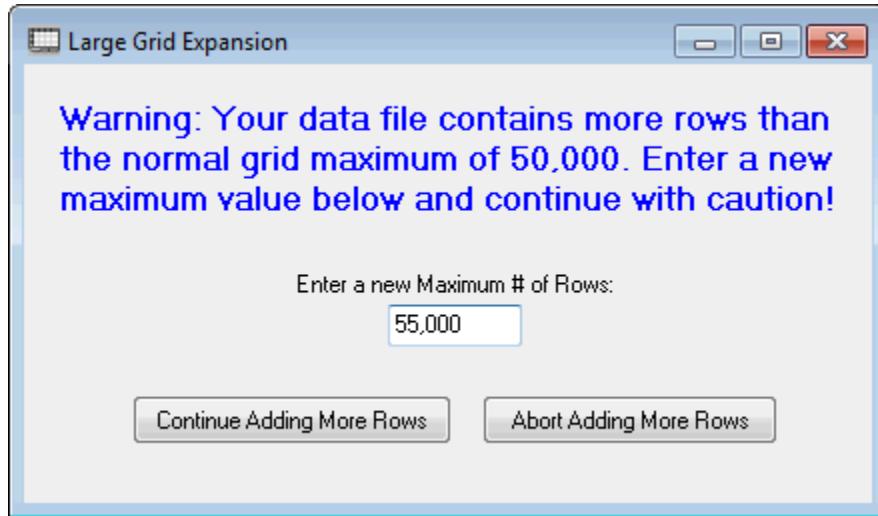
Converting data differs from both formatting and validating because converting can potentially result in a different data type. As an example, we can convert a time value like **03:36:42 PM** so it becomes a decimal number like **0.65048611111111** that might represent the same time value, but expressed and written as a decimal fractional part of a 24-hour day. When one is available, we generally recommend using a **Destination Column** when converting data. Another example would be converting the Roman Numeral **MCMLXII** to the integer value **1962**, or converting the integer **2012** to the Roman Numeral **MMXII**. You should understand that converting is intimately related to validating and formatting because the input value must be correct before it can be converted, and the output function must somehow know the desired output format. All conversion functions and details are under the **Conversions** menu.

It is only when you manually format a block of cells, or perform a validation test, or try to convert data to a different format, that checks are made for consistency and correctness. In all cases you will somehow be telling the program about the expected exact data types for both input and output (including types that are not one of our standard five types). Formatting a block of data cells may automatically change some of those cells. Validating data never changes the string data, but when you use a **Report Column**, you will see error messages and hints in a validation **Report Column** telling you what is potentially inconsistent and what deserves your attention. We generally recommend that you format your data first, then validate it second, and finally convert it, if necessary, last.

Basic Program Limitations

The **CSV Editor** program is a simple utility. It does have three limitations you may need to know. First, we have already mentioned that each cell is limited to having at most 2,048 characters. Second, the program will only read and use the first 200 columns of any **CSV** file. While it is certainly possible for a **CSV** file to have more than 200 columns, that would be unusual. The program simply ignores the data in any column beyond the 200th column.

Third, the program normally only reads about 50,000 lines from a **CSV** file. If the file contains more lines, the program will stop with a message like:



You can enter a number larger than 50,000, but you will have at least been warned of the possibility you could exceed your computer's memory capacity. If you need to work with a **CSV** file that has a very large number of lines, we suggest you split the file into smaller file segments, say in segments of 25,000 lines each, and then use the **CSV Editor** program to work on those smaller segments. This is a case where you may need to first use a regular text editor program like **Notepad** or **TextPad**, to help you split the file into segments.

If you do change the number in the above dialog box, you should know the program only uses that number while you are working on that particular grid. Each time you load a new **CSV** file the 50,000 line limit will be restored. Thus entering a new upper limit in the above dialog is only a temporary change in the limit. In the above example, we have entered a value of 55,000 that would allow reading an additional 5,000 lines over the normal maximum of 50,000.

When entering a new integer value, you can insert commas or spaces in the edit box as needed. However, the program will remove all commas and spaces from the string you enter before it tries to convert that string to an integer value.

The above dialog may appear whenever you perform any function that needs to expand the number of rows in the current grid, and the new number of rows would exceed the 50,000 row limit.

Starting the Program With a Command Line Parameter

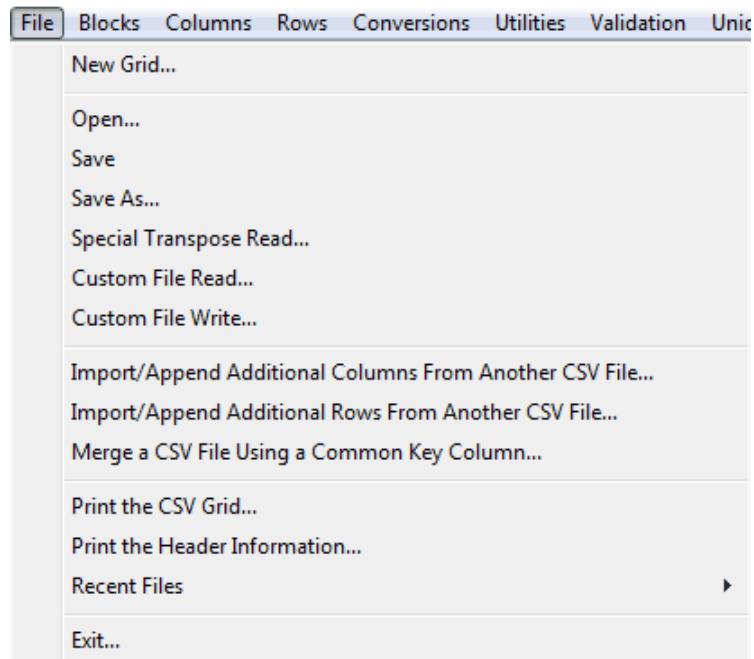
Normally this program will be started without giving it any command line parameters. However, it is an option to open **CSV Editor** with one command line parameter that is the fully qualified path to a text file that can be automatically opened for viewing as a **CSV** file.

The current version of the program only uses at most one command line parameter, but that parameter is optional and will probably be seldom used. In fact, the real purpose of allowing that one command line parameter is so this program can be automatically opened by calling it using a batch file.

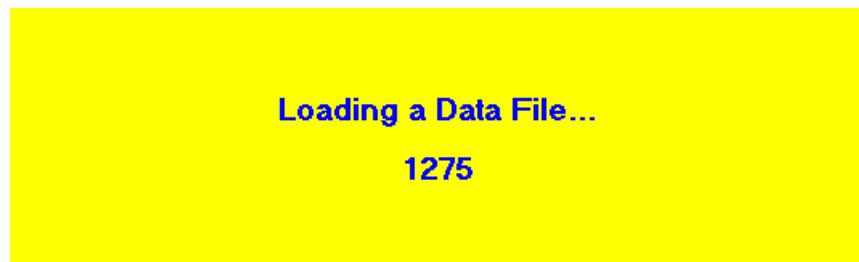
There are other startup options that get executed when the program normally starts without any command line parameter. See also the help topic about the **Options** menu that has a submenu item to **Set the Startup Options....** The **Startup Options Dialog** contains the details of how to setup and manage the program's normal startup options. When the program is started with one command line parameter then all the normal startup options that otherwise try to open a default file are by-passed. The only startup option that is retained is the one to restore the last program window size and window position after any file is opened on startup.

If you have read everything from page 1 up to this point, you should be ready to start experimenting with the program on your own. Feel free to skip around this help file and read about any topics that interest you. The devil is in the details and all the details are in the pages that follow.

The File Menu



The **File** menu contains the usual items to open and save **CSV** files. Whenever you open a large data file, usually one with thousands of lines and many columns of data, you may see a message that shows a progress counter that appears like:



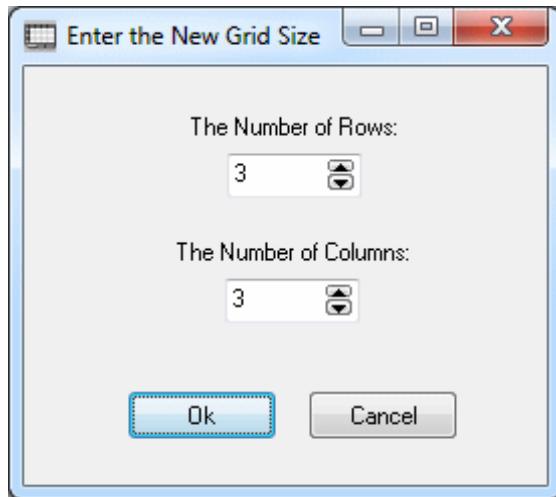
The rapidly changing progress counter indicates the number of lines currently read from the file. When opening most data files you may not see this message because it usually just flashes very briefly and then automatically disappears when the file has been completely read.

Philosophically speaking, the current version of the **CSV Editor** program only works with one data grid. If you have made any editing changes to the current grid at the time you try to open another file, you will first be asked if you would like to save the current grid before opening another grid. However, any grid can be expanded to contain the logical equivalent of two or more grids. This idea is discussed when you merge data from another file. In general however, there is only one opened grid at any one time. Unlike spreadsheets, we don't use multiple sheets. See also the topic **Copy Matching Data** that is discussed under the **Utilities** menu and see the two topics that deal with importing and merging files under the **File** menu.

There is a submenu near the bottom of the **File** menu, **Recent Files**, that will hold up to 15 of the most recently used or opened files. This provides a quick way to re-open any recently used file. Just select any file from this list that is automatically loaded and managed for you by the **CSV Editor** program.

Creating A New Grid

If you want to create a new grid without opening an existing grid then you should select the menu item **File | New Grid...** and you will be prompted to enter the initial grid size:

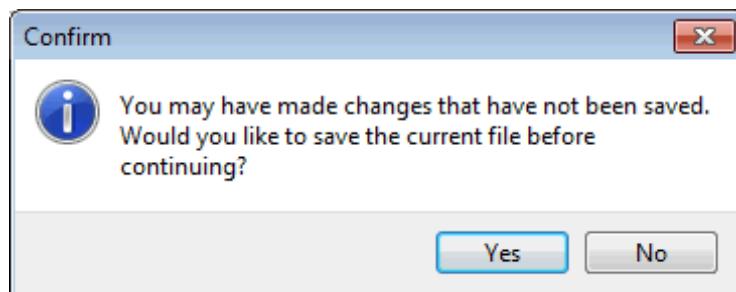


Normally the number of rows should be less than 50,000. The number of columns is limited to no more than 200 columns. You can always add more rows and columns up to these limits and you can also later delete rows and columns to make the numbers you need.

After you click the **Ok** button you will see a blank grid with the number of rows and columns that you specified and then you can begin entering new data in the blank cells.

After creating a new grid you can subsequently read in additional data from files that can be appended as additional rows or columns. The grid will automatically expand as required, up to the program limits.

In fact, if you have made any editing changes to the currently opened grid at the time you try to create a new grid, then you won't see the above new grid dialog until after you have the opportunity to save the existing grid. When you have pending edits or changes that have not been saved, you will first see a message like the following before you can enter the new grid size.



The Special Transpose Read Function

The **File** menu contains a special menu item with the title **Special Transpose Read...**. When you execute this function you will be asked to open a **CSV** file. What is different compared to the normal file **Open** menu item is that the file is loaded into the **CSV** grid so that what would normally be rows will be columns and what would have been columns will be loaded as rows. This is equivalent to having a transpose function if the **CSV** grid were considered to be the equivalent of a mathematical matrix. The transpose function converts rows to columns and converts columns to rows.

We should also mention that it is possible to perform the equivalent of a transpose operation using the **Block-Column Reshaping...** function under the **Blocks** menu. We won't explain how to do that here, but we do explain how to do it at the end of the discussion of the menu command **Blocks | Block-Column Reshaping....**. Using the **File | Special Transpose Read...** function is easier as long as your grid has an appropriate size.

As an example of the Transpose Read function, we will use one of our standard files that has the name **AcademyAwards.csv**. This file normally appears as:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maximilian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Palance	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

The next two figures below show the transpose of the original AcademyAwards grid.

Whenever the **Special Transpose Read** function is executed, the program always automatically turns off the option to show the first row as a header row. Below we can determine there are 6 rows and 22 columns in the transposed grid. Although it may appear as if the original grid had 21 data rows and 6 columns, the first row was displayed as a header row (see the figure on the previous page of this help file) and this fact explains why when the same file is loaded as a transpose then we end up with 22 columns. Note how the first vertical data column in the next figure contains what was originally the horizontal header row.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12	Column 13	Column 14	Column 15	Column 16	Column 17	Column 18	Column 19	Column 20	Column 21	Column 22
1 >	Year:	1960	1961	1962	1963	1964	1965	1966														
2	Best Actor:	Burt Lancaster	Maxmillian Schell	Gregory Peck	Sidney Poitier	Rex Harrison	Lee Marvin	Paul Sc														
3	Best Actress:	Elizabeth Taylor	Sophia Loren	Anne Bancroft	Patricia Neal	Julie Andrews	Julie Christie	Elizabe														
4	Best Director:	Billy Wilder	Robert Wise	David Lean	Tony Richardson	George Cukor	Robert Wise	Fred Zi														
5	Best Picture:	The Apartment	West Side Story	Lawrence of Arabia	Tom Jones	My Fair Lady	The Sound of Music	A Man														
6	Tagged:	T	T	T	T	T	T	T														

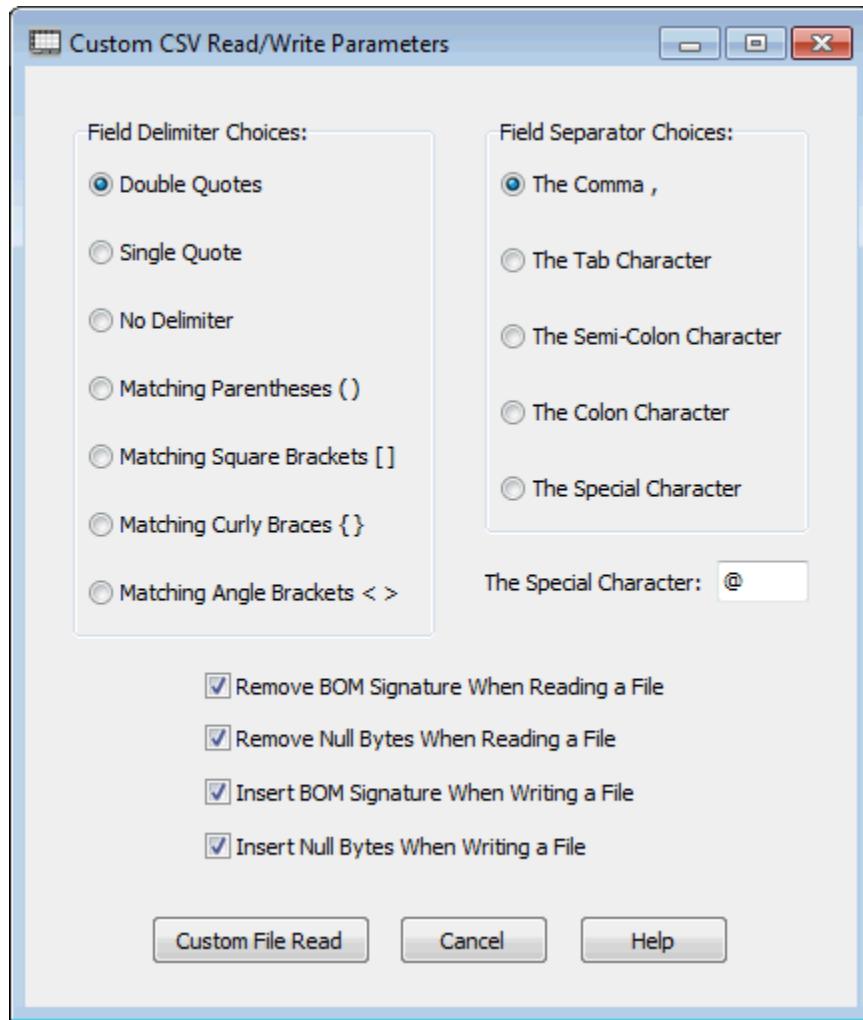
We next scroll to the right to show the last or right-most column of transposed data, Column 22.

<	Column 17	Column 18	Column 19	Column 20	Column 21	Column 22
1 >	1975	1976	1977	1978	1979	1980
2	Jack Nicholson	Peter Finch	Richard Dreyfuss	Jon Voight	Dustin Hoffman	Robert De Niro
3	Louise Fletcher	Faye Dunaway	Diane Keaton	Jane Fonda	Sally Field	Sissy Spacek
4	Milos Forman	John G. Avildsen	Woody Allen	Michael Cimino	Robert Benton	Robert Redford
5	One Flew Over The Cuckoos Nest	Rocky	Annie Hall	The Deer Hunter	Kramer vs Kramer	Ordinary People
6	T	T	T	T	T	T

There is one special restriction of the **Special Transpose Read** function. No more than 200 rows or columns will be created, no matter how many columns or rows are contained in the original CSV file. If you need to compute the transpose of the current grid, we suggest you save the grid and then re-load it using the **Special Transpose Read** function. If needed, then you can save the transposed grid as the new transposed CSV file. You could also export rows in batches or groups of 200 rows so you could later use the **Special Transpose Read** function. See the function under **Rows | Export Using Row Groups...** to do this automatically.

Custom Read-Write Parameters

There are two menu items under the **File** menu that perform custom read/write operations on what are supposed to be **CSV** files. The real problem is that there is no clearly defined or agreed upon format for **CSV** files. This is why we give a set of rules at the end of this help file. But for those times when you need to read or write a specially formatted superficial **CSV** file, you can select either **File | Custom File Read...** or **File | Custom File Write...** and you will see a dialog like the following. The purpose of this dialog is to allow you to define some flexibility in how fields are delimited and how fields are separated. It also allows you to read and write files using a particular form of **UTF-16** encoding related to Unicode characters. We will briefly explain some of the complexities of Unicode text files and **UTF-16** encoding at the end of this help topic.



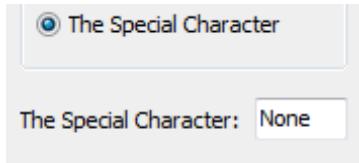
The caption on the button **Custom File Read** will appear as **Custom File Write** whenever you are actually saving or writing a file, instead of reading a file. But the same dialog and controls and their meaning and use will appear regardless of whether you are reading or writing a file.

Although it may appear as if all these controls are independent, we still need to specify exactly what all the choices really mean. We don't cover all the possible ways in which mal-formed **CSV** files are encountered in practice, but we do allow you to read and write files that don't conform to any standard.

As a simple example, sometimes databases write data using the semicolon character ; as a separator rather than a comma. Another example would be when a **TAB** character is used as a field separator character rather than a comma. We can usually work with such files, and many others as well, but we don't claim to work with every possible custom format that someone invents on their own.

In terms of field separator choices, we allow you to set any special character in the edit box, provided you also specifically choose **The Special Character** radio button.

If you make any choice for a **Field Delimiter** that involves matching character pairs, then the program will automatically set the **Field Separator Choice** controls as



where the word **None** means no special field separator character is actually used. Otherwise, if you enter more than one character in the edit box, and you also select the radio button for **The Special Character**, then the program will only use the first character that you enter. If you enter an empty string the program will automatically replace your empty string with a single comma character, which may not be special, but at least it is a character!

We will discuss the last four check boxes at the end of this help topic.

When you click the button with the caption **Custom File Read** or **Custom File Write** you will next be prompted by a corresponding file open or file save dialog where you must select the file to be read or written. After you select the file, the program will perform the file read (open) or file write (save) operation.

We will try to give a few basic but non-conforming examples. It is probably easiest to begin by describing the **Field Delimiter Choices** that involve matching pairs of characters. Just remember that you can try setting and using many possible parameters, but we don't guarantee all combinations will solve your problem when you have a file that does not conform to our standard rules.

First, let's assume we have a standard but simple **CSV** file that has only 1 line, but that line is correctly and fully formatted according to our rules.

```
"Best Actor","Best Actress","Best Director","Best Picture","Year"
```

Now we will make some changes to this line to show what some non-conforming files might look like. The next four lines are examples of using matching character pairs. Note that in all four examples, there is no separator character used at all!

```
(Best Actor)(Best Actress)(Best Director)(Best Picture)(Year)  
[Best Actor][Best Actress][Best Director][Best Picture][Year]  
{Best Actor}{Best Actress}{Best Director}{Best Picture}{Year}  
<Best Actor><Best Actress><Best Director><Best Picture><Year>
```

The one rule the above four examples all adhere to is that none of the field data is allowed to contain any paired matching character. In fact, all of the above examples would work if there were a comma or a space or any other characters between the fields as in:

```
(Best Actor),(Best Actress),(Best Director),(Best Picture),(Year)
(Best Actor) (Best Actress) (Best Director) (Best Picture) (Year)
(Best Actor) @ (Best Actress) @ (Best Director) @ (Best Picture) @ (Year)
```

We ignore any and all characters between fields. This means you have a lot of flexibility when you try using matching paired characters. In fact, all of the previous seven examples would discard characters not considered parts of real data and would load a grid so it looked like the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Best Actor	Best Actress	Best Director	Best Picture	Year

In all other cases where you don't choose matching paired characters for a **Field Delimiter Choice** then the program will always expect to use a single **Field Separator** character. The choice between **Double Quotes** and **Single Quote** should work where you can literally trade these two kinds of quote characters. Our program already handles the case where **No Delimiter** is used, as long as the comma is used as a field separator. This is what our normal **CSV** file read/write functions automatically handle (including optional delimiter double quotes).

When it comes to writing files, we try to honor your parameter choices in the above dialog box. For example, if you saved our standard **Academy Awards** file without a **Field Delimiter** and with a semi-colon for a **Field Separator Character** then the output would look like:

```
Best Actor;Best Actress;Best Director;Best Picture;Year;Tagged
Burt Lancaster;Elizabeth Taylor;Billy Wilder;The Apartment;1960;T
Maxmillian Schell;Sophia Loren;Robert Wise;West Side Story;1961;T
Gregory Peck;Anne Bancroft;David Lean;Lawrence of Arabia;1962;T
Sidney Poitier;Patricia Neal;Tony Richardson;Tom Jones;1963;T
Rex Harrison;Julie Andrews;George Cukor;My Fair Lady;1964;T
Lee Marvin;Julie Christie;Robert Wise;The Sound of Music;1965;T
Paul Schofield;Elizabeth Taylor;Fred Zinnemann;A Man For All Seasons;1966;T
Rod Steiger;Katherine Hepburn;Mike Nichols;In The Heat Of The Night;1967;T
Cliff Robertson;Katherine Hepburn;Sir Carol Reed;Oliver;1968;T
John Wayne;Maggie Smith;John Schlesinger;Midnight Cowboy;1969;T
George C. Scott;Glenda Jackson;Franklin Schaffner;Patton;1970;T
Gene Hackman;Jane Fonda;William Friedkin;The French Connection;1971;T
Marlon Brando;Liza Minelli;Bob Fosse;The Godfather;1972;T
Jack Lemon;Glenda Jackson;George Roy Hill;The Sting;1973;T
Art Carney;Ellen Burstyn;Francis Ford Coppola;The Godfather Part II;1974;T
Jack Nicholson;Louise Fletcher;Milos Forman;One Flew Over The Cuckoos Nest;1975;T
Peter Finch;Faye Dunaway;John G. Avildsen;Rocky;1976;T
Richard Dreyfuss;Diane Keaton;Woody Allen;Annie Hall;1977;T
Jon Voight;Jane Fonda;Michael Cimino;The Deer Hunter;1978;T
Dustin Hoffman;Sally Field;Robert Benton;Kramer vs Kramer;1979;T
Robert De Niro;Sissy Spacek;Robert Redford;Ordinary People;1980;T
```

If your input file was not in a conforming format (according to our rules), and if you need that format for output, then you should write that file using only the **Custom File Write...** function. Do not use our file **Save** function.

We also suggest you look at the output using either a text editor or a hexadecimal editor to insure the binary output is as desired. Again, we can't make guarantees for every custom format that exists, but we think we cover a lot of non-standard formats. We do perform doubling up of quotes as needed if your data contains any quote character (double quote or single quote). However, you cannot append additional columns or rows using non-conforming data.

The only reason we provide our custom read and write functions is so you can benefit from all our other features and functions. We would prefer you convert your non-conforming data so it obeys our sensible rules and we encourage you to enforce those rules as best you can.

Some Background About ASCII and Unicode

Next we will discuss some issues related to Unicode. The **CSV Editor** program has been designed to work with a native text file format that can be described as 8-bit ASCII encoding for the English language on Windows computers. Even this simple statement has a lot of hidden complexity behind it because to fully appreciate it requires detailed knowledge involving subtleties and relationships with newer file formats. Some of those subtleties are related to 7-bit and 8-bit and even 16-bit characters.

So we should first try to describe ASCII characters and how we use them. Each ASCII character can be assumed to occupy 8-bits. Thus each character corresponds to one byte and vice versa, each byte corresponds to one character. This statement is also misleading because we haven't described what a character is and what a character is NOT! In any case, 8-bit bytes are associated with decimal numbers in the range from 0 to 255 inclusive. In practice we only use ASCII characters whose decimal number range is between 9 and 126.

All ASCII characters can be divided into three classes. The first class can be called *control characters* and these are 32 in number and are associated with decimal numbers in the range from 0 to 31. We only use three control characters in this range. Those are the horizontal tab character with decimal number 9, the line feed character with decimal number 10, and the carriage return character with decimal number 13.

The second class can be called the *printable* ASCII characters and these are associated with decimal numbers in the range from 32 to 126. These are 95 characters you can actually see on your screen and these are all characters that you can easily type on your keyboard. In distinction with the first class, you normally never see a character on a screen that is a horizontal tab or a line feed or a carriage return, even though keyboards have a tab key and an ENTER or RETURN key. Even pressing the RETURN key on your keyboard usually (but not always!) inserts the two 8-bit bytes with decimal values 10 and 13. You normally never actually see any of the 32 control characters because they are not printable characters made for viewing or printing.

The third class of ASCII characters are associated with decimal numbers in the range from 127 to 255. These are more nebulous than the first two classes we have described because a few of these may act like control characters while others might be considered printable. Characters in this third class require 8 bits whereas characters in the first two classes can be written using only 7 bits. On the original IBM PC some of these characters were used for line drawing and others were used for some European language characters. At this stage most technical writers would discuss what are called *code pages*, but we will just say that decimal bytes in the range 127 to 255 can be associated with multiple interpretations which means they don't really have any particular interpretation unless you are willing to accept some kind of a default interpretation. We don't use any characters in the decimal range 127 to 255, but Unicode does, but Unicode is only one possible interpretation. In fact, Unicode was invented AFTER many previous interpretations were well established.

ASCII characters were not the first encodings used on computers to represent writing. ASCII characters only became established when different computer systems had to deal with each other. Both ASCII and Unicode are formal sets of agreements on how bytes in computers will be used. Think in the past when roads were used for two-way travel. Somehow, at some time, it became an agreement when people in the United States decided to drive on the right side of the road while people in England decided to drive on the wrong side! Driving works as long as people understand and agree to abide by the rules.

Unicode came about when there was a recognized need to expand the used character sets into one consistent functionality that could operate with all writing systems and computers in the world, including some make-believe characters. Before going further we will introduce some terminology that will aid in understanding some of the basic aspects of Unicode.

A **character** can be loosely defined to be the smallest component of writing that has a semantic meaning. In the English language, characters are put together to form words. Other languages (including math and Braille and Chinese and Hebrew) also group symbols together to form words and ideas. However, **A CHARACTER IS NOT A SYMBOL** even though characters can be represented by symbols. This idea is really very simple and most people should have no trouble understanding it. A math analogy will help with this abstract idea. Suppose I decide to write the number five. For example, the following are nine examples of writing symbols on paper to represent the number five.

$$5 \quad 8 - 3 \quad \frac{10}{2} \quad 35 \div 7 \quad \sqrt{25} \quad \text{cinco} \quad \text{cinq} \quad \text{V} \quad \begin{array}{|c|c|c|}\hline | & | & | \\ \hline\end{array}$$

The number five is an abstract idea. **NONE OF THESE REPRESENTATIONS ARE THE NUMBER FIVE**. Whether it is written in a math language, or the Spanish language or the French language or as a Roman numeral, or as a set of stick figures, the concept of the number five is independent of any one of its many written representations. The same is true of characters. The Unicode system represents abstract characters chosen from all modern and ancient languages, and also includes special technical and math symbols and many other diverse types of characters and non-characters.

A **glyph** is a printed symbol that represents something in particular. That something may be a character or a word or an idea or an object. The following is a glyph that represents a unisex bathroom!



A **font** is a collection of glyphs that usually (but not exclusively) represents characters.

In Unicode a **character repertoire** is an unordered set of characters that would be used together for some common purpose. Character repertoires can be grouped and combined to form large collections. Examples of character repertoires might include the English and Latin and Greek alphabets, or a large set of technical or math symbols, or it might be the ideographic characters used in Chinese, or Japanese or Korean. Unicode contains many different character repertoires.

In Unicode a **character encoding** is a 1-1 mapping of a character repertoire into the set of nonnegative integers. The unsigned integer associated with a character is called the character's **code point**. The range of the code points of a character repertoire is called a **code space**.

A fundamental idea about Unicode is this: *Unicode does not encode glyphs. Unicode only encodes abstract characters.* Think of the letter A in the English alphabet as being like the number 5. The first letter of our alphabet is an abstraction of all the following glyphs:

A A A A A A A A A A

None of the above glyphs are the **character** A. The A character is the abstract semantic unit that they all represent and it is that abstract idea that is the Unicode character A. Sometimes this is confusing because when you read what I write (the letter A) I have to write a particular glyph for the A character. That requires using a particular computer with a particular operating system with a particular word processing program that uses particular fonts to display particular characters. This involves using particular bits and bytes!

When moving from ASCII to Unicode, a difference is that Unicode characters are described using the code point terminology and a special notation with hexadecimal characters. A Unicode character can be specified by writing the symbolism U+HHHHHH where the U+ represents Unicode and the H's are the required hexadecimal characters. The number of H's is usually between a minimum of 4 and a maximum of 6. The nonnegative integers associated with Unicode characters run from decimal 0 to decimal 1,114,111. The largest integer is a 21-bit value. In hexadecimal notation the range is U+0000 to U+10FFFF.

The individual letters in the message **Hello World** could be described in Unicode by writing

U+0048 U+0065 U+006C U+006C U+006F U+0057 U+006F U+0072 U+006C U+0064

However, and it is a big however, Unicode characters are NOT directly tied into bits and bit patterns, even though we normally think that integers are tied to hexadecimal numbers and hexadecimal numbers are easily tied to bits and bit patterns. The magic isn't in the numbers. The magic is in what the numbers represent and in how those numbers get represented in the memory of any particular computer system. Not all computer memory systems are created equal as will be discussed below.

We have not discussed surrogate characters nor combining characters that are sometimes used with Unicode files. The real reason is that our applicable code point values are all in the range between decimal 9 and decimal 126. Thus surrogate pairs and combining characters should never be an issue for this **CSV Editor** program.

We won't delve into all the complexities associated with Unicode, but we must discuss two key concepts that form the large conceptual difference between using ASCII and using Unicode.

1. Endian-ness
2. Unicode transformations.

When a message like the above **Hello World** message is written on a computer system, the Unicode code points must somehow map into bits and bytes and words in a computer's memory. Even more, when a message like this is stored in a file, the bits and bytes in that file must also represent Unicode code points. When that same file needs to be shared among different computer operating systems all over the world, we must somehow take into account some fundamental differences of computer architectures. These concepts are nowhere present in using ASCII. Remember, the whole idea of Unicode goes beyond having a commonly used system for representing characters associated with different languages. We are constantly sharing that information between disparate computer systems. Never forget that Unicode code points generally require 21-bits.

Endian-ness

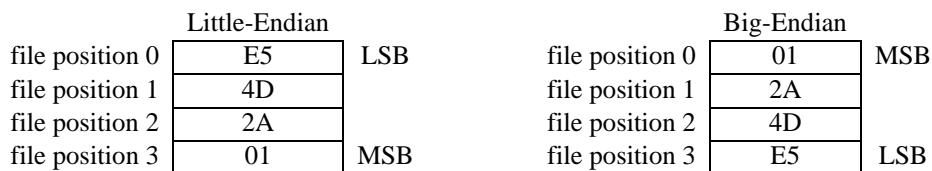
Computer systems based on Motorola or PowerPC processors store and use both 16-bit and 32-bit integers with a different byte order than Intel processors. Inside a given machine the byte order is not relevant because integers are read as complete integers, and the byte storage order is not an issue. However, when a 16-bit or a 32-bit integer is written to a file, there is a process called serialization that does depend on the byte order.

We will give a 32-bit example that should help explain the concept called endian-ness. When written on paper, the decimal integer 19,549,669 consists of eight decimal digits. Converted to hexadecimal notation this value would appear as 01 2A 4D E5 where each two hex digits comprise one byte. A single hex digit by itself represents 4 bits. We can put the hexadecimal notation for the bytes in boxes and write

MSB	LSB
01	2A 4D E5

where **MSB** stands for **Most Significant Byte** and **LSB** stands for **Least Significant Byte**. This hexadecimal format corresponds to one way to write any 32-bit integer on paper. Reading from left-to-right, the most significant parts come first and the least significant parts come last. When written on paper in hex, the 32-bit value 01 2A 4D E5 is the same whether it gets stored in either a Motorola or an Intel type of computer.

The *endian* distinction only comes into play when a multi-byte word must be broken down and written out to a disk file. There are two ways the four bytes of a 32-bit integer could be serialized when written to a disk file. You could write the four bytes from the LSB to the MSB, or you could write them from the MSB to the LSB. In the next figure we show the difference. Regardless of the endian-ness of the computer, the file bytes would get written in the order of file position 0 followed by file position 1 followed by file position 2 followed by file position 3. The reading of the file also starts with position 0 and progresses to position 3.



The terms **Little-Endian** and **Big-Endian** are used to describe these two different orders. Sometimes these terms are abbreviated by writing **LE** and **BE**.

When it comes to writing text files on a disk, there is a standard way of describing the endian-ness of the Unicode character code points that are in a file. Written in hexadecimal notation, the first two bytes of a Unicode text file can be either U+FFFE or they can be U+FEFF. The code point U+FEFF represents the Unicode **Byte Order Mark** also abbreviated as the **BOM**.

An agreed upon standard is that a Big-Endian machine will first write U+FEFF while a Little-Endian machine will first write U+FFFE. Taken together, these two bytes FE and FF form what is called a signature for a Unicode text file. When a given machine reads these first two bytes it can distinguish the byte order of all the remaining 16-bit or 32-bit integers that are in a file. We should also say that the code point U+FFFE is not a printable or legal Unicode character. Thus when a machine sees these two bytes as the first two bytes in a file, that machine can know it should switch the order of all the remaining bytes that it reads and that represent either 16-bit integers or 32-bit integers.

UTF-8 and UTF-16 and UTF-32

This leads us to a discussion of what are called Unicode storage formats. The reason there are multiple storage formats for Unicode characters has to do with compactness, backward compatibility, and robustness of the encoding range. If you are new to storage formats then an analogy with how cans of Coca-Cola are generally made available in four different formats might help your thinking. For instance, you can have individual cans of Coca-Cola, or you can have a 6-pack or a 12-pack or you can have a whole case of 24 cans in one package.

All Unicode point codes have 21-bits that can be mapped into three different standard storage formats. The reason ASCII characters don't involve the storage format concept is because all ASCII characters fit in one byte that acts as a universal storage format and that format has no need for any representation beyond a single simple byte. Bytes are a natural storage format for any kind of data on any computer system.

The simplest of the Unicode storage formats has the name **UTF-32**. The mapping simply takes the 21-bits of any code point and adds 11 leading zero bits to make a full 32-bit value. That's all there is to mapping any Unicode character to a 32-bit value. We don't really use this format at all, but we wanted to mention it because even though it is used less often, or is rarely ever used, it does exist. Its main disadvantage is that it wastes at least 11 bits for each represented character. It consumes 4 bytes per character.

The next simplest storage format or mapping has the name **UTF-16**. We won't fully describe this format either, but we will say that if your code point values are between 0 and 65,535 then they directly map to at most 16 bits. Just think as if you wrote out the full 21-bit value, and you drop the 5 most significant bits which are 0 bits anyway. We do use this format, but we only use it for code point values that are less than 65,535 so we don't need to describe how this format maps code point values that are larger than decimal 65,535.

In our specific usage, each Unicode character consumes 2 bytes of memory. In other environments a code point may require 4 bytes of memory, or two 16-bit words, even when the storage format is **UTF-16**. So **UTF-16** does NOT mean that each character occupies 16-bits. Some characters require more than 16 bits, but if you restrict your character set, then it will likely be that none of your characters will require more than 16 bits. If you restrict your character set enough, none of your characters may require more than 8 bits, but in **UTF-16** all such limited 8-bit code point values will still occupy 2 bytes or 16 bits, where 8 of those bits may be zero bits anyway.

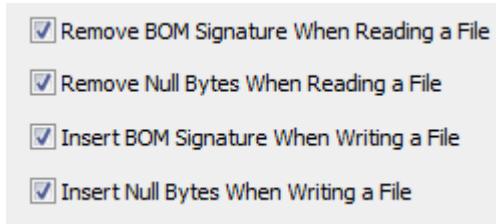
Finally we get to **UTF-8** which is a way of mapping a potential 21-bit value into a sequence of one to four potential 8-bit byte values. In our application, we never use code points above decimal 126. This means any character we are actually going to see and use, when written as a 21-bit value, will have its leading 13 bits be zero bits. This means we can just drop those first 13 zero bits and arrive at a single 8-bit integer or byte value. Other people describe this situation by saying Unicode code points at or below U+007F map directly to 7-bit ASCII values (really 8-bit) and vice versa. This is what allows most users of the English language to use **UTF-8** as if it were the same as 8-bit ASCII. However, 8-bit ASCII and **UTF-8** can be very different when you use code point values above 126 decimal. In **UTF-8** a single Unicode character may require 2 or 3 or 4 bytes.

We know there is much more that can be said about **UTF-16** and **UTF-8**, but we don't ever intend to use code point values larger than decimal 126 whenever we read or write Unicode text files. So we can consider ourselves to have graduated from using ASCII text files with 8-bit integers to a very limited form of Unicode text files using 16-bit integers, when we have to deal with text files using a limited form of **UTF-16** encoding.

Text File Line Endings

The three major computer operating systems are Windows, Macintosh, and Unix. The lines in a text file on a Windows machine are normally terminated by writing the two hex values 0D 0A. The lines in a text file on a Macintosh machine are normally terminated by writing the single hex value 0D. The lines in a text file on a Unix machine are normally terminated by writing the single hex value 0A.

Now that we have discussed ASCII and Unicode characters, and the **BOM**, and storage transformations, and text file line endings, we can discuss the purpose and use of the last four control check boxes that exist in our dialog for setting custom file reading and writing parameters. Those four check boxes appear as:



These four check boxes allow us to read and write Unicode text files that might otherwise be considered as foreign or very different from plain ASCII text file files. The text files that we can read and write are more fully described as being Unicode text files using **UTF-16** encoding with a **BOM** signature that specifies that they are **LittleEndian** where all the characters have code point values less than or equal to decimal 126 or U+007E, but also where each character is represented by a 16-bit integer, and where our line endings use two 16-bit words that are 0D 00 0A 00. Wow, that is quite a mouthfull!

We will first consider the grid that appears as:

<	Column 1	Column 2	Column 3	Column 4
>	Color Name:	Red Value:	Green Value:	Blue Value:
1	Black	0	0	0
2	Cyan	0	255	255
3	Magenta	255	0	255
4	Teal	0	128	128
5	Olive	128	128	0
6	Orange	255	127	0

When written as a **CSV** ASCII text file where there is no delimiter and where the separator character is the comma, the text file would look like:

```
Color Name:,Red Value:,Green Value:,Blue Value:  
Black,0,0,0  
Cyan,0,255,255  
Magenta,255,0,255  
Teal,0,128,128  
Olive,128,128,0  
Orange,255,127,0
```

When this same text file is viewed in its raw byte form, showing hexadecimal values, 16 bytes across on a line, it looks like:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	43	6F	6C	6F	72	20	4E	61	6D	65	3A	2C	52	65	64	20
00000010	56	61	6C	75	65	3A	2C	47	72	65	65	6E	20	56	61	6C
00000020	75	65	3A	2C	42	6C	75	65	20	56	61	6C	75	65	3A	0D
00000030	0A	42	6C	61	63	6B	2C	30	2C	30	2C	30	0D	0A	43	79
00000040	61	6E	2C	30	2C	32	35	35	2C	32	35	35	0D	0A	4D	61
00000050	67	65	6E	74	61	2C	32	35	35	2C	30	2C	32	35	35	0D
00000060	0A	54	65	61	6C	2C	30	2C	31	32	38	2C	31	32	38	0D
00000070	0A	4F	6C	69	76	65	2C	31	32	38	2C	31	32	38	2C	30
00000080	0D	0A	4F	72	61	6E	67	65	2C	32	35	35	2C	31	32	37
00000090	2C	30	0D	0A												

We are next going to show the raw bytes of this **CSV** file when it is written as a Unicode text file that has the **BOM** as a signature and that uses 16-bit integers to represent each character. This file was created using our **Custom File Write** function to save the above example grid. You might note that our line endings use the 0D 00 0A 00 or line feed and carriage return values with each of these occupying a full 16 bits.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	FF	FE	43	00	6F	00	6C	00	6F	00	72	00	20	00	4E	00
00000010	61	00	6D	00	65	00	3A	00	2C	00	52	00	65	00	64	00
00000020	20	00	56	00	61	00	6C	00	75	00	65	00	3A	00	2C	00
00000030	47	00	72	00	65	00	65	00	6E	00	20	00	56	00	61	00
00000040	6C	00	75	00	65	00	3A	00	2C	00	42	00	6C	00	75	00
00000050	65	00	20	00	56	00	61	00	6C	00	75	00	65	00	3A	00
00000060	0D	00	0A	00	42	00	6C	00	61	00	63	00	6B	00	2C	00
00000070	30	00	2C	00	30	00	2C	00	30	00	0D	00	0A	00	43	00
00000080	79	00	61	00	6E	00	2C	00	30	00	2C	00	32	00	35	00
00000090	35	00	2C	00	32	00	35	00	35	00	0D	00	0A	00	4D	00
000000A0	61	00	67	00	65	00	6E	00	74	00	61	00	2C	00	32	00
000000B0	35	00	35	00	2C	00	30	00	2C	00	32	00	35	00	35	00
000000C0	0D	00	0A	00	54	00	65	00	61	00	6C	00	2C	00	30	00
000000D0	2C	00	31	00	32	00	38	00	2C	00	31	00	32	00	38	00
000000E0	0D	00	0A	00	4F	00	6C	00	69	00	76	00	65	00	2C	00
000000F0	31	00	32	00	38	00	2C	00	31	00	32	00	38	00	2C	00
00000100	30	00	0D	00	0A	00	4F	00	72	00	61	00	6E	00	67	00
00000110	65	00	2C	00	32	00	35	00	35	00	2C	00	31	00	32	00
00000120	37	00	2C	00	30	00	0D	00	0A	00						

In the above format we can see the two-byte **BOM** signature at the beginning. We can also see the two bytes associated with each character. For example, in the first line the letter C is represented by the two bytes 43 00. Note that this is a 16-bit value where the 43 comes first. This is a **LittleEndian** representation of the decimal value 67. In hex, decimal 67 is 00 43. In **LittleEndian**, the least significant bits are stored first in the lower memory addresses while the most significant bits are stored later in higher memory addresses.

If we read the above file, but using our **Custom File Read** function where we set the parameters to have no delimiter and to use the comma as the separator character and to remove the **BOM** and remove all 00 or null bytes, then the file read would have looked liked the following (after we strip out the **BOM** and nulls).

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	43	6F	6C	6F	72	20	4E	61	6D	65	3A	2C	52	65	64	20	Color Name:, Red
00000010	56	61	6C	75	65	3A	2C	47	72	65	65	6E	20	56	61	6C	Value:, Green Val
00000020	75	65	3A	2C	42	6C	75	65	20	56	61	6C	75	65	3A	0D	ue:, Blue Value:.
00000030	0A	42	6C	61	63	6B	2C	30	2C	30	2C	30	0D	0A	43	79	.Black, 0, 0, 0.. Cy
00000040	61	6E	2C	30	2C	32	35	35	2C	32	35	35	0D	0A	4D	61	an, 0, 255, 255.. Ma
00000050	67	65	6E	74	61	2C	32	35	35	2C	30	2C	32	35	35	0D	genta, 255, 0, 255.
00000060	0A	54	65	61	6C	2C	30	2C	31	32	38	2C	31	32	38	0D	.Teal, 0, 128, 128.
00000070	0A	4F	6C	69	76	65	2C	31	32	38	2C	31	32	38	2C	30	.Olive, 128, 128, 0
00000080	0D	0A	4F	72	61	6E	67	65	2C	32	35	35	2C	31	32	37	..Orange, 255, 127
00000090	2C	30	0D	0A													, 0..

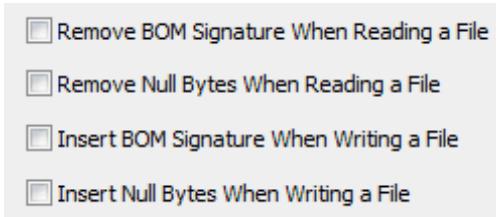
We need only compare this file with that shown near the top of the previous page. The file contents are identical!

This should help explain how we can think of turning a Unicode text file with a **BOM** signature and 16-bit integers (with limited range between 9 and 126 decimal) into an ASCII 8-bit text file by simply removing the **BOM** and removing all 00 or null bytes. The two previous examples also explain how, when we perform a **Custom File Write** function, we can actually write a file that is the same as a Unicode text file that begins with a **BOM** signature and that contains 16-bit integers (again with limited decimal range between 9 and 126). Internally, we really wrote an ASCII text file but we manipulated the lines by inserting null characters after every regular character. This is what creates the illusion of the **Little Endian** format.

We also had to custom write our line endings using the four bytes 0D 00 0A 00.

To effectively use our **Custom File Read** and **Custom File Write** functions requires that you first become familiar with the format of any Unicode text file that you are starting with and intending to read. Usually the only way to really know your beginning format is to view the raw file bytes using what we call a hexadecimal editor program. The above hexadecimal file examples were taken using a hexadecimal editor program named HxD. To obtain this editor visit the web site www.mh-nexus.de.

For those times when you want to read and write normal ASCII text files without any concern for Unicode text, then you should uncheck all four check boxes so they appear as:



There are a few more technical details about ASCII characters you may wish to read about in the help topic **Conversions With ASCII Characters and Numbers** elsewhere in this help file.

This concludes the help topic with the title **Custom Read-Write Parameters**.

Importing and Appending From Other CSV Files

There are two menu items under the **File** menu that allow you to append data to the existing grid, by reading in all the new data from another **CSV** file. There are only two ways in which that additional data can be appended. You can choose to append the new data by adding additional columns to the existing grid, or you can append the new data by adding it as additional rows to the existing grid. Appending data is also called **Importing**. The two **File** menu item titles are:

File | Import/Append Additional Columns From Another CSV File...
File | Import/Append Additional Rows From Another CSV File...

Whether you import new columns or new rows, you will first be presented with a standard file open dialog in which you should choose the data file that contains the data to be imported and appended. The contents inside this file should of course contain regularly formatted **CSV** data. The number of rows or the number of columns in this data file do not have to be related to either the number of columns or rows in the current **CSV** grid.

The program will actually read the new data file twice, the first time it reads the data it remembers how many columns and how many rows are in that file. On the second read, the program fills in the new cells with the new data. Before the second read starts, however, the program will automatically append as many rows or columns of blank data as will be needed to fill out the newly expanded grid. The only limitations on the amount of expansion allowed is that the new grid must never exceed 200 total columns nor 50,000 total rows.

When adding columns, the new columns are added to the right of the existing grid. If the number of rows in the added columns is less than the existing number of rows, then as each column is added, the rows will be filled out with blank data to match the number of rows in the existing grid. If the number of rows in the added columns is more than the number of rows in the existing grid, then all rows in the existing grid will first be expanded with blank entries to match the final number of required columns.

When adding rows, the new rows are added at the bottom of the existing grid. If the number of columns in the added rows is less than the existing number of columns, then as each row is added, the columns will be filled out with blank data to match the number of columns in the existing grid. If the number of columns in the added rows is more than the number of columns in the existing grid, then all rows in the existing grid will first be expanded with blank entries to match the final number of required columns.

If you use the import function to add new columns from another **CSV** file, then you should also know about another special function under the **File** menu that also imports data as additional columns, but in a very special way. What is special is the use of a common key column that is in the file you have open, and also exists in the file to be merged. When matching key columns exist, then this special merge function will reorder the new rows and keep the new row data synchronized by using the matching key values. See also the menu item **File | Merge a CSV File Using a Common Key Column...**

If you use either of the two regular **File | Import/Append** functions we can summarize where and how new cells may get blank filled with four figures shown on the next page. The first two figures are related to appending columns. The last two figures are related to appending rows.

In these figures, the data cells in the original grid are colored blue. The green cells represent the appended CSV file data. The new cells that are automatically made blank are colored gray. In cases where the number of rows or columns of new green cells matches the existing blue cells, then no new gray cells will be made.

The new green cell columns have more rows than the original blue cells.

The new green cell columns have fewer rows than the original blue cells.

The new green cell rows have more columns than the original blue cells.

The new green cell rows have fewer columns than the original blue cells.

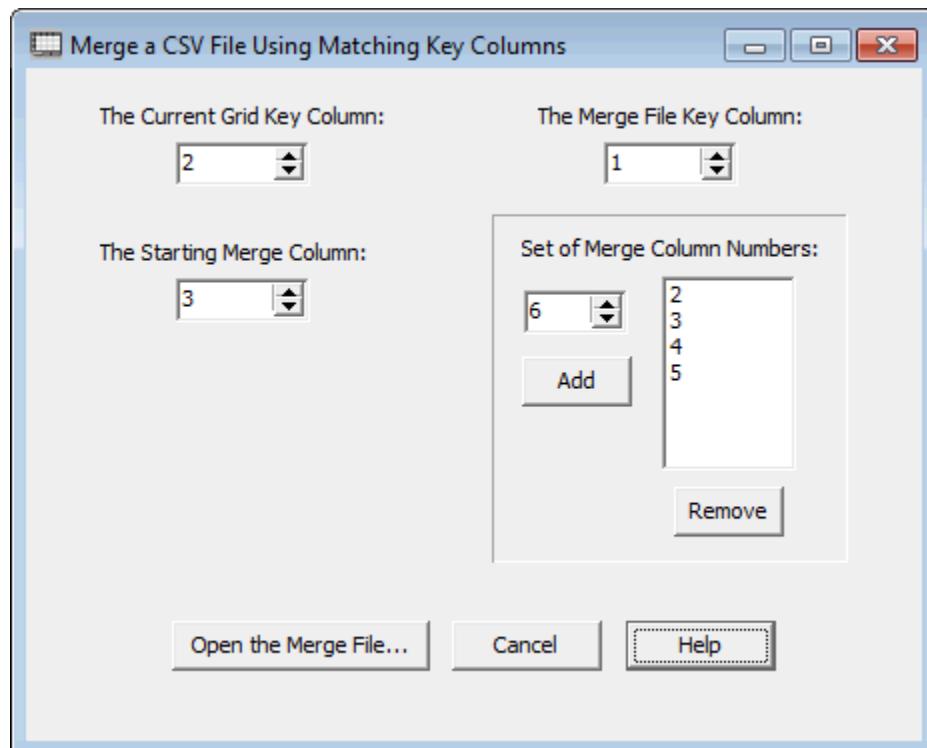
Merging CSV Files With Common Key Columns

You can merge one CSV file with another, provided the two CSV files share a common key column. The first example we will give involves the two existing grids shown below. The left grid is currently opened; the right grid is not currently opened. The right grid contains the data that is to be merged into the grid on the left.

<	Column 1	Column 2
>	Name:	SS Number:
1	Paula Boone	601-12-5515
2	Camila Gray	405-26-8423
3	Sheila Meadows	395-31-2131
4	Clint Burris	802-47-2761
5	Edith Leblanc	624-47-9928
6	Chris Rich	476-61-2002
7	Lydia Ramirez	738-60-9874
8	Benjamin Tanner	774-04-6277
9	James Russell	283-49-7186
10	Tony Walker	399-63-4919
11	Harold Ryan	571-78-8986
12	Lucas Tate	618-56-7242
13	Frank Perkins	632-85-2733
14	Wanda Avila	620-76-9563
15	Anthony Logan	684-21-9560
16	Jessica Cardenas	876-25-3643

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	SS Number:	Work Status:	Age:	Marital Status:	# of Children:
1	876-25-3643	Working	27	Married	3
2	774-04-6277	Retired	71	Divorced	2
3	405-26-8423	Retired	38	Married	1
4	618-56-7242	Retired	77	Single	0
5	738-60-9874	Retired	51	Single	3
6	395-31-2131	Working	36	Divorced	2
7	399-63-4919	Retired	61	Single	3
8	476-61-2002	Retired	45	Married	0
9	571-78-8986	Retired	48	Single	2
10	684-21-9560	Retired	64	Divorced	3
11	601-12-5515	Working	72	Married	1
12	632-85-2733	Retired	33	Married	2
13	624-47-9928	Working	56	Married	3

If we select the command **File | Merge a CSV File Using Key Columns...** we will see the dialog:

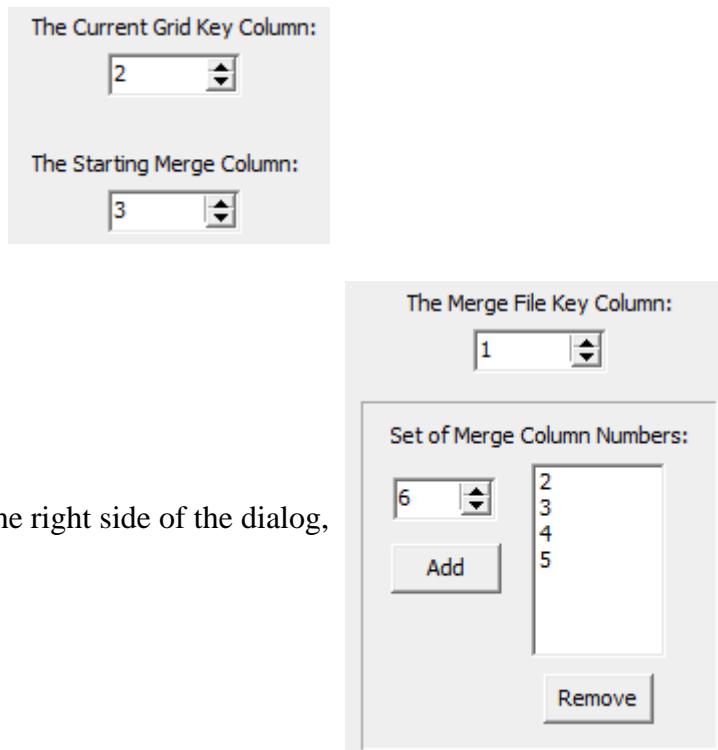


We need to explain how to set all the controls in this dialog. Before you begin, you must be familiar with the CSV file that contains the data that will be merged (the file on the right on the previous page) and you must open the CSV file that will receive the merged data (the file on the left on the previous page).

Look back at the previous page and note how **Column 2** in the grid on the left can be used as a key column that matches the type of information in **Column 1** in the grid on the right. These two grids contain a common key column, but also note that the grid on the right has a different number of rows than the grid on the left, and the two grids do not match in terms of the ordering of their rows. The grid on the left is our original opened grid that will be changed while the grid on the right contains the data that will be merged into the existing grid.

The first two controls you will normally set are those with the labels **The Current Grid Key Column** and **The Merge File Key Column**. These two controls contain the column numbers of the common key column, where **The Current Grid Key Column** is associated with the currently opened grid while **The Merge File Key Column** is associated with the file that contains the data that will be merged into the currently opened grid. The merge file will eventually be selected for use, using what appears as a file open dialog, although the merge file is not really opened and the merge file is an unseen file when this function is executed. Often, you will want to first open the merge file to see and write down the column numbers for the key column and the data columns that are to be merged. Then open the grid that will receive the new columns.

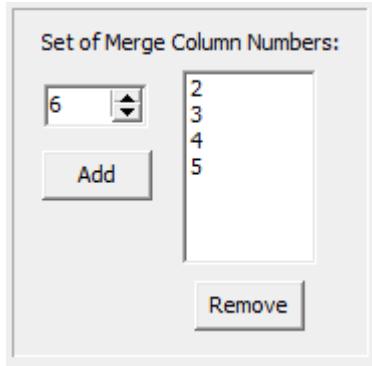
The next important concept is that the new columns that will be merged into the current grid, will be merged in consecutive columns, starting with the column that is identified by **The Starting Merge Column**. What can be confusing here is that all columns involved in this operation either refer to columns in the currently opened grid or they refer to columns in the file that contains the data that is to be merged. You need to know that **The Starting Merge Column** references a column in the currently opened grid. We can try to make the distinction between the two types of column references by stating that the two controls that appear on the left side of the dialog apply to the currently opened grid.



All the remaining controls that appear on the right side of the dialog,

to the file that contains the data that will be merged.

Thus you need to know the structure of the file that contains the data to be merged. The merged data columns are identified using the controls that appear as:



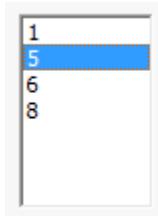
Remember these columns reference columns in the **CSV** file that contains the data that is to be merged. Using the **Add** and **Remove** buttons you can insert any number of required columns from the merge data file. Those numbers will normally be sorted, but you should not worry about the column ordering because you can rearrange all the columns after the merge is complete.

For our example, after we click the button to **Open the Merge File...**, and we select the proper merge file, then the current grid will change to the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Name:	SS Number:				
1	Paula Boone	601-12-5515	Working	72	Married	1
2	Camila Gray	405-26-8423	Retired	38	Married	1
3	Sheila Meadows	395-31-2131	Working	36	Divorced	2
4	Clint Burris	802-47-2761				
5	Edith Leblanc	624-47-9928	Working	56	Married	3
6	Chris Rich	476-61-2002	Retired	45	Married	0
7	Lydia Ramirez	738-60-9874	Retired	51	Single	3
8	Benjamin Tanner	774-04-6277	Retired	71	Divorced	2
9	James Russell	283-49-7186				
10	Tony Walker	399-63-4919	Retired	61	Single	3
11	Harold Ryan	571-78-8986	Retired	48	Single	2
12	Lucas Tate	618-56-7242	Retired	77	Single	0
13	Frank Perkins	632-85-2733	Retired	33	Married	2
14	Wanda Avila	620-76-9563				
15	Anthony Logan	684-21-9560	Retired	64	Divorced	3
16	Jessica Cardenas	876-25-3643	Working	27	Married	3

Note that columns 2,3,4,5 from the merge file have become new columns 3,4,5,6 in the above grid. You may also note that we did not edit the header information for the newly inserted columns. However, after the data is merged you can turn off the header display, then edit the header information, and finally turn the header display back on.

When using the list box of column numbers, you may need to delete a column number. To do that first click on the number in the list and note that number or row will be highlighted in blue. Then you can click the **Remove** button to perform the deletion.

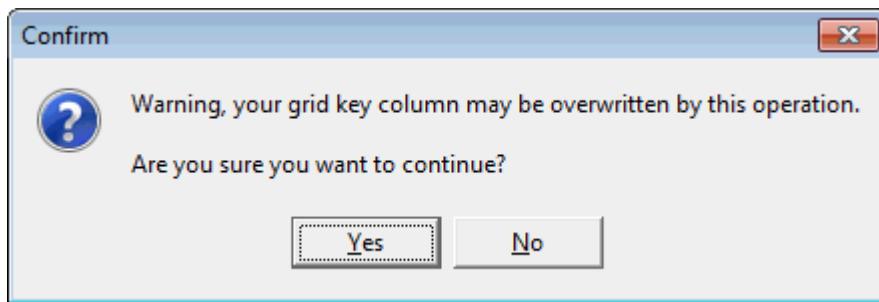


Otherwise, to add a column number to the list, first use the spin control above the **Add** button to dial in the desired number and then click the **Add** button to add that number to the list.

The way the program works when merging data is that each next line is read from the merge file. Then the just read key data info is searched for in the current grid, within **The Current Grid Key Column**, and if the key data is found in some row in the current grid, then the remaining columns from the merge file line are read into consecutive columns in the current grid, starting with the column that is **The Starting Merge Column**. If the key data is not found in **The Current Grid Key Column**, then no changes take place in the current grid. In other words, the only rows that change are those rows that have matching key data within the merge **CSV** file.

The current grid may be expanded by adding columns to accommodate the new data. This primarily occurs when **The Starting Merge Column** is one of the rightmost columns in the current grid. In fact, **The Starting Merge Column** can be set to any number between 1 and the number that is one more than the last column number in the current grid. However, **The Starting Merge Column** will usually be different from **The Current Grid Key Column**, but that is not a requirement. When these two numbers are the same, the key data column in the current grid will be overwritten. In fact, unless you make **The Starting Merge Column** one more than the last column in the current grid, then data in the current grid may be overwritten by the merge function. The current grid will automatically expand up to the maximum limit of 200 columns, but only as needed.

Whenever there is a possibility of **The Current Grid Key Column** data being overwritten by the merged data then you will first see a warning message like the following:



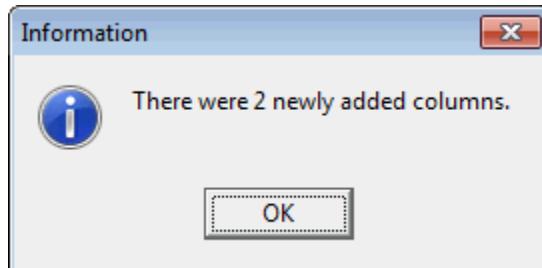
In normal use we recommend you set **The Starting Merge Column** to be one more than the last column number in the current grid. In this fashion, all the merged data will appear in newly created columns.

There are a few more comments we should make about this merge function. There is a possibility that the key data in the merge data file may contain empty strings. We consider this an error because key data should never be the equivalent of a null. In such a case, we never merge any new data unless the key data entry for a given row is not the empty string.

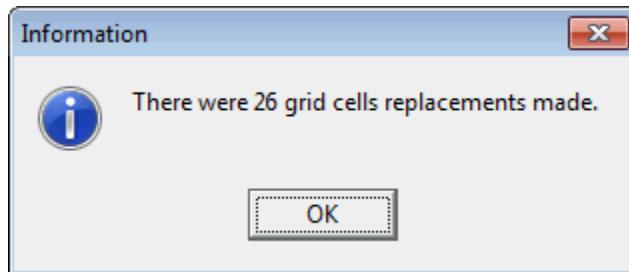
We also check that as each row is read from the merge data file, we never try to access data beyond the number of data columns read for that line. This is one case where if the merge data file were a mal-formed **CSV** file then we do no harm, or make no changes, to the currently opened grid.

It is also possible that in the setup dialog that you enter merge column numbers that are beyond the maximum number of columns read for a data row. In that case we also do nothing to the current grid.

When the merge function completes you may see a couple of information messages. If the merge function needed to insert any additional columns to the current grid then you may see a message like the following:



The merge function keeps track of how many individual cells were replaced with data from the merge file and thus you may also see a message like the following:



Sometimes the information returned by this last message can help you debug other errors in the current grid or in the **CSV** merge file.

We have given only one example for the merge function where it was used to add multiple columns from one **CSV** file into another **CSV** file.

One other common use of the merge function is to determine which rows in one grid appear with key data in another grid. In this special case, you can add a single column to the merge data file that can be filled with a constant letter like Z in all rows. Then add a single blank column to the opened grid. When you perform the merge function, you will use only the Z column from the merge data file to insert the letter Z into those rows in the opened grid that match with a row from the merge data file. In this manner you will know which key elements from the merge data file were found in the open grid. You could then sort the opened grid by the column that contains only blanks and the letter Z.

We will give one example by asking which numbers in Column 2 in the grid on the right on the next page appear in Column 2 of the grid on the left?

<	Column 1	Column 2	Column 3	Column 4
1 >	1/29/2012	978-1-449-30468-3	O'Reilly & Associates	Johnson, Clay
2	2/15/2012	978-0-691-14039-1	Princeton University Press	Bernstein, Dennis S.
3	2/15/2012	978-0-19-974044-4	Oxford University Press	Levitin, Anay; Levitin, Maria
4	2/15/2012	978-0-691-14714-7	Princeton University Press	MacCormick, John
5	2/23/2012	978-0-486-47883-8	Dover Publications	Michaelson, Greg
6	2/27/2012	978-0-465-01775-1	Basic Books	Stewart, Ian
7	3/2/2012	978-0-486-47417-5	Dover Publications	Pinter, Charles C.
8	4/12/2012	978-0-471-11709-4	John Wiley & Sons	Schneier, Bruce
9	4/14/2012	978-0-8218-4418-2	American Mathematical Society	Mullen, Gary L.
10	7/6/2012	978-0-691-15270-7	Princeton University Press	Cook, William J.
11	7/17/2012	978-0-691-14342-2	Princeton University Press	Havil, Julian
12	8/21/2012	978-0-321-62930-2	Addison-Wesley	Vickers, Andrew
13	9/13/2012	978-0-674-05755-5	Belknap Press	Lockhart, Paul
14	9/25/2012	978-1-118-46446-5	Wiley Computer Publishing	Upton, Eben; Halfacree, Gareth
15	10/4/2012	978-0-307-72095-5	Crown Publishers	Anderson, Chris
16	10/4/2012	978-1-59420-411-1	The Penguin Press	Silver, Nate
17	10/4/2012	978-1-59184-492-1	Portfolio/Penguin	Steiner, Christopher
18	10/5/2012	978-0-984-72511-3	Digital Frontier Press	Brynjolfsson, Erik; McAfee, Andrew
19	10/11/2012	978-1-118-20413-9	Wiley Computer Publishing	Thurrott, Paul; Rivera, Rafael
20	10/15/2012	978-0-321-88691-0	Peach Pit Press	Revell, Jeff
21	10/23/2012	978-1-285-42457-6	Cengage Learning	Busch, David
22	11/19/2012	978-0-07-180783-8	McGraw-Hill	Monk, Simon
23	11/23/2012	978-0-470-40765-3	John Wiley & Sons	Kjellstrom, Bjorn; Elgin, Carina
24	12/8/2012	978-1-56881-721-7	CRC Press	Wapner, Leonard
25	12/27/2012	978-0-691-14892-2	Princeton University Press	Van Brummelen, Glen
26	12/28/2012	978-0-7484-0304-2	CRC Press	Snyder, John; Bugayevskiy, Lev
27	1/15/2013	978-0-691-15271-4	Princeton University Press	Gray, Jeremy
28	1/15/2013	978-0-321-81958-1	New Riders Publishing	Kelby, Scott

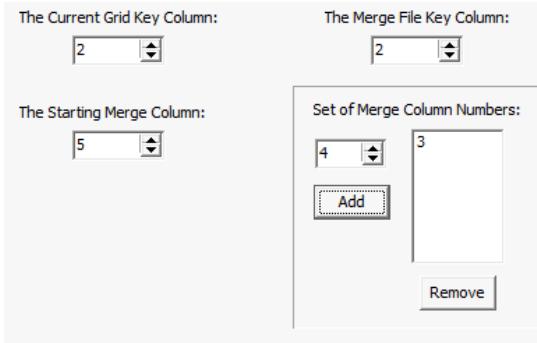
<	Column 1	Column 2
1 >	Cummings	978-1-56881-721-7
2	Spence	978-1-449-30468-3
3	Strong	978-0-486-64725-8
4	Grant	978-0-262-51668-6
5	Vinson	978-0-8176-4704-9
6	Hardin	978-0-486-47417-6
7	Guzman	978-1-4129-1314-6
8	Kennedy	978-0-8176-4372-0
9	Haney	978-0-465-01775-1
10	Andrews	978-0-486-47883-8
11	Hogan	978-0-691-15265-3
12	Gates	978-0-691-14039-1
13	Bennett	978-0-471-11709-4
14	Alvarez	978-0-596-80552-4
15	Shields	978-1-59184-492-1
16	Rosa	978-1-56158-891-6
17	Gray	978-0-486-65241-2
18	Sears	978-0-19-974044-3
19	Anderson	978-0-321-88691-0

To answer the question we will add one blank column to the grid on the left. We will also add a column of all Z's to the grid on the right. However, it is still not easy to determine which numbers in Column 2 in the grid on the right appear in Column 2 of the grid on the left.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	1/29/2012	978-1-449-30468-3	O'Reilly & Associates	Johnson, Clay	
2	2/15/2012	978-0-691-14039-1	Princeton University Press	Bernstein, Dennis S.	
3	2/15/2012	978-0-19-974044-4	Oxford University Press	Levitin, Anay; Levitin, Maria	
4	2/15/2012	978-0-691-14714-7	Princeton University Press	MacCormick, John	
5	2/23/2012	978-0-486-47883-8	Dover Publications	Michaelson, Greg	
6	2/27/2012	978-0-465-01775-1	Basic Books	Stewart, Ian	
7	3/2/2012	978-0-486-47417-5	Dover Publications	Pinter, Charles C.	
8	4/12/2012	978-0-471-11709-4	John Wiley & Sons	Schneier, Bruce	
9	4/14/2012	978-0-8218-4418-2	American Mathematical Society	Mullen, Gary L.	
10	7/6/2012	978-0-691-15270-7	Princeton University Press	Cook, William J.	
11	7/17/2012	978-0-691-14342-2	Princeton University Press	Havil, Julian	
12	8/21/2012	978-0-321-62930-2	Addison-Wesley	Vickers, Andrew	
13	9/13/2012	978-0-674-05755-5	Belknap Press	Lockhart, Paul	
14	9/25/2012	978-1-118-46446-5	Wiley Computer Publishing	Upton, Eben; Halfacree, Gareth	
15	10/4/2012	978-0-307-72095-5	Crown Publishers	Anderson, Chris	
16	10/4/2012	978-1-59420-411-1	The Penguin Press	Silver, Nate	
17	10/4/2012	978-1-59184-492-1	Portfolio/Penguin	Steiner, Christopher	
18	10/5/2012	978-0-984-72511-3	Digital Frontier Press	Brynjolfsson, Erik; McAfee, Andrew	
19	10/11/2012	978-1-118-20413-9	Wiley Computer Publishing	Thurrott, Paul; Rivera, Rafael	
20	10/15/2012	978-0-321-88691-0	Peach Pit Press	Revell, Jeff	
21	10/23/2012	978-1-285-42457-6	Cengage Learning	Busch, David	
22	11/19/2012	978-0-07-180783-8	McGraw-Hill	Monk, Simon	
23	11/23/2012	978-0-470-40765-3	John Wiley & Sons	Kjellstrom, Bjorn; Elgin, Carina	
24	12/8/2012	978-1-56881-721-7	CRC Press	Wapner, Leonard	
25	12/27/2012	978-0-691-14892-2	Princeton University Press	Van Brummelen, Glen	
26	12/28/2012	978-0-7484-0304-2	CRC Press	Snyder, John; Bugayevskiy, Lev	
27	1/15/2013	978-0-691-15271-4	Princeton University Press	Gray, Jeremy	
28	1/15/2013	978-0-321-81958-1	New Riders Publishing	Kelby, Scott	

<	Column 1	Column 2	Column 3
1 >	Cummings	978-1-56881-721-7	Z
2	Spence	978-1-449-30468-3	Z
3	Strong	978-0-486-64725-8	Z
4	Grant	978-0-262-51668-6	Z
5	Vinson	978-0-8176-4704-9	Z
6	Hardin	978-0-486-47417-6	Z
7	Guzman	978-1-4129-1314-6	Z
8	Kennedy	978-0-8176-4372-0	Z
9	Haney	978-0-465-01775-1	Z
10	Andrews	978-0-486-47883-8	Z
11	Hogan	978-0-691-15265-3	Z
12	Gates	978-0-691-14039-1	Z
13	Bennett	978-0-471-11709-4	Z
14	Alvarez	978-0-596-80552-4	Z
15	Shields	978-1-59184-492-1	Z
16	Rosa	978-1-56158-891-6	Z
17	Gray	978-0-486-65241-2	Z
18	Sears	978-0-19-974044-3	Z
19	Anderson	978-0-321-88691-0	Z

We finish by leaving the grid on the left open, and we perform a merge operation where we set the controls as follows:



and where we use the merge file as the grid on the right. The results are shown in the next grid below.

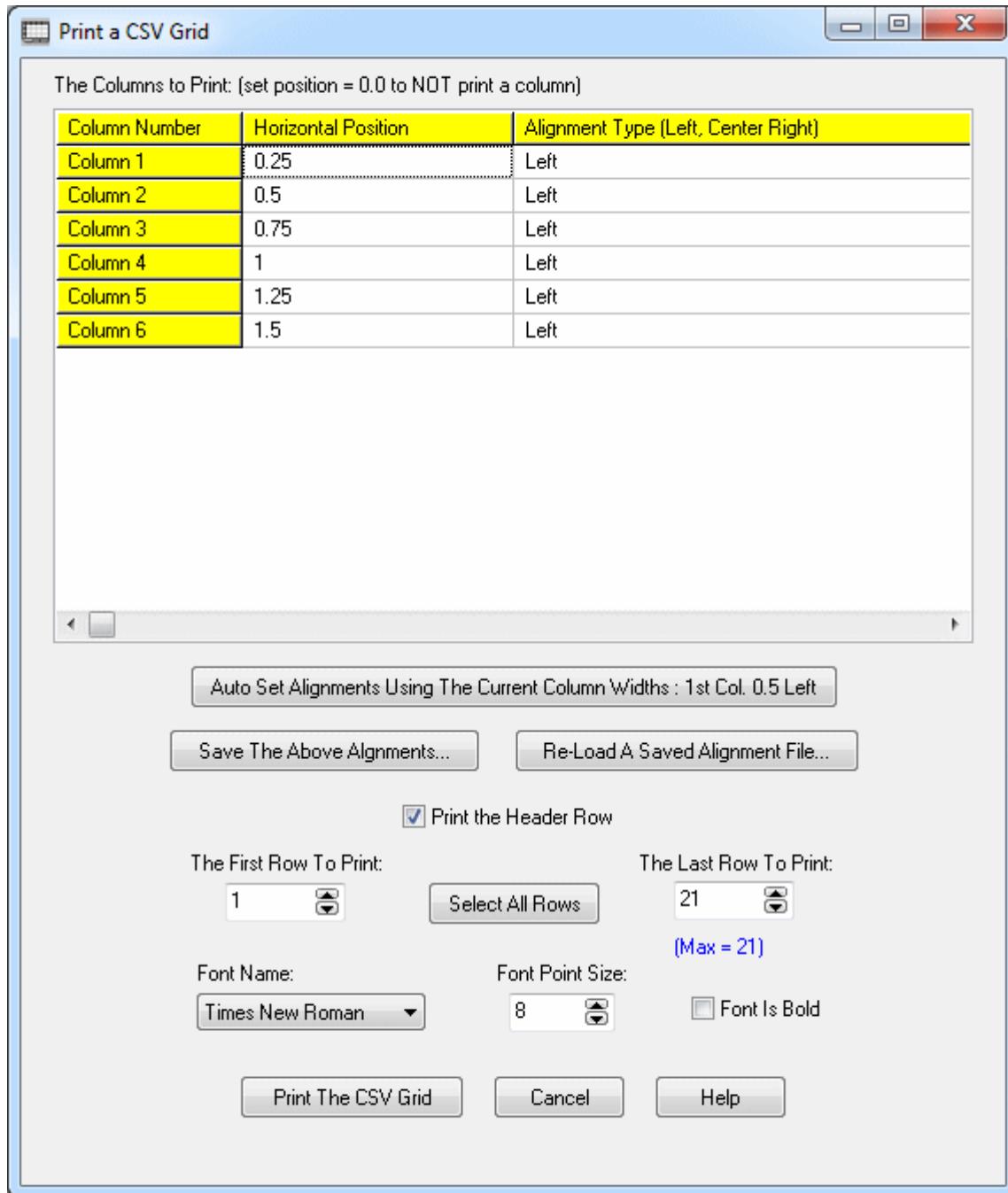
<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	1/23/2012	978-1-449-30468-3	O'Reilly & Associates	Johnson, Clay	Z
2	2/15/2012	978-0-691-14039-1	Princeton University Press	Bernstein, Dennis S.	Z
3	2/15/2012	978-0-19-974044-4	Oxford University Press	Levitin, Anay; Levitin, Maria	
4	2/15/2012	978-0-691-14714-7	Princeton University Press	MacCormick, John	
5	2/23/2012	978-0-486-47883-8	Dover Publications	Michaelson, Greg	Z
6	2/27/2012	978-0-465-01775-1	Basic Books	Stewart, Ian	Z
7	3/2/2012	978-0-486-47417-5	Dover Publications	Pinter, Charles C.	
8	4/12/2012	978-0-471-11709-4	John Wiley & Sons	Schneier, Bruce	Z
9	4/14/2012	978-0-8218-4418-2	American Mathematical Society	Mullen, Gary L.	
10	7/6/2012	978-0-691-15270-7	Princeton University Press	Cook, William J.	
11	7/17/2012	978-0-691-14342-2	Princeton University Press	Havil, Julian	
12	8/21/2012	978-0-321-62930-2	Addison-Wesley	Vickers, Andrew	
13	9/13/2012	978-0-674-05755-5	Belknap Press	Lockhart, Paul	
14	9/25/2012	978-1-118-46446-5	Wiley Computer Publishing	Upton, Eben; Halfacree, Gareth	
15	10/4/2012	978-0-307-72095-5	Crown Publishers	Anderson, Chris	
16	10/4/2012	978-1-59340-411-1	The Penguin Press	Silver, Nate	
17	10/4/2012	978-1-59184-492-1	Portfolio/Penguin	Steiner, Christopher	Z
18	10/5/2012	978-0-984-72511-3	Digital Frontier Press	Brynjolfsson, Erik; McAfee, Andrew	
19	10/11/2012	978-1-118-20413-9	Wiley Computer Publishing	Thurrott, Paul; Rivera, Rafael	
20	10/15/2012	978-0-321-88691-0	Peach Pit Press	Revell, Jeff	Z
21	10/23/2012	978-1-285-42457-6	Cengage Learning	Busch, David	
22	11/19/2012	978-0-07-180783-8	McGraw-Hill	Monk, Simon	
23	11/23/2012	978-0-470-40765-3	John Wiley & Sons	Kjellstrom, Bjorn; Elgin, Carina	
24	12/8/2012	978-1-56881-721-7	CRC Press	Wapner, Leonard	Z
25	12/27/2012	978-0-691-14892-2	Princeton University Press	Van Brummelen, Glen	
26	12/28/2012	978-0-7484-0304-2	CRC Press	Snyder, John; Bugayevskiy, Lev	
27	1/15/2013	978-0-691-15271-4	Princeton University Press	Gray, Jeremy	
28	1/15/2013	978-0-321-81958-1	New Riders Publishing	Kelby, Scott	

By reading the rows with Z's in **Column 5** we can clearly see the answer to the question of what rows in the original grid on the right had key data in Column 2 in the grid on the left. While this example used grids with only one or two dozens of rows, you can imagine how powerful this function would be when applied to grids with thousands of rows in each grid.

You may wish to compare everything we have explained in this help topic with another help topic that discusses how to copy data after manually computing a set difference or a set intersection. Here we are essentially creating extra columns of data (that's why we call it a merge operation), but only for rows that have matching key data. In a sense, this operation appends data by adding it in new columns, but only data that represents a special kind of intersection for associated key column cells. See also the help topic **Copying Matching Data** under the **Utilities** menu and see **Export Matching Key Rows** under the **Rows** menu. In database terminology, what we are discussing is a special kind of a join operation. Another related topic worth reading about is **Conversions Using a Replacement List** under the **Conversions** menu.

Printing the CSV Grid

For those times when you need to print your grid, you should go to the **File** menu and pull down the menu item that says **Print the CSV Grid...**. You should then see a dialog similar to the following, although the number of columns and the number of rows will vary with the grid that is currently open:



The above dialog is used to specify the columns to be printed and to specify a range of consecutive rows, starting with **The First Row To Print** and ending with **The Last Row To Print**.

In the above grid, the **Horizontal Position** numbers are decimal values. In fact, the default values are spaced every 1/4 of an inch which is very unrealistic. Clicking the **Auto Set Alignments Using the Current Column Widths** button will make a better first cut for these values. These values represent the number of inches each corresponding column will be printed from the left edge of your paper. It may seem a little strange, but this program does not care if your paper is in either a portrait or a landscape orientation.

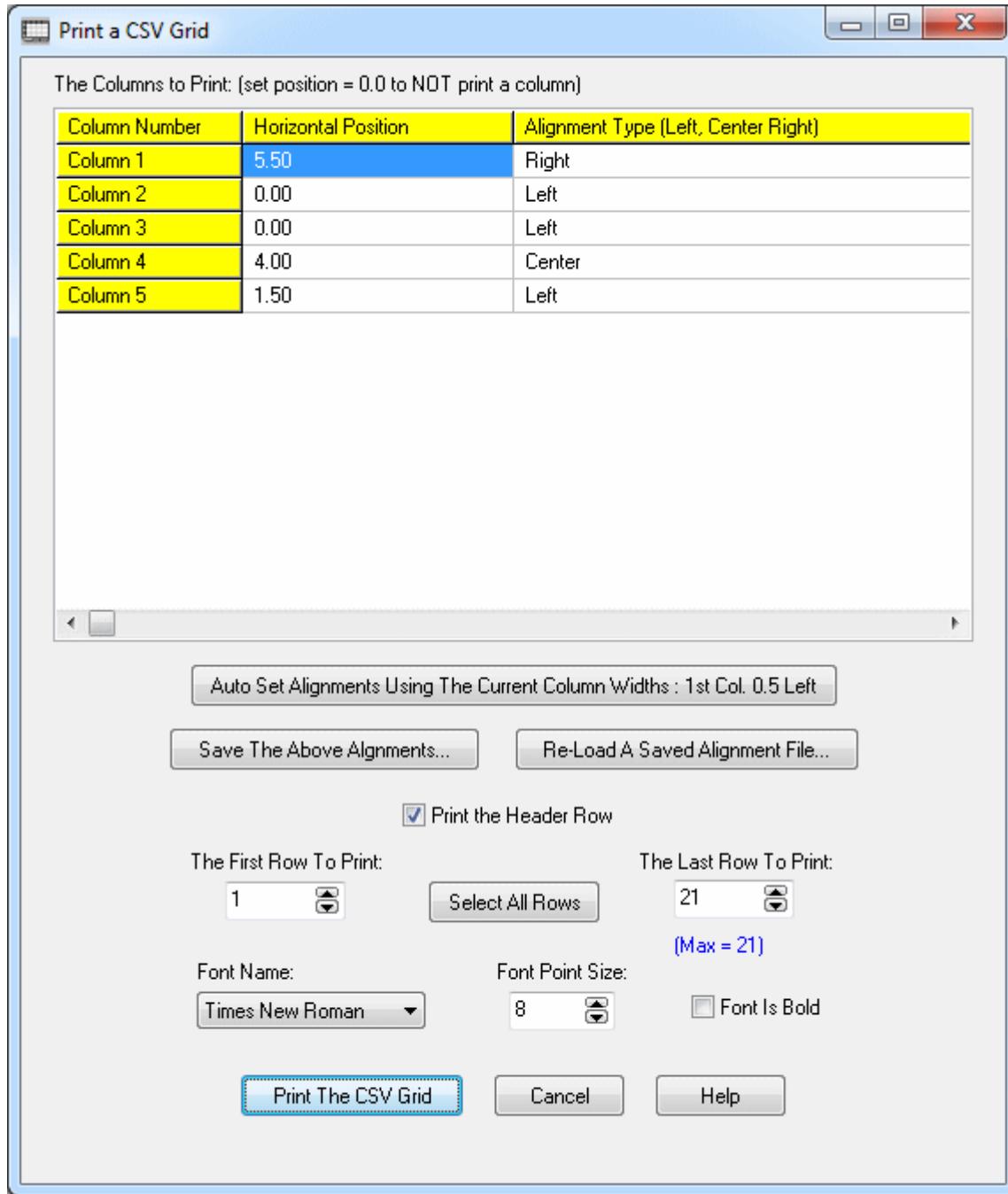
You can change the type of alignment that references each **Horizontal Position**. The default alignments are all **Left**, but you can type in either the word **Center** or **Right** instead of **Left** to change the type of the alignment. If you don't want to print a particular column you should set the **Horizontal Position** value to **0.0** and this means the corresponding column won't print at all.

Since setting up the alignment positions for each column can take some time, you have the option to save the print layout grid after you fill in all the values. We recommend you always do this. Saving your print layout is especially useful when you tend to work with the same grid most of the time. Just before printing, you can Reload a saved grid print layout. By default, whenever the above dialog is opened, the program will make an unrealistic set of positions every quarter of an inch with all Left alignments. You will normally need to change all the **Horizontal Position** numbers.

We will show one example. Consider a list we call the Academy Awards that appears in a **CSV** grid as:

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Year	Best Actor	Best Actress	Best Director	Film Title
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia
4	1963	Sidney Potier	Patricia Neal	Tony Richardson	Tom Jones
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People

Now suppose we open the **File** menu item, **Print the CSV Grid...** and enter the values as shown below.



Giving Column 2 and Column 3 a **Horizontal Position** of **0.00** means those two columns won't print. Columns 1, 4, and 5, will be printed, but their left to right print order will be different from the left to right order that is shown in the above CSV file view. Also note that these three columns have been given three different kinds of Alignment (**Right**, **Center** and **Left**).

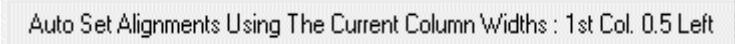
Column 1 output is the Film Title, and it has a **Left** alignment at the **Horizontal Position** of **1.50** inches.
 Column 4 output is the Best Director, and it has a **Center** alignment at the **Horizontal Position** of **4.00** inches.
 Column 5 output is the Year, and it has a **Right** alignment at the **Horizontal Position** of **5.50** inches.

In the next figure, the black vertical line represents the left edge of the paper. The blue lines and blue text don't actually print, but they are shown below so you can understand the nature of the output. The blue lines with the arrows measure absolute distances, in inches, from the left edge of the paper.

<u>Film Title</u>	<u>Best Director</u>	<u>Year</u>
The Apartment	Billy Wilder	1960
West Side Story	Robert Wise	1961
Lawrence of Arabia	David Lean	1962
Tom Jones	Tony Richardson	1963
My Fair Lady	George Cukor	1964
The Sound of Music	Robert Wise	1965
A Man For All Seasons	Fred Zinnemann	1966
In The Heat Of The Night	Mike Nichols	1967
Oliver	Sir Carol Reed	1968
Midnight Cowboy	John Schlesinger	1969
Patton	Franklin Schaffner	1970
The French Connection	William Friedkin	1971
The Godfather	Bob Fosse	1972
The Sting	George Roy Hill	1973
The Godfather Part II	Francis Ford Coppola	1974
One Flew Over The Cuckoos Nest	Milos Forman	1975
Rocky	John G. Avildsen	1976
Annie Hall	Woody Allen	1977
The Deer Hunter	Michael Cimino	1978
Kramer vs Kramer	Robert Benton	1979
Ordinary People	Robert Redford	1980

After studying the above example you should have a better idea of how you can make the printed output match your printing needs.

One final note of caution. If you take the time to fill out many horizontal print positions and alignments, we highly recommend you always save your alignments in a file before you try to print. Otherwise, if your printer is out of paper, or if your printer jams, or runs out of ink, or if anything else goes wrong, you will have to manually re-enter all your alignments before you can properly print. But if you save your alignments you can then quickly re-load them when things do go wrong. Just remember Murphy's Law. **If anything can go wrong it will go wrong!**

Clicking the button that appears as  will cause the first column to start at 0.5 inches with a Left type of alignment. All the following columns will also be left aligned, but their widths will correspond to about how the columns appear on the current display. We generally recommend you click this button when you are using the print feature for the first time. This should give you a rough idea of the correct values for the current display. You can then further customize the current print job by editing the values the program has automatically determined.

As a practical matter, you can only normally print just a few columns. Using a landscape orientation for your paper will give you room for perhaps a couple of more columns, but then you will also be limited by the space allowed for the number of rows that will fit on one page. Unless you need real paper output, another trick is to print to a **PDF** file in which case you may be able to specify a more wide virtual sheet.

Also note that in setting up a print job you can specify the font name and the font point size and you have the option of turning the bold attribute on or off. The default values of a plain Times New Roman 8-point font size give you a small printout that is very readable and that can fit a large table on a single page. You may need to choose portrait or landscape when the printer selection comes up, but you can experiment to determine the best type of font to use. Using the Courier New font gives a fixed size for all letters in the font, much like an old typewriter. If you have an Adobe **PDF** printer (distiller) you can try printing to that printer to save paper until you get everything setup just right. Then you can print to a real printer using real paper as needed.

In the above example dialog box, because we checked the checkbox with the caption **Print The Header Row**, each new page of output will start at its top with the column header information printed in bold and underlined. That header row will be followed by a blank row, and then the actual data will start and will continue down the page. If not all of the lines to be printed will fit on one page, then as each new page gets started, a new header row will be printed at its top. Although the header rows are printed with a bold and underlined font, the normal data rows will be printed without the underline font attribute and the bold attribute will only be applied if you checked the checkbox with the caption **Font Is Bold**.

We consider the primary purpose of printing on paper to make the output readable. So while there is a practical limit to the number of columns that will fit on a piece of paper, you can print out lists that are readable and useable. Of course lists with many rows of data will consume more sheets of paper.

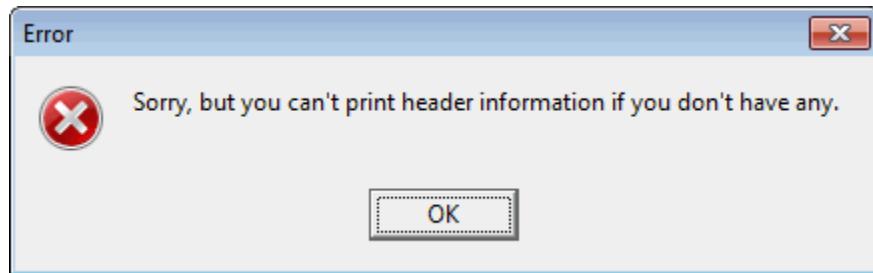
The format of the information in a file that contains printing alignment information is very simple. For the above example the contents of the text file that holds the alignment information is as follows:

```
Column 1
5.50
Right
Column 2
0.00
Left
Column 3
0.00
Left
Column 4
4.00
Center
Column 5
1.50
Left
```

Printing the Header Information

You can print the Header Information for the currently opened grid by selecting the menu item **File | Print the Header Information....** When you do this you will see a standard dialog for printing. There are no parameters that need to be setup for this specialized printing. Just select your printer and go!

In case you don't have any header information you will see an error message like:



An example of the output is shown below for the Academy Awards sample file.

Column #: Header Line:

Column 1	Year
Column 2	Best Actor
Column 3	Best Actress
Column 4	Best Director
Column 5	Best Picture
Column 6	Tagged

The file = C:\Documents\AcademyAwards.csv

The number of data rows = 21

In this example we can see the header information is printed down the rows in the output. The last two lines show the name of the file and the number of data rows that the file had at the time of printing.

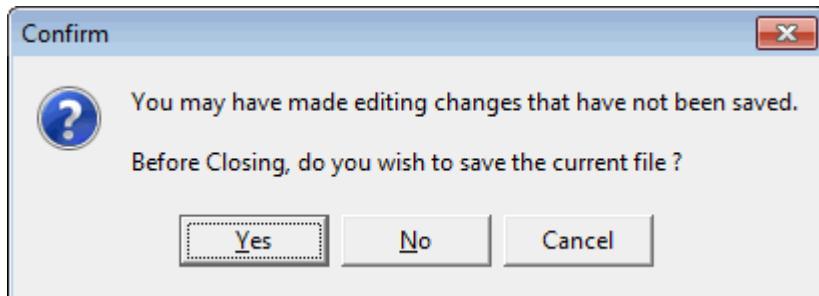
Printing header information probably only makes sense when you have a large number of columns that you can't see all at once. This information is also useful whenever you work with one or more grids that will be combined in some way that requires identifying common header information.

Recent Files

The **File** menu has a menu item **Recent Files ►** that will show a submenu containing up to 15 of the most recently used files. Thus if you need to re-open a recently used file all you need do is click on any of the listed filenames to immediately re-open that file.

Exiting the Program

When you want to exit from the program you have a couple of choices. You can click the close button in the upper-right program window or you could select the menu item **File | Exit**. In either case, if you have made any editing changes that have not been saved you will first be prompted by:

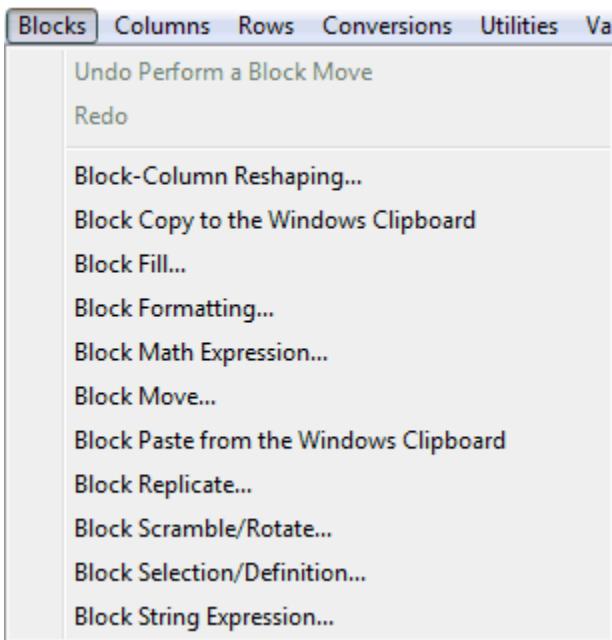


In fact the program will display an asterisk character * at the end of the filename in the main program window caption line when you have not saved editing changes that should be saved. This visual clue is useful whenever you aren't sure if you need to save anything. The asterisk character disappears from the filename in the window top caption line each time you save the current file. Then the asterisk will re-appear as you make more editing changes.

The Blocks Menu

The five main menus following the **File** menu can all be considered to contain the main editing functions of the program. These are the **Blocks**, **Columns**, **Rows**, **Conversions**, and **Utilities** menu items. We will discuss these five main menus in the order just listed. This is not meant to slight either the **Validation** menu or the **Uniqueness** menu which contain very important functions for analyzing your data. It is just that the functions under the **Validation** and **Uniqueness** menus are usually not used to actually edit or change your data. The submenu functions under these two main menus are normally only used to make reports about the nature of your data.

The **Blocks** menu contains functions that apply to any rectangular block of cells, even those with only 1 column or only 1 row.

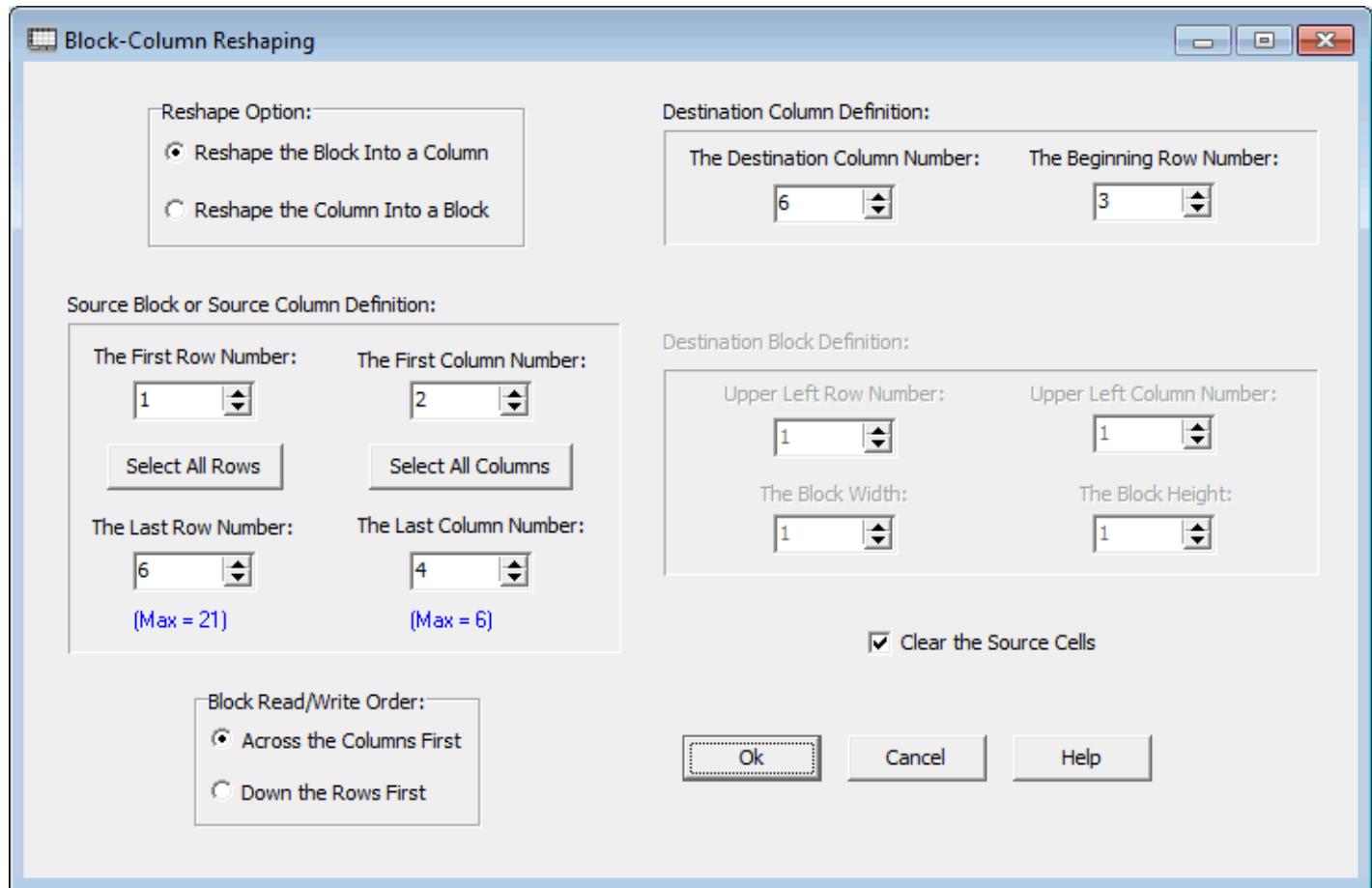


You should simply note that until you perform an operation, the **Undo** and **Redo** menu items will appear inactive and grayed out. There is only one level of **Undo** or **Redo**.

The **Block Copy to the Windows Clipboard** and **Block Paste from the Windows Clipboard** menu items are standard in that they either copy the currently selected block of cells to the Windows clipboard, or they paste a new block of cells into the current grid, starting at the current grid position.

Block-Column Reshaping

Under the **Blocks** menu you can select the menu item **Block-Column Reshaping...** and you will see the dialog that is used to either reshape a rectangular block into a single column, or to reshape a single column into a rectangular block. These are two significant grid manipulation functions that allow you to apply any column function to any block of cells because you can first convert the block to a column, then perform the column function, and finally reshape that column result back into the original rectangular block shape. In fact, the Block-Column reshape functionality can also be used to perform Block-Block reshaping as we explain at the end of this topic. Block-Block reshaping can also be used to make the transpose of any grid.



On the next page we show a part of the Academy Awards grid that we use for several examples in this help file. In the next grid shown below we have selected a rectangular block of cells that is 6 rows tall and 3 columns wide. If we use the settings as shown in the above dialog, then the operation would reshape and move the rectangular block into a single column, in column 6 and beginning in row 3. Moreover, the operation would clear all the cells in the original selection, but it would copy those cells in the order that you would read a newspaper by first going across the columns before moving down the rows.

If you compare the two figures on the next page you can see the before and after effects of this reshape function. Reshaping a block is also a special case of a block move because the original block now appears not only with a column shape, but all the selected data has been moved into column 6.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T

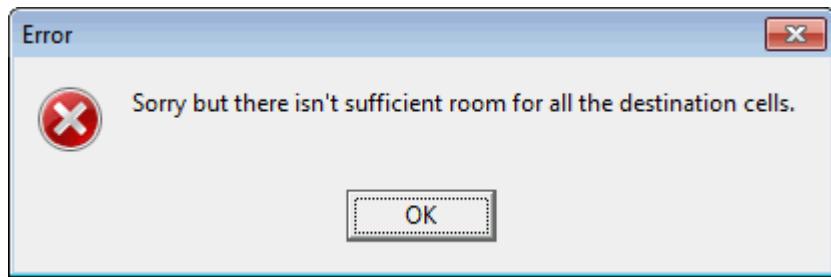
The same block cells with a column shape in a different location is shown in the next figure below.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960				The Apartment	T
2	1961				West Side Story	T
3	1962				Lawrence of Arabia	Burt Lancaster
4	1963				Tom Jones	Elizabeth Taylor
5	1964				My Fair Lady	Billy Wilder
6	1965				The Sound of Music	Maxmillian Schell
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	Sophia Loren
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	Robert Wise
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	Gregory Peck
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	Anne Bancroft
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	David Lean
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	Sidney Poitier
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	Patricia Neal
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	Tony Richardson
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	Rex Harrison
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	Julie Andrews
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	George Cukor
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	Lee Marvin
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	Julie Christie
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	Robert Wise
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

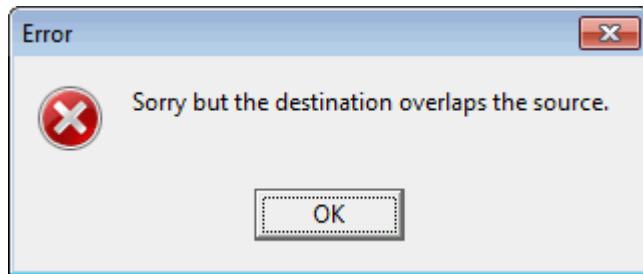
In this example we also chose to clear the original cells when the block was moved and reshaped. We have also subsequently manually selected the overwritten cells that are in the destination column, that contain the moved data.

Whenever you reshape a rectangular block into a column or reshape a column into a rectangular block, there are a few restrictions that must be satisfied. First, the source cells and the destination cells must not overlap in any way. Second, the position of the destination cells must be such that all those cell positions exist within the grid at the time the operation is performed. In other words, the grid will not be expanded to accommodate any new cells. So you want to insure the position and extent of the destination cells is within the boundary of all the defined grid cells.

If you violate the restriction of keeping all the destination cells within the existing grid boundaries then you may see an error message like the following before you can even close the above dialog.



If you violate the restriction that the source and destination cells must never overlap then you may see a different error message like the following.

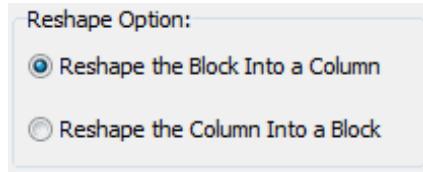


If you violate both conditions then you may see both error messages. In either case, you must change either the source or destination definitions or you must enlarge the entire grid before you can perform the reshape function.

There are perhaps a few subtle options that you should know regarding the controls in the **Block-Column Reshaping** dialog. Not all controls are used or enabled, depending on which options are selected.

The group of controls near the left middle of this dialog are used to either define the source rectangular block or to define the source column. When defining a source column, the spin edit control with the caption **The Last Column Number** is not used at all. Instead, **The First Column Number** and both row number controls are used to completely specify all the source column cells. When defining a rectangular source block, all four controls in the upper left are used to define the rectangular block.

The control that appears as



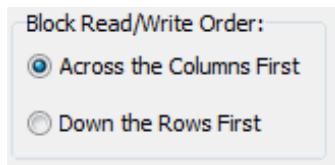
is used to select one of the two functions that

turn rectangular blocks into columns or that turn columns into rectangular blocks. This choice of reshape type is the main control you always need to set and should be set first. That is why this control appears first in the upper-left of the dialog. This choice automatically determines if and how the other controls get enabled/disabled and used.

When you choose to **Reshape the Block Into a Column**, then the two controls that appear under the **Destination Column Definition** are used to define the destination column cells and the other four controls under the **Destination Block Definition** are not used at all and will be disabled. In this case the size of the source block will determine the number of rows used in the destination column, starting with **The Beginning Row Number**.

When you choose to **Reshape the Column Into a Block**, then the two controls that appear under the **Destination Column Definition** are not used at all and will be disabled, and the four controls under the **Destination Block Definition** are used to define the destination block. In this case the size of the source column will determine how much of the destination block gets filled. In some cases, parts of the destination block may go unused or unfilled.

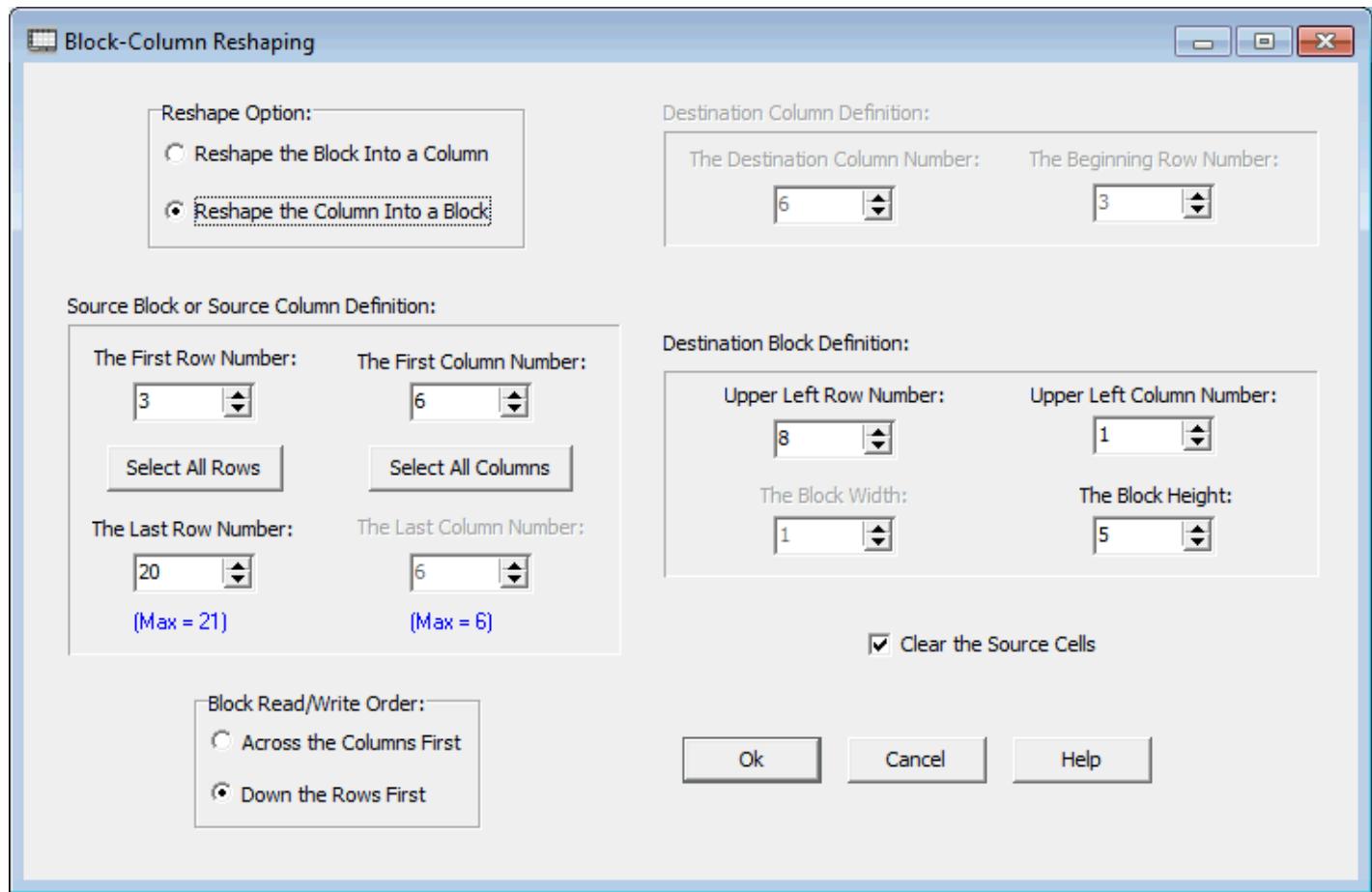
In any case, the control that appears as



always gets used because a block is either a

source that is read or is a destination that is written. When the block is used as a source, then the order chosen is a read order. When the block is used as a destination, then the order chosen is a write order. The state of this control can set or disable the **Block Width** and **Block Height** controls, when a **Destination Block Definition** is needed.

For our next example, we will begin with the selected column cells shown in the last figure two previous pages, and we will set the controls in the **Block-Column Reshaping** dialog to be as shown below. In this example the source is a column. Note **The Last Column Number** control has been automatically disabled because we have selected the **Reshape Option** to start with a column as the source. Only one column number is necessary for this case.



You should note the upper left corner of the **Destination Block Definition** is in row 8 and column 1. After the reshape command is executed, the 18 cells in the source column will fill in the 18 cells shown with a gray background in the next figure.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960				The Apartment	T
2	1961				West Side Story	T
3	1962				Lawrence of Arabia	
4	1963				Tom Jones	
5	1964				My Fair Lady	
6	1965				The Sound of Music	
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	
8	Burt Lancaster	Robert Wise	Patricia Neal	Lee Marvin	In The Heat Of The Night	
9	Elizabeth Taylor	Gregory Peck	Tony Richardson	Julie Christie	Oliver	
10	Billy Wilder	Anne Bancroft	Rex Harrison	Robert Wise	Midnight Cowboy	
11	Maxmillian Schell	David Lean	Julie Andrews	Franklin Schaffner	Patton	
12	Sophia Loren	Sidney Poitier	George Cukor	William Friedkin	The French Connection	
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	

In this example the program used the value of 5 for **The Block Height**, and it ignored **The Block Width** value that was 1. In other words, because the block write order was **Down the Rows First**, only **The Block Height** value was used. This example also illustrates how the final block size would have been 5 rows high by 4 columns across, but the last name used was Robert Wise that was the 18th cell in the source column. So the final used block is not a full 5×4 rectangle with 20 used cells. Only the first 18 cells in the block are actually used. This explains why the two names Franklin Schaffner and William Friedkin have not been overwritten and are not shaded gray. In actual use, you won't see any gray shading. The gray shading in the above figure is only used for explanatory purposes to show where and how the destination block gets filled using the 18 source column cells.

In a different example, if a rectangular block is the destination, and if the block write order is chosen to be **Across the Columns First**, then **The Block Height** value would be disabled, while **The Block Width** value would be enabled instead.

You should always carefully consider how you set the **Block Read/Write Order** option because it has a great effect on the final output.

It is possible to use the **Block-Column Reshaping** dialog to turn a row into a column or to turn a column into a row. This would work because a rectangular block that is only 1 row tall is really just a row. Or a rectangular block that is only 1 column wide is really just a column.

You also have the option to **Clear the Source Cells**, or not, depending on whether you turn on the corresponding checkbox. In the above examples we cleared the source cells so you could see the vacated cell positions after the move.

If any control you want to set is disabled, you should first change one of the radio group buttons to enable it. All controls are enabled or disabled depending on the choices you make in the two sets of radio group buttons.

One of the main uses of block reshaping into a column (and back into a block) is that you can apply column only functions like **Sorting** or **Sum and Average the Entries** to any block of cells. In fact, the **Block Read/Write Order** allows you to write the sorted entries in a block in two different ways.

Our final comment is that you can actually use **Block-Column Reshaping** to perform **Block-Block Reshaping**. In other words, you can reshape any rectangular block into another rectangular block but with a changed rectangular shape that does not have to be a single column. An example will explain how you can do this in two steps.

Imagine you have a rectangular block that is 6×8 in size (6 columns and 8 rows). You can re-shape this as a 3×16 shape (or any other rectangular shape) by first converting the original 48 cells into a single vertical first column 48 rows tall. Then change the output rectangular block definition to the desired final 3×16 rectangular shape and convert the first column into the new changed rectangular shape. Although this takes two steps, it does accomplish **Block-Block** reshaping. You could also re-shape the 6×8 block into a 5×11 block where there might be 7 unused extra cells in the 5th and final column because the destination block is larger than the source block. So the source and destination blocks never even have to match exactly in terms of their dimensions.

The destination block size depends on the **Write Order** and the corresponding enabled **Destination Block Definition** dimension that is active.

For example, we can continue discussing the 48-cell single vertical column example and show several ways this might be re-shaped back into some kind of a rectangular block. Let's assume the column contains the numbers from 1 to 48, in order, from top to bottom.

If the **Write Order** is chosen as **Across the Columns First**, and if the **Block Width** value is 5, then we can compute the first multiple of 5 that covers 48 is 50. This means the final destination block would be 5 columns wide and 10 rows high. In this case there would be 2 unused cells in the last row of the destination 5×10 block, as shown in the next figure.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40
41	42	43	44	45
46	47	48		

Now if the **Write Order** is chosen as **Down the Rows First**, and if the **Block Height** value is 11, then we can compute the first multiple of 11 that covers 48 is 55. This means the final destination block would be 11 rows high and would be 5 columns across. This is when there would be 7 unused cells in the last column of the destination 5×11 block, as shown in the next figure.

1	12	23	34	45
2	13	24	35	46
3	14	25	36	47
4	15	26	37	48
5	16	27	38	
6	17	28	39	
7	18	29	40	
8	19	30	41	
9	20	31	42	
10	21	32	43	
11	22	33	44	

If we tried to make the 48 cells into a square block, then we could set both the **Block Width** and the **Block Height** values to 7, and then the **Write Order** would not really matter in terms of size in that the first multiple of either 7 value that covers 48 would be 49. In this case there would be just 1 extra cell left over in the last row and the last column of the destination block. However, there is a big difference in the two possible outputs as shown in the two figures on the next page.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	

The above figure had a **Write Order of Across the Columns First**.

1	8	15	22	29	36	43
2	9	16	23	30	37	44
3	10	17	24	31	38	45
4	11	18	25	32	39	46
5	12	19	26	33	40	47
6	13	20	27	34	41	48
7	14	21	28	35	42	

The above figure had a **Write Order of Down the Rows First**.

We have now given all the technical details related to the **Block-Column Reshaping** function. We will finish this topic with a practical example. The list below contains GPS information for multiple cities, where each city has four pieces of information in consecutive rows, and where all the information is in one very tall single column.

<	Column 1
1 >	Adelaide
2	-34.921971
3	138.581543
4	Australia
5	Akron
6	41.08
7	-81.52
8	Ohio
9	Albuquerque
10	35.12
11	-106.62
12	New Mexico
13	Alexandria
14	38.82
15	-77.09
16	Virginia
17	Amarillo
18	35.20
19	-101.82
20	Texas

Our original list consisted of 1032 rows. What we needed to do was to extract the data from the list and put it into four distinct columns. Before we did the reshaping, we added four columns to the above grid. Then we performed the reshaping function that converted all the information in **Column 1** with 1032 rows, into a 4-column block that began in row 1 and **Column 2**. After the reshaping we deleted **Column 1** and we deleted the extra rows that were no longer used at the bottom of the grid. Finally we added a header line so you could better read and see the data. We show the first 20 rows of the final result as:

<	Column 1	Column 2	Column 3	Column 4
>	City Name	City Latitude:	City Longitude:	City Location:
1	Adelaide	-34.921971	138.581543	Australia
2	Akron	41.08	-81.52	Ohio
3	Albuquerque	35.12	-106.62	New Mexico
4	Alexandria	38.82	-77.09	Virginia
5	Amarillo	35.20	-101.82	Texas
6	Anaheim	33.84	-117.87	California
7	Anchorage	61.217379	-149.922729	Alaska
8	Arlington	32.69	-97.13	Texas
9	Arlington	38.88	-77.10	Virginia
10	Atlanta	33.76	-84.42	Georgia
11	Auckland	-36.844461	174.745239	New Zealand
12	Augusta-Richmond	33.46	-81.99	Georgia
13	Aurora	39.71	-104.73	Colorado
14	Aurora	41.77	-88.29	Illinois
15	Austin	30.31	-97.75	Texas
16	Baghdad	33.330528	44.416077	Iraq
17	Bakersfield	35.36	-119.00	California
18	Baltimore	39.30	-76.61	Maryland
19	Bangkok	13.752725	100.501282	Thailand
20	Barcelona	41.393294	2.171997	Spain

You should now appreciate the advantage of reshaping the single tall column list into a shorter height 4-column table. When the information was all in a single tall column it was difficult to read. Now each city occupies just one line in the table and all the information for that city is spread across into the four columns. Since $1032 \div 4 = 258$, our table has 258 rows of data. When we save this table with the header row we end up with a **CSV** file that has 259 lines of text.

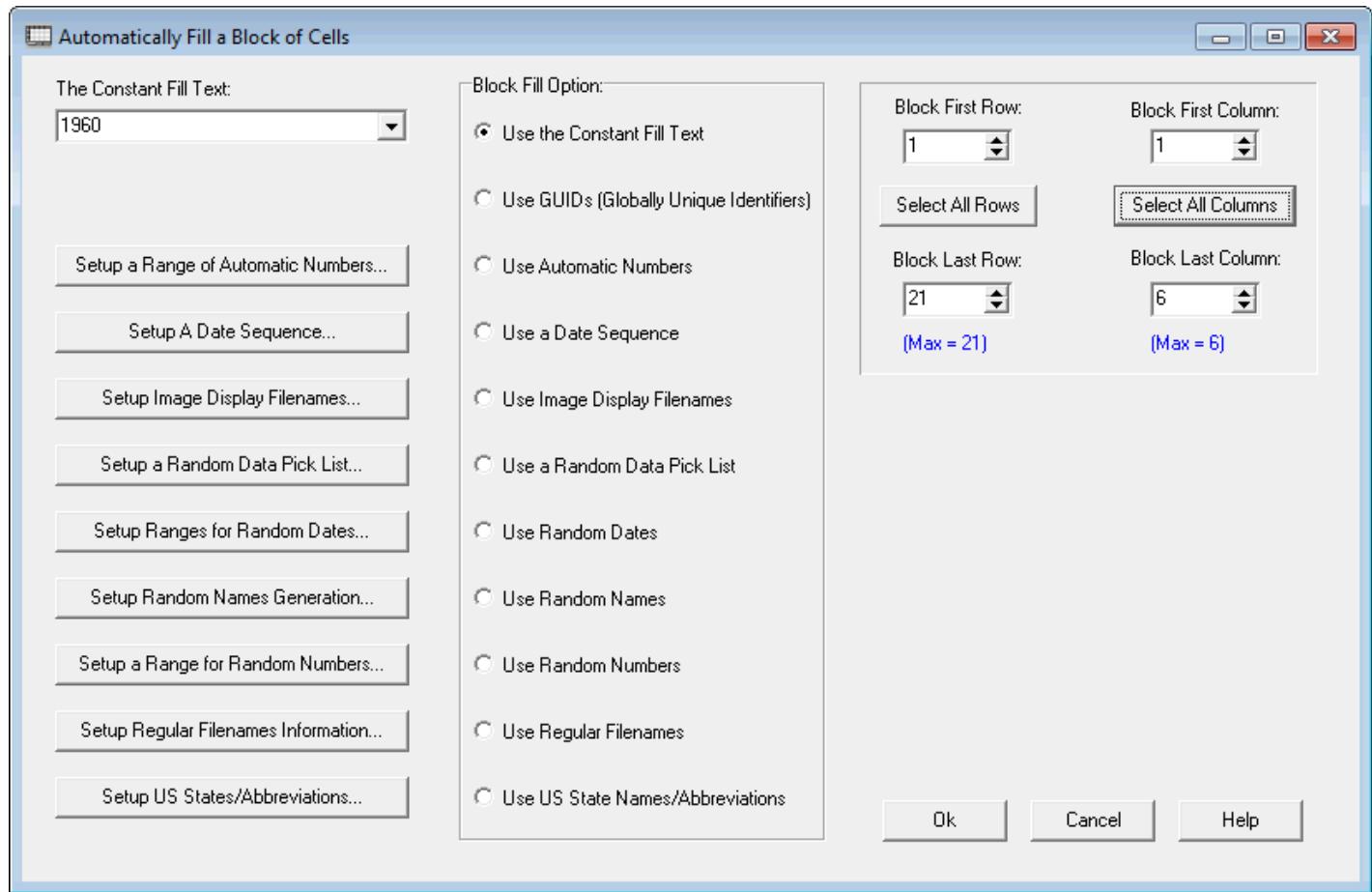
Using the **Block-Column Reshaping...** function twice, it is possible to perform the equivalent of a transpose operation on any rectangular block of cells. To do so, first note the width of the original rectangular block and then reshape the original rectangular block into a single column where the **Block Read/Write Order** is **Across the Columns First**. Then reshape that single column back into another rectangular block, but as you do so, make the new block height the width of the original rectangular block and set the **Block Read/Write Order** to **Down the Rows First**. You now have two different ways to accomplish a transpose. Another way is to directly use the **File | Special Transpose Read...** function.

Filling A Contiguous Block

The **Blocks** menu item to **Block Fill...** can be used in many different ways. As a first example, suppose a column in a grid is to contain the 2-letter abbreviation for the state that a person lives in. If you had 300 rows that represented all people from the state of California you don't have to enter 300 cells and type in CA 300 times. You could just setup the **Block Fill** parameters using CA as the constant fill text and choose the block as a series of rows in a single column. As a second example, if you had a list of products and one of your columns was to contain a serial number that you create, you could use the **Automatic Numbers Form** to automatically generate a range of systematic serial numbers.

There are nine dialog boxes we should show and discuss that are related to filling a continuous block. Those nine dialogs correspond to the nine buttons in the left side of the following dialog. First, the main **Block Fill** setup dialog appears as:

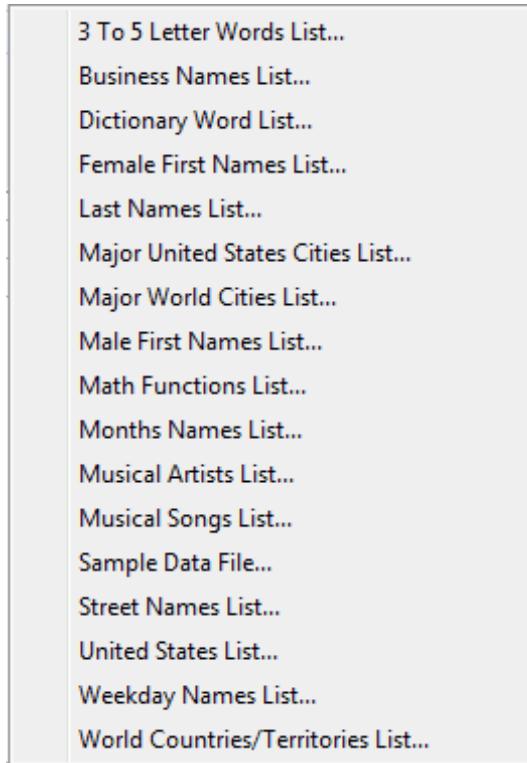
The Main Block Fill Setup Dialog



The six controls in the upper-right are used to define a rectangular block within the existing grid. The maximum row and column numbers (shown in blue) are determined by the grid that exists at the time this dialog is opened. These six controls are standard for defining any rectangular block of cells that you wish to operate on.

The **Block Fill Option** is where you make the main choice for the type of **Block Fill**. There are eleven choices. You can fill a block using the same constant text string, or you can fill a block with Globally Unique Identifiers, or you can fill a block using specially generated automatic numbers, or you can fill a block with a sequence of dates, or you can fill it with image display filenames, or you can fill it with data chosen randomly from a finite **Pick List**, or you can fill it with Random Dates, or you can fill it with human names that are chosen at random, or you can fill a block with random numbers, or you can fill it with file information, and finally you can fill a block with either US State names or US State Abbreviations.

Using a **Pick List** can be a very general way of filling a block with almost any kind of data. We provide 17 different lists of data, any one of which could be used as a **Pick List**. So if you wanted a random list of business names as opposed to names of people, then you could load your **Pick List** with our business names list. If you wanted a list of Rock 'n' Roll artists you could load your **Pick List** with our musical artists list. The following is a list of our sample data lists.



As a special case, we should point out that when you generate random human names, then only the **Block First Column** number is used because part of the setup for random human names determines whether an additional one or two consecutive columns will also be used. In any case, only the **Block First Column** number determines the first column used with random human names generation. The same is true if you choose to **Use Image Display Filenames** because the filenames are only filled in one column. If you choose **Use Regular Filenames** then up to four consecutive columns may be automatically filled.

Filling With GUIDs

In the **Block Fill** dialog box you can select the **Block Fill Option** as the one that says **Use GUIDs (Globally Unique Identifiers)**. This provides a means of creating unique key entries. As an example, we could insert a new blank column in a grid and then have the program fill all rows with **Globally Unique Identifiers**, also known as GUIDs. The following shows the Academy Awards grid with a new first column containing GUIDs in that column.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>		Year	Best Actor	Best Actress	Best Director
1	{B7818E5F-0E8A-4E84-8AEF-2A2317BDBF49}	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder
2	{2787DFA2-2648-4A1B-BCC8-3059BDA0BC51}	1961	Maxmillian Schell	Sophia Loren	Robert Wise
3	{DBAAED2D-819D-485B-ABD9-025C7C38838A}	1962	Gregory Peck	Anne Bancroft	David Lean
4	{053494E0-018E-472C-8DC8-1D345A8E741D}	1963	Sidney Poitier	Patricia Neal	Tony Richardson
5	{5638AD6B-FE7F-41C4-BF62-A0C5FE41F249}	1964	Rex Harrison	Julie Andrews	George Cukor
6	{AFC932C9-DB4A-4675-80F5-ACAE98FB5C07}	1965	Lee Marvin	Julie Christie	Robert Wise
7	{7335B9F7-F8C3-4A21-9725-C0AAF950D7F6}	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann
8	{1089A08A-DFA7-40F0-A8A7-8C64F1F1E320}	1967	Rod Steiger	Katherine Hepburn	Mike Nichols
9	{9E48E966-96A0-4026-80EF-BBEF51FDEA47}	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed
10	{FE4FC9F7-B279-48D3-8DB8-6C32F0925AC3}	1969	John Wayne	Maggie Smith	John Schlesinger
11	{1DB93DAF-0598-40AF-8D7B-B899DFB742EA}	1970	George C. Scott	Glenda Jackson	Franklin Schaffner
12	{E1E99330-DF3B-4D4D-A512-3E2466C3D705}	1971	Gene Hackman	Jane Fonda	William Friedkin
13	{E82534CB-51C7-43EA-AA57-7F47225C3EBF}	1972	Marlon Brando	Liza Minelli	Bob Fosse
14	{A9299DF3-A903-477D-88E7-1B5AC0871161}	1973	Jack Lemon	Glenda Jackson	George Roy Hill
15	{95A6A8EF-C08A-4A09-ADE9-300060ED3060}	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola
16	{8829AF1E-BC95-449E-AA94-C99E75A7969F}	1975	Jack Nicholson	Louise Fletcher	Milos Forman
17	{C8D1247C-EFC6-40B7-8B43-135C902F89F3}	1976	Peter Finch	Faye Dunaway	John G. Avildsen
18	{C84CDD25-9653-4BF1-912D-2F0EC3ACD6EB}	1977	Richard Dreyfuss	Diane Keaton	Woody Allen
19	{270E74BD-9A5F-4180-B742-AD657DEAA3C1}	1978	Jon Voight	Jane Fonda	Michael Cimino
20	{B605F411-7D22-404E-9777-5E79BD1C93BC}	1979	Dustin Hoffman	Sally Field	Robert Benton
21	{AC0DC223-6978-4A50-9945-475D8E6162B4}	1980	Robert De Niro	Sissy Spacek	Robert Redford

GUIDs are guaranteed to be unique because they are generated using many random parameters, including but not limited to the date and the exact time the GUID was created. In the above example we can see that each GUID is basically a string of hexadecimal digits. Each hex character is either a digit from 0-9 or a letter from A-F. Each GUID is made up of five blocks of random hex characters and those blocks are separated using hyphen characters. In total there are 32 hex characters ($8 + 4 + 4 + 4 + 12 = 32$) that constitute 128 total bits or 16 bytes. Each GUID shown above is also enclosed by 2 curly braces for a total GUID string length of 38 ASCII characters.

The most fascinating aspect of GUIDs is that they should be unique for all computers, even those computers that will be built in the future. This feature of a GUID is why they are called globally unique. Theoretically no computer software ever used on any future computer should ever generate a GUID that matches any GUID produced by any past computer or software system, regardless of the operating system for those two computers.

We will now discuss several examples that will provide some background for your conceptual understanding of GUIDs. It turns out that in the modern world, almost everyone naturally accepts and has experience with using what are called ***unique identifiers***. If you have either a savings or a checking account with a bank, then you already know about account numbers. A savings account or a checking account has associated with it what is called an account number. It turns out the name "account number" is mostly inaccurate, because in use, account numbers are not used like other numbers are used. Fundamentally, numbers are used to either add or subtract or multiply or divide quantities, or they are used to measure quantities. But no one ever adds or subtracts or multiplies or measures anything with an account number. An account number is an example of a unique identifier. It is rather peculiar that numbers are used as unique identifiers in computer databases.

Account numbers are unique in that your account numbers are different from anyone else's account numbers. They apply uniquely to you. Second, account numbers are used as a means to identify you. If you call your bank on the telephone to ask what your current checking account balance is, your bank will ask for your checking account number. They may ask for your name and Social Security number as well, but it is your account number that really uniquely identifies your account within any particular banking system.

The next most common example of unique identifiers might be your car's license plate number or your car's VIN, known as the Vehicle Identification Number. If you get a parking ticket, your car is identified (and subsequently you are identified) by its license plate number. License plate numbers are unique. License plate numbers seem shorter and simpler than banking account numbers, but that is primarily because of the way they are used. Car license plate numbers usually employ a combination of both letters and numbers, but we will still call them numbers. An automobile VIN is also called a number but the way it is used differs from a license plate number. In fact, license plate numbers may only be required to be unique within a given state, whereas an automobile VIN is unique for the entire world. That is the primary reason why VINs are longer than license plate numbers, or why license plate numbers can be shorter than VINs.

The next example of a unique identifier should be familiar to anyone who makes an airplane reservation. As an example, the number L9K9BT could represent an airplane reservation number for a particular flight between Los Angeles and Barcelona, Spain. This number is somewhat like a car's license plate number. It identifies a unique airline flight for a given passenger, but this number has limited use in terms of a time limit. A typical airline reservation number might be valid for only a few months, or perhaps a year at most, whereas a car's license plate number might need to remain valid for ten or more years.

Our next examples of unique identifiers are serial numbers as might be found on appliances like refrigerators or televisions. If you need to repair a refrigerator and you call the company that made it, they might ask for the serial number so they can uniquely identify the exact type or model, but if you registered it when you bought it, such a number could also be used to uniquely identify you, the owner. Such a number might also be used to verify that your device is still under warranty. In this sense, a serial number is a unique identifier that may have an associated time limit.

Our next to last example is a serial number on a dollar bill. In this case, the serial number is not used to uniquely identify a person, but it is used to uniquely identify the dollar bill.

Our last example of unique identifiers is probably the most ubiquitous. A credit card number is an example of a unique identifier. This is somewhat like a combination of a car's license plate number and bank account number. A credit card number is used to uniquely identify the card itself, as well as the owner or user of the card.

The real reason unique identifiers are usually in some form of number or combination of letters and numbers is to avoid real person's names that are never unique. Unique identifiers can be sorted to avoid duplicates. Sorting also allows for extremely fast look ups by database systems.

Now you should begin to appreciate what a GUID can be used for. In a technical sense, any GUID like those used in our example, is a 128-bit number. It is a unique identifier, but it is globally unique in that it has no real future or past time limit. A GUID does not have any geographical limit either. It is global in terms of both time and area of use.

Filling A Block Using Automatic Numbers

When you **Block Fill** using automatic numbers, you can choose one of two options related to the **Fill Order**. The next table below shows the result of filling a block with five columns where the automatic numbers first increase going down the rows.

Fill Order:

Numbers First Increase Down the Rows

Numbers First Increase Across the Columns

<	Column 1	Column 2	Column 3	Column 4	Column 5
1	1	23	45	67	89
2	2	24	46	68	90
3	3	25	47	69	91
4	4	26	48	70	92
5	5	27	49	71	93
6	6	28	50	72	94
7	7	29	51	73	95
8	8	30	52	74	96
9	9	31	53	75	97
10	10	32	54	76	98
11	11	33	55	77	99
12	12	34	56	78	100
13	13	35	57	79	101
14	14	36	58	80	102
15	15	37	59	81	103
16	16	38	60	82	104
17	17	39	61	83	105
18	18	40	62	84	106
19	19	41	63	85	107
20	20	42	64	86	108
21	21	43	65	87	109
22	22	44	66	88	110

The next table below shows the result of filling the same block of five columns where the automatic numbers first increase going across the columns.

Fill Order:

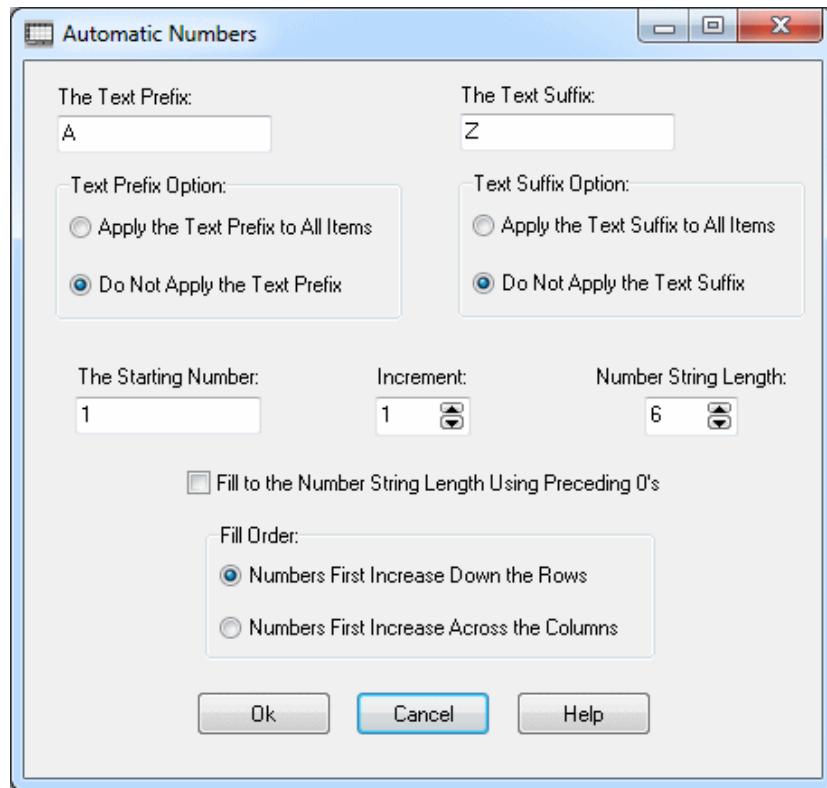
Numbers First Increase Down the Rows

Numbers First Increase Across the Columns

<	Column 1	Column 2	Column 3	Column 4	Column 5
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20
5	21	22	23	24	25
6	26	27	28	29	30
7	31	32	33	34	35
8	36	37	38	39	40
9	41	42	43	44	45
10	46	47	48	49	50
11	51	52	53	54	55
12	56	57	58	59	60
13	61	62	63	64	65
14	66	67	68	69	70
15	71	72	73	74	75
16	76	77	78	79	80
17	81	82	83	84	85
18	86	87	88	89	90
19	91	92	93	94	95
20	96	97	98	99	100
21	101	102	103	104	105
22	106	107	108	109	110

Although automatic numbers can be more complicated than the simple integers shown in the above two examples, the only point we are trying to illustrate by these two examples is to show how the numbers can increase in two different ways in left-right and top-bottom orders.

When you click the button with the caption **Setup a Range of Automatic Numbers..** you will see the next dialog box appear as shown on the next page.



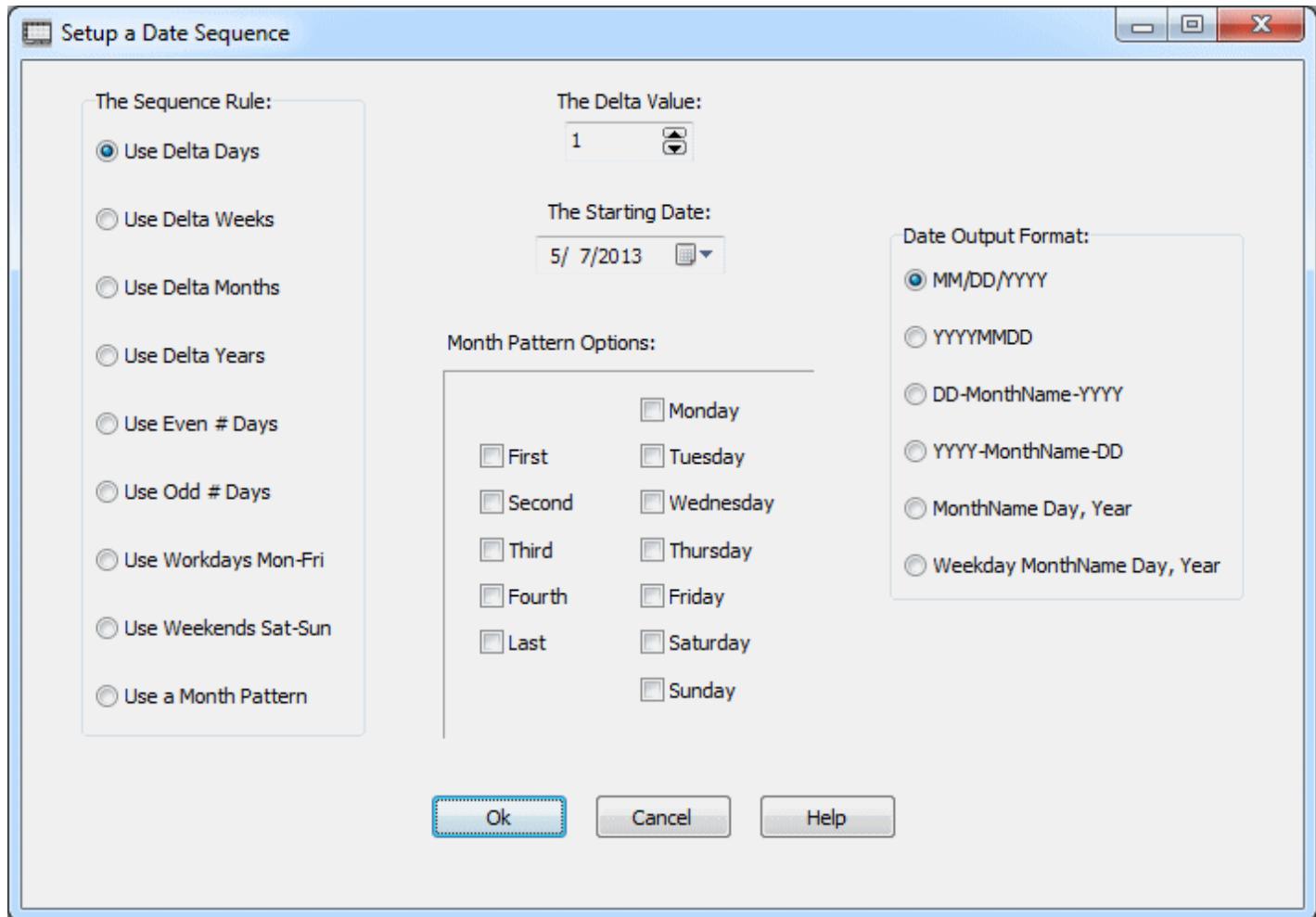
This dialog is used to determine all the parameters associated with what we are calling Automatic Numbers. You could also think of these as specialized Serial Numbers that can also begin and end with optional constant text elements. Such extra text elements are called **The Text Prefix** or **The Text Suffix**.

The numbers that are generated are whole numbers, starting with the value you enter as **The Starting Number**. Each previous number is incremented by the **Increment** value to make the next number for the next cell to be filled. The default **Increment** value is 1. The **Increment** value can range between 1 and 999.

The **Number String Length** value is ignored, unless you check the checkbox with the caption **Fill to the Number String Length Using Preceding 0's**. As an example, if the whole number were 123, and if you set a **Number String Length** of 6, and if you wanted to fill the number with preceding 0's, then the number that is placed in the grid would appear as 000123. Otherwise, without filling with preceding 0's the whole number would just appear as plain old 123.

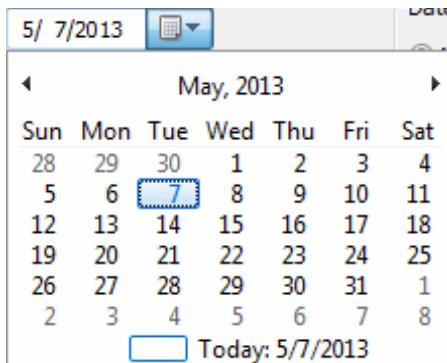
Filling Using A Date Sequence

In the main dialog for setting up a **Block Fill** operation, if you click the button with the caption **Setup a Date Sequence...** then you will see the following dialog.



We define a **Date Sequence** as a rule that creates dates that fit a chosen (but flexible) pattern. In the above dialog you can choose only one rule at a time from a list of nine rules. The other controls in this dialog may or may not be used, depending first on the **Sequence Rule** that you choose.

The one control that is always used is the one that says **The Starting Date**. This date determines a starting point for applying a date sequence rule. The starting date will always revert to the first day of the month whenever you choose to **Use a Month Pattern**. Unless the starting date already matches a sequence rule, then the first date that gets made can be one or more days after your starting date. If you click the down-arrow for **The Starting Date** you will bring up a regular calendar that appears as shown on the next page.



Using this calendar control, you can change the month, or the day, or the year for the starting date.

As with all block functions, you should normally select the block you want to fill before you open the main block fill dialog. We will give several examples of block fills using various kinds of date sequences. You can read the block examples shown below as if reading lines in a book. Read each line from left to right.

For our first example, suppose you live in a desert community in which water is limited and you are subject to a rule that says you can only water your lawn or plants on even numbered days. If you select the **Sequence Rule** that says **Use Even # Days** then you could fill a block to look like:

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Wednesday May 08, 2013	Friday May 10, 2013	Sunday May 12, 2013	Tuesday May 14, 2013	Thursday May 16, 2013
2	Saturday May 18, 2013	Monday May 20, 2013	Wednesday May 22, 2013	Friday May 24, 2013	Sunday May 26, 2013
3	Tuesday May 28, 2013	Thursday May 30, 2013	Sunday June 02, 2013	Tuesday June 04, 2013	Thursday June 06, 2013
4	Saturday June 08, 2013	Monday June 10, 2013	Wednesday June 12, 2013	Friday June 14, 2013	Sunday June 16, 2013

As another example, if you choose to use the first rule **Use Delta Days**, and if you set the **Delta Value** to be six, then with a starting date of **05/07/2013**, the program will fill the same block where each next date differs from the previous date by exactly 6 days. In this case we choose the **Date Output Format** to make dates in the form **MM/DD/YYYY**.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	05/07/2013	05/13/2013	05/19/2013	05/25/2013	05/31/2013
2	06/06/2013	06/12/2013	06/18/2013	06/24/2013	06/30/2013
3	07/06/2013	07/12/2013	07/18/2013	07/24/2013	07/30/2013
4	08/05/2013	08/11/2013	08/17/2013	08/23/2013	08/29/2013

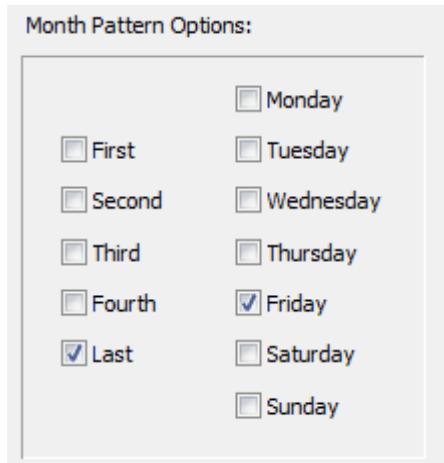
An example similar to the one above would be to **Use Delta Weeks** and set a **Delta Value** of 3. In this case we see below that successive dates differ by exactly 21 days each, that is the same as three weeks.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	05/07/2013	05/28/2013	06/18/2013	07/09/2013	07/30/2013
2	08/20/2013	09/10/2013	10/01/2013	10/22/2013	11/12/2013
3	12/03/2013	12/24/2013	01/14/2014	02/04/2014	02/25/2014
4	03/18/2014	04/08/2014	04/29/2014	05/20/2014	06/10/2014

For our next example we will choose to make weekend days and we change the output format. Note how the list below only shows Saturdays and Sundays for successive weekends from May 11, 2013 through July 14, 2013.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Saturday May 11, 2013	Sunday May 12, 2013	Saturday May 18, 2013	Sunday May 19, 2013	Saturday May 25, 2013
2	Sunday May 26, 2013	Saturday June 01, 2013	Sunday June 02, 2013	Saturday June 08, 2013	Sunday June 09, 2013
3	Saturday June 15, 2013	Sunday June 16, 2013	Saturday June 22, 2013	Sunday June 23, 2013	Saturday June 29, 2013
4	Sunday June 30, 2013	Saturday July 06, 2013	Sunday July 07, 2013	Saturday July 13, 2013	Sunday July 14, 2013

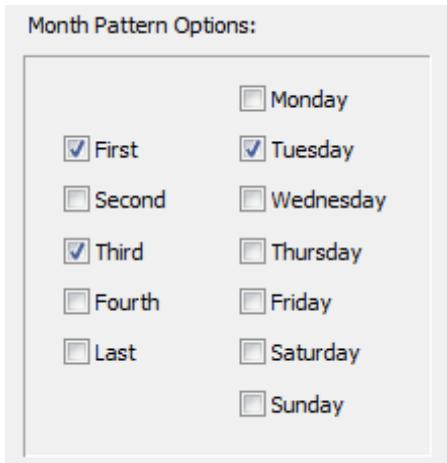
The most involved date sequences are those that **Use a Month Pattern**. For example, if we wanted a list of dates that were the last Friday of each month we would setup the **Month Pattern Options** by checking the **Last** checkbox and the **Friday** checkbox.



After selecting the **Sequence Rule, Use a Month Pattern**, the dates that would be in this sequence would be the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Friday May 31, 2013	Friday June 28, 2013	Friday July 26, 2013	Friday August 30, 2013	Friday September 27, 2013
2	Friday October 25, 2013	Friday November 29, 2013	Friday December 27, 2013	Friday January 31, 2014	Friday February 28, 2014
3	Friday March 28, 2014	Friday April 25, 2014	Friday May 30, 2014	Friday June 27, 2014	Friday July 25, 2014
4	Friday August 29, 2014	Friday September 26, 2014	Friday October 31, 2014	Friday November 28, 2014	Friday December 26, 2014

As our last example, suppose you belong to a social club that meets on the 1st and 3rd Tuesdays of every month. In this case you would set the **Month Pattern Options** as shown on the next page. This example illustrates that you can check more than one checkbox in the **Month Pattern Options**. You could also check more than one day name checkbox, as needed.



The output from this example is:

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Tuesday May 07, 2013	Tuesday May 21, 2013	Tuesday June 04, 2013	Tuesday June 18, 2013	Tuesday July 02, 2013
2	Tuesday July 16, 2013	Tuesday August 06, 2013	Tuesday August 20, 2013	Tuesday September 03, 2013	Tuesday September 17, 2013
3	Tuesday October 01, 2013	Tuesday October 15, 2013	Tuesday November 05, 2013	Tuesday November 19, 2013	Tuesday December 03, 2013
4	Tuesday December 17, 2013	Tuesday January 07, 2014	Tuesday January 21, 2014	Tuesday February 04, 2014	Tuesday February 18, 2014

We should point out that the **Delta Value** is only used if you select one of the first four **Delta Sequence Rules**. For all other **Sequence Rules** the **Delta Value** is ignored. When used however, the **Delta Value** is subject to interpretation by the chosen **Sequence Rule**. Such a value might represent days, or weeks, or months, or years. If you chose the starting date as your next birthday date, and if you chose the rule to **Use Delta Years**, you could make a list of all your future birthdays for the rest of your life!

As with all of our functions that write dates, you can choose from any one of six **Date Output Formats**. After generating dates, you can later use Block Formatting to change the date format. You could also use the **Conversions** menu to convert the dates back into actual **Julian Day Numbers**.

All date **Sequence Rules** are such that dates will continue to be generated until your selected block is filled with dates. The program usually starts with the **Starting Date** and as it generates successive dates it checks to see if the date fits your chosen **Sequence Rule**. If it does, that date is output in the chosen output format and the next date is tested. The next date is usually just the next calendar day, unless you have chosen a **Delta Rule** in which case multiple days may be skipped automatically.

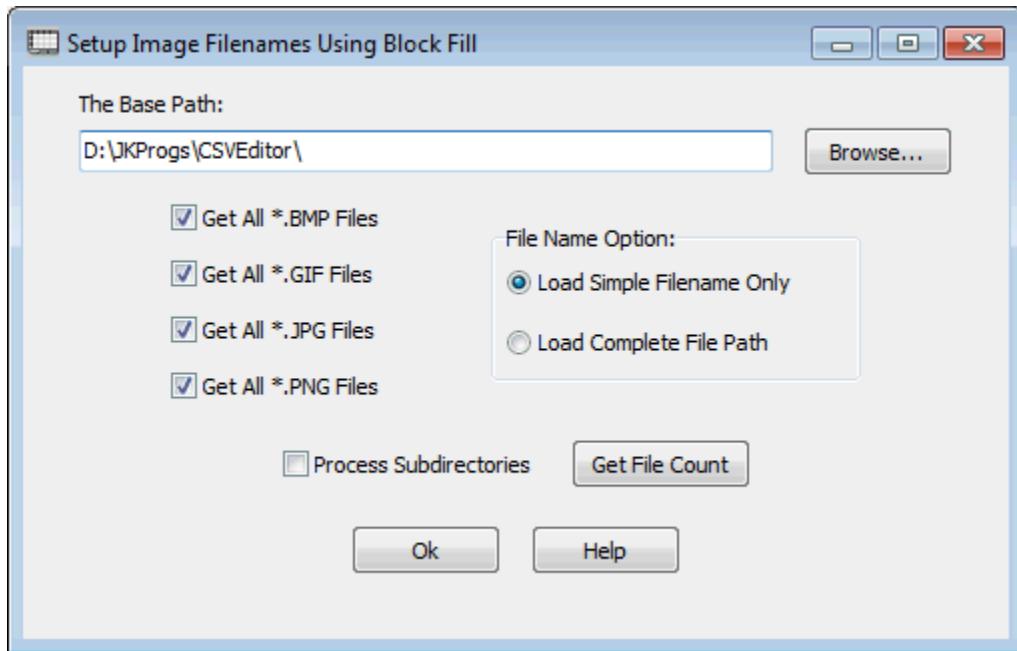
If you choose to **Use a Month Pattern**, but you don't check any of the count checkboxes or day name checkboxes, then the program will not let you close the dialog. Instead, it will give you a warning message to check a checkbox. In this case you could also make a different selection than the **Use a Month Pattern** option without checking any checkbox. The **Month Pattern Options** do not apply if you choose any option except the one to **Use a Month Pattern**.

Filling Cells With Image Filenames

You can fill a column of cells where you let the program automatically choose existing filenames to fill the cells. This can save you a lot of searching and typing. For this special function, the filenames are intended to be for picture images that can be either ***.BMP** or ***.GIF** or ***.JPG** or ***.PNG** image files. These four file types are the only allowed picture types. When you click the button



you should see the following dialog.



You can type in **The Base Path** and you can then choose any combination of the four possible file types. The final **File Name Option** is whether you want the program to load simple filenames only, or whether you want to load the complete path to each file. The filenames/filepaths are what get loaded into the main grid cells.

Another option is controlled by the checkbox with the caption **Process Subdirectories**. When this is checked, the program will recursively inspect all subdirectories under **The Base Path**. Be sure to allow enough empty rows in your grid to accommodate a large number of files when this option is checked.

It is an option to click the button with the caption **Get File Count** to just get a report on the number files that will be found when the function executes. This report can help you determine ahead of time how many additional rows you might want to consider to add to your current grid.

After you click the **Ok** button you will be back in main Block Fill dialog where the program should automatically have chosen the radio button with the caption **Use Image Display Filenames**.

For this special function, the program will only try to fill names within a single column. That column is the one defined by the **Block First Column**. The **Block Last Column** number is not used at all. However, the block range of rows can be used to limit how many rows in the **Block First Column** are used.

Thus you should insure the range of rows is set as desired before you click the **Ok** button that performs the block fill function. The program will scan all the image files and try to load the names into the range of selected rows. Of course, not all the rows will get filled if your directory (or subdirectories) contains only a few picture files. If you have more picture images than there are rows in the selected block, then some of the image files will not get listed.

You can use the **Browse...** button to automatically change **The Base Path**. The only unusual feature to know about using the **Browse...** button is that you must click on an image file in any subdirectory as if you were going to open that file, but the program will simply set the path for the file in **The Base Path** edit box, without actually opening any file.

Once the **Block First Column** has been loaded with image filenames you can click any cell in that column and then select the menu item **Utilities | View Images Using Grid Cell Filenames...** to view the picture images. You can scroll up and down through the list to quickly view the images one at a time.

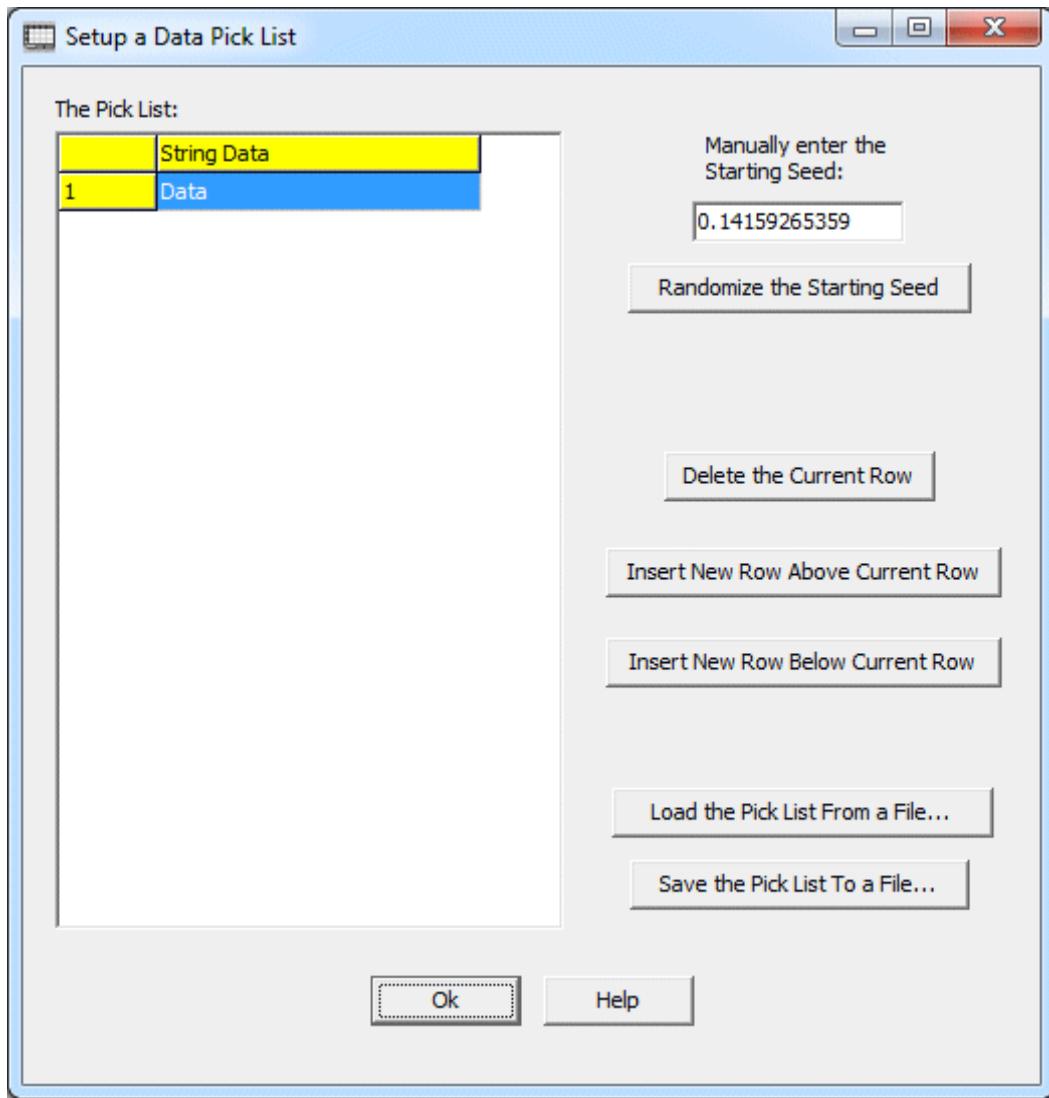
An example listing with a mixture of image filenames is the following. In this example we list simple filenames and not complete file pathnames.

<	Column 1
>	Picture Names:
1	BabyPicture.bmp
2	CanyonFog.gif
3	ClockPicture.jpg
4	Color2.png
5	DeerInCanyon.jpg
6	Eclipse.png
7	Family001.jpg
8	Family002.jpg
9	Family003.jpg
10	Fullmoon.bmp
11	Jupiter.bmp
12	Mars.gif
13	Mercury.bmp
14	MonaLisa.gif
15	Neptune.bmp
16	People.gif
17	Pluto.jpg
18	Saturn.bmp
19	Scanner.png
20	Sun.bmp

See also the Help topic **Filling Cells With File Information**.

Using a Pick List to Fill a Block

You can fill a block of cells where you randomly choose the data from a pre-determined list that we call a **Pick List**. In the Block Fill dialog, if you click the left-side button with the caption **Setup a Random Data Pick List...** you will bring up the following dialog box:



This dialog box is used to fill in a list of items that form what we call a **Pick List**. A **Pick List** is usually just a list of strings from which you can randomly pick items. In fact, this dialog box functions almost identically to the dialog box that is used to create a list of **Validation Strings**.

Using the controls to the right of the grid you can manually create your own custom **Pick List**. As an example, suppose you have a column in a grid that is to contain the names of some prominent golfers. By editing the **Pick List** and adding new rows you could generate a list that looks as follows:

The Pick List:	
	String Data
1	Tiger Woods
2	Rory McIlroy
3	Phil Mickelson
4	Arnold Palmer
5	Jack Nicklaus
6	Bubba Watson
7	Gene Sarazen
8	Sam Snead
9	Bobby Jones
10	Lee Trevino

We have listed only 10 names, but a given **Pick List** can contain any number of items that you need.

When a **Pick List** is used to randomly fill a block of cells, the action that takes place can be described very simply. Each time you need to fill a cell you just make a random pick from the **Pick List**. A given item from the **Pick List** might be chosen more than once, especially if the number of cells in the block is very large compared to the number of items in your **Pick List**.

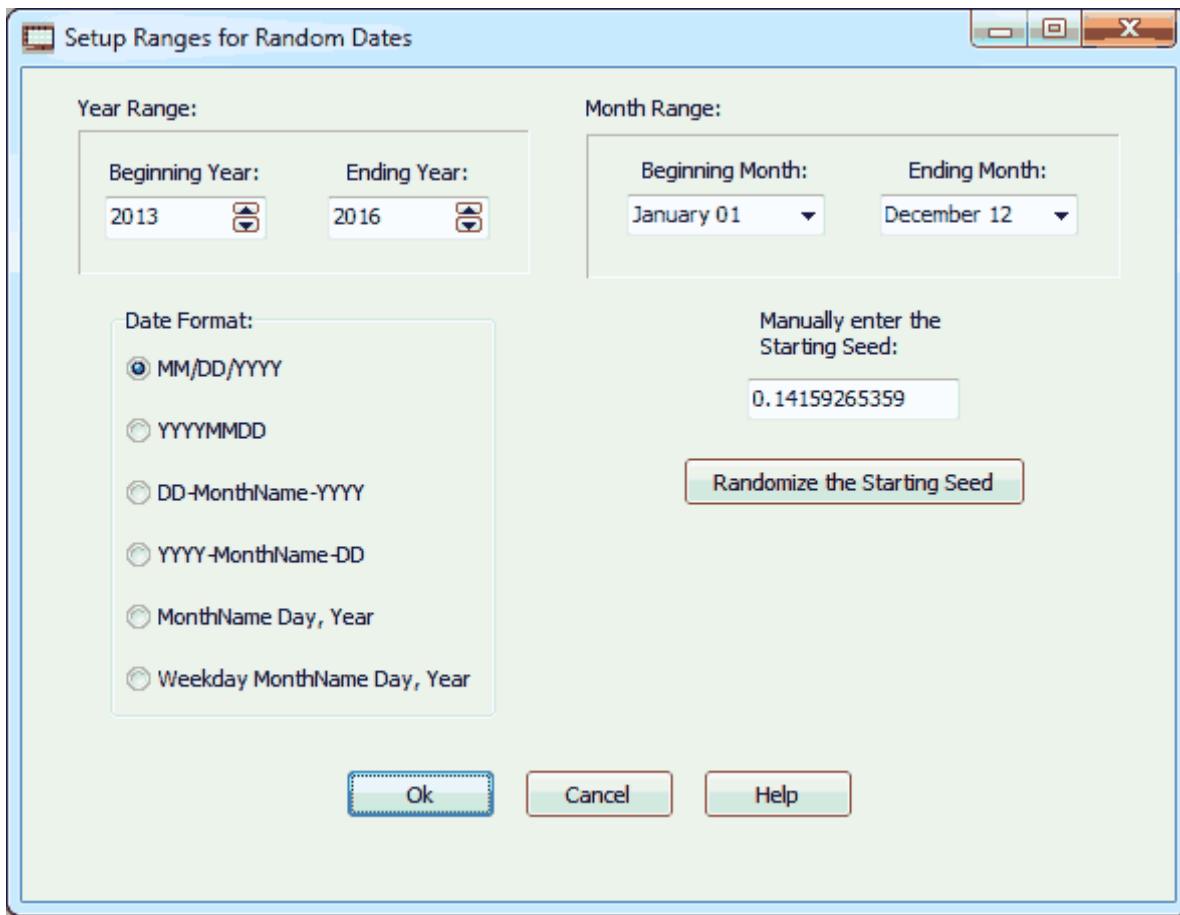
The **Pick List** dialog also has a couple of items used to setup the random number generator seeds. You can manually type in a starting seed or you can let the program automatically create a random starting seed. The seeds used in this dialog differ from the seeds used to generate plain random numbers or the seeds used to generate random dates. Allowing you to enter a starting seed gives you the ability to exactly repeat a previous random sequence at any later time.

When you save a **Pick List** in a file, that file will be a **CSV** file with just one column of data. You can also load a **Pick List** with an ordinary text file. If you know you are going to use your **Pick List** more than once then it is a good idea to save your **Pick List** in a file.

Although this program has a special function devoted to generating random names, we could generate random names using multiple **Pick Lists**. We could use one **Pick List** to generate first names in a certain column, then we could use another **Pick List** to generate last names in another column. The **CSV Editor** program is supplied with multiple name lists that can be used as **Pick Lists**. To generate real random names it is easier to use the built-in function that helps with this automatically. Distinct from human names, we also provide a list of business names that could also be used as a **Pick List** for when you need random names of businesses.

Filling With Random Dates

In addition to filling a block with random numbers, you can also fill a block with random dates. Random dates are different from Date Sequences. To fill with random dates you can bring up the command **Blocks | Block Fill** and once you see the block fill dialog box that was shown above in this help file, you can click the button with the caption **Setup Ranges for Random Dates...**. This will bring up the dialog box that appears as:



In this dialog you can setup a range of dates that will include a set of consecutive years called the **Year Range** and you can setup a range of consecutive months called the **Month Range**. There is no Day Range because the program will automatically generate random numbers in the range from 1-31 that represent random days within any month. In fact, the process will account for leap years for the month of February that can have at most either 28 or 29 days and it will distinguish other months that have either 30 or 31 days. Thus all random dates that are generated are truly random dates, but they always represent valid dates.

You can choose any one of six output formats for the dates that can be generated by choosing one of the radio buttons in the **Date Format** group of radio buttons. These are the six formats that the program uses for almost all of its date processing functions.

We should also mention that the **Starting Seed** used to generate random dates is different from the **Starting Seed** shown in the dialog that generates random numbers. You can manually enter any **Starting Seed** or you can press the button to have the program randomize the **Starting Seed**. Manually entering a **Starting Seed** is an option that provides the ability to exactly repeat a random experiment with dates data.

When you generate Random Dates, the program will generate dates for the ranges you set. It will generate as many dates as needed until the block is completely filled. Depending on the size of your block and the ranges of your dates, it is possible that any generated date may appear more than once. This will be rare when you set a large range for your dates, or when your block is relatively small. You should just be aware that we don't try to generate unique dates. Later, you could check the results for uniqueness using the **Uniqueness** menu.

When you set the ranges for the dates, the **Ending Year** must never be strictly smaller than the **Beginning Year**. If you should violate this restriction, the program will automatically change the **Ending Year** value to make it the same as the **Beginning Year** value. You can have the same value for the **Beginning** and **Ending Years**. In that case the dates that are generated will all be within the same year.

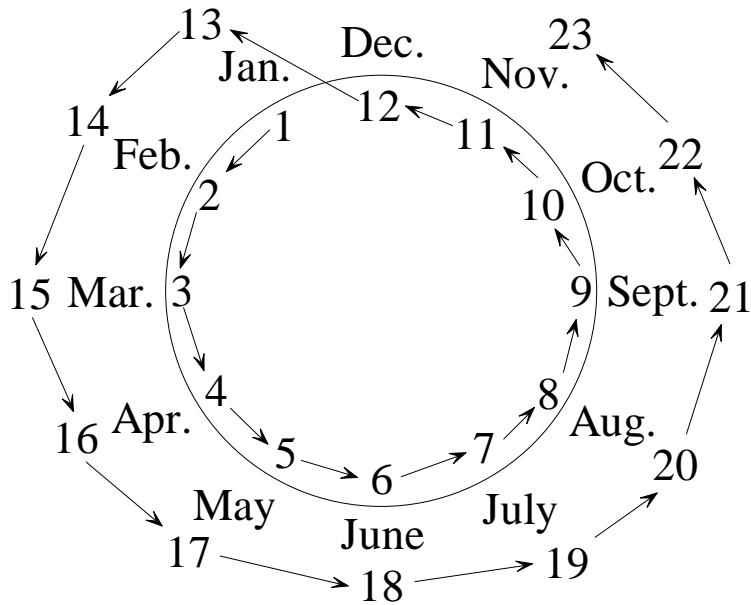
Note that having a **Month Range** allows you to generate dates for things like summer months or months that fall within a given quarter of a year or half of a year, or any other desirable range. The **Month Range** we actually use has an unusual property that differs from the **Year Range** property. As just stated above, your **Ending Year** value must be at least as large as the **Beginning Year** value. This same property is not true for Months. The **Ending Month** can be either the same or can be before the **Beginning Month**.

As an example of how our special algorithm works for Months, if you choose October as the **Beginning Month** and choose March as the **Ending Month**, the numeric month range is not the interval [10, 3]. Instead, when the **Ending Month** number is smaller than the **Beginning Month** number, we make the month range the special interval [10, 15] where we just add 12 to the smaller ending number. Then we cyclically continue double numbering the months until we reach the number 23. We then generate random month numbers using the interval [10, 15]. If any generated value is larger than 12, we subtract 12 from that number. This guarantees the final Month number will be in the normal range from 1 to 12. The Months that will be chosen for the [10, 15] interval would correspond to:

October(10) → November(11) → December(12) → January(13) → February(14) → March(15)

In particular, if 14 were a generated random Month number in the interval range [10, 15], after we subtract 12 from 14 we get 2, that really represents the Month of February.

To more completely understand what we are doing, you should visualize the months as in the circular pattern shown in the next figure. You can see the extended special double numbering outside the circle and you can see how it continues around in a counter-clockwise direction.



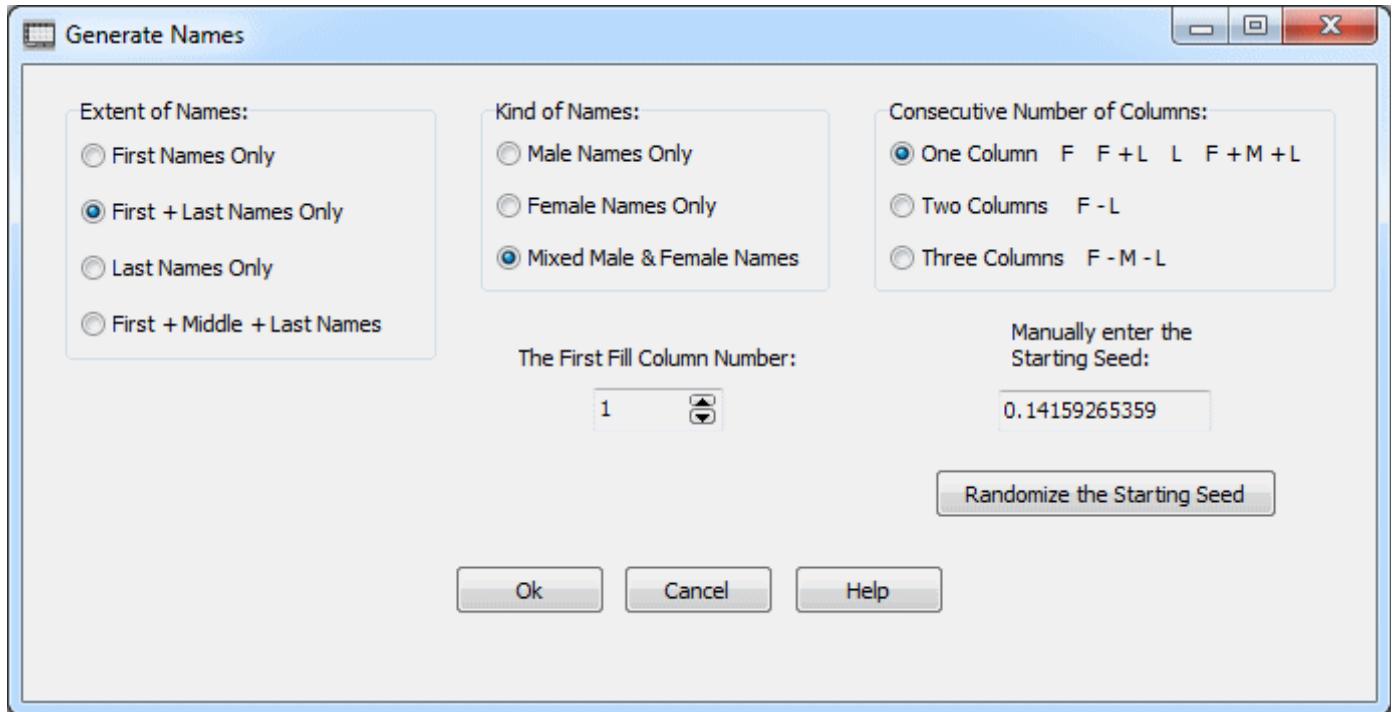
With this special numbering algorithm, you can select a range of months that can span a winter season with true consecutive months. As illustrated by the above example, the months could go from October to March in a counter-clockwise direction, taking on the intermediate consecutive months of November, December, January, and February.

An example of generating random dates for this range appears below.

<	Column 1	Column 2	Column 3	Column 4
1 >	12/28/2013	01/17/2013	10/02/2013	11/10/2013
2	12/13/2013	11/11/2013	01/08/2013	11/05/2013
3	03/16/2013	11/30/2013	02/03/2013	12/06/2013
4	12/21/2013	10/02/2013	10/02/2013	12/10/2013
5	01/30/2013	10/08/2013	11/14/2013	12/27/2013
6	11/20/2013	02/18/2013	11/24/2013	12/10/2013
7	12/22/2013	03/13/2013	10/10/2013	01/31/2013
8	11/25/2013	03/17/2013	11/22/2013	12/12/2013
9	12/22/2013	11/05/2013	01/10/2013	11/08/2013
10	11/07/2013	10/30/2013	10/22/2013	11/20/2013

Filling a Block With Random Names

When you press the Block Fill dialog button with the caption **Setup Random Names Generation** you will bring up the following dialog box.



There are more than a few subtle options to understand here. First, you can choose any one of four kinds of combinations of names that can include **First Names Only**, or **First+Last Names**, or **Last Names** only, or **First+Middle+Last Names**.

The next choice has to do with gender. You can generate male names only, or generate female names only, or you can generate both male and female names. You might note these choices only affect what gets chosen for both first and middle names. Also, we make an effort to not choose the same middle name as the first name.

The last set of choices has to do with how many columns you want the generated names to occupy. You can only occupy one or two or three columns. Any of the first four name extent choices can be placed in a single column. If you choose to use two columns, then you must choose to generate only the first and last names. When and only when you choose to use three columns then you must also generate all three of the first and middle and last names.

You can set any column as a first column, but after that, if you plan to use either two or three columns, then any and all columns required after the first column must follow the first column in order and they must be consecutive columns.

Like all random data generating schemes used by the **CSV Editor** program, you can set a starting seed that is used to control the generation of random numbers.

We might also mention that when the **CSV Editor** program first loads, it will look for three special **CSV** files that contain male names, female names, and last names. These files are distributed with the program and should remain undisturbed in the main directory where the **CSV Editor** program is installed. If these files are changed or deleted then you won't be able to generate random names.

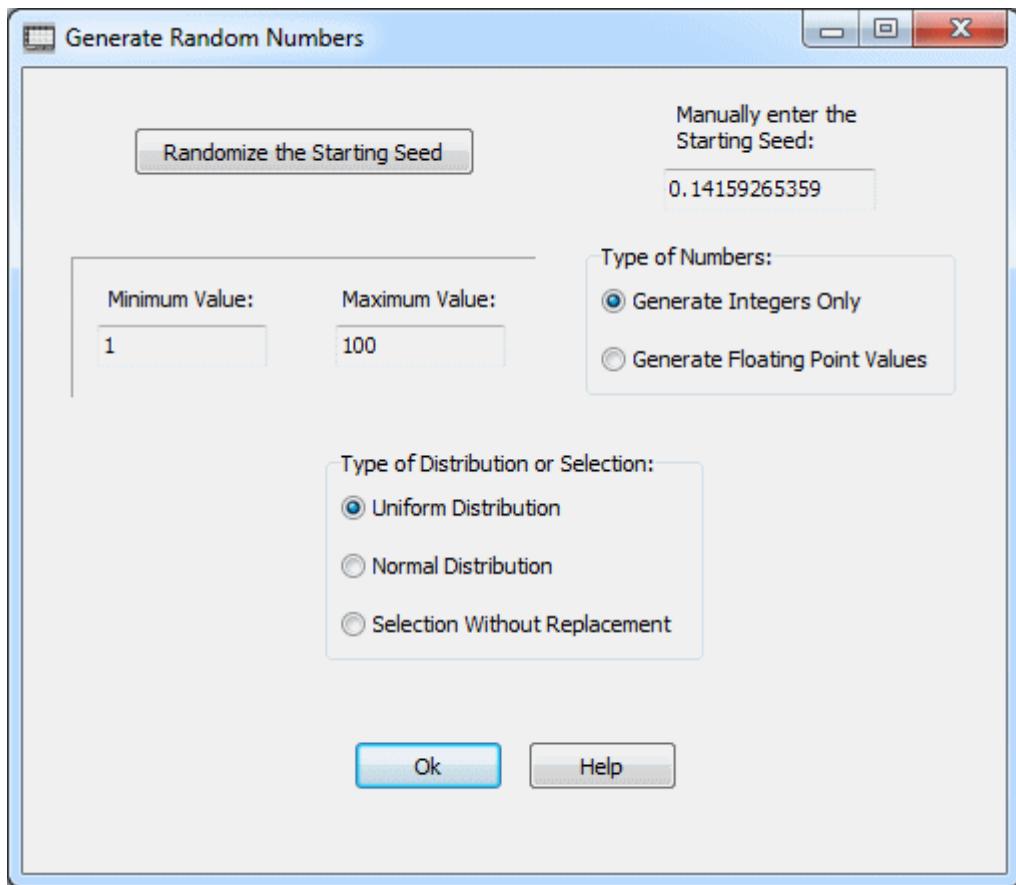
When you click the **Ok** button you will be back in the main dialog for doing a Block Fill.

We should also point out that although the main Block Fill dialog allows you to select multiple columns, the generation of random names only uses the **Block First Column** number in that dialog. This is because the random names setup dialog determines how many additional columns will be used (if any) beyond the **Block First Column**. It might be best to just think of filling a block with random names as being a single column block fill operation (even though as many as 3 consecutive columns may actually be used). Just make sure whenever your random names will occupy a 2nd or even a 3rd column, that you have created one or two extra blank columns, as needed. Otherwise this function will overwrite any existing data when it fills in one or two extra columns beyond the **Block First Column** number.

This help topic has been concerned with generating names of persons. If you need to generate names of businesses then we recommend you load the file **BusinessNamesList.csv** into the dialog used to **Setup a Random Data Pick List** after you select the function **Blocks | Block Fill...**.

Filling With Random Numbers

In the main dialog for setting up a **Block Fill** operation, if you click the button with the caption **Setup a Range for Random Numbers...** then you will see the following dialog in which you can determine all the parameters associated with generating random numbers.



What is called the **Starting Seed** should be a decimal number in the range between 0 and 1. The reason for allowing you to manually set the starting seed is so you can generate results that can later be repeated exactly. If you don't care about repeating the same sequence of numbers at another time, then you can click the button with the caption **Randomize the Starting Seed** and the program will automatically determine another random seed value for you, based on a random time determined by your computer's system clock at the time you pressed the button. All seed values are decimal values in the range between 0 and 1.

The main parameters are the minimum and maximum values. These two numbers determine a range for the random numbers that will be generated, especially when the **Uniform Distribution** type is selected. The Maximum value must be strictly larger than the Minimum value, or you will see an error message after the program automatically enforces this restriction by setting the Maximum value as 1 more than the Minimum.

If you choose the **Normal Distribution** then the mean and standard deviation are automatically determined by the range of the minimum and maximum values. In this case the mean would be automatically determined as the midpoint between the minimum and maximum values. The standard deviation would be the number that is the maximum value minus the minimum value, all divided by 6.

With this scheme it is easy to understand why we would have about three standard deviations above and below the mean. This implies almost all (99.7%) numbers that are generated with the **Normal Distribution** can be expected to lie within the minimum and maximum range.

If you want a different mean or a different standard deviation all you have to do is define appropriate minimum and maximum values. If μ denotes the desired mean and σ denotes the desired standard deviation, then you can compute that: Minimum Value = $\mu - 3\sigma$ and Maximum Value = $\mu + 3\sigma$.

Going the other way around:
$$\mu = \frac{(\text{Maximum Value}) + (\text{Minimum Value})}{2}$$

$$\sigma = \frac{(\text{Maximum Value}) - (\text{Minimum Value})}{6}$$

If you choose **Selection Without Replacement** then the program will automatically switch to make the type of numbers generated integers only. In this case the program will normally select all integers between the minimum and maximum values inclusive. However, the main difference between the **Uniform Distribution** and **Selection Without Replacement** is that with a **Uniform Distribution** it is possible that a given integer may be selected more than once, especially if the number of selections (i.e. the number of block cells) is much larger than the range of integers.

Using **Selection Without Replacement** can be thought of like having a bag full of numbered ping pong balls, where the numbers range between the minimum integer and the maximum integer that you enter in the dialog. Generating a new number is like reaching into the bag and randomly selecting a ball and then removing that ball from the bag so it can't be selected again. You would keep extracting balls from the bag until the bag is empty or until your block of cells is filled, whichever occurs first.

You would normally choose **Selection Without Replacement** when you want to generate all numbers in a random order, but you don't want to select a given number more than once.

As an example, suppose a football stadium has a block of consecutive empty seats numbered between 23 and 56 inclusive. If you had a group of 34 football fans, you could randomly assign each fan a seat number in the range between 23 and 56. In this case you would want to assign each fan a random seat, but you would not want to assign more than one fan to a seat and you would not want any seat to go empty. You would setup the range of numbers for **Selection Without Replacement** as:

Minimum Value:	Maximum Value:
23	56

When you apply **Selection Without Replacement** there is a subtle relationship between the range of integers you specify and the size of the block you choose to be filled. If the number of integers between the minimum and maximum that you specify is the same as the number of cells in the block you specify then your block will be exactly filled by all the integers. If the block contains more cells than there are integers in your range then after all the integers have been selected, all remaining block cells will simply be filled with blank entries. This guarantees you never generate integers outside the range that you specify, and that you never generate any integer more than once.

If the block you specify contains fewer cells than there are integers between your minimum and maximum values inclusive, then you may have unused or unselected integers that don't appear anywhere in the output block. Whenever we apply **Selection Without Replacement** we always try to insure the number of cells in our block exactly matches the range of integers that we define.

In the above stadium seat example you might note that $34 = (56 - 23) + 1$.

In general, the number of block cells should normally be the maximum integer minus the minimum integer, plus one.

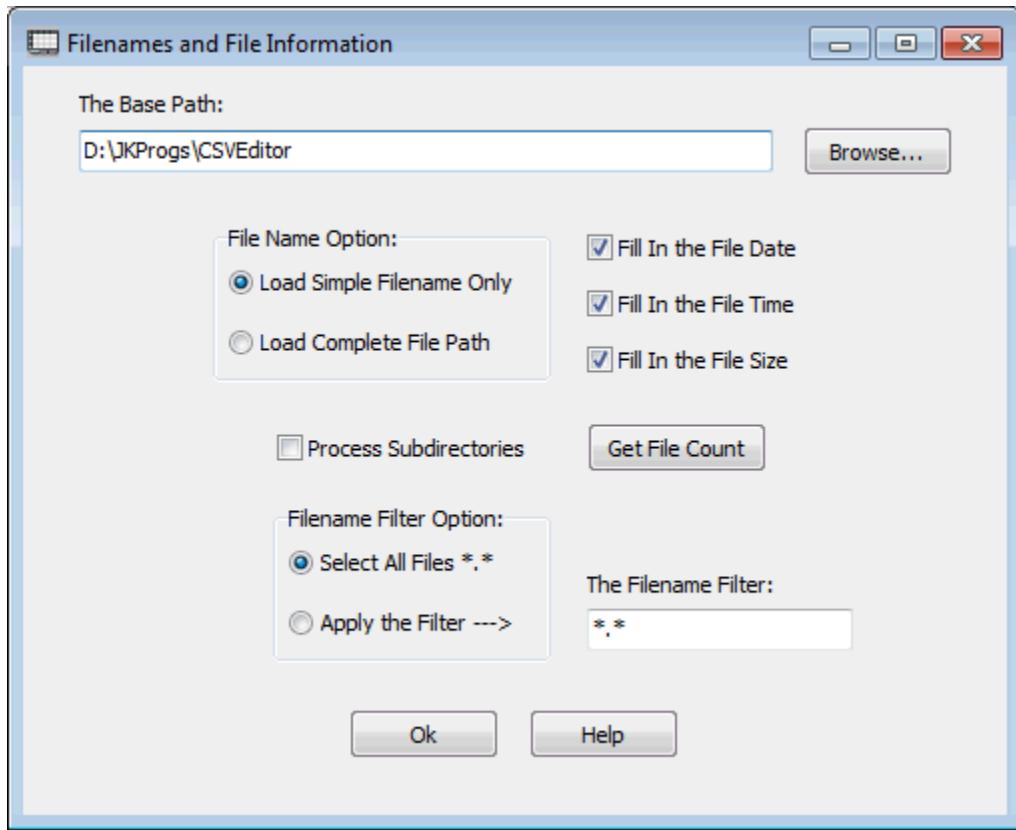
In general, there are two block fill functions that you might think are missing in this program. One function would be for generating random boolean values and the other would be for generating random times. To create a random block fill for boolean values we suggest using a Pick List with only the two choices of 0 and 1. Later, you could reformat these value as F and T values or any other boolean format that you prefer.

To generate random times you can use the random number generator where you generate decimals that are only in the range between 0 and 1. Then convert those random decimals to random times by applying the function under **Conversions | Convert Times <--> Decimal Values....** Thus this program really does provide for generating any kind of random values that we support as our native data types.

As a more specific time example, if you wanted to generate random times during the normal daytime working hours between 9:00 AM and 5:00 PM, then you should choose the decimal range to be between 0.375 and 0.70833.

Filling Cells With File Information

If you are in the main **Block Fill** dialog and you click the button with the caption **Setup Regular Filenames Information...**, you will see the following dialog:



The above dialog is used to set a directory from which file information can be subsequently loaded into an existing **CSV** grid. **The Base Path** edit box is used to select any base directory path. You can also use the **Browse** button to navigate to any directory, but if you do, then you should simply double click any file within that directory to have that directory path automatically entered for you in the above edit box. If there are no filenames for you to click on, then you should just manually type in **The Base Path** without using the **Browse** button.

Next you should choose one of the two **File Name Options**. The program can enter simple filenames or it can load complete file paths, including the filenames. Next, you can check any or all or none of the three check boxes to fill in either the **File Date** or the **File Time** or the **File Size** information.

Another option is controlled by the checkbox with the caption **Process Subdirectories**. When this is checked, the program will recursively inspect all subdirectories under **The Base Path**. Be sure to allow enough empty rows in your grid to accommodate a large number of files when this option is checked.

It is an option to click the button with the caption **Get File Count** to just get a report on the number files that will be found when the function executes. This report can help you determine ahead of time how many additional rows you might want to consider adding to your current grid.

There is a **Filename Filter Option** that can either **Select All Files** or you can **Apply the Filter** where you can enter a filetype in the edit box. Filetypes should be entered like the examples ***.mp3** or ***.txt** or ***.csv** or ***.pdf** where the filetype must end with a period character followed by some ordinary text that is usually a small number of characters.

Then when you click the above **Ok** button you will be back in the main **Block Fill** dialog where you can further select a range of rows. For this block fill function, the program does not really use the range of columns in the **Block Fill** dialog, but it does use the **Block First Column** to determine the first column used when loading the file information. Then when you perform this special block fill function, the program will load up to four consecutive columns worth of information. An example is shown below where we have inserted a header row that identifies the type of file information shown in each column. Note that this example has filled four columns, starting with Column 1.

<	Column 1	Column 2	Column 3	Column 4
>	The Filename/Filepath:	File Date:	File Time:	File Size (Bytes):
1	C:\MyDocuments\Hp\35s.zip	12/10/2011	01:41:22 AM	7,352,849
2	C:\MyDocuments\Hp\Bp.dsk	05/25/1995	12:06:44 AM	671
3	C:\MyDocuments\Hp\Chhu11.ra	12/22/2005	10:24:36 AM	33
4	C:\MyDocuments\Hp\Chhu12.ra	12/22/2005	10:25:12 AM	33
5	C:\MyDocuments\Hp\Directions.txt	11/28/2000	10:33:20 PM	3,815
6	C:\MyDocuments\Hp\FirstPageHP12C.pdf	08/07/2011	04:03:30 PM	128,772
7	C:\MyDocuments\Hp\Grob2tif.exe	11/17/1989	05:57:22 PM	5,472
8	C:\MyDocuments\Hp\Hp-42.txt	11/07/1992	07:41:48 PM	42,477
9	C:\MyDocuments\Hp\hp12c.zip	01/04/2012	02:54:42 PM	1,071,909
10	C:\MyDocuments\Hp\HP12CBByJKN3.jpg	08/04/2011	01:27:54 AM	4,298,750
11	C:\MyDocuments\Hp\HP12CN2ByJK.jpg	08/04/2011	01:50:14 AM	1,031,800
12	C:\MyDocuments\Hp\HP12CPlatinumCalculatorSpecSheet.pdf	07/27/2011	11:50:58 AM	429,684
13	C:\MyDocuments\Hp\HP12CPlatinumKeyboardView.JPG	07/27/2011	01:37:42 PM	101,813
14	C:\MyDocuments\Hp\HP12CPlatinumKeyboardView.pdf	07/27/2011	01:41:20 PM	91,235
15	C:\MyDocuments\Hp\HP12CPlatinumUsersGuide.pdf	08/07/2011	04:05:00 PM	2,444,338
16	C:\MyDocuments\Hp\HP12CQuickStartGuide.pdf	07/29/2011	01:29:42 PM	691,195
17	C:\MyDocuments\Hp\Hp12csolutionshandbook.pdf	07/26/2011	08:53:22 PM	843,112
18	C:\MyDocuments\Hp\HP15cUserManual.pdf	11/30/2011	03:56:46 PM	3,938,941
19	C:\MyDocuments\Hp\Hp25.pdf	12/21/2005	10:10:00 AM	12,763,847
20	C:\MyDocuments\Hp\hp35.pdf	05/16/2005	10:04:56 AM	7,411,452
21	C:\MyDocuments\Hp\HP3in1Emu.zip	01/04/2012	02:56:48 PM	2,501,657

The first used column will always hold the filename or the complete filepath. The next column will hold the file date. The column after that will hold the file time. The column after that will hold the file size in bytes. Only any of these last three columns will be used if you checked the corresponding check boxes. The number of extra columns used (following the filename/filepath column) depends on how many check boxes you checked.

If you don't check any of the three check boxes then only the filename column will be filled with information. If you check only 1 or 2 check boxes, then only 1 or two additional columns will be used after the filename column. In the example shown on the next page, we checked all three check boxes.

Depending on your selected row range and the number of files contained in **The Base Path** (and other subdirectories), the program will fill as many rows with file information as it can. However, it will not fill any rows outside of your selected row range, and if you have only a few files, it will stop entering file information before it finishes using all the grid rows that you have chosen. The loading stops whenever the program first runs out of rows or first runs out of files.

In case you don't have three columns after the **Block First Column**, then the program will only fill in file date and file time and file size information for as many columns as are actually available, past the filename/filepath column. So the program does NOT expand the grid to add any additional columns to accommodate all the file information. If any files in your list have a filesize of 0 you can try turning off the Read Only attribute of the file and then reload the list.

You need to be careful about overwriting data in columns that are to the right of the **Block First Column**. So if you check any of the three optional check boxes you need to be aware that data in any extra columns can be overwritten with the extra file information. A good practice is to always have or to always insert three blank columns after the **Block First Column**, before you apply this block fill function.

After a grid has been loaded with file information you may wish to perform further editing to eliminate any undesired files. In practice you may find it easier to work with blocks when you fill a block with the full filename paths as opposed to just using the simple filename only.

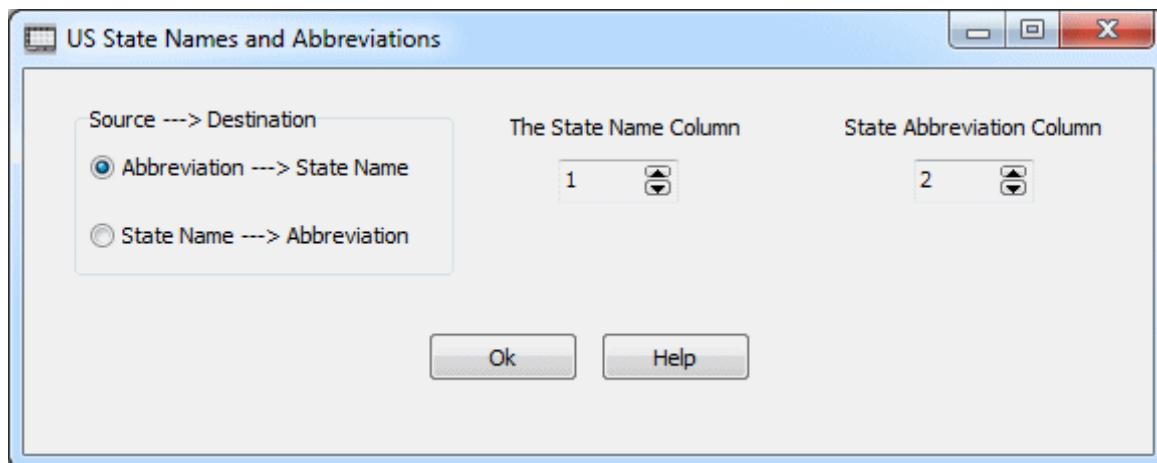
In addition to just creating a list of existing files, once you make such a list you can select any filename cell in your list and then select the function **Utilities | Launch a File From the Current Column...** to bring up a dialog box that allows you run a program associated with that particular filename. If the filename itself is an executable file, then when you launch that file the **CSV Editor** program will run that program. This works best when you load complete file paths.

Otherwise, if the file is not an application or program, the **CSV Editor** program will try to run an application that is associated with that file. For example, if the file is a text file, the **CSV Editor** program will try to open a word processor for that file. If the file is a *.mp3 music file then the **CSV Editor** program will try to play that music file. If the file is a picture image, the **CSV Editor** program will try to run a program that can display that picture file. Thus the ability to launch a single file is a powerful tool.

As discussed earlier in this help file, if you want to view a series of picture images then you should not try to load regular filename information. Instead, you should select the button under the **Block Fill** dialog that has the caption **Setup Image Display Filenames**. After that, select **Utilities | View Images Using Grid Cell Filenames....**

Automatic Fill In of US State Names or Abbreviations

If you are in the main **Block Fill** dialog and you click the button with the caption **Setup US States/Abbreviations** you will bring up the following dialog box.



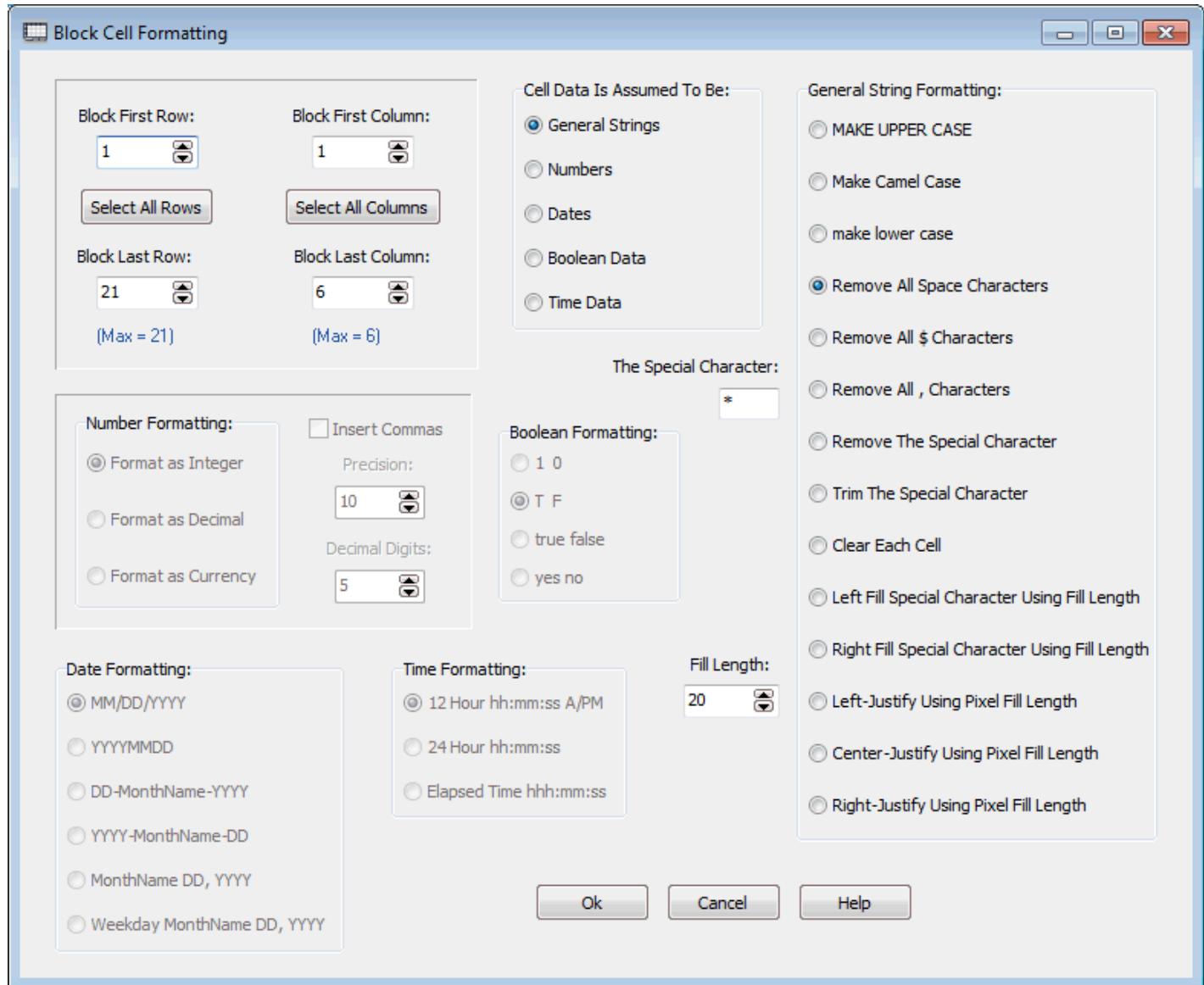
The idea here is that you have already opened a string grid which has two columns, one containing the US State Names and another containing the US State Abbreviations. Actually, only one of these columns needs to contain the real information and the other column could be blank, or not. The whole purpose of this automatic block fill operation is to save you from doing any manual typing. In other words, depending on what State information you already have, this function will automatically fill in the other type of State information.

If you already have the 2-letter State Abbreviations and want to fill in the full State Names then you would select the first radio button in the group on the left. If you already have the full State Names and you want to automatically fill in the State Abbreviations then you should select the second radio button in the group on the left. After that, just set the two column numbers and when you click the **Ok** button you will be back in the main Block Fill dialog where you need only select the range of rows to which you wish to apply this operation. In this special case you might note that the Block Fill column numbers in the main dialog are ignored because this function will get the two column numbers from the dialog just shown above.

If you don't have a column with either kind of state information, then you could apply the method that uses a **Pick List** and open the file **UnitedStatesList.csv** as a finite **Pick List** and use that to fill your block with random state information.

Block Formatting

Under the **Blocks** menu you can select the menu item **Block Formatting...**. This menu item is used to automatically format all the cells in a selected block of cells. When you do this you will see a dialog box that appears similar to the following:



The controls in the upper-left corner of this dialog are used to define the block to be formatted. Note that you can click a button to either **Select All Rows** or **Select All Columns** if you want the largest possible block.

Many times the block you define will be limited to a single column or a specific range of rows in a single column. In any case, the first decision you must make concerns what the cell data is assumed to represent. We assume all the cells in your block should be of the same type of data that can only be one of the five choices: **General Strings, or Numbers, or Dates or Boolean Data or Time Data.**

The purpose of automatically formatting a block of cells is to insure the data in the block is all formatted in exactly the same and precise and consistent way. Depending on what the data is assumed to be, several of the controls will be disabled. To enable a disabled control, first change the assumed data type.

There are many options for **General Strings**. You can choose to change the text case to one of **MAKE UPPER CASE** or **Make Camel Case**, or **make lower case**. You can also choose to remove specific single characters like the space character, or the \$ character, or the comma character, or you can enter a single special character and have the program remove all instances of that special character. You can also choose to **Clear Each Cell** so it contains empty data. You can either Left or Right fill a cell to the **Fill Length** using **The Special Character**. You can also format strings so they are left-justified, or centered, or right-justified within their cells.

If you select **Trim The Special Character** then the program will delete the special character from the left or right ends of the current cell, but it won't delete the character if it appears in the middle of the cell. This is the main difference between trimming the special character and removing the special character. Removing the special character means removing all instances of that character. Trimming only applies to the left and right ends of a cell.

As an example of using Block Formatting where we fill cells to a given length, imagine you have a column with some numbers that appears as shown below on the left. If you make **The Special Character** a 0 and if you set the **Fill Length** to 4 and choose **Left Fill Special Character Using Fill Length**, then the selected block would change to the column shown on the right.

43	0043
131	0131
25	0025
70	0070
103	0103
39	0039
126	0126
102	0102
9	0009
98	0098
20	0020
82	0082
193	0193
95	0095
55	0055
57	0057
66	0066
169	0169
199	0199
43	0043
21	0021

The above example would make even more sense if you needed to sort the column using the strings on the right. In this case you could choose to leave the data type for the column as strings, because they all have the same length that is 4 characters wide. In other words, the column on the right could be sorted using either strings or numbers. However, the column on the left could not be sorted using strings. If you wanted to sort the column on the left then in the sort function dialog you would have to select the type of source as numbers.

As another example, if you make the **Fill Length** 6 and keep **The Special Character** a 0 and choose to **Right Fill Special Character Using Fill Length** to the fill length then the before and after results should appear as shown next.

43	430000
131	131000
25	250000
70	700000
103	103000
39	390000
126	126000
102	102000
9	900000
98	980000
20	200000
82	820000
193	193000
95	950000
55	550000
57	570000
66	660000
169	169000
199	199000
43	430000
21	210000

The control that sets the **Fill Length** has two very different interpretations, depending on which of the last five choices is made for formatting general strings. In the above two examples we set this fill length to be either 4 or 6 and we applied character fills that used **The Special Character**. For those examples the **Fill Length** that was applied was a character length. In other words, it was a maximum value for the string length in terms of characters.

If you choose any of the last three string formatting or justification choices, then the **Fill Length** is what we call a **Pixel Length**. This is the number of screen pixels that measure the extent of the string in the grid cell. We show the three different kinds of justifications on the next page.

The big difference in these two uses of the **Fill Length** is the range of numbers used. When filling characters with **The Special Character**, the **Fill Length** will nominally be a number that is 10 or less. When justifying cell contents (either left or center or right), then the **Fill Length** represents the maximum width in pixels and then the nominal value will probably be at least 100 but could often be larger than 100.

Column 2
File Bytes:
467,039,849
1,042,606,629
23,518,417
770,610,051
7,168,361
75,532,041
506,374,511
5,312,614
24,907,165
267,995,520
1,559,175,575

Column 2
File Bytes:
467,039,849
1,042,606,629
23,518,417
770,610,051
7,168,361
75,532,041
506,374,511
5,312,614
24,907,165
267,995,520
1,559,175,575

Column 2
File Bytes:
467,039,849
1,042,606,629
23,518,417
770,610,051
7,168,361
75,532,041
506,374,511
5,312,614
24,907,165
267,995,520
1,559,175,575

The left column of numbers appear as normal strings that are left-justified. The middle column has the numbers centered with each other. The right column has been formatted using **Right-Justify Using Pixel Fill Length** where the **Fill Length** was set at 100. We would say the left example is left-justified, the center example is center-justified, and the right example has the numbers right-justified. Perhaps more often than not, you may want to make your numbers appear right-justified. When that is the case, you might first perform a string format where you trim all the blank characters from the left and right ends of the string. Second, choose the **Right-Justify Using Pixel Fill Length** where you set an appropriate value in screen pixels. A typical column like any of the three columns shown above might be 100 pixels wide.

The formatting for **Dates** is the next simplest kind of formatting to explain. **Dates** can be in many input formats, and we don't claim to cover all possible kinds of input date formats. In fact, we only output six different date formats. If the program cannot reasonably determine your input date, then it will not make any changes to the cell. The following table shows how some date input cells will get converted to the standard **MM/DD/YYYY** format. This table should give you some idea of the kinds of input cells that can be automatically converted as dates. The last two rows in the table below exhibit no change to the cell data because the format is not recognized.

Input Cell	Output Date
8/3/99	08/03/1999
5/03	05/01/2003
January 5, 2012	01/05/2012
48	01/01/1948
1902	01/01/1902
05-Jun-12	06/05/2012
9.15.2010	09/15/2010
5 January 2013	5 January 2013
2010.9.15	2010.9.15

We can briefly try to describe how we handle various kinds of date strings. After checking for one of our standard output date formats, we re-format any spelled out or abbreviated month names using numbers. We also convert any dashes or periods to slash characters. After this initial re-formatting we check that all remaining characters from the cell are either digits or slash characters.

If any non-digit or non-slash characters remain, then we ignore the cell. Otherwise, we count the number of slash characters. If there are only digits and no slash characters then we assume the cell only contains a year number and we convert the final output to show the equivalent of January 1 for that year. If a year only consists of two digits, and if those digits make a number between 0 and 29 then we add 2000 to that. But if the year number is between 30 and 99 then we add 1900 to that. If the year number turns out to have more than 4 digits then we ignore that cell because we don't try to do any interpretation in this case.

If there is only one slash character then we try to determine the numbers before and after that slash character, assuming the number before is a month number and assuming the number following is a year number. We then treat the year number as just described above.

When the cells are known to contain **Numbers**, then we can format those numbers in basically one of three ways by choosing a number formatting type as either **Decimal**, or **Integer**, or **Currency**. For both **Decimal** and **Currency** you can also further select two integers named the **Precision** and the **Decimal Digits**. These two integer values are ignored if the **Integer** format type is selected.

The **Precision** value basically tells how many total significant digits to limit the value (minimum is 2; default is 10; maximum is 18). The **Decimal Digits** value tells how many digits to write after the decimal point (minimum is 0; default is 5; maximum is 15) and the program will round the number to that many decimal place values. The program may automatically switch to scientific notation, depending on the given values.

When you select **Currency** the program will automatically set the **Precision** to 18 and it will set **Decimal Digits** to 2. You could later change these although the default values are probably what you want. Together the **Precision** and **Decimal Digits** values determine how the number is rounded.

If the number contained in a cell is represented by the string data 3.14159265359 then the output can be interpreted as shown in the following table. The program adds a \$ to **Currency** values and it will round values to the nearest penny and it will insert commas every three digits to the left of the decimal point. The following table contains examples of cell input and output related to how number data gets formatted.

Cell Input	Number Type	Precision	Decimal Digits	Cell Output
3.14159265359	Decimal	10	5	3.14159
3.14159265359	Decimal	10	3	3.142
3.14159265359	Decimal	10	8	3.14159265
12800.	Decimal	5	2	12800.00
12800.	Decimal	4	2	1.28E04
12800.	Decimal	3	2	1.28E04
12800.	Decimal	2	2	1.3E04
<hr/>				
3.14159265359	Integer	ignored	ignored	3
6.00	Integer	ignored	ignored	6
6.	Integer	ignored	ignored	6
<hr/>				
3.14159265359	Currency	18	2	\$3.14
314159265359	Currency	18	2	\$314,159,265,359.00
6.0	Currency	18	2	\$6.00
6.	Currency	18	2	\$6.00
6.005	Currency	18	2	\$6.01

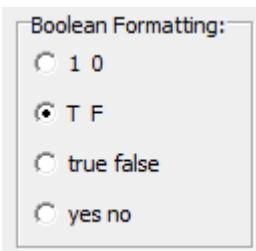
When formatting numbers, if any cell contains a string that cannot be converted to some kind of a real number, then you will see an error message for that cell and the formatting process will abort further processing of other cells. This handling for numbers differs from how dates are handled. Bad date cells are simply ignored while bad number cells will be flagged with an error and location message. Empty strings in cells will automatically be converted to zero number values without any warning or error messages being given.

If you edit any bad number cell, and then re-try the block formatting process, you should be able to get a good result for the selected block. The program will remove all \$ and comma characters and blank characters from strings before it tries to convert any cell to a number. If a valid number can be obtained, then that number is what gets re-formatted for the cell, using the **Number Type** and the **Precision** and the **Decimal Digits** you selected for the new output.

If you want commas inserted into **Integers** then you should check the checkbox with the caption **Insert Commas**. This checkbox only applies when you choose the number formatting to be applied to **Integers**. Commas are always automatically inserted into currency values, but usually commas are not inserted into other decimal floating point values.

In the above table you might note the decimal value 12800. shows different types of output, depending on the **Precision** and **Decimal Digits** values. Three of these examples have automatically switched to scientific notation and one of them has rounded the value to show 1.3E04.

The **Boolean Data** type is to represent values that are either true or false. You normally have only two choices for this data type. We allow four variations that can be the spelled out words **true**, **false** or **yes**, **no** or we allow the single letters **T** and **F** or the single digits **1** and **0**. When the program tries to automatically format your Boolean data, it only looks at the first letter in the existing string. On the basis of that letter it decides what the truth value should be and then it fills the cell based on the desired format that you choose. The only four format choices are:



If the existing string is empty, or if the program cannot determine the truth value from the 1st letter, then the program will display an error message for the first cell it finds that is in error and it will abort further processing.

There are three different **Time Data** formats that are used for formatting output. Times can be in a 12-hour format where the times end in either **AM** or **PM**. Times can also be in a 24-hour military format where there is no **AM** or **PM** indicator and the largest time can be 23:59:59. These two formats are assumed to represent specific times of the day.

A third and different kind of time represents what is called an elapsed time. This type of time is somewhat unusual and is less commonly used. An example for this kind of time could represent the time it takes for a train to run from San Francisco to Boston. If we assume such a time could be as long as $2\frac{1}{2}$ days, then we would want to allow for the number of hours to exceed the 24 hour limit of a normal day.

All times only measure accurately to the nearest second. Also, seconds that are 60 or more are converted to additional minutes, and minutes that are 60 or more are converted to additional hours. If you choose a time format other than an elapsed time, then hours that are 24 or above are essentially ignored.

We will use the following grid to test some individual time formats. This grid has both valid and invalid formats. The program not only has to make a proper output format, it also has to read all possible time inputs and reject those that are invalid. In formatting times, the program always stops with the first cell in the selected block that cannot be converted into a valid time. There can be many reasons why a string is an invalid time. The program will always provide a corresponding error message.

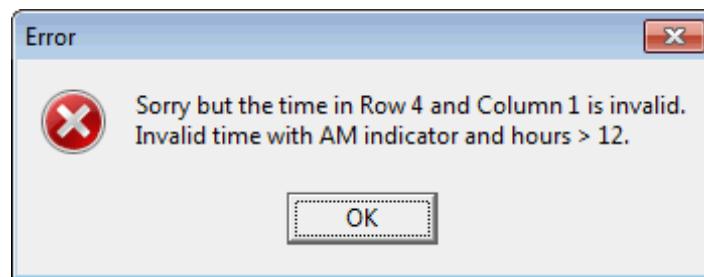
<	Column 1	Column 2	Column 3
>	AM/PM Times:	Elapsed Times:	24-Hour Times
1	02:03:04 AM	0	23:59:59
2	02:05:06 PM	59:159:180	24:60:60
3	14:00:00	124:40:38	0:0:0
4	14:00:00 AM	-5	testing
5	14:00:00 PM	01:02:03x	
6	11:00:00 PM		
7	12345		

To begin to understand how we handle times, lets start with the times in the first three rows of Column 1. When we try to read these and format them for the 12-hour output format we generate the following results. The left figure is the input while the right figure is the output.

02:03:04 AM	02:03:04 AM
02:05:06 PM	02:05:06 PM
14:00:00	02:00:00 PM

Here we can see that essentially no changes were made in the first two rows, but the third row changed 14:00:00 to 02:00:00 PM.

If you try to format 14:00:00 AM into any type of time output you will simply generate an error message like the following:



On the other hand, 14:00:00 PM will convert to 02:00:00 PM in the 12-hour format or it will come back as 14:00:00 for either a 24-hour format or an elapsed time format. Thus in some cases an AM or PM indicator will be tolerated on input, even when there should not be any AM or PM indicator in the output.

For a 12-hour output, the input of 12345 will get converted to 09:00:00 AM. The reason this works, is that all hours above 24 hours are first converted to hours values with less than 24 hours. In other words, the value 12345 is first seen as exactly that many hours, then multiples of 24 are thrown away until we get down to the value of 33. Then one more 24 value is thrown away and this is why we end up with 09:00:00 AM as a result.

When 12345 is converted to the 24-hour military format we also get back 09:00:00 as the new formatted time. But when 12345 is converted to elapsed time, then it comes back as 12345:00:00 because elapsed times can have hours greater than either 12 or 24.

The next two figures show how three values get converted to the 12-hour format.

0	12:00:00 AM
59:159:180	01:42:00 PM
124:40:38	04:40:38 AM

The second row above is first converted to 59:162:00 because 180 seconds counts as 3 additional minutes. Next the 162 minutes gets converted to 2 hours and 42 minutes. So we then have 61:42:00. Finally, the hours are reduced by multiples of 24 until we have 13 hours left. So 13 hours and 42 minutes is finally converted to 01:42:00 PM. The third row is simply reduced by 24 hours until we get down to 4 hours. So 04:40:38 AM is the final result.

When these same values are converted to 24-hour format we see the before and after results as:

0	00:00:00
59:159:180	13:42:00
124:40:38	04:40:38

Next we show the before and after results when formatting what should be 24-hour values into 12-hour values.

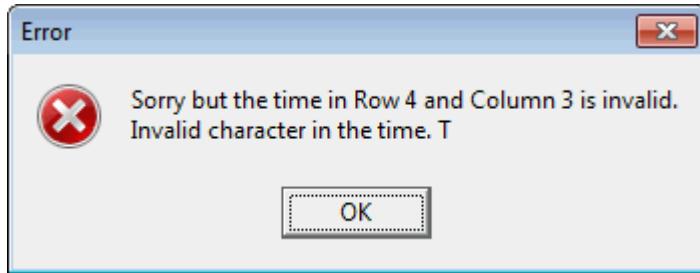
23:59:59	11:59:59 PM
24:60:60	01:01:00 AM
0:0	12:00:00 AM

The second row above can be understood by considering that 60 seconds converts to an additional minute and then 61 minutes gets converted to an additional hour with 1 minute left over, and finally 24 hours gets bumped up to 25 hours before it gets reduced back down to 1 hour. Thus 24:60:60 converts to 01:01:00 AM.

Our final idea about formatting times is that we ignore blank cells. In other words, blank cells are tolerated and simply remain unchanged as blank cells.

Blanks are removed from times when times get formatted. All times must only contain valid characters.

If you see an error message like:



it really means either the character t or T was found to be invalid. Characters are converted to upper-case so AM and PM will not be formatted as am or pm. An invalid character might be any lower-case letter even though the error message will only show upper-case letters.

We will finish this help topic on formatting by making a couple of comments about the looks of grids. First, this program is not intended to be used for making data look pretty. Instead, this program is intended to make data look consistent. The three string formatting functions that can perform left and center and right justifications can help some grids look more attractive. This effect is mostly cosmetic and is not a requirement for any data with which we are familiar. Pure data that is properly formatted normally doesn't require any further cosmetic adornment!

In practice you might never find it necessary to actually perform the left-justification function because by default all strings in cells are automatically displayed with left justification. One reason for not using left-justification is that for the most part this function just wastes space. About the only time it might be desirable to force left-justification is when you want to have some control over the width of a column and you don't want to manually set the column grid width. You can try to let the trailing spaces determine the width of the column and you can use the automatic adjustment function determine a column width. This assumes the trailing blank spaces can be considered benign to your data.

There are times when it is nice to be able to right-justify numbers, whether they be ordinary decimal values, but especially when they are currency values and when you might be taking a screen shot of a grid. When currency or other number values are right-justified then all the decimal points should appear to be lined up vertically. However, just before importing data into a spreadsheet or a database table, you may wish to reformat numbers as regular values without any leading spaces. When printing grids you can easily make numeric or other values have right justification without having to first insert leading spaces by using the formatting right-justification function. This is especially true when you are printing a grid to make a **PDF** file. This same comment applies when it is desirable to have all the data values centered in a column.

If you are using any of the three justification choices for string data, and if you edit any entries in the block, then we suggest you re-select your block and trim the space character from all the cells and finish by re-formatting the block. This manual re-formatting is required because we don't do automatic block formatting that would require maintaining extra meta-data along with a grid.

Applying Math Expressions

The **CSV Editor** program has limited math capabilities, but it can apply a wide variety of mathematical functions to individual cells. In fact, you can write any general math formula or expression, and that expression can then be applied individually to each cell in a block of cells that you select. Your math expression or formula can include any of the functions normally found on a scientific calculator. We will deliberately try to keep our examples simple.

Applying a math expression to a block of cells first requires that every cell in the selected block contain only numeric data. As an example, consider the block in the next grid shown below. We have five rows and six columns and the data contains the numbers from 1 to 30, in order, going across the rows first.

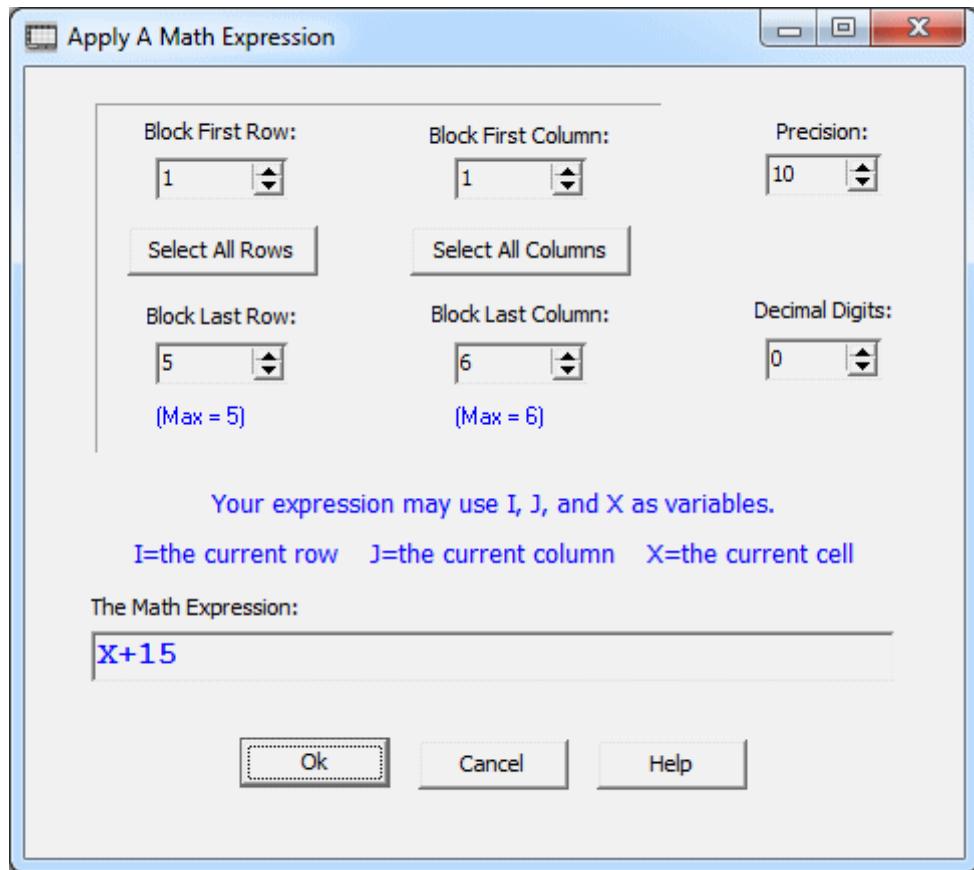
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18
4	19	20	21	22	23	24
5	25	26	27	28	29	30

Now imagine that we want to add the number 15 to every cell in this block. To do this we select the menu item **Blocks | Block Math Expression...** and when the dialog box comes up we select the above block and we type in our math expression as:

x+15

We also set the Decimal Digits to 0 because we want to format the results using whole numbers only. In other situations you can format the results using any precision and any number of decimal places as you require.

The dialog should appear as shown on the next page.

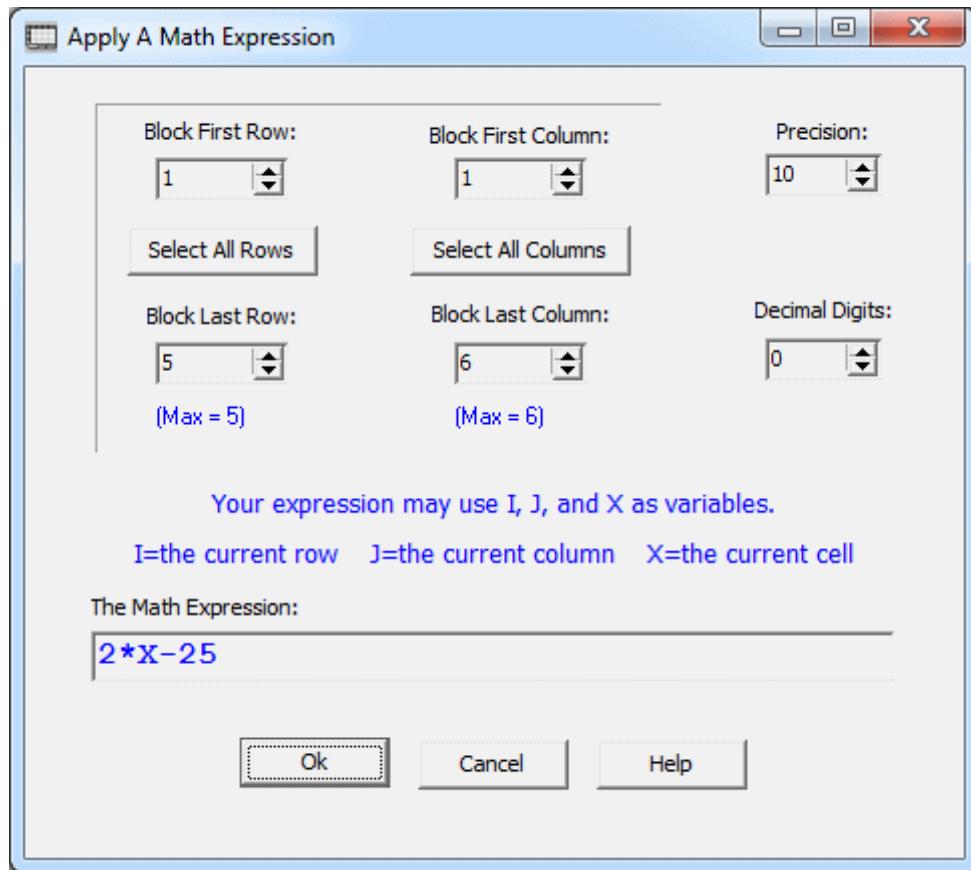


In the above dialog, you type in your math expression using the edit box for **The Math Expression**. In this example, we typed **x+15** for the expression. When we click the **Ok** button, the new grid should appear as:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	16	17	18	19	20	21
2	22	23	24	25	26	27
3	28	29	30	31	32	33
4	34	35	36	37	38	39
5	40	41	42	43	44	45

Note how the number 15 has been added to every cell in the grid. In this example, as in all examples, you can use **X** as a variable in your formula. What **X** really represents is each cell in the block because the program will visit each cell in the block. When a cell is visited, first the value of **X** is initialized with the existing number in the cell, then the expression formula gets evaluated using the current value of **X**, and the numerical result from the formula evaluation gets stored back in the same cell. Then the program will visit the next cell in the block and repeat the steps just described. The block cells are visited one at a time until the entire block is processed.

As a second example, if we keep this last grid but change to a new formula, like **2*x-25**, then we would first bring up the dialog and make the settings as shown below.



Now after we click **Ok** we see the new grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	7	9	11	13	15	17
2	19	21	23	25	27	29
3	31	33	35	37	39	41
4	43	45	47	49	51	53
5	55	57	59	61	63	65

If you look at the next to last grid in the first row and first column you could calculate that $2*16-25=7$ and **7** is the new value in the immediately above grid. If you look at the next to last grid in the last row and the last column you could calculate that $2*45-25=90-25=65$ and **65** is the new value in the immediately above answer grid, in its last row and last column.

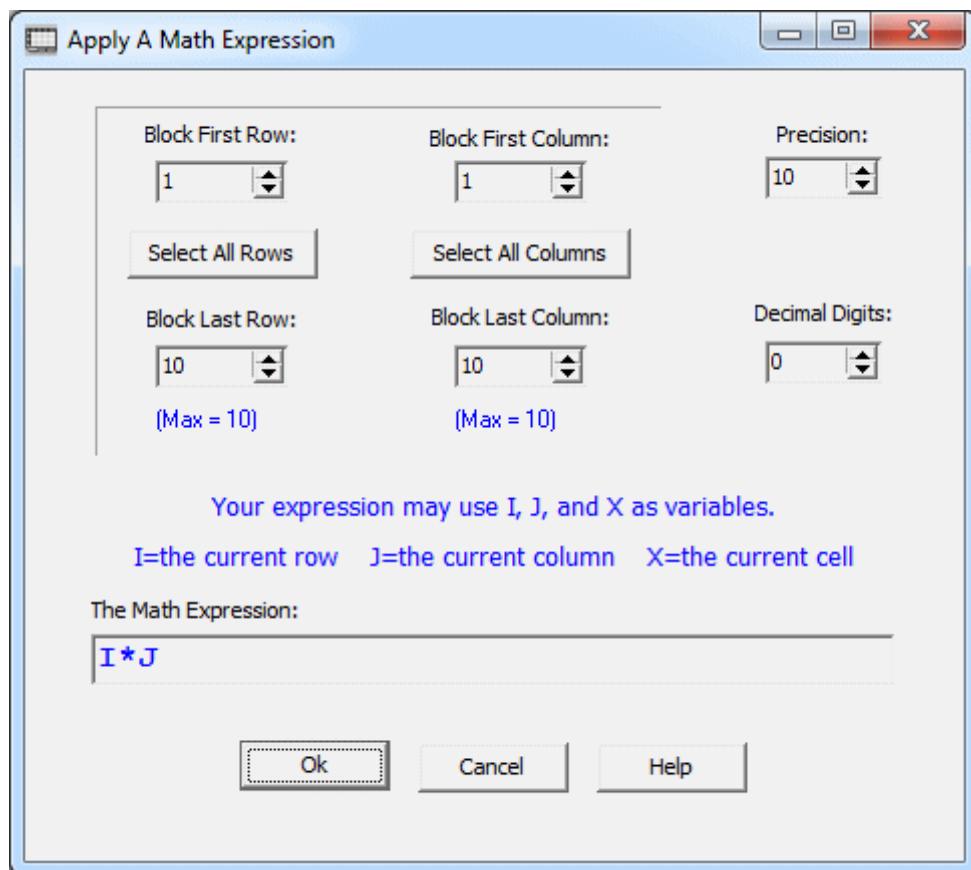
Usually any formula you enter will probably somehow involve using the variable **X** as we have done in our first two examples. However, you don't have to use **X** in your formula. If you enter a constant number as your formula expression, then the program would evaluate that expression as it visits each cell and it would enter the constant number in the cell. In other words, your formula doesn't always have to use **X**, and if it does not use **X**, the expression will still be evaluated anew as each cell in the block is visited.

Now we want to keep things simple, but you should know there are two other variables besides **X** you can use in your formula. Those variables are **I** and **J** where **I** refers to the current row and **J** refers to the current column. Just keep in mind that like **X**, the variables **I** and **J** vary each time you visit another block cell. **I** always represents the row number and **J** always represents the column number of the cell currently being visited.

As our last example, we have created a grid with 10 rows and 10 columns in which every entry is the number 0.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
1 >	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

Next we select the entire grid and we enter the math expression as **I*J**.



We still keep the **Decimal Digits** value at 0 because we want to display whole numbers. After we click the **Ok** button we see the result.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
1 >	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

The above table should remind you of a standard multiplication table. Note the number in the cell in row **I** and column **J** is the product **I*J**.

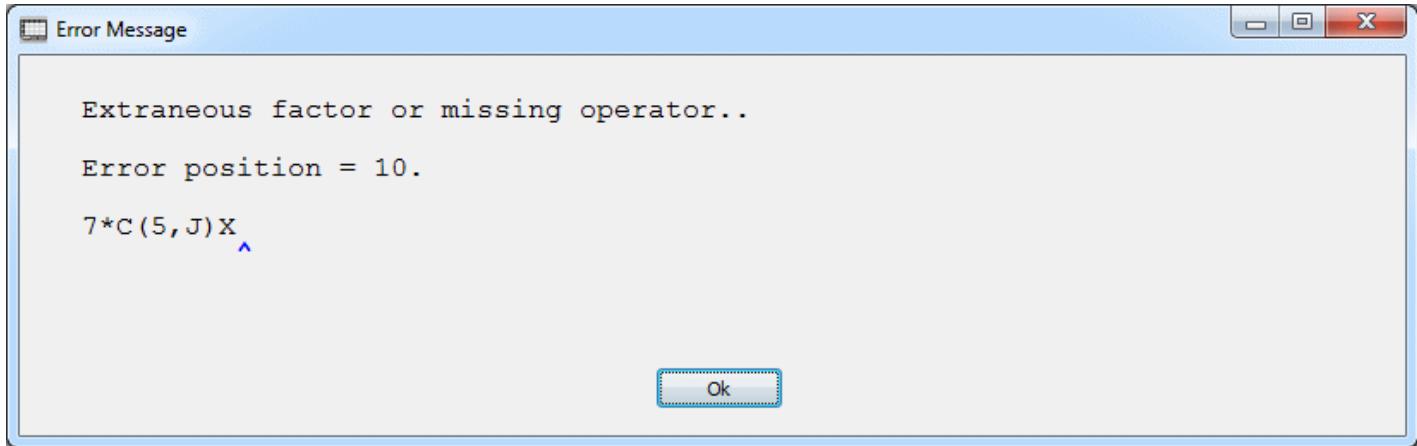
Expressions like **C(I,5)** or **C(4,J)** can be used to reference individual grid cells, where the subscripts are a mixture of constants and the special variables **I** and **J**. You can also reference a constant cell by writing **C(4,6)** as part of an expression. This is known as a constant cell reference because both **4** and **6** are constants. Such constants can be outside the range of the selected block as long as they are within range for the entire grid. In other words, a constant cell reference like **C(24,37)** would not make any sense if the current grid had less than **24** rows or less than **37** columns. Using such a reference anywhere within an expression would generate an error because it would refer to a non-existent cell. In a math expression, all cell references must be for cells that contain numeric strings, not general strings. The syntax format for a cell reference is to write **C(row,column)** where **row** and **column** are themselves expressions that evaluate to numbers within proper ranges for the current grid. In particular, you should note that as a block gets scanned, **X=C(I,J)**.

Next, we give an example that tells how to add a multiple of one row to another row in a grid. If you choose your block to be all the columns of row **8**, and if you used an expression like, **7*C(5,J)+X**, the effect would be to multiply the **5th** row by **7** and add that to row **8**. All entries in rows **5** and **8** must contain strings that can be converted to numbers. None of these cells can contain a non-numeric string.

You never know when you might use a math expression. Suppose you had a grid with a column of numbers that represented pages in a book, where the pages were part of an index for the book and where the adjacent column contained the corresponding index word or phrase. Assume the book was over 500 pages long. After the book was edited, suppose all the page numbers after page 11 were increased by 3 because there were 3 additional pages inserted into the book between the original page numbers of 11 and 12. You could select the appropriate block and enter and apply the math expression **X+3**. This action would automatically re-index the entire book, but more important, it would save you a lot of boring manual editing, not to mention a whole lot of time!

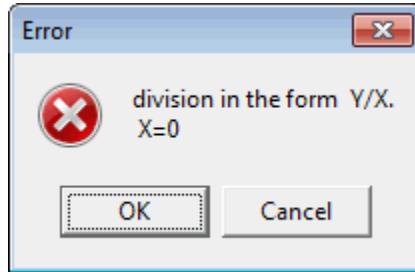
There are a few more things you should know about entering expressions. First, the edit box will automatically convert everything you type into UPPER CASE letters. You can't enter anything in lower case. Second, if you make a mistake entering an expression, (i.e. you enter an invalid syntax or an illegal built-in function name) the program will stop and show a message and it will try to explain what went wrong.

An example syntax error message might appear like the following:



In this case there is a missing operator that should be in front of the **X** symbol. We should emphasize that the program will try to point out the position in your math expression where it encountered an error, but the true error position may be other than where the **^** symbol points. In this example the **^** symbol points past the last character in the expression. Internally the program found **7*C(5,J)** to be Ok and it found the **X** symbol Ok, but it could not determine how to combine these two subexpressions. If the invalid expression is corrected by inserting a **+** symbol in front of **X**, so the expression becomes **7*C(5,J)+X**, then the program would parse it correctly. You should always be able to recover from any kind of a parsing error.

A different kind of error occurs when the program is executing or evaluating any math function expression and it attempts to do something mathematically illegal, like dividing by zero. This could be considered as a run-time error. You will always see an error message that details what is mathematically illegal. As an example:



Later in this help file we show how to convert between two different temperature scales using a menu item under the **Conversions** menu. The following expressions can be used to accomplish temperature conversions on a block that contains temperature values. In case you wish to try temperature conversions using the **Math Expression** method, the following two formulas could be used.

(X-32)*5/9 ← use this formula to convert a block of cells from Fahrenheit to Celsius

(X*9/5)+32 ← use this formula to convert a block of cells from Celsius to Fahrenheit

For the first formula you should select a block that contains Fahrenheit temperatures. You may wish to make a copy of an existing block first and operate on that copy. Then apply the first formula to that block.

For the second formula you should select a block that contains Celsius temperatures. You may wish to make a copy of an existing block first and operate on that copy. Then apply the second formula to that block.

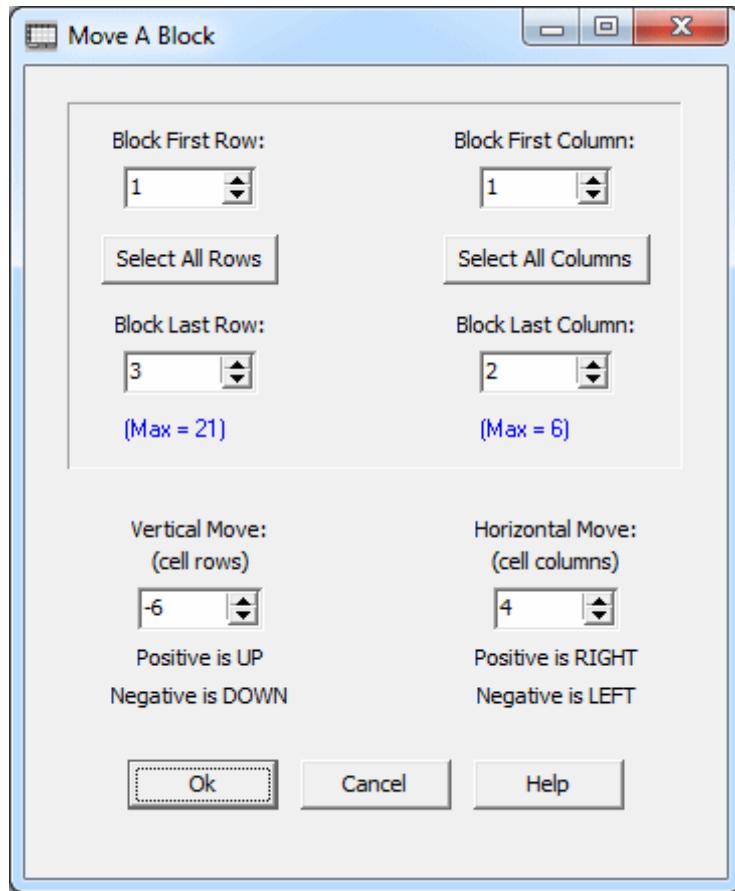
We conclude this discussion of applying math expressions by telling you that there is a lot more to learn. Just choose **Help | Math Expressions** to learn more technical details related to this topic.

Not related to applying math expressions is the ability of the program to perform Linear Regression. Just select the menu item **Utilities | Linear Regression Report...** and then click the **Help** button in the dialog that appears to read about this other mathematical functionality that can automatically analyze data and find an equation for the mathematical curve that best fits that data.

If your need is for performing financial calculations, then we recommend you try **Utilities | Create a Financial Schedule....** This menu item allows you to easily create four different basic kinds of tables related to loans and savings and spending and managing accounts that earn or pay out compound interest.

Moving a Block

If you select the **Blocks** menu item **Block Move...** you will see the following dialog.



As with all block functions, the upper controls in this dialog allow you to precisely define a rectangular block of cells. The next two controls down allow you to set **Vertical** and **Horizontal Move** values. Such values are in terms of cell counts. For the **Vertical Move**, a positive value will move the selected block up while a negative value will move the same block down. A zero value causes no change in the vertical position of the block. Similarly, a **Horizontal Move** value that is negative will cause the block to move left while a positive value will move the block right. A zero **Horizontal Move** value will not change the block's horizontal position. You would use a zero horizontal value when you want to move a block vertically up or down within the same set of columns that the block already occupies.

There is a significant difference between doing a block copy and paste operation and doing a block move operation using the above dialog. A block move will always leave blank cells behind where any portion of the block makes a vacated cell. As an example of a block move, consider the next two figures shown on the next page. The first figure contains some Academy Awards information.

If we set the controls as shown in the above dialog box, which means the selected block will be moved down 6 cell rows and moved across 4 cell columns to the right, then after we click the **Ok** button we will see the block change as shown in the second figure on the next page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1			Elizabeth Taylor	Billy Wilder	The Apartment	T
2			Sophia Loren	Robert Wise	West Side Story	T
3			Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	1960	Burt Lancaster
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	1961	Maxmillian Schell
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	1962	Gregory Peck
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

Note how the block has moved and the move has left six empty cells that the original position of the block occupied before the block was moved. You should note that the block remains selected, but it is in its new position.

If we were to perform a further move up +5 vertical cell rows (make the **Horizontal Move** value 0) then the above figure would further change to the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1			Elizabeth Taylor	Billy Wilder	The Apartment	T
2			Sophia Loren	Robert Wise	1960	Burt Lancaster
3			Anne Bancroft	David Lean	1961	Maxmillian Schell
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	1962	Gregory Peck
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann		
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols		
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed		
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

You can never move a block further in any direction than what the size of the current grid will allow. You may notice that as you change values in the two spin edit controls the program will automatically switch a value to 0 so as to not allow the any part of the block to move outside the current grid boundaries.

The real need for a block move function can be seen by inspecting the left grid on the next page. In that grid the second column is to contain a list of television stations in the Los Angeles area. The only problem is that the listed **Channel Descriptions** are off by exactly three rows in terms of their vertical alignment row positions. We would like to move the **Channel Descriptions** in **Column 2** up three rows, but at the same time we don't want the **Channel Number** information in **Column 1** to move or change. So we can't simply delete the three extra cells at the top of **Column 2**.

Instead, we select the block as shown in the middle figure on the next page and then we move only the selected entries in **Column 2** up three cell rows, but within that same **Column 2**. The final result is shown in the third figure on the right. The result is that the **Channel Number** information and the **Channel Description** information now line up correctly row by row.

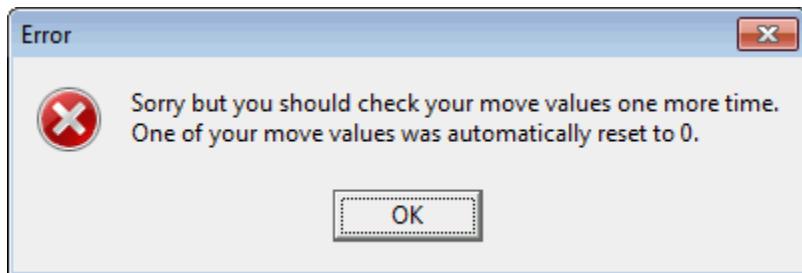
<	Column 1	Column 2
>	Channel Number:	Channel Description:
1	2	
2	4	
3	5	
4	7	KCBS (CBS)
5	9	KNBC (NBC)
6	11	KTCA
7	13	KABC
8	18	KCAL
9	20	KTTV
10	22	KCOP
11	24	KSCI
12	28	KVME
13	30	KWHY
14	31	KVCR
15	34	KCET
16	44	KPKN
17	46	KVMD
18	50	KMEX
19	52	KXLA
20	54	KFTR
21	56	KOCE
22	57	KVEA
23	58	KAZA
24		KDOC
25		KJLA
26		KLCS

<	Column 1	Column 2
>	Channel Number:	Channel Description:
1	2	
2	4	
3	5	
4	7	KCBS (CBS)
5	9	KNBC (NBC)
6	11	KTCA
7	13	KABC
8	18	KCAL
9	20	KTTV
10	22	KCOP
11	24	KSCI
12	28	KVME
13	30	KWHY
14	31	KVCR
15	34	KCET
16	44	KPKN
17	46	KVMD
18	50	KMEX
19	52	KXLA
20	54	KFTR
21	56	KOCE
22	57	KVEA
23	58	KAZA
24		KDOC
25		KJLA
26		KLCS

<	Column 1	Column 2
>	Channel Number:	Channel Description:
1	2	KCBS (CBS)
2	4	KNBC (NBC)
3	5	KTCA
4	7	KABC
5	9	KCAL
6	11	KTTV
7	13	KCOP
8	18	KSCI
9	20	KVME
10	22	KWHY
11	24	KVCR
12	28	KCET
13	30	KPKN
14	31	KVMD
15	34	KMEX
16	44	KXLA
17	46	KFTR
18	50	KOCE
19	52	KVEA
20	54	KAZA
21	56	KDOC
22	57	KJLA
23	58	KLCS
24		
25		
26		

If you tried to do this using the block copy and paste operations you would have to do a little extra editing in order to remove a few rows that would remain behind and doubly exposed. In general, Block Moves are slightly more efficient than using copy and paste operations.

When using the above dialog, depending on the order you make changes and the values you enter, when you try to close the dialog you may see a message like the following. Although rare, this message can appear even when you click the **Cancel** button to close the dialog.



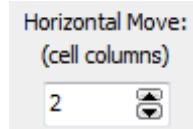
If you see this message, you can continue editing and insure the block you set is properly defined and make sure the two move values are appropriate before you try to close the dialog. Setting both move values to 0 means no move is really defined, but that is ok, and in that case the dialog will close without generating a potential error message.

One of the more common uses of the block move function occurs in maintaining a book index as shown on the next page.

Below we show an index for a document.

116	field names	288	383				
117	file count	57	69				
118	file information	57	69-	71	307-	308	
119	File menu	14					
120	filename filter	69-	71				
121	filenames	57	69-	71	307-	308	

If you look at row 118, suppose you need to insert two page numbers after page 57 and before page 69-. In this case you should select the block in row 118 that contains the page numbers 69- to 308. Then select **Blocks | Block Move...**. When the dialog appears enter the number positive 2 into the Horizontal Move position.



116	field names	288	383					
117	file count	57	69					
118	file information	57	69-	71	307-	308		
119	File menu	14						
120	filename filter	69-	71					
121	filenames	57	69-	71	307-	308		

Then when you execute this function you will move the selected block 2 cells horizontally to the right.

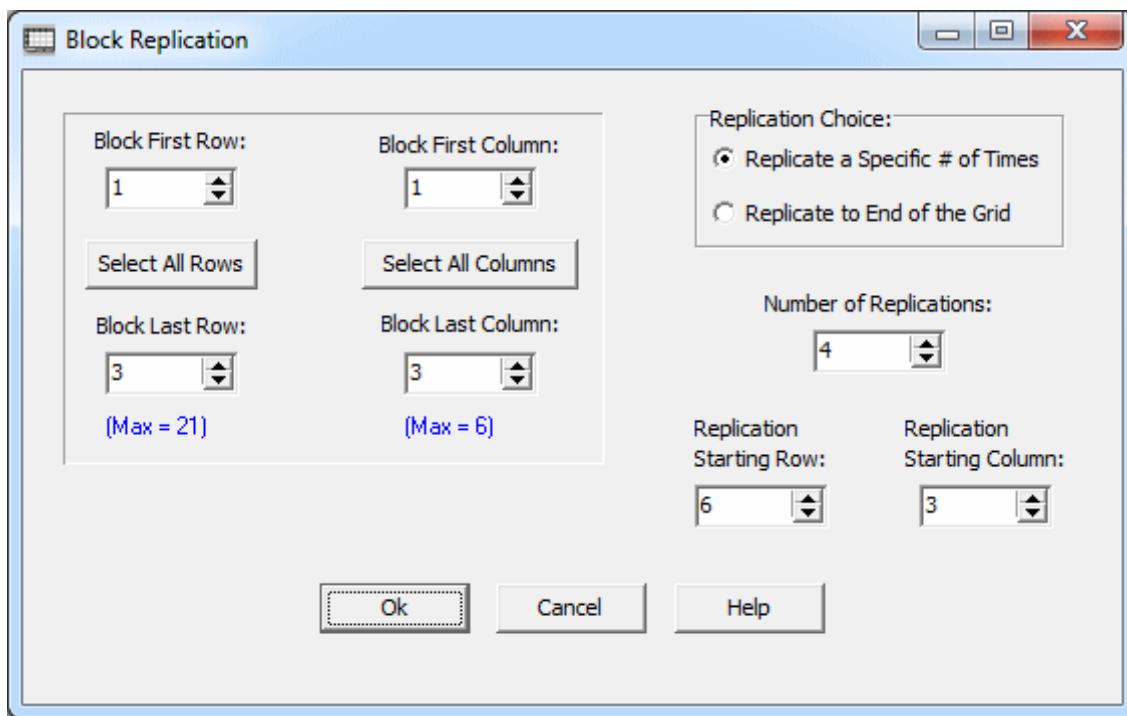
116	field names	288	383					
117	file count	57	69					
118	file information	57			69-	71	307-	308
119	File menu	14						
120	filename filter	69-	71					
121	filenames	57	69-	71	307-	308		

Now you have 2 open cells into which you can type the new page numbers.

Although the above example is simple because we moved only 4 cells, it illustrates an essential function that we use all the time. This simple function saved us from having to re-type the entire row just to put in 2 page numbers.

Replicating a Block

The **CSV Editor** program has a special function for automatically replicating any rectangular block a certain number of times. When we use the word *replicate* it means to copy a block, but in an automatic and repeated manner. If you first select a rectangular block of cells in an upper area of a grid, you can then select **Blocks | Block Replicate...** and you will see the following dialog that is used to setup either of two replication choices. Since the replications take place in a downward fashion, the block you select must always be in cells above where you want the block copies to be made.



After defining an appropriate block using the six controls in the upper-left of the dialog, the only choice you need to make is whether you want to specify a particular number of replications, or whether you want the replications to continue automatically until the bottom row of the current grid is reached. You only need to enter the **Number of Replications** when you choose the first radio button. Otherwise, the **Number of Replications** control will be disabled when you choose to **Replicate to End of the Grid**. The only other thing you need to specify is the position of the upper-left corner of where you want the replications to start. You do this by entering the **Replication Starting Row** and the **Replication Starting Column**. These two values are not to be confused with the **Block First Row** and the **Block First Column** which are used to define the upper-left corner of the selected block. The **Replication Starting Row** must always be larger than the **Block Last Row** because the replications must start below the selected block.

As a first example, the next figure shows the Academy Awards grid in which we have blanked out all the cells except for the block of cells in the very upper left corner. The selected block consists of three rows and three columns.

We are going to execute the Block Replication function using the values that are set in the above dialog box. This means we are going to repeat the selected block a total of 4 times, beginning with the upper-left corner at row 6 and column 3. The two figures on the next page show the before and after results.

The original block selection before it is replicated.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor			
2	1961	Maxmillian Schell	Sophia Loren			
3	1962	Gregory Peck	Anne Bancroft			
4						
5						
6						
7						
8						
9						
10						

The replicated data starts in row 6 and occupies columns 3, 4, and 5, and finishes in row 17.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor			
2	1961	Maxmillian Schell	Sophia Loren			
3	1962	Gregory Peck	Anne Bancroft			
4						
5						
6		1960	Burt Lancaster	Elizabeth Taylor		
7		1961	Maxmillian Schell	Sophia Loren		
8		1962	Gregory Peck	Anne Bancroft		
9		1960	Burt Lancaster	Elizabeth Taylor		
10		1961	Maxmillian Schell	Sophia Loren		
11		1962	Gregory Peck	Anne Bancroft		
12		1960	Burt Lancaster	Elizabeth Taylor		
13		1961	Maxmillian Schell	Sophia Loren		
14		1962	Gregory Peck	Anne Bancroft		
15		1960	Burt Lancaster	Elizabeth Taylor		
16		1961	Maxmillian Schell	Sophia Loren		
17		1962	Gregory Peck	Anne Bancroft		
18						
19						
20						
21						

Looking at the last figure we can see that indeed the original block has been replicated 4 times, starting in row 6 and column 3. Each time the block is replicated, it is copied starting with the next row after the last used row from the previous copy. Thus all block copies are vertically adjacent to each other, going down the grid.

The next figure is where we go back and start with the first figure on the previous page. Instead of replicating the block a set number of times, we simply let the program continue making block copies going down the grid until we run out of rows at the bottom of the grid. In the next example we also chose the replication starting upper-left corner to be at row 5 and column 1.

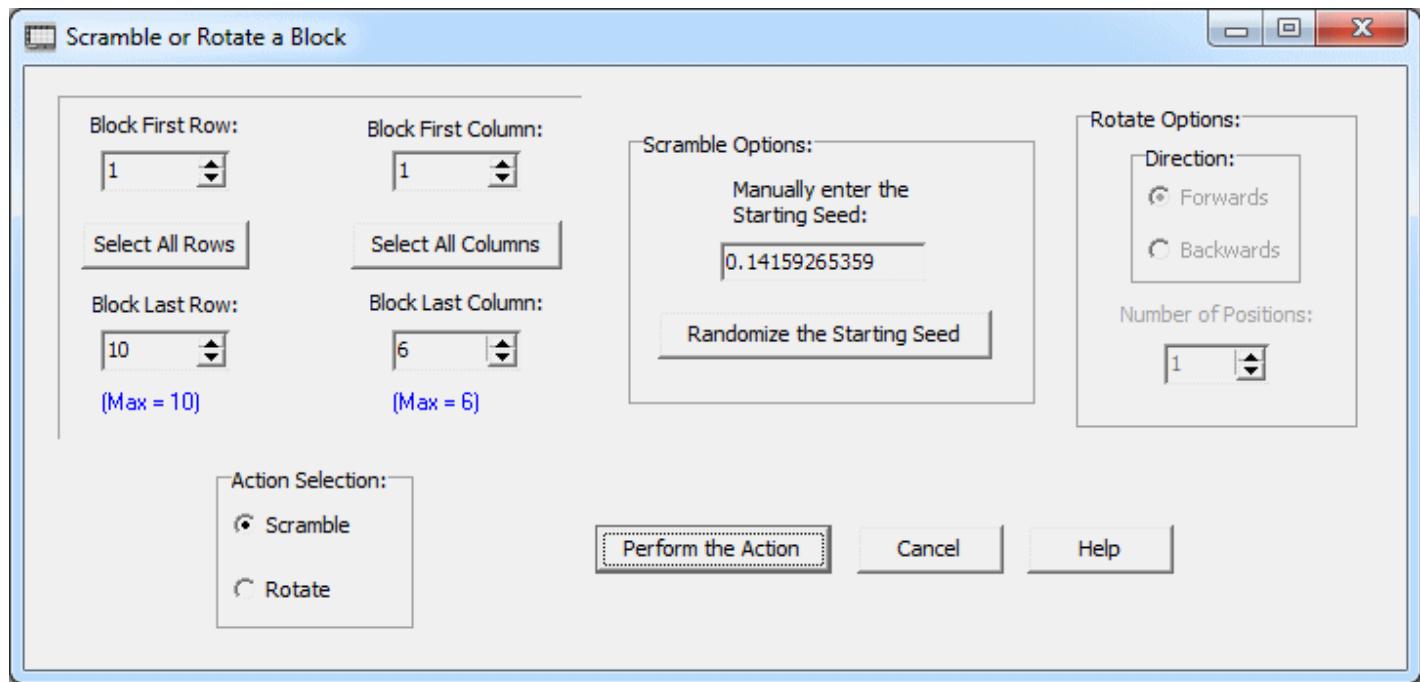
	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
<	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor			
2	1961	Maxmillian Schell	Sophia Loren			
3	1962	Gregory Peck	Anne Bancroft			
4						
5	1960	Burt Lancaster	Elizabeth Taylor			
6	1961	Maxmillian Schell	Sophia Loren			
7	1962	Gregory Peck	Anne Bancroft			
8	1960	Burt Lancaster	Elizabeth Taylor			
9	1961	Maxmillian Schell	Sophia Loren			
10	1962	Gregory Peck	Anne Bancroft			
11	1960	Burt Lancaster	Elizabeth Taylor			
12	1961	Maxmillian Schell	Sophia Loren			
13	1962	Gregory Peck	Anne Bancroft			
14	1960	Burt Lancaster	Elizabeth Taylor			
15	1961	Maxmillian Schell	Sophia Loren			
16	1962	Gregory Peck	Anne Bancroft			
17	1960	Burt Lancaster	Elizabeth Taylor			
18	1961	Maxmillian Schell	Sophia Loren			
19	1962	Gregory Peck	Anne Bancroft			
20	1960	Burt Lancaster	Elizabeth Taylor			
21	1961	Maxmillian Schell	Sophia Loren			

In the above example the final block part ended with row 21. So the original block was copied $5\frac{2}{3}$ times. The final row in the original block with the two names Gregory Peck and Anne Brancroft got cut off at the bottom of the grid because we ran out of grid rows. This is normal and should be expected.

Whenever the program tries to replicate the original selected block, it will only copy as many cells as will fit in the remaining grid space. So if you run out of rows at the bottom, or if you run out of columns on the right, the program will not expand the grid beyond its current boundaries. This means some block parts may get cut off. If you always allow a sufficient number of columns, then only the last rows of the block will get cut off at the bottom of the grid, as in the above example. Using Block Replication, you could select a single row and replicate that row 5,000 times, if the existing grid had a sufficient number of rows.

Scrambling or Rotating a Block

The **CSV Editor** program has a couple of special functions whose purpose is to reorder the elements in any selected block. The first function or action is called scrambling. You could just think that scrambling works by first removing all the elements from the block and then it puts them back in, one at a time, but in a completely random order. To apply this function select the menu item **Blocks | Block Scramble/Rotate...** and you will see the following dialog.



The controls in the upper-left are standard for selecting a block of cells. When scrambling, you also want to make sure the **Action Selection** has the **Scramble** radio button selected. As with any block function, we generally recommend you select the block before you open any dialog function that will operate on a block.

For our first example, let's start with the following grid in which the integers from 1 to 60, in order, fill the grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	1	11	21	31	41	51
2	2	12	22	32	42	52
3	3	13	23	33	43	53
4	4	14	24	34	44	54
5	5	15	25	35	45	55
6	6	16	26	36	46	56
7	7	17	27	37	47	57
8	8	18	28	38	48	58
9	9	19	29	39	49	59
10	10	20	30	40	50	60

If we applied the block scramble function to the entire grid, then the next figure could show the result of scrambling all these cells.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	37	39	59	6	19	58
2	29	45	50	17	26	38
3	33	44	34	3	54	35
4	7	48	30	23	13	43
5	28	49	40	53	12	18
6	31	46	42	20	16	10
7	4	32	8	1	9	60
8	27	55	5	52	14	51
9	47	41	2	11	15	21
10	56	36	24	25	22	57

Note how the integers from 1 through 60 now appear in a random order within the grid.

As a second example, we show two lists below. The list on the left is sorted in alphabetical order using titles of movies. When this list is scrambled it might look like the list on the right where the movie titles appear in a random order. Both lists contain exactly the same elements, they are just in two different orders, with the list on the right being completely random.

Column 5
Best Picture
A Man For All Seasons
Annie Hall
In The Heat Of The Night
Kramer vs Kramer
Lawrence of Arabia
Midnight Cowboy
My Fair Lady
Oliver
One Flew Over The Cuckoos Nest
Ordinary People
Patton
Rocky
The Apartment
The Deer Hunter
The French Connection
The Godfather
The Godfather Part II
The Sound of Music
The Sting
Tom Jones
West Side Story

Column 5
Best Picture
Oliver
Patton
The Godfather
The Sound of Music
Tom Jones
Rocky
Ordinary People
A Man For All Seasons
The Sting
My Fair Lady
Annie Hall
Midnight Cowboy
West Side Story
The French Connection
The Deer Hunter
In The Heat Of The Night
The Godfather Part II
Kramer vs Kramer
One Flew Over The Cuckoos Nest
Lawrence of Arabia
The Apartment

We should emphasize that when you scramble the cells in a block, only that block changes. In general, this means no entire rows or columns are changed, except for the selected block cells. This action is somewhat like sorting in place, except scrambling is contrary to sorting.

When scrambling, the random number generator is controlled by the starting seed in the above dialog. You can manually enter any seed as long as it is in the range between 0 and 1. You can also press the button with the caption **Randomize the Starting Seed** to cause the program to generate a random seed. If you intend to exactly repeat a random experiment, then you can always do so by using the same starting seed between two different invocations of this function. The random number seed in this dialog is distinct from the seeds used by other random functions in the program.

Because it doesn't make much sense to scramble or rotate a block that has only one or two cells, we require that your block have a minimum of three cells. If your selected block doesn't satisfy this requirement then you will generate the following error message:

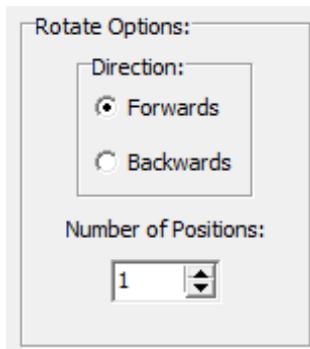


The main choice you will make in the **Scramble or Rotate a Block** setup dialog is the **Action Selection**. Depending on the choice you make, either the **Scramble Options** group of controls or the **Rotate Options** group of controls will be enabled while the other of these two groups will be disabled.

If you need to scramble the rows in a grid to make a random ordering of the rows, you should select the menu item **Rows | Scramble a Range of Rows....**

Rotating

Let's now assume you want to rotate a block. Then after selecting **Rotate** in **Action Selection**, you will see the following couple of controls become active.



With these controls you select the direction of rotation and you select the number of positions or cells that you want to rotate. The simplest blocks are those that consist of only one row or one column of cells. We will discuss examples for these simple kinds of blocks before we move onto more general rectangular blocks.

Below we show an original block column that has 10 rows and that contains the numbers from 1 to 10, in order, from top to bottom. Then we show the result of that same block with two different examples of rotation.

<	Column 1
1 >	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

The numbers 1-10 in order.

<	Column 1
1 >	8
2	9
3	10
4	1
5	2
6	3
7	4
8	5
9	6
10	7

Rotate 3 Positions Forwards.

<	Column 1
1 >	5
2	6
3	7
4	8
5	9
6	10
7	1
8	2
9	3
10	4

Rotate 4 Positions Backwards.

From the above column examples you can infer the meaning of **Forwards** is basically down while the meaning of **Backwards** is basically up. All cells remain within the original selected block and wrap around.

We should also explain that the **Number of Positions** will always be a whole number larger than or equal to 1. This number can never be 0 or negative. In this case the word **Positions** means the same as cells.

Next we show three blocks with the numbers from 1-12 in a horizontal row. The row on top is the original block with the numbers in order from left-to-right. The next two rows show examples of different rotations. Again, all the cells remain within the original selected block and wrap around.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12
1 >	1	2	3	4	5	6	7	8	9	10	11	12

The numbers from 1-12 in order from left-to-right.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12
1 >	11	12	1	2	3	4	5	6	7	8	9	10

Rotated 2 Positions Forwards.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12
1 >	6	7	8	9	10	11	12	1	2	3	4	5

Rotated 5 Positions Backwards.

From these row examples you can infer the meaning of **Forwards** is to the right while the meaning of **Backwards** is to the left.

Finally we show an original rectangular grid that has 5 rows and 9 columns. We have filled the grid with the numbers from 1-45 in order, reading across the rows, as you would read a paragraph of text.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
1 >	1	2	3	4	5	6	7	8	9
2	10	11	12	13	14	15	16	17	18
3	19	20	21	22	23	24	25	26	27
4	28	29	30	31	32	33	34	35	36
5	37	38	39	40	41	42	43	44	45

The numbers from 1-45 with the cells laid out like words in a paragraph.

When we select all these cells as one block, and rotate it **Forwards** by 4 **Positions** the result appears next.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
1 >	42	43	44	45	1	2	3	4	5
2	6	7	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31	32
5	33	34	35	36	37	38	39	40	41

The cells rotated 4 Positions Forwards.

If we rotate the original grid 4 **Positions Backwards**, the result would appear as:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
1 >	5	6	7	8	9	10	11	12	13
2	14	15	16	17	18	19	20	21	22
3	23	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39	40
5	41	42	43	44	45	1	2	3	4

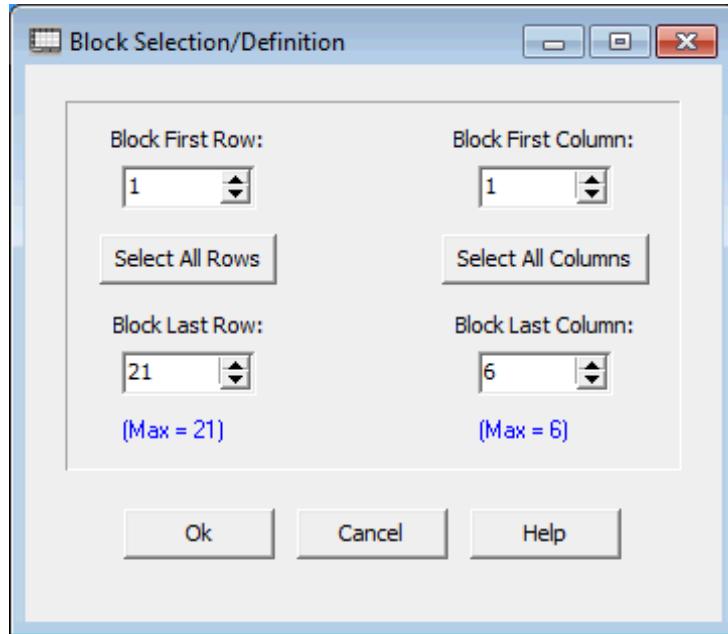
The cells rotated 4 Positions Backwards.

From these last two examples we can infer that **Forwards** means to the right and down while **Backwards** means to the left and up. It is easiest to remember these directions if you think about the order you would read the words in a paragraph of text. Reading forwards is always to the right and down while reading backwards would be to the left and up.

We assume these six examples of block rotations will put you in good standing regarding how block rotations work. Probably more often than not, when you need to perform a block rotation it will be for a single column of cells. Just remember to try to select the block before you choose **Blocks | Block Scramble/Rotate....** Once the dialog is open, first select either **Scramble** or **Rotate** as the desired **Action Selection**.

Selecting a Block

There may be times when you need to either select or precisely define a large block of cells, and you would prefer to do it without using the keyboard or your mouse. For such unusual cases you can select the command **Blocks | Block Selection/Definition...** and you should bring up the following simple dialog.



After defining an appropriate block using the six controls in the dialog, you can just click the **Ok** button and you should see your selected block of cells will be highlighted in the color blue.

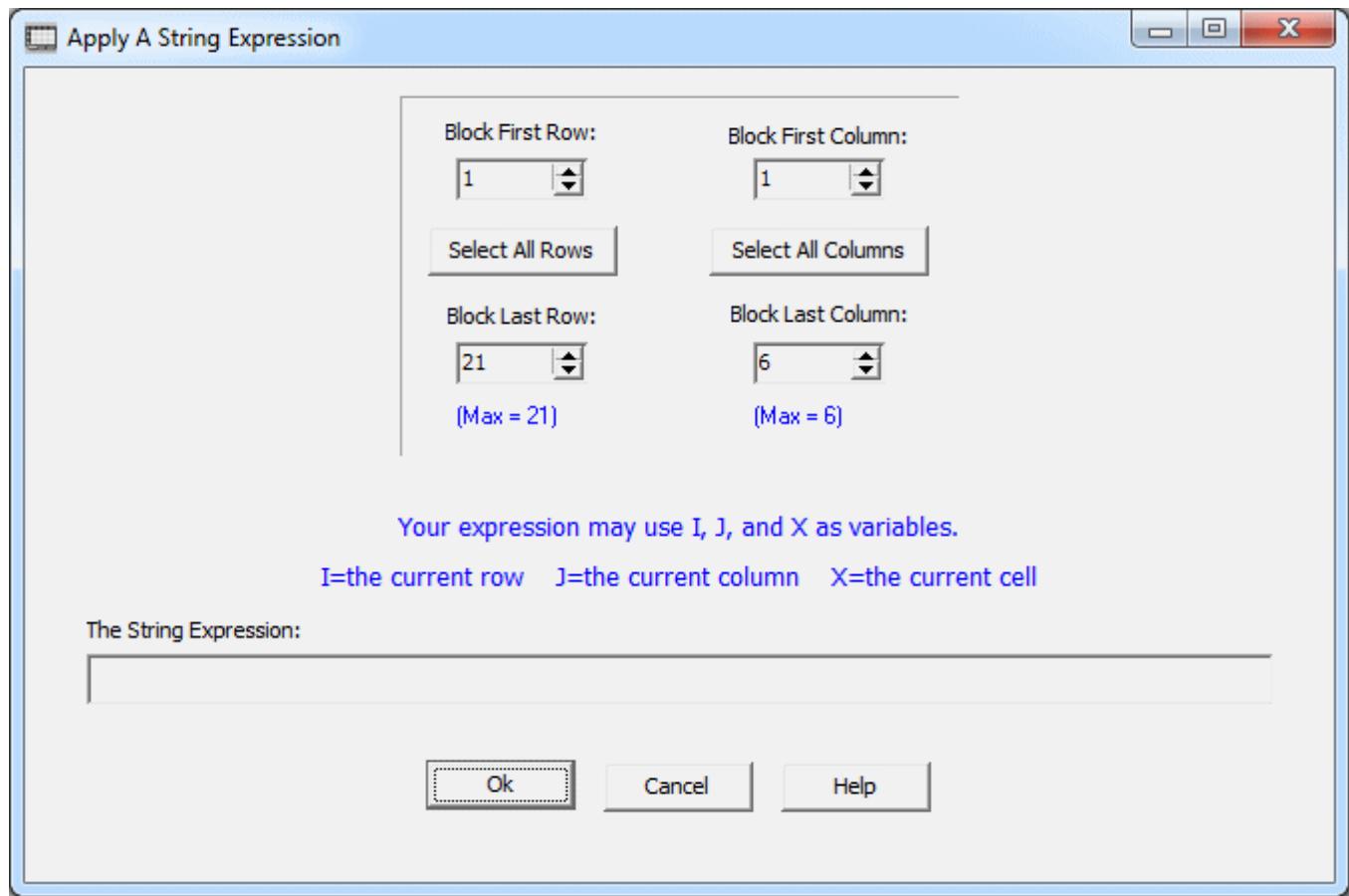
So this particular function doesn't really do very much except help you precisely select any block of cells. This may be rarely needed because most functions that require operating on a block of cells already contain the controls that allow you to precisely define the block. However, there are a few block functions like making a block copy where this function may be useful.

After you select a block, you may wish to open the menu **View | Report the Block Size...** to simply verify that the block you just defined is exactly what you thought you specified.

Applying String Expressions

In addition to math expressions, the **CSV Editor** program has the ability to apply what are called string expressions that are functions that can be used to build the contents of grid cells. A string is just another name for a collection of characters that are grouped together. Grid cells contain strings. Applying string expressions works very similar to applying math expressions, but string expressions don't have to involve numbers. String expressions only involve collections of characters. If you need numbers, use a math expression; otherwise use a string expression.

When you select the **Blocks** menu item **Block String Expression...** you will see the following dialog.



This dialog works in a manner very similar to that for applying math expressions. The upper controls define a rectangular block of cells that will get scanned or visited on a cell by cell basis. There is a wide edit box into which you are to type a string expression that will perform a particular task.

To understand string expressions you need only be introduced to a few kinds of string expressions. In fact, there are only six basic string functions and one operator that can combine string expressions. Understanding these seven basic things will make string expressions easy to apply. We will begin with a relatively simple example. The first figure on the next page shows a list of some of the major cities in the United States.

	Column 1	Column 2	Column 3	Column 4
<	City Name:	Latitude:	Longitude:	Location:
1	Akron	41.08	-81.52	Ohio
2	Albuquerque	35.12	-106.62	New Mexico
3	Alexandria	38.82	-77.09	Virginia
4	Amarillo	35.20	-101.82	Texas
5	Anaheim	33.84	-117.87	California
6	Anchorage	61.18	-149.19	Alaska
7	Arlington	32.69	-97.13	Texas
8	Arlington	38.88	-77.10	Virginia
9	Atlanta	33.76	-84.42	Georgia
10	Augusta-Richmond	33.46	-81.99	Georgia
11	Aurora	39.71	-104.73	Colorado
12	Aurora	41.77	-88.29	Illinois
13	Austin	30.31	-97.75	Texas
14	Bakersfield	35.36	-119.00	California
15	Baltimore	39.30	-76.61	Maryland
16	Baton Rouge	30.45	-91.13	Louisiana

Let's assume we want to combine the state locations that are in **Column 4** with the city names in **Column 1**. We can select **Column 1** by clicking on its column header and then we would enter the following string expression into the above string expression dialog box.

X + ", " + C(I,4)

When we click the **Ok** button to apply the string expression, the grid cells will change to look as follows:

	Column 1	Column 2	Column 3	Column 4
<	City Name:	Latitude:	Longitude:	Location:
1	Akron, Ohio	41.08	-81.52	Ohio
2	Albuquerque, New Mexico	35.12	-106.62	New Mexico
3	Alexandria, Virginia	38.82	-77.09	Virginia
4	Amarillo, Texas	35.20	-101.82	Texas
5	Anaheim, California	33.84	-117.87	California
6	Anchorage, Alaska	61.18	-149.19	Alaska
7	Arlington, Texas	32.69	-97.13	Texas
8	Arlington, Virginia	38.88	-77.10	Virginia
9	Atlanta, Georgia	33.76	-84.42	Georgia
10	Augusta-Richmond, Georgia	33.46	-81.99	Georgia
11	Aurora, Colorado	39.71	-104.73	Colorado
12	Aurora, Illinois	41.77	-88.29	Illinois
13	Austin, Texas	30.31	-97.75	Texas
14	Bakersfield, California	35.36	-119.00	California
15	Baltimore, Maryland	39.30	-76.61	Maryland
16	Baton Rouge, Louisiana	30.45	-91.13	Louisiana

When you inspect **Column 1** you can see that it now contains two new string elements. These elements were each created when the string expression **X + ", " + C(I,4)** was applied to each cell in **Column 1**.

The letter **X** represents the cell contents in **Column 1** as the selected block gets scanned. To this is added the string expression **", "**. This string expression is called a constant string because it contains no variables. The actual string content is what is contained between the two double quote characters. This is just two characters, a comma character and a space character. Then to all that is added the string represented by **C(I,4)**. This string is not constant because it is a cell reference and it contains the variable **I** that represents the current row number as the block is scanned. This cell reference refers to the constant column **4** that is the column that contains the state names. In string expressions, cell references must refer to strings, whereas cell references must eventually resolve to numeric values when used in math expressions. The syntax format for a cell reference is to write **C(row,column)** where **row** and **column** are themselves expressions that evaluate to numbers within proper ranges for the current grid. As any block gets scanned you should note that, **X=C(I,J)**.

The **+** symbol is used to denote string concatenation. Concatenation is just a fancy word that adds strings together as strings, which means that each next string gets appended to the right side of the previous string. The two **+** symbols add the three string parts together that make the final result. Because **Column 1** was selected as the block, it is only the cells in **Column 1** where the program deposits the result computed by the string expression.

Our next example shows a grid that contains some random dates that are assumed to be in the format **MM/DD/YYYY**.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
1 >	11/05/2013	05/17/2013	01/09/2013	01/25/2013	06/13/2013	05/09/2013	05/25/2013
2	03/05/2013	02/20/2013	01/17/2013	06/28/2013	12/05/2013	03/27/2013	11/18/2013
3	04/22/2013	10/22/2013	05/17/2013	01/10/2013	05/04/2013	05/01/2013	01/29/2013

We want to select all these cells and re-arrange their content so that it appears as **YYYY--DD** in which the month information is removed and the order of the year and day information gets switched from right to left. The string expression that will do this for us is:

Substring(X, 7, 4) + "--" + Substring(X, 4, 2)

When this expression gets applied, the new grid appears as:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
1 >	2013-05	2013-17	2013-09	2013-25	2013-13	2013-09	2013-25
2	2013-05	2013-20	2013-17	2013-28	2013-05	2013-27	2013-18
3	2013-22	2013-22	2013-17	2013-10	2013-04	2013-01	2013-29

As before, **X** represents the original cell contents. The function **Substring** is new and takes three arguments. The first argument is any string expression. The second argument is a starting position that is just a count of characters, starting with 1 as the first or left-most character in the string expression. The third argument is the

number of characters to be extracted from the string expression, starting with the start position character. No matter how large the 3rd argument, you never get characters beyond the end of the string. This means it is possible for **SubString** to return fewer characters than the 3rd argument specifies.

So we can understand that **SubString(X, 7, 4)** extracts the **4**-digits that represent the year when **X** is a string like **MM/DD/YYYY**. The first **Y** character is in the **7**th position.

Also, **SubString(X, 4, 2)** extracts the **2**-digits that represent the day number. The day number starts in position **4** in a string like **MM/DD/YYYY**. The first **D** character is in the **4**th position.

The middle expression "**--**" that gets added to the two **Substrings** is another example of a constant string. In addition to cell references like **X** and **C(I,J)**, the only other expressions that create strings are constant strings and the **SubString** and **Length** functions. In fact, **SubString** creates a string from the string expression that is its first argument.

The only three string manipulation functions we haven't discussed yet are named **Reverse** and **Length** and **Token**. **Reverse** takes only one argument that is any string expression. **Reverse** then reverses all the characters in the string expression and it always returns a string of the same length that it starts with.

If we were to apply the expression **Reverse(X)** to the last grid shown, that grid would change to look as follows. Now it would no longer make sense to think of these numbers as representing any kind of date information, but if you compare this grid with the one above you will easily understand how simple the **Reverse** function is.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
1 >	50-3102	71-3102	90-3102	52-3102	31-3102	90-3102	52-3102
2	50-3102	02-3102	71-3102	82-3102	50-3102	72-3102	81-3102
3	22-3102	22-3102	71-3102	01-3102	40-3102	10-3102	92-3102

The string function named **Length** is used to determine the number of characters in any string expression, and although the value returned represents a number, the value returned is really a string that represents a number. For example, imagine you have a grid that appears as follows in which we have ten rows, but where Column 2 is empty and we select the block that consists of only the elements in Column 2.

<	Column 1	Column 2
>	Best Picture	
1	The Apartment	
2	West Side Story	
3	Lawrence of Arabia	
4	Tom Jones	
5	My Fair Lady	
6	The Sound of Music	
7	A Man For All Seasons	
8	In The Heat Of The Night	
9	Oliver	
10	Midnight Cowboy	

If we then apply the string function/expression: **Length(C(I,1))**

the new elements in Column 2 will be the lengths of the strings in Column 1, in the same row. The answer grid should appear as:

<	Column 1	Column 2
>	Best Picture	
1	The Apartment	13
2	West Side Story	15
3	Lawrence of Arabia	18
4	Tom Jones	9
5	My Fair Lady	12
6	The Sound of Music	18
7	A Man For All Seasons	21
8	In The Heat Of The Night	24
9	Oliver	6
10	Midnight Cowboy	15

If you count the number of characters in each movie name in Column 1 you should find the same answer is the string that is in Column 2.

The next string expression example we will give is one for which we have a mostly empty grid with 3 columns and 49 rows and one fixed header row. The header row can be seen in the grid that is on the next page. Initially, the only nonempty cell is in row 1 column 3 and that cell contains the special sentence: **This is a test of the emergency broadcast system.** This sentence contains exactly 49 characters or letters, including the final period character. We select the block as all the cells in rows 1 through 49 inclusive, in column 1. The block string expression we are going to apply is:

Substring(C(1,3), I, 1)

The result is shown in the grid on the following page. The effect of the above string expression is to extract the individual letters from the special sentence and to place those letters in consecutive rows in column 1. The grid on the next page shows all 49 rows.

Reading down the first column letters yields the special sentence that has been re-shaped into a vertical format.

<	Column 1	Column 2	Column 3
>	The Letters:		
1	T		This is a test of the emergency broadcast system.
2	h		
3	i		
4	s		
5			
6	i		
7	s		
8			
9	a		
10			
11	t		
12	e		
13	s		
14	t		
15			
16	o		
17	f		
18			
19	t		
20	h		
21	e		
22			
23	e		
24	m		
25	e		
26	r		
27	g		
28	e		
29	n		
30	c		
31	y		
32			
33	b		
34	r		
35	o		
36	a		
37	d		
38	c		
39	a		
40	s		
41	t		
42			
43	s		
44	y		
45	s		
46	t		
47	e		
48	m		
49	.		

The next string expression examples are based on the grid shown below. The expression we will first apply is

`SubString(C(I,1),Length(C(I,1))-12,10)`

This expression uses a composition of the `Length` function and the element reference that is `C(I,1)`. We subtract `12` from `Length(C(I,1))` to determine the starting position from which we will try to extract the next `10` characters, if that many characters remain in the string.

We will apply this expression to the block that consists of all rows in Column 2. The grid on the left shows the starting grid before the expression is applied, while the grid on the right shows the same grid after applying the expression.

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	
2	Sophia Loren	
3	Anne Bancroft	
4	Patricia Neal	
5	Julie Andrews	
6	Julie Christie	
7	Elizabeth Taylor	
8	Katherine Hepburn	
9	Katherine Hepburn	
10	Maggie Smith	

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	zabeth Tay
2	Sophia Loren	
3	Anne Bancroft	Anne Bancr
4	Patricia Neal	Patricia N
5	Julie Andrews	Julie Andr
6	Julie Christie	ule Chris
7	Elizabeth Taylor	zabeth Tay
8	Katherine Hepburn	erine Hepb
9	Katherine Hepburn	erine Hepb
10	Maggie Smith	

If we analyze the above string expression, we can determine that the variable `I` is only used to track the row numbers that range between 1 and 10. The expression itself is used to extract substrings from the strings in Column 1. Those substrings are deposited in Column 2 and begin `12` characters back from the last letter in each string, and they then use the next `10` characters going to the right from that starting position. It turns out that the names **Sophia Loren** and **Maggie Smith** consist of exactly `12` letters, so when we back up `12` positions from the ends of those names we end up outside the original string boundaries and this is why the strings that are finally returned for these two names are the empty strings.

Note that the name **Anne Bancroft** is 13 letters long, so when we backup `12` letters from the 13th letter we end up at the 1st letter in the name. When we then take the next `10` letters we end up with the incomplete name **Anne Bancr**. The names **Patricia Neal** and **Julie Andrews** are also 13 letters long and are handled similarly.

The names **Elizabeth Taylor** and **Katherine Hepburn** are more than 13 letters long so when the above string expression is applied to these names we get a result that starts after the first letter in the name and finishes before the last letter. Note we extract at most `10` letters from any name. In fact, the name **Julie Christie** is such that we miss the first letter **J** and we also miss the last two letters **ie**.

If we change the above expression to the following:

`SubString(C(I,1),Max(1,Length(C(I,1))-12),10)`

then we will get a different result with the same starting grid.

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	
2	Sophia Loren	
3	Anne Bancroft	
4	Patricia Neal	
5	Julie Andrews	
6	Julie Christie	
7	Elizabeth Taylor	
8	Katherine Hepburn	
9	Katherine Hepburn	
10	Maggie Smith	

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	izabeth Tay
2	Sophia Loren	Sophia Lor
3	Anne Bancroft	Anne Bancr
4	Patricia Neal	Patricia N
5	Julie Andrews	Julie Andr
6	Julie Christie	ule Chris
7	Elizabeth Taylor	izabeth Tay
8	Katherine Hepburn	erine Hepb
9	Katherine Hepburn	erine Hepb
10	Maggie Smith	Maggie Smi

We can explain this example by stating that the expression `Max(1,Length(C(I,1))-12)` determines the maximum numeric value between `1` and the number returned by the expression `Length(C(I,1))-12`. This means the starting point before extracting the next `10` letters may include the first letter. The returned substring in this case will never be empty.

If we change the expression to:

`SubString(C(I,1),Min(1,Length(C(I,1))-12),10)`

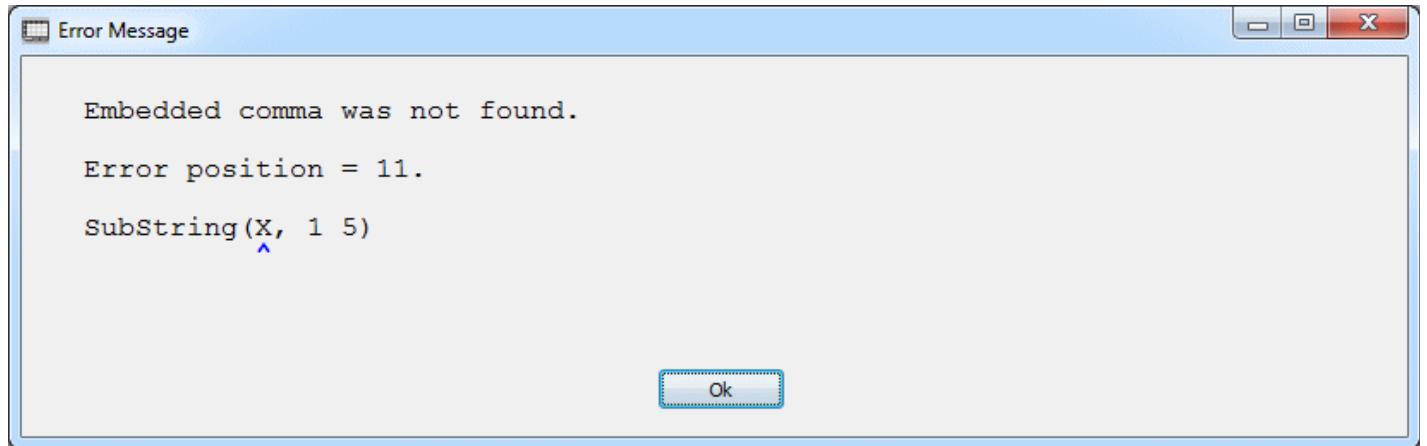
we get yet another result that returns the first `10` letters of the name, but only when the starting position is not 0 or a negative number that would make an empty string as in the first example. The 2nd argument expression, `Min(1,Length(C(I,1))-12)`, returns the minimum between `1` and the number `Length(C(I,1))-12`. It is only when the minimum would be 0 or a negative that we could again get an empty string.

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	
2	Sophia Loren	
3	Anne Bancroft	
4	Patricia Neal	
5	Julie Andrews	
6	Julie Christie	
7	Elizabeth Taylor	
8	Katherine Hepburn	
9	Katherine Hepburn	
10	Maggie Smith	

<	Column 1	Column 2
>	Best Actress	Tagged
1	Elizabeth Taylor	Elizabeth
2	Sophia Loren	
3	Anne Bancroft	Anne Bancr
4	Patricia Neal	Patricia N
5	Julie Andrews	Julie Andr
6	Julie Christie	Julie Chri
7	Elizabeth Taylor	Elizabeth
8	Katherine Hepburn	Katherine
9	Katherine Hepburn	Katherine
10	Maggie Smith	

The last two examples made creative use of the two numeric functions named `Max` and `Min`. These two functions `Max` and `Min` are often used in conjunction with the `Length` and `SubString` functions, especially with the two numeric arguments of the `SubString` function. You can learn about the `Max` and `Min` functions by selecting **Help | Math Expressions....**

If you should enter a string expression that has any kind of a string syntax error, then you may see an error message like that shown below.



We want to emphasize the program will try to point out the position in your expression string where it encountered an error, but the true error position may be other than where the `^` symbol points. In the above example, it is actually a second embedded comma that is missing between the 1 and 5 characters in the expression string. In this example the `^` symbol points to the last known good position in the expression.

The last string manipulation function we will briefly discuss is named **Token**. This function can be thought of as being somewhat related to the **Substring** function in that it returns substring characters, but **Token** takes different arguments and works in a very different manner. **Token** takes three arguments, where the first argument must be a specially formatted string. The second argument is what is called the token separator character, and the third argument is an item count number.

For our first example, consider the specially formatted constant string `"45;93;87;21;39;44;61;26"`. What makes this string special is that it is supposed to contain a series of eight items, separated by the semicolon character `;`. In this example all of the items are numbers, but in general the items can be any series of characters as long as they don't include the `;` character. Now the purpose of the **Token** function is to allow us to extract the numbers from this specially formatted string, one number at a time, by entering the position of the number we want. For example, making the function call

```
Token("45;93;87;21;39;44;61;26", ;, 5)
```

where the third argument is the number `5` should return the substring `39` that is the `5th` item in the specially formatted string that is the first argument.

So you should begin to understand that **Token** takes a specially formatted string as its first argument. The second argument to **Token** is what we call the token separator character. In the above example the token separator character was the semicolon character. Normally the token separator character can be any character that suits your needs and separates the items or substrings that are listed. The only exceptions are left and right parentheses. The token separator character should never be a left or a right parenthesis. The third argument is simply the count of the item that is to be extracted. Now we can give another and perhaps more practical example of using the **Token** function. Suppose you have a grid that looks like the following:

<	Column 1	Column 2	Column 3
1 >			This is a test of parsing some individual words.
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Now imagine you want to extract the individual words from the sentence contained in the cell in Row 1 and Column 3. If you were to select the first 12 rows in Column 1 and apply the string expression `Token(C(1,3), ,I)` to that block of cells, the resulting grid would look like:

<	Column 1	Column 2	Column 3
1 >	This		This is a test of parsing some individual words.
2	is		
3	a		
4	test		
5	of		
6	parsing		
7	some		
8	individual		
9	words.		
10			
11			
12			

In this example, the program used the space character as the second argument to the `Token` function. That means the token separator character is the space character. The `Token` function works by scanning across the string that is its first argument. The string is scanned from left to right and the `Token` function counts token separator characters until it reaches the count of its third argument that is the item counter. The `Token` function will normally return the substring that exists in front of that token separator character.

Because the token separator character in this example was the space character, the function call `Token(C(1,3), ,I)` was able to return each the 9 distinct words contained in the sentence. Note however that the special variable `I` ranged from 1 to 12 when we selected the 12 rows of Column 1 as the block to which the string expression was applied. When the item count number `I` went beyond the number of items contained in the specially formatted list, then the substring returned was the empty string. So we should also understand from this example that rows 10, 11, and 12, in Column 1, were all assigned the empty string.

The specially formatted string does not normally need to end with a token separator character, but in terms of how the **Token** function works, it would return the same answer as if we had an imaginary separator character as the last character in the string expression. The **Token** function will return the empty string whenever you scan past the end of the string without reaching the item count. It will also return the empty string if its first argument is the empty string or if the item count is less than 1. The third argument that is the item count should normally be a positive integer greater than or equal to 1.

You may find it interesting to consider that any **CSV** file line that does not contain any double quotes, and only contains separator commas between items, is an example string where the commas may be considered to be the token separator character that separates the items contained in the line.

Imagine you have a string grid which contains a block of dates that are in the questionable date format **mm/dd/yy** where the month and day numbers may or may not have 2-digits, and where the year value only has two digits and is missing the century information. Using only the **Token** function, you could apply the string expression

```
Token(X,,,1) + "/" + Token(X,,,2) + "/" + "20" + Token(X,,,3)
```

to reformat those dates so they were in the 21st century (i.e. having a 2000 year century). This means those dates would be given the better format **mm/dd/20yy**. You could change the string **"20"** to **"19"** in the above expression if you wanted the years to be in the 1900s instead of the 2000s.

An equivalent string expression that accomplishes the same task, but uses the **SubString** and **Length** functions would be:

```
SubString(X,1,Length(X)-2) + "20" + SubString(X,Length(X)-1,2)
```

We think using the **Token** function expression is the better alternative, but both string expressions will accomplish what is needed to fix the incomplete date format problem.

As a final example, we explain how you can quickly create Social Security Numbers. The easiest way to do this is to select **Blocks | Block Fill**, and click the button to **Setup a Range For Random Numbers...** and set the **Minimum Value** as 100000000 and set the **Maximum Value** as 999999999. Click the radio button to **Generate Integers Only**. Finally select a **Normal Distribution** and click the **Ok** button. Set the proper range for the block and click another **Ok** button. After the block is filled with 9-digit integers, select the block of cells and choose **Blocks | Block String Expression...** and use the following string expression:

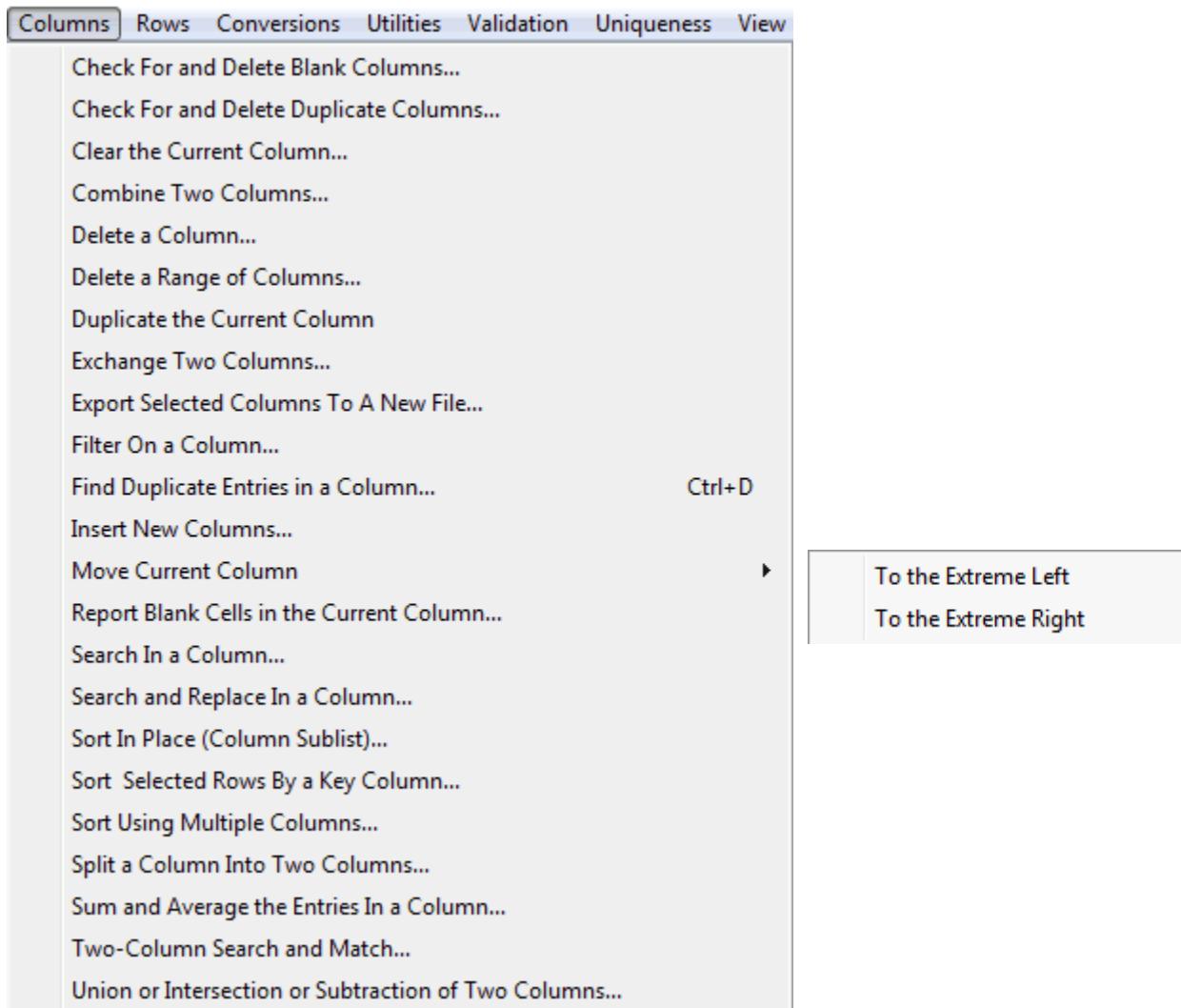
```
SubString(X,1,3) + "--" + SubString(X,4,2) + "--" + SubString(X,6,4)
```

You should then see nicely formatted numbers that look like real Social Security numbers.

There is a separate **Help** menu item **String Expressions...** that gives more information and details about string expressions and how to apply them. We have only covered the basics in the above examples, but these examples should be enough to get you started using string expressions. Learning about **Math Expressions** will also help you with applying **String Expressions** so you may wish to open **Help | Math Expressions....**

The Columns Menu

The **Columns** menu contains functions that apply to working with columns and the opened menu list appears as follows:

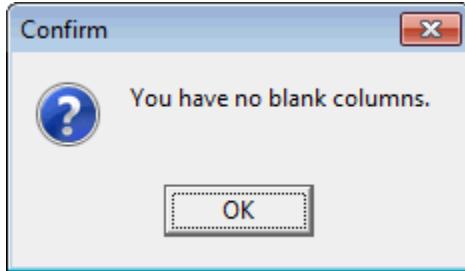


Keep in mind that every column can also be considered as a block. Some blocks are also just columns. So whenever you are looking for a function, that function could be under the **Columns** menu or the **Blocks** menu. If you don't find the function you need under the **Columns** menu, then you might try looking under the **Blocks** menu, and vice versa.

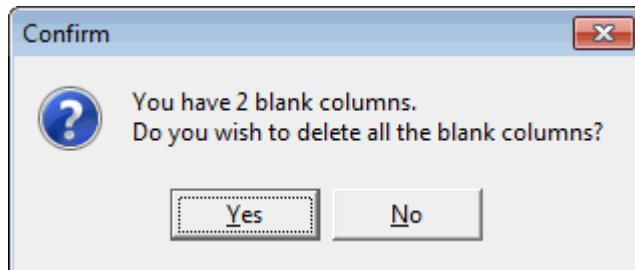
Check For and Delete Blank Columns

If you select the menu item **Columns | Check For and Delete Blank Columns...** you will see one of two messages.

The first message occurs after the program checks and finds no blank columns.



On the other hand, if at least one blank column exists you will see a message similar to the following:

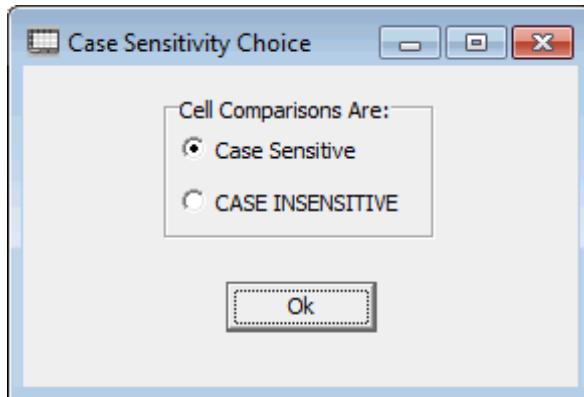


If you click the **Yes** button, the program will then try to delete all the blank columns. However, the program never deletes all the columns in the grid. So only in the rare case that the entire grid is blank, you can be left with exactly one blank column.

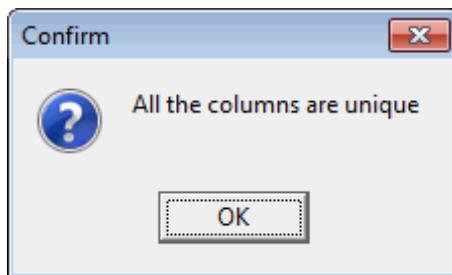
Of course if you click the **No** button then no further action is taken.

Check For and Delete Duplicate Columns

If you select the menu item **Columns | Check For and Delete Duplicate Columns...** the program will first prompt you to choose a case sensitive option for how cell comparisons should be made.

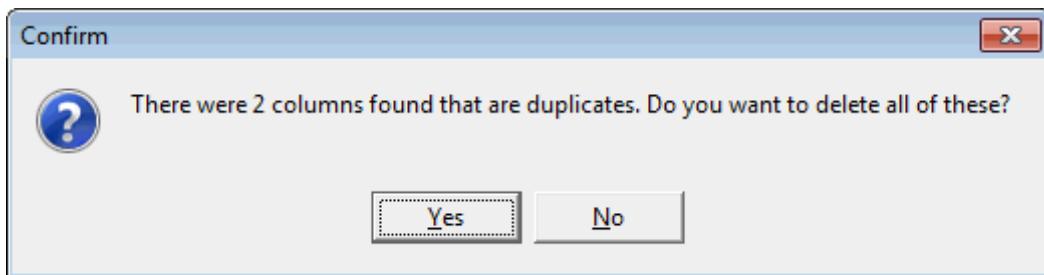


After you click **Ok**, the program will check to see how many duplicate columns you have, if any. You may see a message like:



In this case you don't have any duplicate columns so the function comes to an end.

On the other hand, if you have any duplicate columns you may see a message like:



If you click **No** then the function ends.

If you click **Yes**, then the program will delete all the duplicate columns while still preserving the columns that can be considered as the originals of those duplicates.

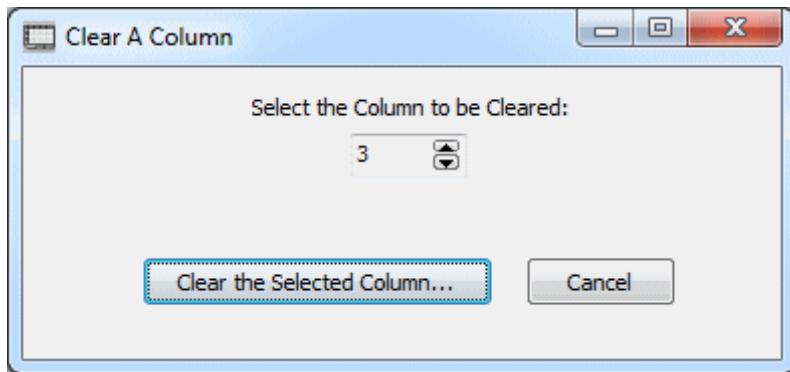
In other words, only the duplicates that are not needed and are probably just wasting space will be eliminated, without eliminating the needed original columns that can be considered firsts. After this, the **Undo** button in the toolbar will become active, just in case you need to undo the column deletions.

This duplicate column delete function differs from the function that deletes duplicate rows in that it does not offer you a choice of a range of columns because the range of columns spans all columns.

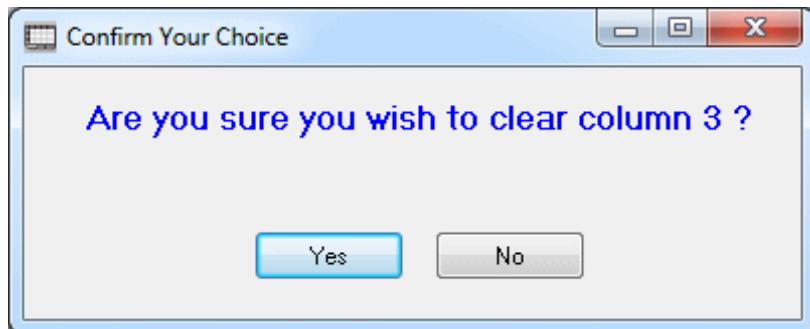
This menu item is repeated under the **Uniqueness** menu. If you only have two columns that you want to compare, then you might also consider a specialized variation of this function by using another function **Uniqueness | Compare Two Columns....**

Clearing the Current Column

If you leave the selection in any row of a column you can clear that column by making empty cells in all rows. Just select the menu item **Columns | Clear the Current Column...** and you will be given a chance to select any column to be cleared, but the default value should already be filled in for you.



If you click the button to **Clear the Selected Column** then the program will ask you to confirm your choice:



If you click **Yes**, the program will empty each cell in the selected column. If you click **No**, the program will ignore the command to clear the column.

An alternative technique to clear a single column is to left click the mouse on the column number to first select the entire column. Then you can press the **DELETE** key on your keyboard. Whatever cells are selected at the time you press the **DELETE** key on your keyboard will be cleared to blank entries. You could also select a block of cells that comprises multiple consecutive columns and then press the **DELETE** key on your keyboard.

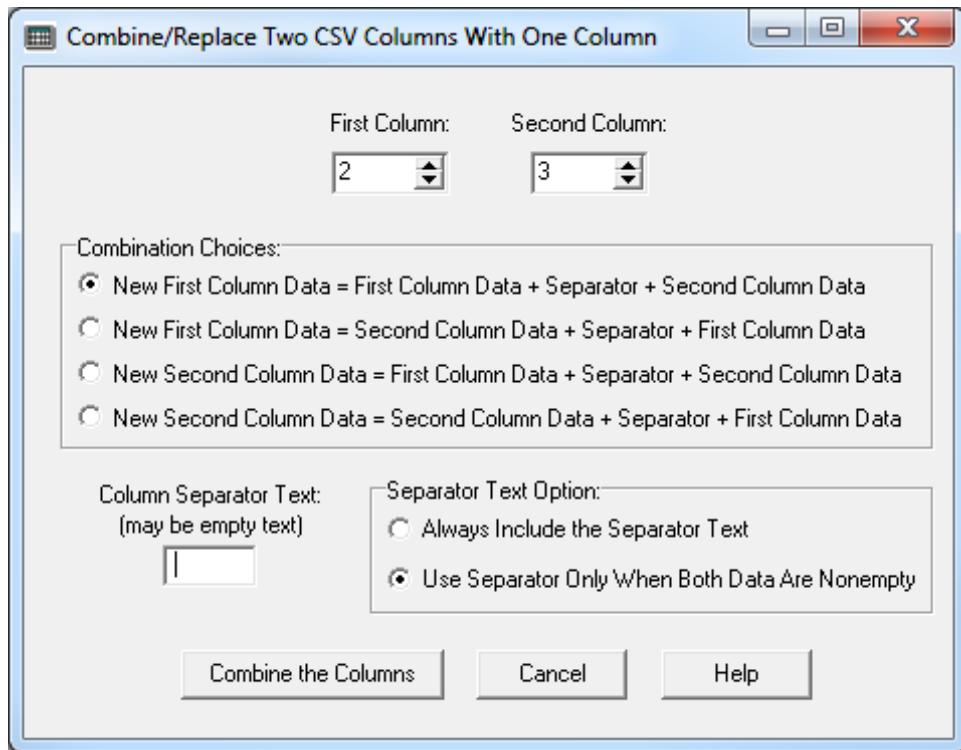
Combining Two Columns

There may be times when it is desirable to combine two columns to make a new single column. An example of needing to do this could be when you have a database table of names. Suppose you have three columns that contain the First Name, Middle Name, and Last Name of employees in your company.

While it is common to sort by the Last name, suppose you want to combine the First and Middle names, in that order, and place the combination back into the original First Name column.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
>	Title	First Name	Middle Name	Last Name	Street Address	City	State	ZIP Code
1	Mrs.	Alma	Ruth	Brantley	1109 Sunset Blvd.	Los Angeles	CA	90021
2		Jay	Norton	Bratton	23234 9th St.	Los Angeles	CA	92323
3		Jeff	Bruce	Bremer	1212 Nineteenth Street	San Diego	CA	93432
4	Mr.	William	Charles	Brendon	6938 West 77th Street	New York	NY	10014
5		Andrea	Janice	Chasin	2424 Culver Blvd.	Chicago	IL	21121
6		Cindy	Cayla	Chou	3717 Bagley Ave.	Santa Monica	CA	90401
7		Dorothy	Ellie	Cummings	1919 Broadway Blvd.	Chicago	IL	23231
8	Mr.	John	Frederick	Doe	12345 Any Street	New York	NY	10001
9	Mrs.	Mary	Jane	Doe	12345 Any Street	New York	NY	10001
10		Brian	Jon	Foudy	2024 Holt Ave.	Los Angeles	CA	93458
11		Gilbert	Lyle	Garcia	12120 Washington Place	San Gabriel	CA	90032
12		Jose	Ferro	Garcia	11921 Malibu St.	Santa Monica	CA	90410
13	Dr.	Layla	Kay	Gelf	3333 Ventura Blvd.	Encino	CA	93421
14		John	Jeff	Hollister	2790 Club Drive	Chicago	IL	22213
15		Robert	Allan	Humphrey	1407 Brockton Ave.	Riverside	CA	93481
16	Mr.	Joe	Brian	Johnson	54321 Elm Street	Salt Lake City	UT	84119
17	Mr.	Donald	Bob	Langston	7418 89th Avenue	Chicago	IL	23121
18		Peter	Scott	Lucas	3576 Ocean Ave.	St. Louis	MO	42123
19		Jean	Claudia	Mackey	717 Manchester	Salt Lake City	UT	81143
20		Steve	Terry	Madden	1337 Saltair Ave.	Los Angeles	CA	98743

To combine the First Name and the Middle Name and to place the result in the position of the First Name column, first select the menu item **Columns | Combine Two Columns...**



Then you need to enter the **First Column** as the number **2** and enter the **Second Column** as the number **3** and you would want to select the first radio button that will combine the **First Column** and **Second Column** in that order, but also put the result back into the first column. Be sure to enter a single blank space character as the **Column Separator Text**. For this example you should also check the radio button to **Always Include the Separator Text**. Your dialog should look exactly like the one shown above.

When you click the button to **Combine the Columns** the program will automatically do everything it needs to do. For this example, you should see the following where the changed **Column 2** contains the combined names.

Also note that this operation was also applied to the column header information.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
>	Title	First Name Middle Name	Last Name	Street Address	City	State	ZIP Code
1	Mrs.	Alma Ruth	Brantley	1109 Sunset Blvd.	Los Angeles	CA	90021
2		Jay Norton	Bratton	23234 9th St.	Los Angeles	CA	92323
3		Jeff Bruce	Bremer	1212 Nineteenth Street	San Diego	CA	93432
4	Mr.	William Charles	Brendon	6938 West 77th Street	New York	NY	10014
5		Andrea Janice	Chasin	2424 Culver Blvd.	Chicago	IL	21121
6		Cindy Cayla	Chou	3717 Bagley Ave.	Santa Monica	CA	90401
7		Dorothy Ellie	Cummings	1919 Broadway Blvd.	Chicago	IL	23231
8	Mr.	John Frederick	Doe	12345 Any Street	New York	NY	10001
9	Mrs.	Mary Jane	Doe	12345 Any Street	New York	NY	10001
10		Brian Jon	Foudy	2024 Holt Ave.	Los Angeles	CA	93458
11		Gilbert Lyle	Garcia	12120 Washington Place	San Gabriel	CA	90032
12		Jose Ferro	Garcia	11921 Malibu St.	Santa Monica	CA	90410
13	Dr.	Layla Kay	Gelf	3333 Ventura Blvd.	Encino	CA	93421
14		John Jeff	Hollister	2790 Club Drive	Chicago	IL	22213
15		Robert Allan	Humphrey	1407 Brockton Ave.	Riverside	CA	93481
16	Mr.	Joe Brian	Johnson	54321 Elm Street	Salt Lake City	UT	84119
17	Mr.	Donald Bob	Langston	7418 89th Avenue	Chicago	IL	23121
18		Peter Scott	Lucas	3576 Ocean Ave.	St. Louis	MO	42123
19		Jean Claudia	Mackey	717 Manchester	Salt Lake City	UT	81143
20		Steve Terry	Madden	1337 Saltair Ave.	Los Angeles	CA	98743

Now we can illustrate one more little technique with the program that allows you to edit the header titles. When the columns were combined, the column headers were also combined and we would like to change the header in **Column 2** that is shown above. The only problem is that you can't edit the text that is highlighted in yellow.

Click the toolbar button  . After you do this you should see the header line is now available for editing.

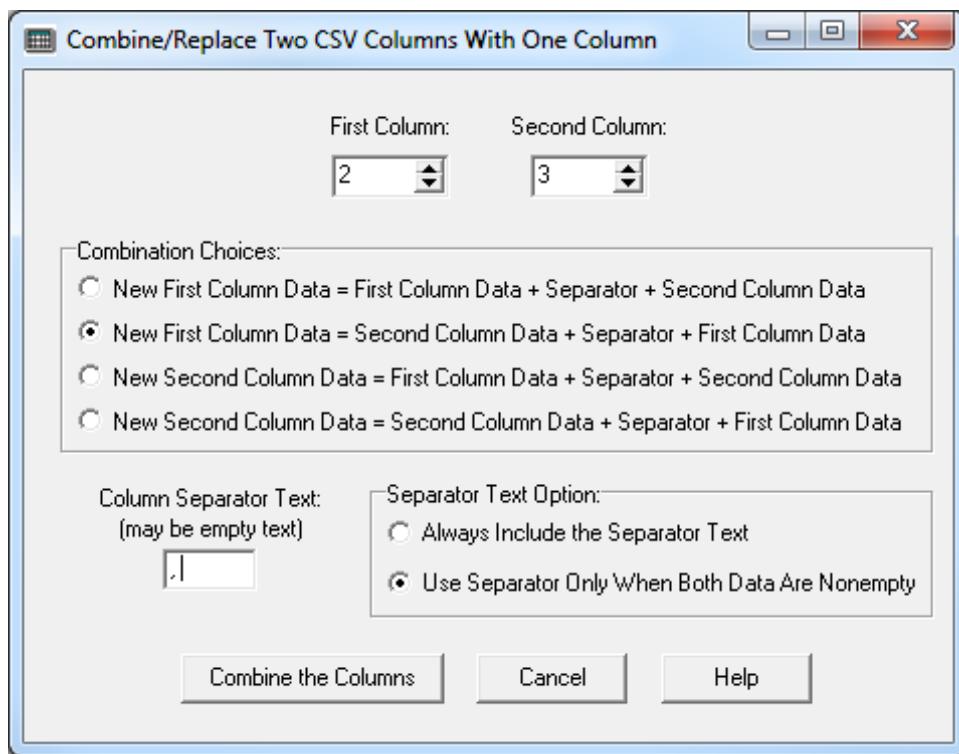
<	Column 1	Column 2	Column 3	Column 4
>	Title	First Name Middle Name	Last Name	Street Address
1	Mrs.	Alma Ruth	Brantley	1109 Sunset Blvd.
2		Jay Norton	Bratton	23234 9th St.

We will edit the **Column 2** header to read **First Names** and then we click the same toolbar button that appears as  . That same button will again show as  , meaning the first line will now be displayed and treated as a header line.

<	Column 1	Column 2	Column 3	Column 4
>	Title	First Names	Last Name	Street Address
1	Mrs.	Alma Ruth	Brantley	1109 Sunset Blvd.
2		Jay Norton	Bratton	23234 9th St.

In any other application where you need to combine two columns, just remember that you have the option to replace either of the two columns being combined, and you could actually change the Left-Right order of those two columns at the time they are combined.

For example, we are now going to combine columns 2 and 3, but we want the new column to start with the last name on the left, followed by a comma and a space and then the first names. The dialog to accomplish that would look like the following where the **Column Separator Text** is a comma and a blank space. Note the second radio button is chosen in the **Combination Choices** radio group.



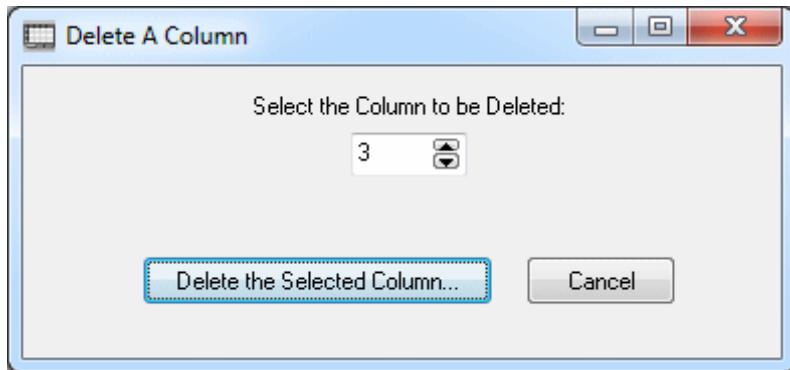
After we open the new file we could change the single name column to have the header title Name, the final result should then appear as:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Title	Name	Street Address	City	State	ZIP Code
1	Mrs.	Brantley, Alma Ruth	1109 Sunset Blvd.	Los Angeles	CA	90021
2		Bratton, Jay Norton	23234 9th St.	Los Angeles	CA	92323
3		Bremer, Jeff Bruce	1212 Nineteenth Street	San Diego	CA	93432
4	Mr.	Brendon, William Charles	6938 West 77th Street	New York	NY	10014
5		Chasin, Andrea Janice	2424 Culver Blvd.	Chicago	IL	21121
6		Chou, Cindy Cayla	3717 Bagley Ave.	Santa Monica	CA	90401
7		Cummings, Dorothy Ellie	1919 Broadway Blvd.	Chicago	IL	23231
8	Mr.	Doe, John Frederick	12345 Any Street	New York	NY	10001
9	Mrs.	Doe, Mary Jane	12345 Any Street	New York	NY	10001
10		Foudy, Brian Jon	2024 Holt Ave.	Los Angeles	CA	93458
11		Garcia, Gilbert Lyle	12120 Washington Place	San Gabriel	CA	90032
12		Garcia, Jose Ferro	11921 Malibu St.	Santa Monica	CA	90410
13	Dr.	Gelf, Layla Kay	3333 Ventura Blvd.	Encino	CA	93421
14		Hollister, John Jeff	2790 Club Drive	Chicago	IL	22213
15		Humphrey, Robert Allan	1407 Brockton Ave.	Riverside	CA	93481
16	Mr.	Johnson, Joe Brian	54321 Elm Street	Salt Lake City	UT	84119
17	Mr.	Langston, Donald Bob	7418 89th Avenue	Chicago	IL	23121
18		Lucas, Peter Scott	3576 Ocean Ave.	St. Louis	MO	42123
19		Mackey, Jean Claudia	717 Manchester	Salt Lake City	UT	81143
20		Madden, Steve Terry	1337 Saltair Ave.	Los Angeles	CA	98743

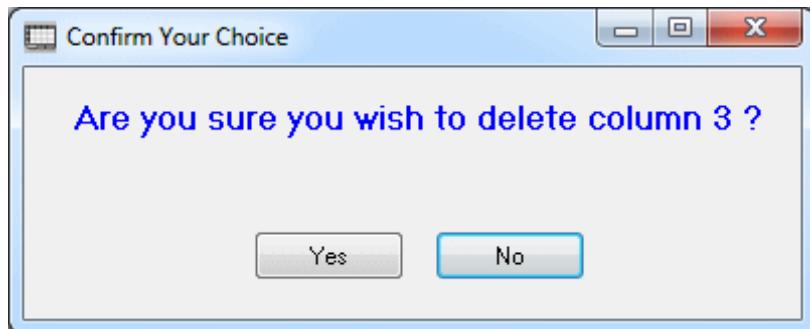
By now you should be able to combine any two columns as needed in any order as needed. Don't forget to set the **Column Separator Text** as appropriate.

Deleting a Single Column or a Range of Columns

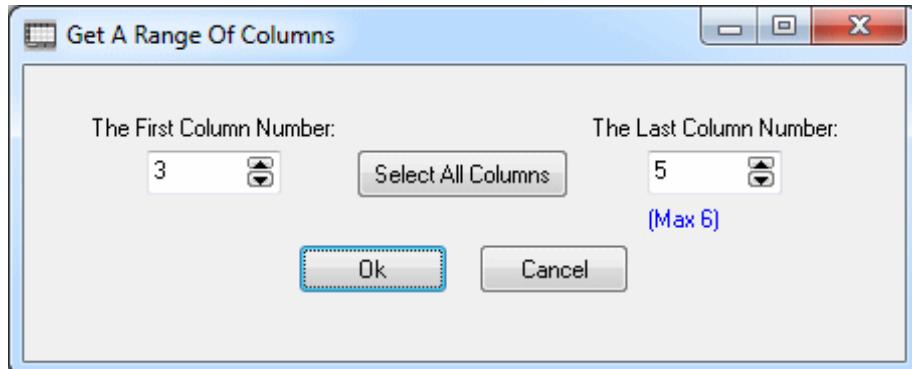
You can delete a single column by clicking the button  in the toolbar, or you can select the menu item **Columns | Delete a Column...**. When you do this you will be prompted by an initial dialog in which you can select the column to be deleted:



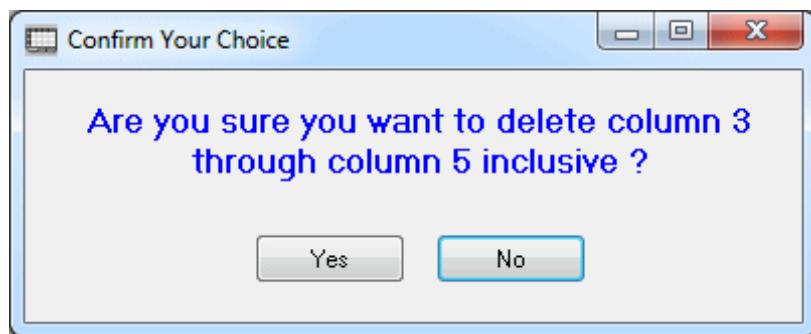
If you proceed you will be given a chance to confirm your intentions:



If you know you want to delete many consecutive columns in one step, then you can select **Columns | Delete a Range of Columns...** and you will be prompted to select a range of columns:



If you continue you will be given one last chance to change your mind.



All columns get renumbered after you delete any column or range of columns.

Also note that you can press the undo button in the toolbar if you accidentally delete the wrong columns.

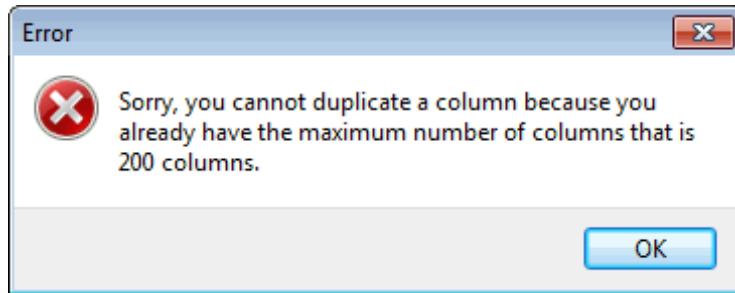
Duplicating the Current Column

You can duplicate the current column by clicking the button  in the toolbar or you can select the menu item **Columns | Duplicate the Current Column**.

When you do this the action that takes place is automatic. A new column is created immediately to the right of the current column and the contents in all rows will be filled with the entries of the previous column. Before the insertion takes place, all columns to the right of the current column, if any, will shift one column position to the right. So duplicating the current column never overwrites any existing data. After the insertion takes place, all columns will get renumbered as needed.

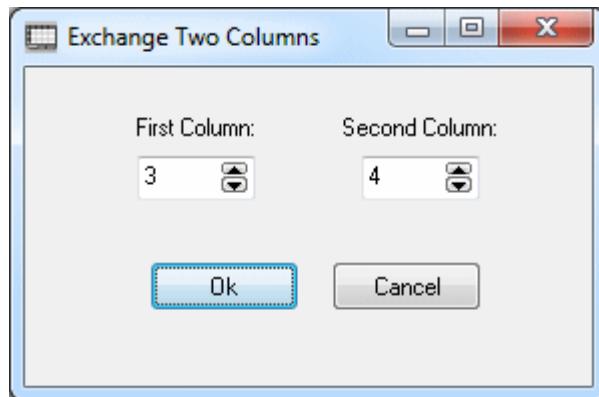
After the current column is duplicated, if you decide to change your mind you can click the undo  button in the toolbar to get back to where you started.

The above description assumes your grid still has room to add a new column. If your grid already has the maximum number of columns then you will see the message:



Exchanging Two Columns

You can exchange any two columns by selecting the menu item **Columns | Exchange Two Columns...**. You will then see a dialog that allows you to select the two columns to be exchanged:



When you click the **Ok** button, the program will swap the contents of the two columns, including any header information as necessary. There is no need to renumber any columns. Only the column contents get exchanged. However, the displayed width for those two columns may be swapped as well so that those two columns keep the same widths they had before they were exchanged.

Exporting Columns

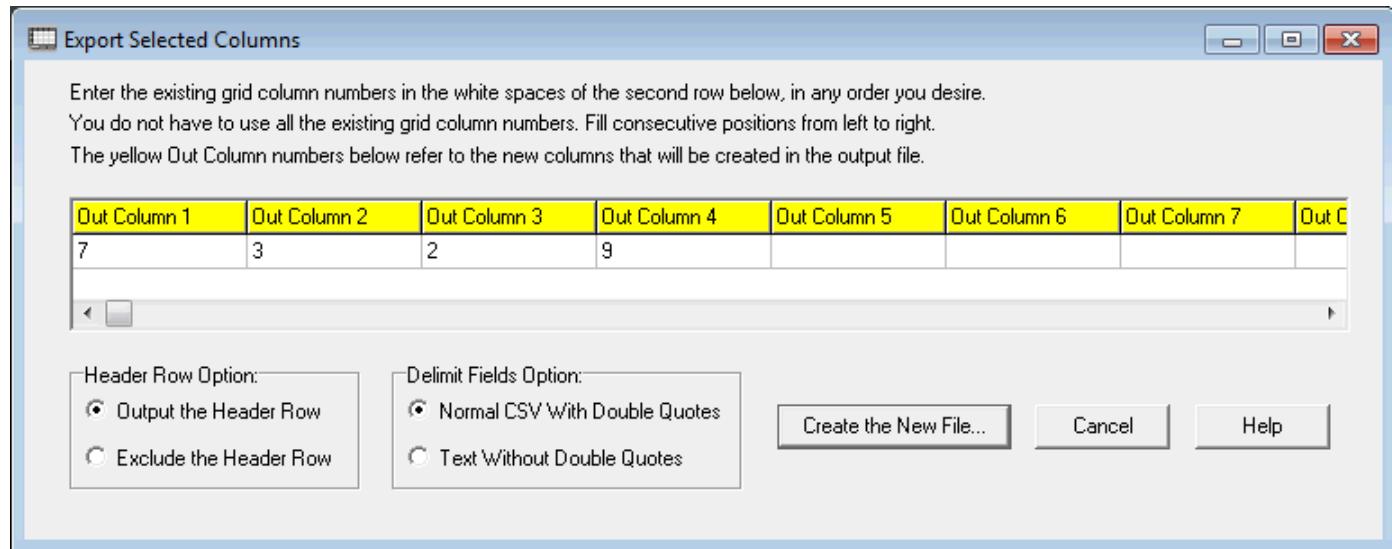
There may be times when you need to export certain columns from a **CSV** file. In fact, one reason might be to simply re-order the columns, although most of the time you would want to select a subset of the existing columns. Under the **Columns** menu is a submenu item with the title **Export Selected Columns To a New File....** In any case, to export any columns and save them in a new order always means creating a new **CSV** file. To get started doing this you would first start filling in the second row in the table shown in the next figure below. You would type in numbers that represent columns from the currently open **CSV** grid.

You don't have to use all the existing column numbers, and you can change the order of the columns as you extract columns. In the case where you want to permute the order of all existing columns, you should enter all existing column numbers, but in the new order you want, in the list in the dialog box shown below.

There is a first option called the **Header Row Option** that you can set as needed, depending on whether you want to export any header row information or not. You won't see the **Header Row Option** control if you are not displaying any header information because you should not see this option if there is no header row information that is currently being displayed.

A second option is called the **Delimit Fields Option**. This option allows you to choose the type of delimiting that is applied to the fields that are output. You can choose between making a normal **CSV** file with double quotes for field delimiters, or you can make a regular text file in which the fields are not delimited by double quotes. In either case, if your field data contains double quotes then those elements will be output where each quote is written as a pair of double quotes. Fields will always be separated with comma characters. A reason for having this option is so that programs not expecting to use true **CSV** files as input, and not capable of removing double quotes, can still read and use the text data that is output. The default is to make a **Normal CSV With Double Quotes**.

As an example, imagine you filled in the first few entries of the grid as shown below:



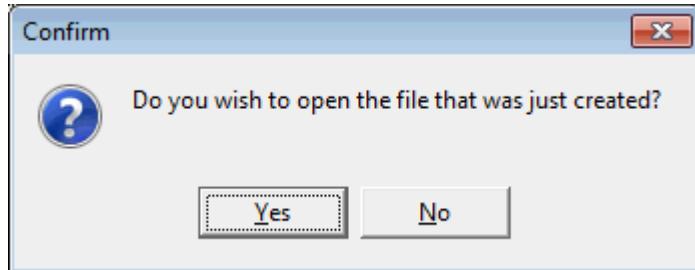
The meaning would be that you intended to extract exactly four columns from the existing grid. The first column to be extracted would be column number 7 from the existing grid, the second column to be extracted would be column 3 from the existing grid, the third column to be extracted would be column 2 from the existing grid, and the fourth and last column to be extracted would be column 9 from the existing grid.

Thus you should understand that when you export columns from an existing grid, you can take the columns in an order different from the order in which they appear in the existing grid.

One other point is needed. We intended to export only 4 columns in the above example. Thus we used the first four **Out Columns** for the output. You will need to fill in your **Out Column** numbers from left to right, consecutively until you enter your last **Out Column**. The first several **Out Columns** thus will always be consecutively filled.

You don't have to use all the **Out Columns** (in the above example we only used the first four **Out Columns**), but any existing column numbers that you enter must start with the first **Out Column** and fill consecutive **Out Columns** moving from left to right. The only blanks allowed are in all the remaining **Out Column** positions that will go unused.

When you click the button to **Create the New File**, you will be prompted to enter the new filename. If you enter the name of an existing file you will be given a chance to change your mind, or you can overwrite the existing file. You will also be given a chance to load the file you just created by responding to the following prompt.

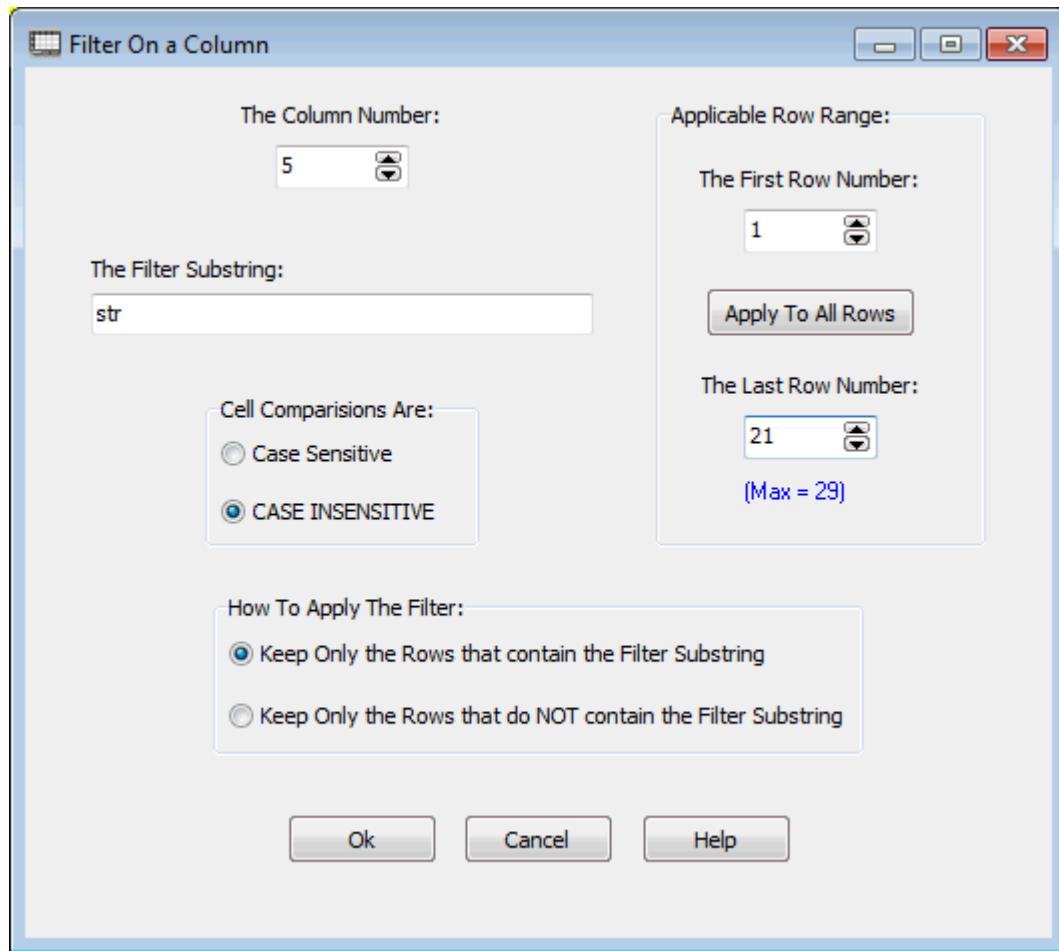


Filter On A Column

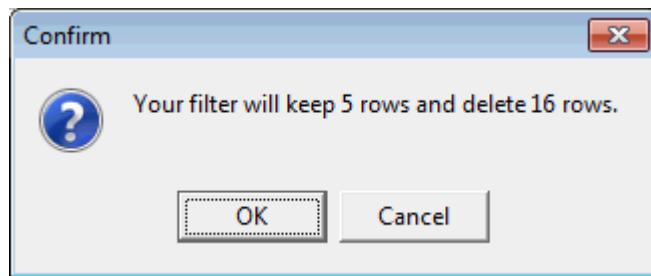
An item under the **Columns** menu allows you to filter rows, by either keeping or deleting, only those rows that contain a particular string, or substring, within a given column. As an example of the use of this feature, consider the following **CSV** example file:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
>	Title	First Name	Middle Name	Last Name	Street Address	City	State	ZIP Code
1	Mrs.	Alma	Ruth	Brantley	1109 Sunset Blvd.	Los Angeles	CA	90021
2		Jay	Norton	Bratton	23234 9th St.	Los Angeles	CA	92323
3		Jeff	Bruce	Bremer	1212 Nineteenth Street	San Diego	CA	93432
4	Mr.	William	Charles	Brendon	6938 West 77th Street	New York	NY	10014
5		Andrea	Janice	Chasin	2424 Culver Blvd.	Chicago	IL	21121
6		Cindy	Cayla	Chou	3717 Bagley Ave.	Santa Monica	CA	90401
7		Dorothy	Ellie	Cummings	1919 Broadway Blvd.	Chicago	IL	23231
8	Mr.	John	Frederick	Doe	12345 Any Street	New York	NY	10001
9	Mrs.	Mary	Jane	Doe	12345 Any Street	New York	NY	10001
10		Brian	Jon	Foudy	2024 Holt Ave.	Los Angeles	CA	93458
11		Gilbert	Lyle	Garcia	12120 Washington Place	San Gabriel	CA	90032
12		Jose	Ferro	Garcia	11921 Malibu St.	Santa Monica	CA	90410
13	Dr.	Layla	Kay	Gelf	3333 Ventura Blvd.	Encino	CA	93421
14		John	Jeff	Hollister	2790 Club Drive	Chicago	IL	22213
15		Robert	Allan	Humphrey	1407 Brockton Ave.	Riverside	CA	93481
16	Mr.	Joe	Brian	Johnson	54321 Elm Street	Salt Lake City	UT	84119
17	Mr.	Donald	Bob	Langston	7418 89th Avenue	Chicago	IL	23121
18		Peter	Scott	Lucas	3576 Ocean Ave.	St. Louis	MO	42123
19		Jean	Claudia	Mackey	717 Manchester	Salt Lake City	UT	81143
20		Steve	Terry	Madden	1337 Saltair Ave.	Los Angeles	CA	98743
21	Dr.	Francis	Benjamin	Madril	234 Sherman Way	Santa Monica	CA	90342
22		Pedro	Alan	Martinez	1905 Capri Drive	Santa Monica	CA	90532
23	Mr.	Michael	Don	Morley	219 Banyon Street	Santa Monica	CA	92323
24		Peter	Michael	Moy	1700 Decker Street	Chicago	IL	22212
25	Mrs.	Verna	Ethel	Scott	2323 Pier Street	Long Beach	CA	91122
26	Mrs.	Nancy	Fran	Silton	2616 Victoria Ave.	Chicago	IL	21321
27	Mr.	David	John	Simons	2616 Venice Ave.	San Francisco	CA	93413
28		John	Jay	Stern	2221 Hilltree Street	Los Angeles	CA	90342
29		Andrew	Richard	Vogel	2417 Hill Street	San Mateo	CA	90121

We first click on column 5 to select it and then we choose the **Columns | Filter On a Column** menu item. This brings up the following dialog box in which **The Column Number** is already 5 because we previously clicked on column 5 before we chose the **Filter On a Column** menu item. Whenever we are going to operate using a particular column, we always click on some entry in that column, or we **CTRL** click the column header title number to automatically select all rows in that column. In this case we are selecting rows 1 through 21 inclusive as the **Applicable Row Range**. Note however the full grid contains 29 rows.



We make **The Filter Substring** the three letters "str". We chose the case sensitivity option as **CASE INSENSITIVE**. Then when we click the **Ok** button we will see the following confirmation dialog box.



For this example, the numbers apply to the 21 rows we selected, even though the original grid contained 29 total rows.

In this example when we click the **OK** button to continue we will see the resulting 5 rows that are kept by our special filter. If we determined that our filter was deleting too many rows we would click the **Cancel** button instead, and the entire filter operation would be aborted without making any changes to the current grid. Otherwise, when we click the **OK** button, the filter is actually applied.

Also note that the complementary result can be obtained by changing the radio button for **How To Apply The Filter**. In this example we are only keeping the rows that contain the filter as a substring. We could just as easily change this to delete the rows that match the filter and keep the rows that Do NOT match. Thus to obtain a complementary or exactly opposite result is easy. In any case, the following shows the resulting grid contents for this example. The first 5 rows are the 5 rows kept by the filter, but rows 6 through 13 are rows that were not involved in our row selection, so these 8 other rows were moved up from the bottom of the grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
>	Title	First Name	Middle Name	Last Name	Street Address	City	State	ZIP Code
1		Jeff	Bruce	Bremer	1212 Nineteenth Street	San Diego	CA	93432
2	Mr.	William	Charles	Brendon	6938 West 77th Street	New York	NY	10014
3	Mr.	John	Frederick	Doe	12345 Any Street	New York	NY	10001
4	Mrs.	Mary	Jane	Doe	12345 Any Street	New York	NY	10001
5	Mr.	Joe	Brian	Johnson	54321 Elm Street	Salt Lake City	UT	84119
6		Pedro	Alan	Martinez	1905 Capri Drive	Santa Monica	CA	90532
7	Mr.	Michael	Don	Morley	219 Banyon Street	Santa Monica	CA	92323
8		Peter	Michael	Moy	1700 Decker Street	Chicago	IL	22212
9	Mrs.	Verna	Ethel	Scott	2323 Pier Street	Long Beach	CA	91122
10	Mrs.	Nancy	Fran	Silton	2616 Victoria Ave.	Chicago	IL	21321
11	Mr.	David	John	Simons	2616 Venice Ave.	San Francisco	CA	93413
12		John	Jay	Stern	2221 Hilltree Street	Los Angeles	CA	90342
13		Andrew	Richard	Vogel	2417 Hill Street	San Mateo	CA	90121

Note that only those five rows that contain the word "Street" in column 5 have been kept. All other selected rows have been deleted.

Our filter was "str" and because we chose the case sensitivity as **CASE INSENSITIVE**, the filter matched all rows that contained the word "Street".

If no rows are found that match your search criterion, then the program will warn you that no matching rows were found. If your filter would have the effect of deleting all existing rows in the entire grid you will generate a warning to that effect and you will be able to cancel the filter operation and preserve all your data.

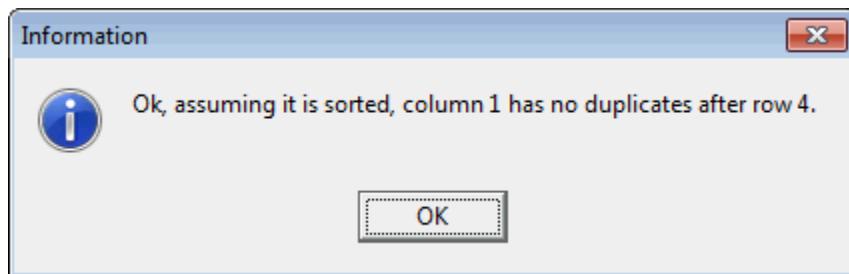
When **The Filter Substring** is the empty string, then we have to carefully consider how we handle the cases of when the compared cell is also empty or not, but depending on whether we want to keep the row or filter it out. The empty string is always considered a substring of another empty string but the empty string is never a substring of a non-empty string. A nonempty string is never a substring of an empty string.

Applying a filter in a column can also be accomplished by clicking the toolbar button .

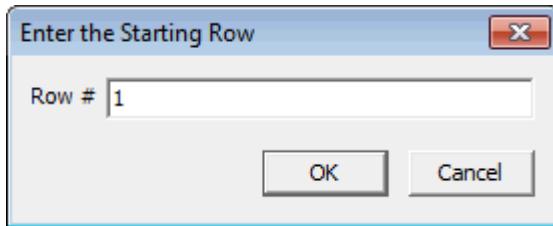
Another function that can be used to determine certain rows is under the **Rows** menu. That function is called **Export Matching Key Rows...** and is an alternative to using a string filter as described in this current help topic. Yet another function is under the **Rows | Filter Rows With Matching Cells...** menu that filters any group of rows based on comparing cells in two columns of the grid, where the cells are in the same row.

Finding Duplicates In A Column

When you try to find duplicates in a column, the program assumes you previously sorted that column. Thus you should try sorting a column just before you look for duplicates. Because sorting can be time-consuming, the program does NOT automatically sort a column before applying the duplicate search function, unless you set a program option to make it do this. Thus you might need to sort the search column yourself. However, when the program does NOT find any duplicates, it will give you a message like the following that is reminding you that it assumes the search column was already sorted.



In any case, when you select the menu item **Find Duplicate Entries in a Column...**, or press **CTRL+D** on the keyboard, you will first be prompted by a simple dialog that asks you to enter the starting row number for where the search is to start.



You don't always have to start in Row # 1, but after a duplicate is found, when you try to find subsequent duplicates in that same column, this dialog will automatically fill in a suggested row number for you that should be the desired starting row, starting from the last duplicate found row.

If the program finds duplicate entries within the search column, it will flash the message

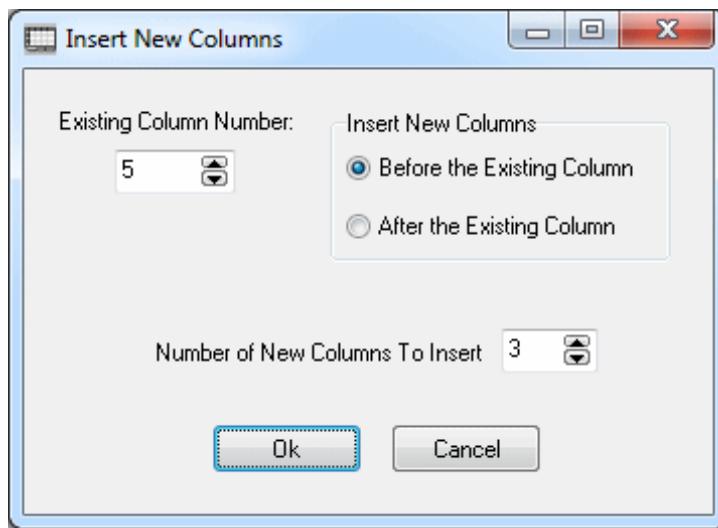


in the middle of the screen. It will also place the cursor/highlight on the row containing the duplicate. You need to manually inspect the column entry in the row above the highlighted cursor row. You should then see at least two consecutive same entries. Just remember to check the row above the cursor whenever the duplicate check function flashes the above message.

A different function under the **Uniqueness** menu can be used to find duplicate cells in any block of cells. In this special case, no form of sorting will be necessary. The block you select might consist of a single column.

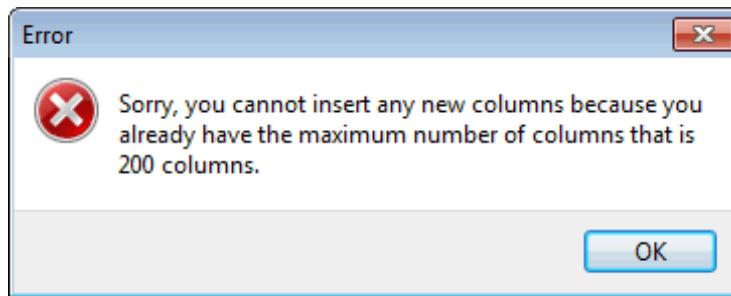
Inserting New Columns

Whenever you need to insert new columns, you can select the menu item **Columns | Insert New Columns...** or you can just click the button  in the toolbar. You will be prompted by a dialog in which you can select the existing column and select whether the insertion is before or after that existing column, and finally you can tell how many new columns you want to insert.



When you click the **Ok** button the program will create the new columns and fill them with blank entries. All columns will get renumbered as needed.

The above description assumes your grid still has room to add some new columns. If your grid already has the maximum number of columns then you will see the message:



Moving Columns

There are at least two simple ways in which you can effectively move a column left or right, any number of column positions. First, there is a menu item under the **Columns** menu that allows you to **Exchange Two Columns**. If you want to move a given column only a few positions away from its current column position, then you can accomplish that by simply making a series of adjacent column exchanges. If you need to move a column more than a few columns away from its current position, then another technique is to use a column insert followed by an exchange, followed by a column delete. Exactly how you do this is described next.

As an example, suppose you have a grid with 20 columns and you want to move column 17 left so that it is essentially inserted between existing columns 5 and 6. Click the cursor anywhere in column 5 and then click the toolbar button to insert a new column. Insert the new column just after column 5.

You will have a new blank column 6, and what was previously column 17 has just moved right by one column to become a new column 18. Now choose the **Columns** menu item **Exchange Two Columns**. When the dialog appears, make the first column to exchange column 6 and make the second column to exchange column 18. Click **Ok** to exchange those two columns. Then click anywhere in column 18 and click the toolbar button to delete a column. You should be prompted to delete column 18 which is the empty column.

In terms of what was the original grid, we can summarize what the changes are. First, the original columns 1 to 5 will not have changed. The new column 6 is what was the original column 17. The new columns 7 through 17 are the original columns 6 through 16 as if they all moved right by one position. The new columns 18 through 20 are the same as the original columns 18 through 20.

Another menu item allows you to move the current column all the way to the left or all the way to the right of the current grid. Just open **Columns | Move Current Column ▶** and then further select to move **To the Extreme Left** or **To the Extreme Right**. These operations move the current column to **Column 1** (Extreme Left) or to the last or rightmost column of the current grid (Extreme Right).

Move Current Column to the Extreme Left or to the Extreme Right

Selecting the menu item **Columns | Move Current Column ►** gives you a two-submenu choice as to position:



You can move the current column to the extreme left of the grid or you can move it to the extreme right of the grid. No prompting is needed for this operation. You can immediately undo this by clicking the **Undo** button in the toolbar.

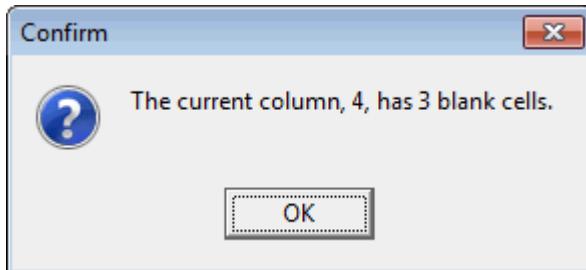
This action is often used when you need to move a column all the way left or right to either get it out of the way or to move it where you can work on it. Of course any criteria can be applied to place a set of columns at the extreme left and right sides of the grid. Here Extreme Left means the first column and Extreme Right means the last column.

This function will probably be used most often when you have a large number of columns that do not display all at once on the screen, and you would like to move a few of those columns so they become adjacent columns that can all be worked on and seen together. You would select each column in turn and move them using the same extreme direction.

Report Blank Cells in the Current Column

You can select **Columns | Report Blank Cells in the Current Column...** to have the program count and report the number of blank cells that are in the current column. This function is often used whenever you are performing validation in a column and you aren't sure how you want to handle blank entries.

A typical report might appear like the following:

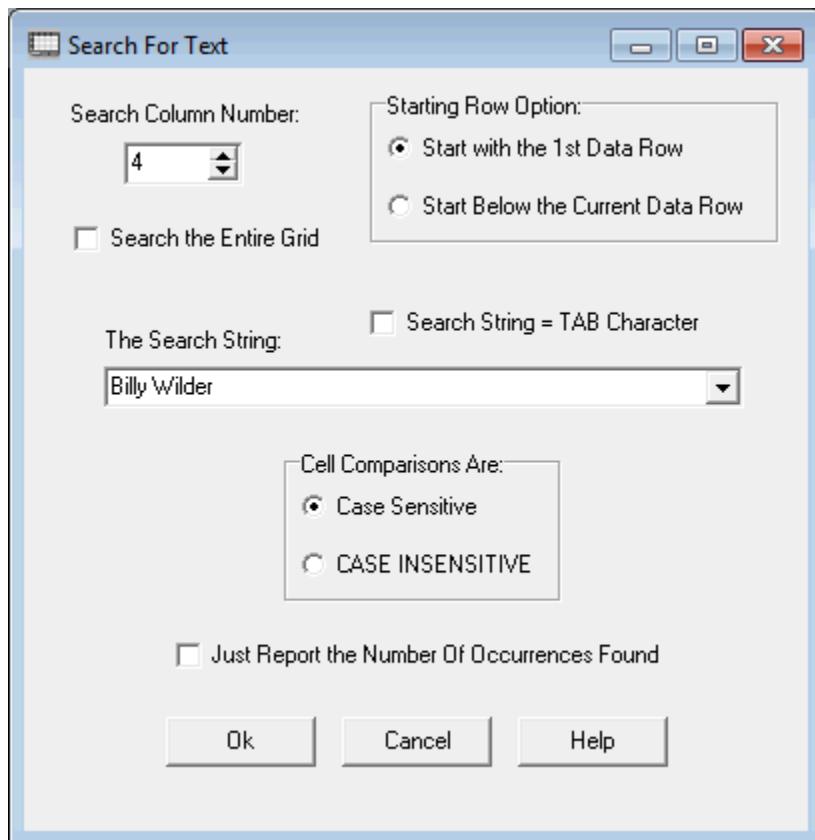


Before applying this function you should click on any cell in the column whose entries you want to check; this will select that column as the current column.

Searching For Text

The **Columns** menu functions pertain to operations on columns. In general, we recommend you click on the column you want to operate on before you select one of these submenu items or before you use an equivalent button in the toolbar. By clicking on a column before you operate on that column you will cause the program to automatically select that column number in any dialog that gives you the opportunity to change the column number of the column to be worked on. This is especially true before sorting or doing a search or a search and replace operation. It can also be helpful to first select a block of columns if you know you want to operate on multiple columns.

You should note one special distinction between plain searching, and searching and replacing text. If you are just searching for text, then you have the option to **Search the Entire Grid**. Sometimes this is useful when you have opened a very large **CSV** file and you aren't sure which particular column to search in. However, after you find the first text occurrence, you may then want to make further searches within one column only. The **Find Next** toolbar button only searches within the current column, regardless of the state of the **Search Entire Grid** checkbox. After you find the search text, you can click on another column in the grid and continue by clicking the **Find Next** button in the toolbar without having to reopen the **Search For Text** dialog just to change the search column to another column. Whatever column is the currently active column is used with the **Find Next** button.

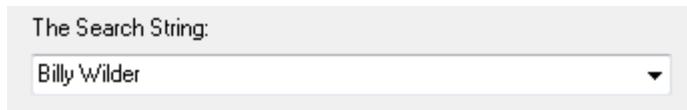


However, when **Searching And Replacing** text you don't have the option to work with the entire grid at once. So searching and replacing text is more restrictive than just searching for text. You may often find that you only want to search or search and replace within a single column anyway. By default, the **Search the Entire Grid** checkbox is NOT checked in the **Search For Text** dialog.

Note that if you need to find TAB characters only, then you need only turn on the check box for that option. The program only reports one occurrence per cell. In other words, the program does not try to find all occurrences within a single cell when more than one exists within a single cell. In fact, this statement is true of all searches and all search and replacements. Only the first occurrence of the search string within a cell is used and/or counted. If you know your cells have multiple occurrences, then you may need to run the search or search and replace function multiple times for the same column or the same block.

Another option with searching is to **Just Report the Number Of Occurrences Found** that matched the search criterion. As an example, if one of the columns in your grid contained 2-letter state abbreviations, and if you searched for the string CA, you should see a report telling how many rows there were of people in California.

The **Search For Text** dialog contains a drop-down list box that holds all the strings you may have tried in previous searches.



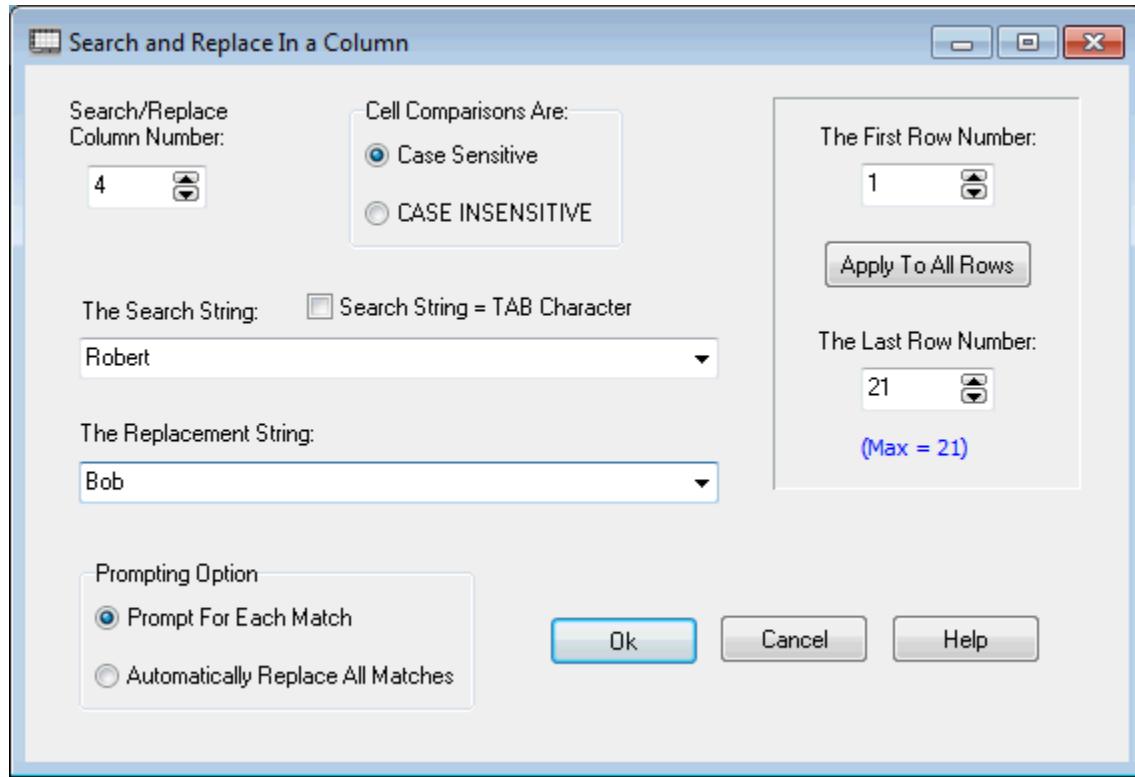
This means you don't have to re-type a previous search string, if you just select it from the history list of previous searches. Just click the down arrow ▼ that is on the far right of **The Search String** edit box, to open up the history list.

The Search String can be the empty string, in which case a match is made only for those cells that are also the empty string.

An alternative to using this main search function is to open an **Auto Search Window**, whose functionality is described in this help file as part of the functions under the **View** menu. The **Auto Search Window** provides a series of alphanumeric buttons that you can click to quickly find the first cell in a column whose first character matches the button you pressed. See also the **View** menu near the end of this Help file.

Searching And Replacing Text

The **Search and Replace In a Column** dialog can be brought up by choosing **Columns | Search And Replace In a Column...**. You should then see a dialog similar to the following.

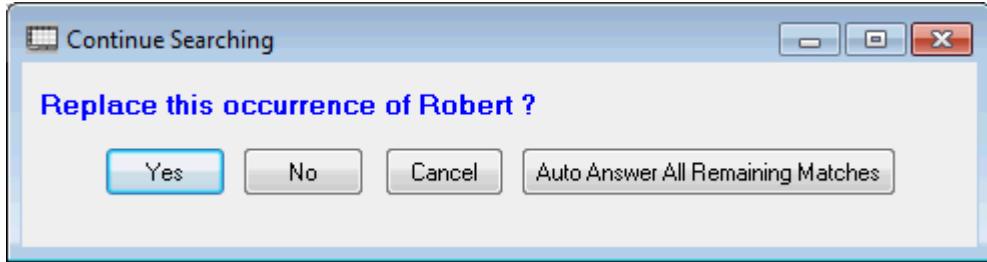


Note how this dialog differs from the **Search For Text** dialog because this dialog allows you to set a specific range of rows that can limit the possible search and replace range of rows. Also, this dialog does NOT have an option to work across the entire grid. You must make all replacements within a single column only.

The search type can be made **Case Sensitive** or **CASE INSENSITIVE** and you have to the option to prompt for each replacement occurrence or you can make all replacements automatically.

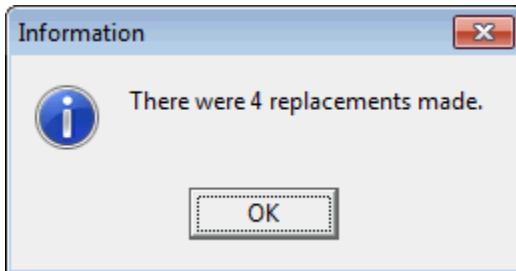
Even if you choose to prompt for each replacement, during each prompt, you will have a chance to continue and make all remaining replacements automatically, or you will be able to cancel any remaining replacements. The next page shows what a typical replacement prompt looks like. Of course, at each prompt you can click the **Yes** or **No** button that only applies to the current replacement.

An example replacement prompt might appear like:



In this case, clicking the button **Auto Answer All Remaining Matches** means to answer **Yes** to all remaining matches that are found without the benefit of seeing any of those matches. Pressing **Cancel** will abort the function immediately. Otherwise pressing **Yes** or **No** will process your click and the program will continue searching for other matches until no more matches are found.

After all replacements have been made you should see a summary message similar the following.



Either **The Search String** or **The Replacement String** can be the empty string. When searching, a match is found only for those cells that are also the empty string. When **The Search String** is not empty, but **The Replacement String** is empty, then the effect of a search and replace is to simply remove **The Search String** part from each matching cell.

Another thing to note is that when this function is performed, only the first match within a cell gets replaced. As an example, when **The Search String** is the single letter e and when **The Replacement String** consists of the two letters ZZ, then when a cell containing the name Katherine gets processed, the result will look like KathZZrine and it will not look like KathZZrinZZ. So there is only one replacement made per cell.

A function similar to this Search and Replace function is one named a Search and Match function. To learn about this other related function see also [Columns | Two-Column Search and Match....](#)

Sorting

The **CSV Editor** program provides three different types of sorting functions under the **Columns** menu. Under the hood, all sorting is done using an efficient implementation of the QuickSort algorithm. That algorithm defaults to an Insertion Sort when there are 16 or fewer elements to be sorted. We provide for both single-column and multiple-column sorting for grids, as well as what we call in-place sorting for single-column blocks that only affects the cells (not the grid rows) in the single-column block.

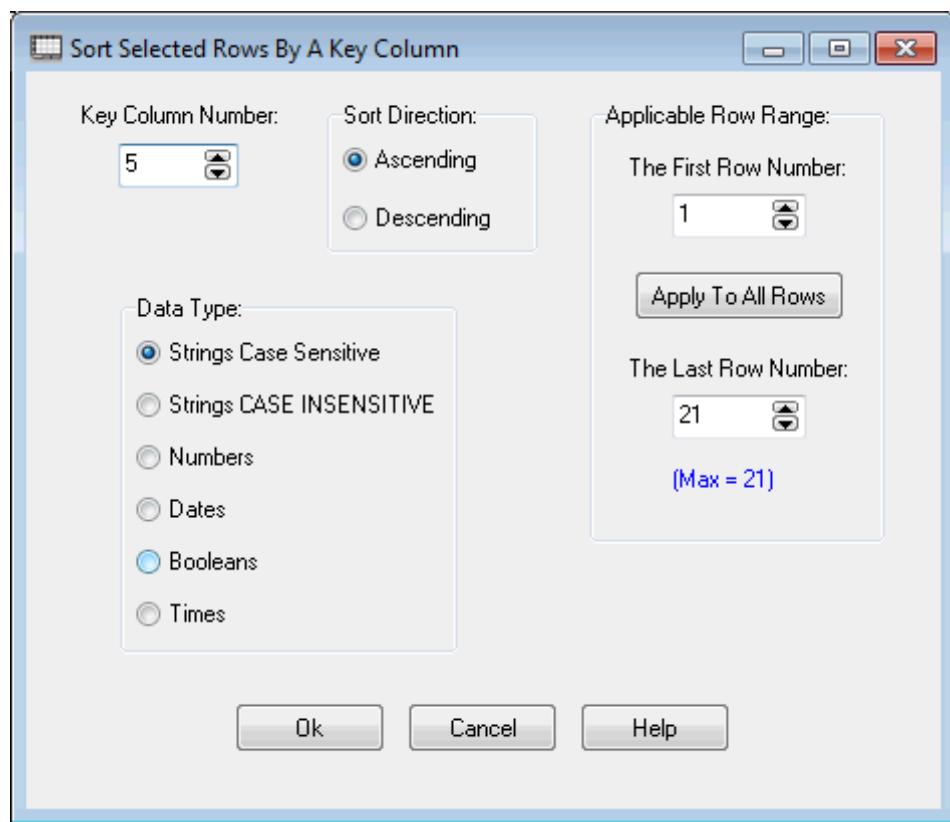
The most common type of sort has the title **Sort Selected Rows By a Key Column....**. For this type of sort, only the rows within the selected range of rows change, and even then, those rows are simply re-ordered so the entries in the key column appear sorted when you read up or down the key column within those rows. This same sorting function can also be executed by using the sort button in the toolbar that appears as: .

To see an example of this type of sort, below we show the AcademyAwards.csv file that we use for many examples in this help file. We have selected all the entries in Column 5.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

All the films in the above grid are sorted by Column 1, the year in which the films were made. If we wanted to sort the films by a different column, say by the **Best Picture** column, we could first **CTRL** click on the **Column 5** header to select all rows in that column. Then we would click the toolbar sort button , or we would select the **Columns** menu item with the caption **Sort Selected Rows By a Key Column....**.

Either way we would bring up the following dialog box whose caption indicates we are sorting using a Key Column:



In the above dialog we could select a range of row numbers to define the rows that are to be sorted. All other rows would remain unchanged.

For this example we would click the button with the caption **Apply To All Rows** and then we would click the **Ok** button to actually perform the sort operation.

The result should look like the grid shown on the next page. The entries in Column 5 are sorted in ascending alphabetical order. Note the year numbers in Column 1 are no longer sorted and appear as if in a somewhat random order.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
2	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
3	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
4	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
5	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
6	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
7	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
8	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
9	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
10	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
13	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
14	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
15	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
16	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
17	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
18	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
19	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
20	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
21	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T

If you compare this list with that shown on two pages previous, you can see that all the rows in the grid have been sorted, such that the entries in Column 5 (also known as the key column), the **Best Picture** titles, have been put in ascending alphabetical order.

If you choose a **Data Type** that assumes the cells contain anything other than **Strings** (i.e., **Numbers**, **Dates**, **Booleans**, or **Times**), then before any sorting takes place, the program will first verify that every cell in the block can be converted to a valid numeric value. In particular, **Numbers** will handle US currency values by first temporarily removing any and all \$ and commas and spaces. If any cells are found to contain invalid data, then you will see an error message telling you which cell could not be converted to a valid numeric value. You should manually edit the text in the cell before trying to sort the block again.

Empty strings in cells will automatically be converted to special numeric values for sorting comparison purposes, without any warning or error messages being given for such cells. In other words, an empty cell is tolerated in the sorting process, but no empty cell is actually permanently converted to a special numeric value, so the same cell remains empty when the sort function finishes. All empty cells will appear mixed together in the sort positions where other special numeric values would otherwise appear.

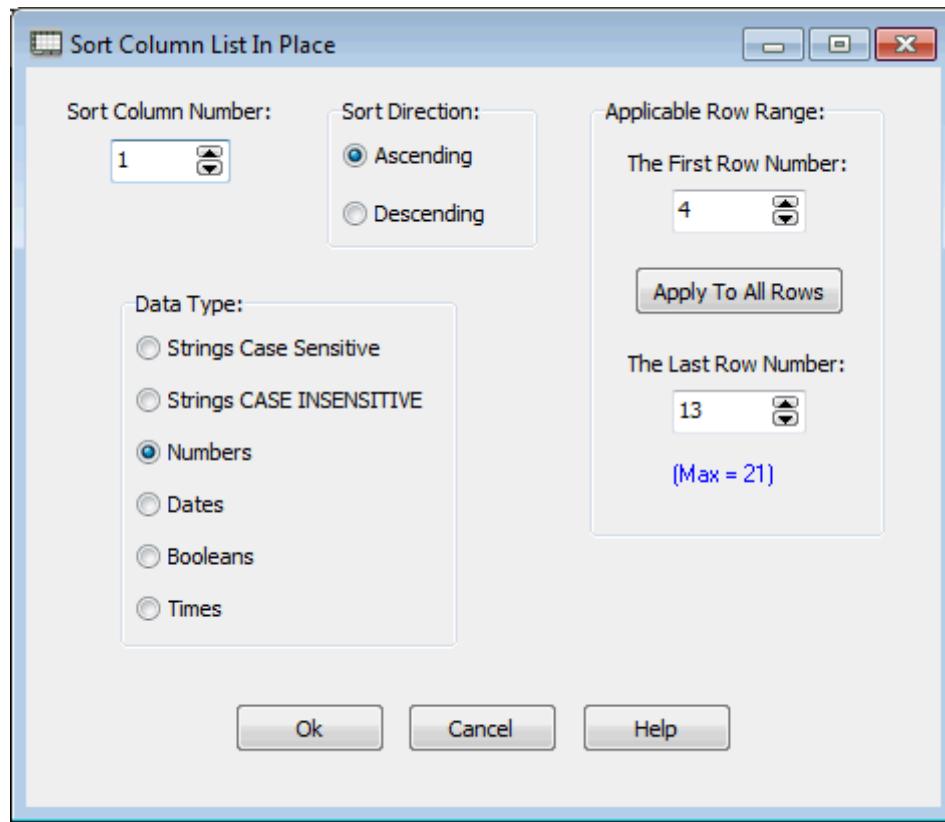
For **Numbers**, the special value for an empty cell is the number **0**. For **Dates**, the special value for an empty cell is the date **01/01/0000**. For **Booleans**, the special value for an empty cell is the **0** boolean. For **Times**, the special value for an empty cell is the time **00:00:00**.

Sorting In Place

A special kind of sort is available under the **Columns** menu with the caption **Sort In Place (Column Sublist)**.... To understand this type of sort we will select rows 4 through 13 in the grid that is the Academy Awards grid after we have first sorted that grid with Column 5, the Best Picture names. We select only within Column 1. The selected cells make a little sublist within a single column. We only show a partial grid below and on the next page.

2	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
3	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
4	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
5	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
6	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
7	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
8	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
9	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
10	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
13	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
14	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
15	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
16	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T

With the entries selected in a single column, we can sort this selected little block *in place*. What we mean when we say *in place*, is that only the entries in the selected block will be sorted, and as they are sorted, no other row information will change. If you select the **Columns** menu item **Sort In Place (Column Sublist)**... you will see the following dialog in which we have chosen the **Data Type** radio button with the title **Numbers**. Note the title for this dialog says **Sort Column List In Place**.



Now when we click the **Ok** button we will see the result of this special *in place* sort.

2	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
3	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
4	1960	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
5	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
6	1964	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
7	1968	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
8	1969	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
9	1970	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
10	1975	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T
11	1976	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1979	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
13	1980	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
14	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
15	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
16	1972					

For this example, you should note that the year numbers in Column 1 for the selected cells are now sorted in ascending order. Only these selected cells have changed from the previous grid picture.

While this result may not make sense because these year numbers now don't match the actors or picture title rows for the corresponding years, we need only understand that when you sort entries in a column *in place*, only the selected cells change to become sorted. None of the other row information changes with this type of sort. This is very different from the type of sort where the information in each entire row moved when the corresponding row is moved to a new row position.

While it may be rare to sort *in place* like this, it is a nice feature to have when editing a **CSV** string grid.

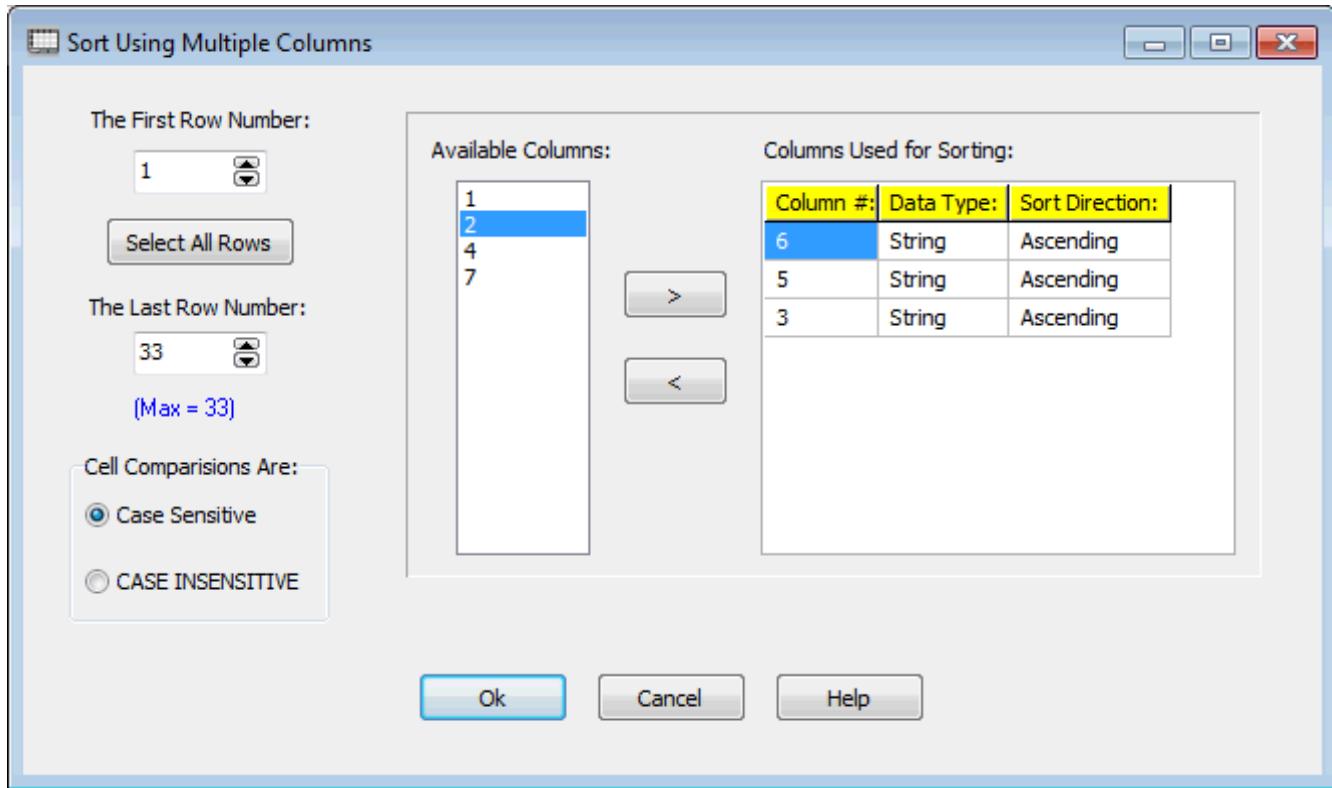
If you choose a **Data Type** that assumes the cells contain anything other than **Strings** (i.e., **Numbers**, **Dates**, **Booleans**, or **Times**), then before any sorting takes place, the program will first verify that every cell in the block can be converted to a valid numerical value. In particular, **Numbers** will handle US currency values by first temporarily removing any and all \$ and commas and spaces. If any cells are found to contain invalid data, then you will see an error message telling you which cell could not be converted to a valid numeric value. You should manually edit the text in the cell before trying to sort the block again.

Empty strings in cells will automatically be converted to special numeric values for sorting comparison purposes, without any warning or error messages being given for such cells. In other words, an empty cell is tolerated in the sorting process, but no empty cell is actually permanently converted to a special value, so the same cell remains empty when the sort function finishes. All empty cells will appear mixed together in the sort positions where other special numeric values would otherwise appear.

For **Numbers**, the special value for an empty cell is the number **0**. For **Dates**, the special value for an empty cell is the date **1/1/0000**. For **Booleans**, the special value for an empty cell is the **0** boolean. For **Times**, the special value for an empty cell is the time **00:00:00**.

Sorting Using Multiple Columns

There is a third sorting function under the **Columns** menu that has the title **Sort Using Multiple Columns....**. When you select this menu item you should see a dialog that appears like that shown below.



Using this dialog you can first select a range of rows to be used in the sorting process. This means you can apply multiple column sorting using any subset of consecutive rows. A standard option is to select the case sensitive choice that determines how the **Cell Comparisons** are made for **String** data. There are two lists on the right. Initially the list with the title **Columns Used for Sorting** will be empty and the **Available Columns** list will contain the column numbers for all the columns in your grid. In the above figure pay particular attention to the rows in these grids that contain a blue highlight. In the above example we have already moved available columns 6, 5, and 3, in that order, into the **Columns Used for Sorting** list.

You use the two buttons to move a column number from one list to the other. The first aspect to learn about using these buttons is that you must always first click on a row in either grid to highlight an item row in blue before you can move the corresponding column number from one list to the other list.

When you first select a column number from the **Available Columns** list and then click the right-arrow button, then that number will be removed from the **Available Columns** list and it will be added to the **Columns Used for Sorting** list. The left-arrow button performs the opposite function, removing a column number from the **Columns Used for Sorting** list and putting it back in the **Available Columns** list. Taken together, the two lists comprise all column numbers. A given column number must appear in exactly one of these two lists at any time. In other words, a given column number is either used, or not.

For each row in the **Columns Used for Sorting** list, you have a choice of making the **Data Type** either **String** or **Numeric** or **Date** or **Boolean** or **Time**. The **Case Sensitive** option only applies to all **String** data and is otherwise ignored. You have the choice of making the **Sort Direction** either **Ascending** or **Descending**. To change any of these choices, just click on the cell that you want to change and the description in that cell will automatically switch to the next available description. To give an example of how and why multiple column sorting is used, consider the following grid that contains a list of names and addresses.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
>	Title:	First Name:	Last Name:	Street Address:	City:	State:	ZIP:
1	President	George	Washington	1600 Pennsylvania Avenue NW	Washington	DC	20500
2	Mr.	Joe	Johnson	54321 Elm Street	Salt Lake City	UT	84119
3		Baseball	Hall of Fame	P.O. Box 590	Cooperstown	NY	13326
4	Mr.	John	Doe	12345 Any Street	New York	NY	10001
5	Mrs.	Mary	Doe	12345 Any Street	New York	NY	10001
6	Mrs.	Alma	Brantley	1109 Sunset Blvd.	Los Angeles	CA	90021
7	Mr.	William	Brendon	6938 West 77th Street	New York	NY	10014
8		Cindy	Chou	3717 Bagley Ave.	Santa Monica	CA	90401
9	Mr.	David	Simons	2616 Venice Ave.	San Francisco	CA	93413
10		Andrew	Vogel	2417 Hill Street	San Mateo	CA	90121
11	Mrs.	Verna	Scott	2323 Pier Street	Long Beach	CA	91122
12	Mr.	Donald	Langston	7418 89th Avenue	Chicago	IL	23121
13		Peter	Lucas	3576 Ocean Ave.	St. Louis	MO	42123
14		John	Hollister	2790 Club Drive	Chicago	IL	22213
15	Dr.	Layla	Gelf	3333 Ventura Blvd.	Encino	CA	93421
16		Gilbert	Garcia	12120 Washington Place	San Gabriel	CA	90032
17		Dorothy	Cummings	1919 Broadway Blvd.	Chicago	IL	23231
18		Jeff	Bremer	1212 Nineteenth Street	San Diego	CA	93432
19		Jay	Bratton	23234 9th St.	Los Angeles	CA	92323
20		Andrea	Chasin	2424 Culver Blvd.	Chicago	IL	21121
21		Peter	Moy	1700 Decker Street	Chicago	IL	22212
22		Jose	Garcia	11921 Malibu St.	Santa Monica	CA	90410
23		John	Stern	2221 Hilltree Street	Los Angeles	CA	90342
24	Mrs.	Nancy	Silton	2616 Victoria Ave.	Chicago	IL	21321
25		Steve	Madden	1337 Saltair Ave.	Los Angeles	CA	98743
26	Mr.	Michael	Morley	219 Banyon Street	Santa Monica	CA	92323
27		Jean	Mackey	717 Manchester	Salt Lake City	UT	81143
28	Mr.	Benjamin	Whitlock	942 Beaver Road	Los Angeles	CA	92398
29	Mrs.	Alison	Zune	823 Century Blvd.	Chicago	IL	24321
30		Pedro	Martinez	1905 Capri Drive	Santa Monica	CA	90532
31		Robert	Humphrey	1407 Brockton Ave.	Riverside	CA	93481
32		Brian	Foudy	2024 Holt Ave.	Los Angeles	CA	93458
33	Dr.	Francis	Madril	234 Sherman Way	Santa Monica	CA	90342

If we choose **Columns | Sort Using Multiple Columns...** and set the controls as shown in the dialog on the previous page, then when we click the **Ok** button in that dialog, the grid will get sorted using first the state column and then the city column and finally the last name column and the result will appear as shown on the next page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
>	Title:	First Name:	Last Name:	Street Address:	City:	State:	ZIP:
1	Dr.	Layla	Gelf	3333 Ventura Blvd.	Encino	CA	93421
2	Mrs.	Verna	Scott	2323 Pier Street	Long Beach	CA	91122
3	Mrs.	Alma	Brantley	1109 Sunset Blvd.	Los Angeles	CA	90021
4		Jay	Bratton	23234 9th St.	Los Angeles	CA	92323
5		Brian	Foudy	2024 Holt Ave.	Los Angeles	CA	93458
6		Steve	Madden	1337 Saltair Ave.	Los Angeles	CA	98743
7		John	Stern	2221 Hilltree Street	Los Angeles	CA	90342
8	Mr.	Benjamin	Whitlock	942 Beaver Road	Los Angeles	CA	92398
9		Robert	Humphrey	1407 Brockton Ave.	Riverside	CA	93481
10		Jeff	Bremer	1212 Nineteenth Street	San Diego	CA	93432
11	Mr.	David	Simons	2616 Venice Ave.	San Francisco	CA	93413
12		Gilbert	Garcia	12120 Washington Place	San Gabriel	CA	90032
13		Andrew	Vogel	2417 Hill Street	San Mateo	CA	90121
14		Cindy	Chou	3717 Bagley Ave.	Santa Monica	CA	90401
15		Jose	Garcia	11921 Malibu St.	Santa Monica	CA	90410
16	Dr.	Francis	Madril	234 Sherman Way	Santa Monica	CA	90342
17		Pedro	Martinez	1905 Capri Drive	Santa Monica	CA	90532
18	Mr.	Michael	Morley	219 Banyon Street	Santa Monica	CA	92323
19	President	George	Washington	1600 Pennsylvania Avenue NW	Washington	DC	20500
20		Andrea	Chasin	2424 Culver Blvd.	Chicago	IL	21121
21		Dorothy	Cummings	1919 Broadway Blvd.	Chicago	IL	23231
22		John	Hollister	2790 Club Drive	Chicago	IL	22213
23	Mr.	Donald	Langston	7418 89th Avenue	Chicago	IL	23121
24		Peter	Moy	1700 Decker Street	Chicago	IL	22212
25	Mrs.	Nancy	Silton	2616 Victoria Ave.	Chicago	IL	21321
26	Mrs.	Alison	Zune	823 Century Blvd.	Chicago	IL	24321
27		Peter	Lucas	3576 Ocean Ave.	St. Louis	MO	42123
28		Baseball	Hall of Fame	P.O. Box 590	Cooperstown	NY	13326
29	Mr.	William	Brendon	6938 West 77th Street	New York	NY	10014
30	Mr.	John	Doe	12345 Any Street	New York	NY	10001
31	Mrs.	Mary	Doe	12345 Any Street	New York	NY	10001
32	Mr.	Joe	Johnson	54321 Elm Street	Salt Lake City	UT	84119
33		Jean	Mackey	717 Manchester	Salt Lake City	UT	81143

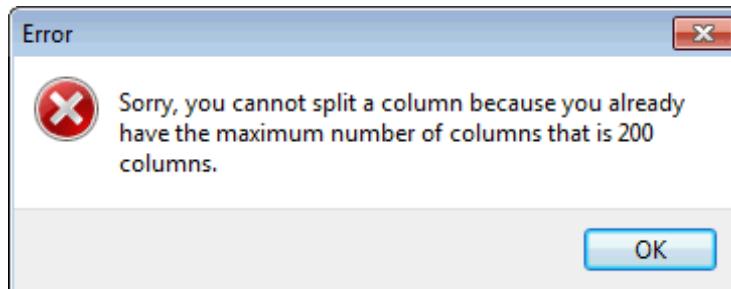
If you study the above grid you will see that **Column 6** is sorted by state, and furthermore, within a given state, the entries are sorted by the city in **Column 5**. Finally note that for cities that appear more than once, the entries are sorted within a city by the last name in **Column 3**. The numbers 6, 5, and 3 were placed in the **Columns Used for Sorting** with this same order from top to bottom. Comparing the previous two grids should help you understand more about how the multiple column sort function works.

Although we have given only one example, this should be sufficient for you to understand how to setup and use a multiple column sort. It would be very rare that you would use more than two or three columns for any multiple column sort. It all depends on how many columns you have and it depends on the nature of your data!

Splitting a Column

There may be times when it is desirable to split a single column into two separate columns. The most common use of this feature would probably occur when a single column contains a list of names where the first and last names are separated by a single space character.

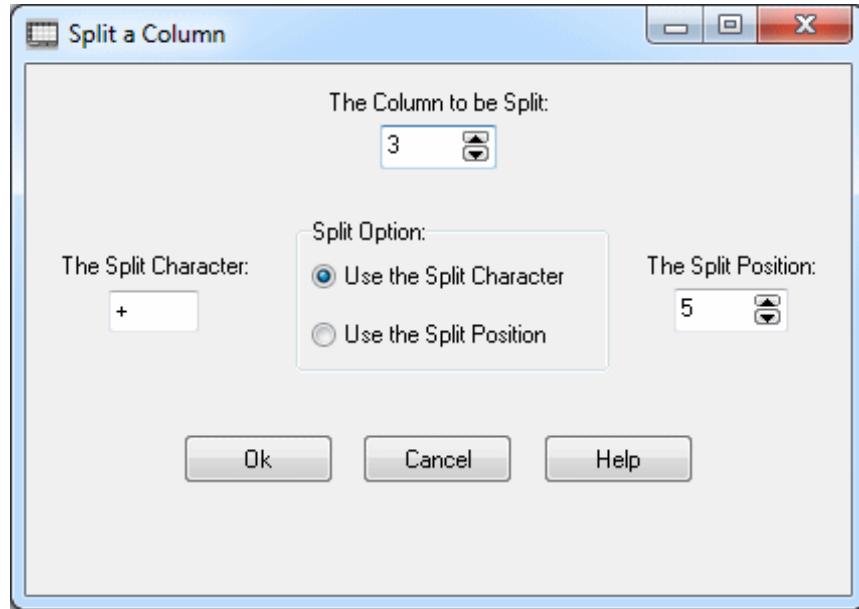
Normally you won't see the following error message when you execute this function, unless your grid has 200 columns.



As a particular example, consider the following grid that holds some Academy Awards data.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Year	Best Actor	Best Actress	Best Director	Film Title
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People

Now look at the Best Actress column that is Column 3. The data in that column consists of first and last names. To split that single column into two columns, first click the mouse in Column 3 and then select the menu item **Columns | Split a Column Into Two Columns...**. You should see the following dialog box.



The column number should already be filled in as column number 3. We are going to enter one space character that is called **The Split Character**. The purpose of **The Split Character** is to determine where to split a given string into two substrings. When a given string gets split, the program will make two substrings. The first substring is all the text that occurs before **The Split Character**, and the second substring is all the text that occurs following **The Split Character**. In fact, **The Split Character** is removed in this process and does not occur in either of the two substrings. When you click the **Ok** button in the above dialog you should see the new grid appear as follows:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best	Actress	Best Director	Film Title
1	1960	Burt Lancaster	Elizabeth	Taylor	Billy Wilder	The Apartment
2	1961	Maxmillian Schell	Sophia	Loren	Robert Wise	West Side Story
3	1962	Gregory Peck	Anne	Bancroft	David Lean	Lawrence of Arabia
4	1963	Sidney Poitier	Patricia	Neal	Tony Richardson	Tom Jones
5	1964	Rex Harrison	Julie	Andrews	George Cukor	My Fair Lady
6	1965	Lee Marvin	Julie	Christie	Robert Wise	The Sound of Music
7	1966	Paul Schofield	Elizabeth	Taylor	Fred Zinnemann	A Man For All Seasons
8	1967	Rod Steiger	Katherine	Hepburn	Mike Nichols	In The Heat Of The Night
9	1968	Cliff Robertson	Katherine	Hepburn	Sir Carol Reed	Oliver
10	1969	John Wayne	Maggie	Smith	John Schlesinger	Midnight Cowboy
11	1970	George C. Scott	Glenda	Jackson	Franklin Schaffner	Patton
12	1971	Gene Hackman	Jane	Fonda	William Friedkin	The French Connection
13	1972	Marlon Brando	Liza	Minelli	Bob Fosse	The Godfather
14	1973	Jack Lemon	Glenda	Jackson	George Roy Hill	The Sting
15	1974	Art Carney	Ellen	Burstyn	Francis Ford Coppola	The Godfather Part II
16	1975	Jack Nicholson	Louise	Fletcher	Milos Forman	One Flew Over The Cuckoos Nest
17	1976	Peter Finch	Faye	Dunaway	John G. Avildsen	Rocky
18	1977	Richard Dreyfuss	Diane	Keaton	Woody Allen	Annie Hall
19	1978	Jon Voight	Jane	Fonda	Michael Cimino	The Deer Hunter
20	1979	Dustin Hoffman	Sally	Field	Robert Benton	Kramer vs Kramer
21	1980	Robert De Niro	Sissy	Spacek	Robert Redford	Ordinary People

Note how the Best Actress names now appear with the first name in Column 3 and the last name in the new Column 4. Even the header information has been split into two parts.

It could happen that in a given row, **The Split Character** does not appear anywhere within the string data. In that case, the first column will end up with the entire string and an empty string will appear in the new column, for that row. It could also happen that **The Split Character** appears more than once in a given string. In that case, only the first appearance of **The Split Character** is used to split the string into two parts.

In general, **The Split Character** can be any character like + or ; or @ or anything else you need. You could even do a **Search and Replace** operation to insert a special character into a column. You might even need to do some manual editing within a column to insert **The Split Character** exactly where you want it.

In practice, you need only use a single character as **The Split Character**. In fact, **The Split Character** can be a string that consists of multiple characters, although that would be rare. In that case, all of **The Split Character** string will disappear when the text gets split into two parts.

If you don't want to use a **Split Character**, you can instead define and use a **Split Position**. The **Split Position** number is simply the number of the character, counting from 1, starting with the leftmost character in the string. When using a **Split Position**, all the characters in the cell get used in some way in the two new columns. If the **Split Position** refers to a position beyond the last character in a cell, then the first new split column will contain all the original cell characters and the second split column cell will be empty.

Summing the Entries In a Column

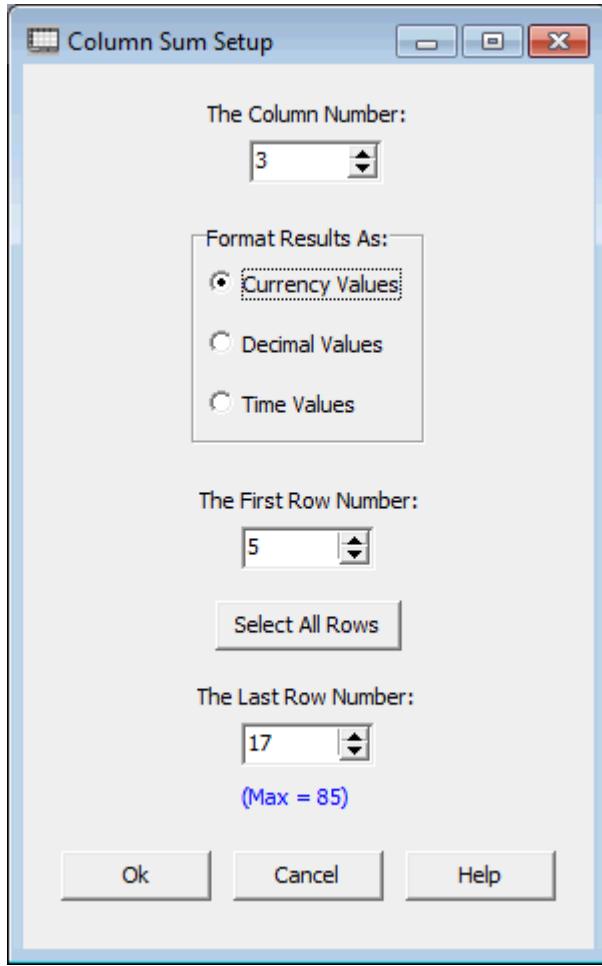
The submenu item to **Sum and Average the Entries In a Column** is used to add numerical values and compute their average. The **CSV Editor** program is not a spreadsheet program, but it does have the ability to work with numeric data. As an example of its numerical capabilities, consider the following **CSV** file. The first column represents dates, the second column represents transaction numbers and the third column represents debits.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	The Date:	The Number:	Debit Amount:	Credit Amount:	Current Balance:
1	01/09/2009	T000621	\$2,850.35		\$6,729.03
2	01/21/2009	T000622	\$205.34		\$4,839.99
3	02/06/2009	T000623	\$3,878.34		\$12,908.57
4	02/10/2009	T000624	\$5,000.00		\$7,908.57
5	03/04/2009	T000625	\$200.00		\$11,534.15
6	03/06/2009	T000627	\$61.84		\$11,472.31
7	03/13/2009	T000626	\$2,961.61		\$7,811.38
8	03/31/2009	T000627	\$200.00		\$5,551.21
9	04/10/2009	T000628	\$1,334.50		\$14,612.35
10	04/13/2009	T000629	\$7,885.00		\$5,762.31
11	04/21/2009	T000630	\$205.34		\$5,525.82
12	05/01/2009	T000631	\$200.00		\$10,571.55
13	05/08/2009	T000632	\$63.90		\$10,448.06
14	05/15/2009	T000633	\$2,991.77		\$5,866.70
15	06/03/2009	T000634	\$200.00		\$10,370.58
16	06/12/2009	T000635	\$3,239.63		\$6,102.55
17	07/01/2009	T000637	\$200.00		\$10,929.74
18	07/06/2009	T000636	\$63.90		\$9,721.46
19	07/15/2009	T000637	\$1,783.45		\$10,030.56
20	07/20/2009	T000639	\$18.09		\$8,899.03

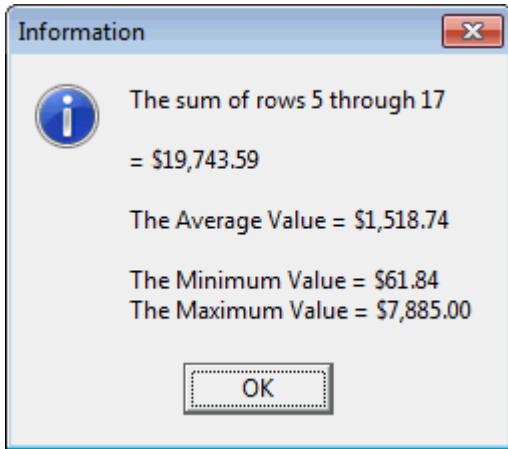
Suppose we want to find the sum of all the debits in rows 5 through 17 inclusive. To do that we would first click on the third column and then we would select the **Columns** submenu item **Sum and Average the Entries In a Column**. That would bring up the following dialog box in which we would set **The First Row Number** to 5 and **The Last Row Number** to 17. We also check the checkbox to **Format Results As Currency Values**.

Incidentally, here is a good place to note that if a cell contains an empty string then that row is skipped in computing both the sum and the average values. If a cell is non-empty, but has a non-numeric value, then you will generate an error message that tells you the entry is invalid. Thus empty cells are tolerated by skipping over them, but non-empty cells within your selected row range must contain valid numbers. The program removes "\$" and "," characters from strings that should otherwise represent valid floating point numeric values.

We setup the next dialog box as:



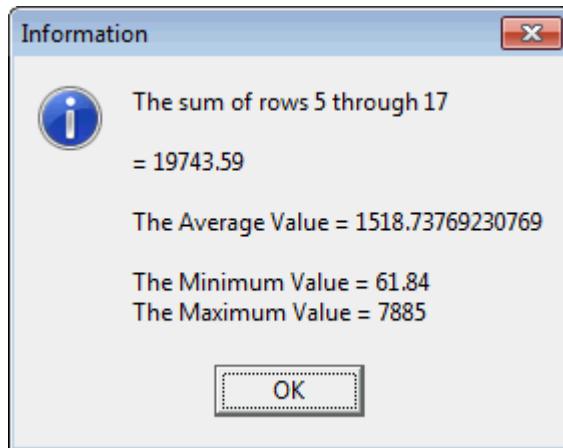
When we click the **Ok** button we see the following message:



In this example, the Average Value has been rounded up to the nearest penny, \$1,518.74, or 74 cents to the nearest penny. We also note the information also shows the minimum and maximum values from the selected rows.

When you compute a column sum, the answer to the sum is saved on the Windows clipboard as a small amount of text. If you want to insert the sum answer into another grid cell, just select that other cell for editing and then type **CTRL+V** on your keyboard to paste the answer into that cell.

If we chose to **Format Results As Decimal Values** we would see the following different message:



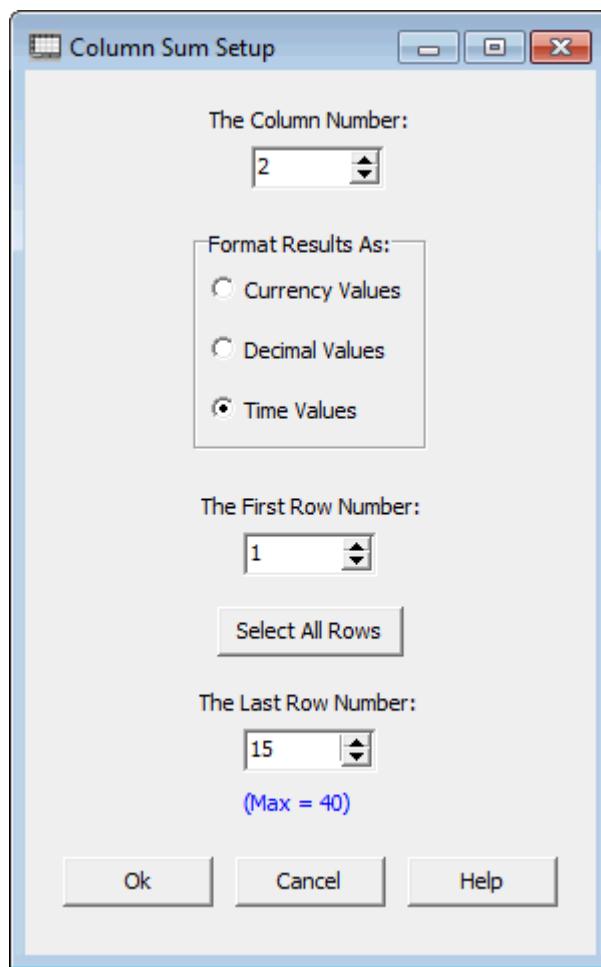
Reading the decimal places allows us to understand why the currency Average Value, \$1518.73769230796, was rounded up to \$1,518.74. Sometimes you want to round currency values and at other times you may have data values that are not related to money, or do not need to be rounded to the nearest hundredth. Just uncheck the checkbox to format results as currency when rounding is not desired.

If you need to perform a more detailed statistical analysis of your data then you should consider the functionality that is available for performing **Linear Regression**. Select **Utilities | Linear Regression Report...** and when the dialog opens click the **Help** button.

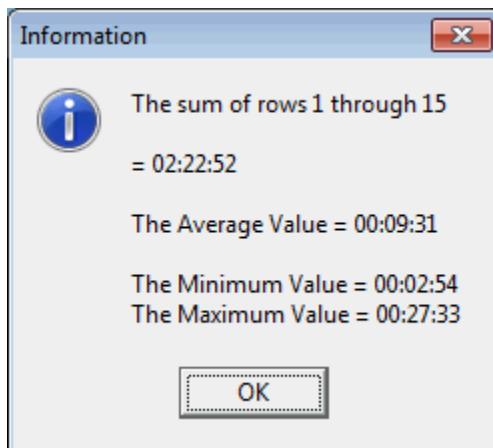
For our final example of computing a sum of values in a column, consider the grid that appears as:

<	Column 1	Column 2	Column 3
>	Video Title:	Video Elapsed Time:	File Size In MB:
1	The Fastest Simplest Way To Start	00:02:54	3.34
2	Introduction	00:07:16	8.24
3	Sheets Templates Keynames	00:11:35	14.40
4	Basic Editing	00:27:33	47.80
5	Line Objects	00:06:22	6.69
6	Rectangles and Ovals	00:09:55	11.1
7	Strings	00:05:41	5.66
8	1D Barcodes	00:10:39	10.80
9	PDF-417 Barcodes	00:13:02	14.10
10	Picture Objects	00:10:07	10.70
11	Block Text Objects	00:08:32	11.60
12	Wave Objects	00:13:45	17.80
13	Arrow Objects	00:04:50	6.45
14	StarPolygon Objects	00:04:21	4.88
15	Quadrilateral Objects	00:06:20	6.64

We select the first 15 rows in this grid and note that Column 2 consists of time values that represent elapsed times for a set of video files. We setup the sum dialog as shown next, in which we choose to **Format Results As Time Values**.



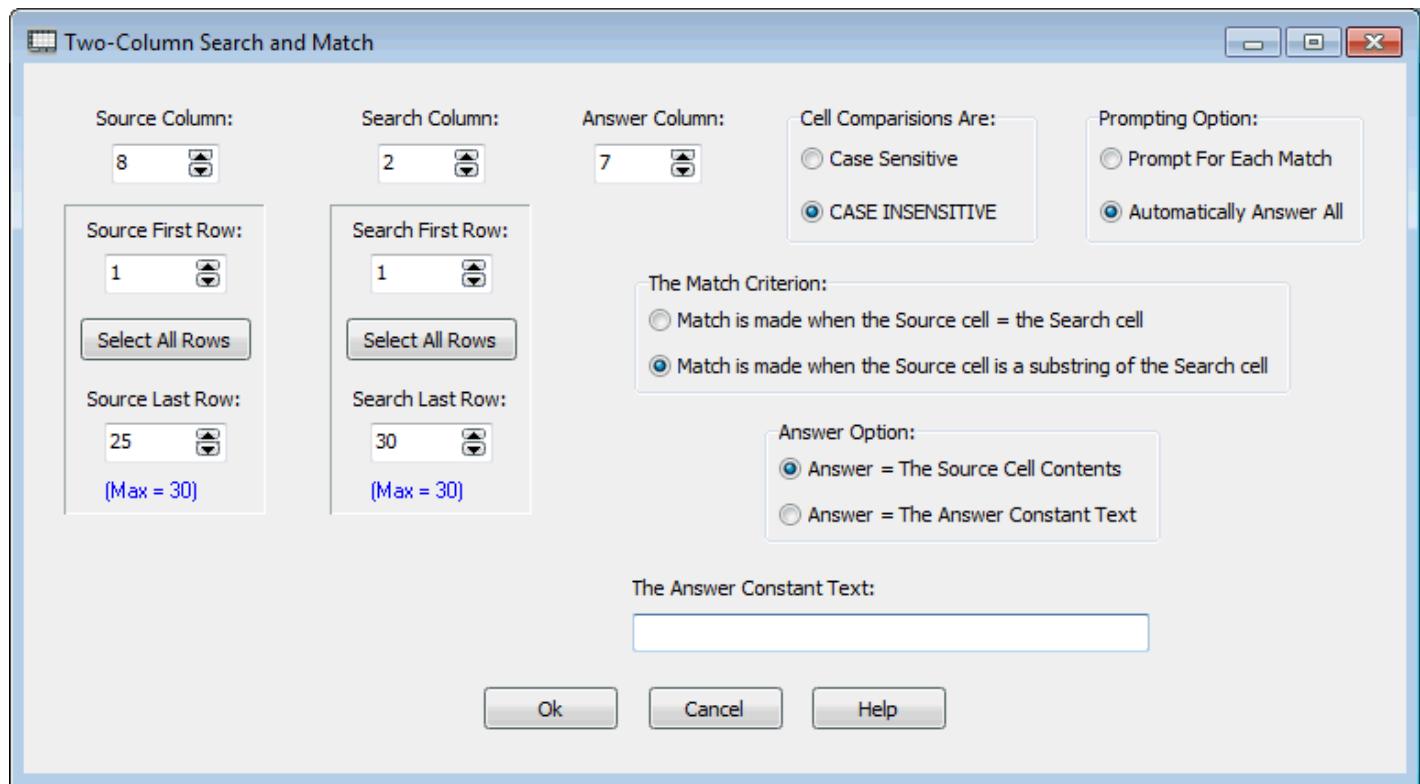
After we click **Ok** the result that comes back is:



The sum represents 2 hours and 22 minutes and 52 seconds. This represents a fair amount of computing power because all the time values were summed to get the final value, not a thing many programs can do so easily!

Two-Column Search and Match

The **CSV Editor** program has a special Search and Match function that involves using three columns. Essentially this special function can be thought of as nearly doing multiple search and replacements, where the source cells come from a given column called the **Source Column**, and where all the searching is made in a column called the **Search Column**, and where, when a match is found in the **Search Column**, then an answer result is placed in a third column called the **Answer Column**. Select the menu item **Columns | Two-Column Search and Match...** to bring up the dialog that appears as:



We call this a Search and Match function because it differs from a true Search and Replace. In the above dialog you normally first define the three special columns named the **Source Column** and the **Search Column** and the **Answer Column**. For the **Source** and **Search** columns you can specify a different range of rows.

The **Search Column** is so named because all searches will be made in that column. The source strings that are searched for are all the selected cells in the **Source Column**. The **Answer Constant Text** is the same constant string for all matches that are made. In the above dialog this text is empty. There is an **Answer Option** that you can set to make a choice of what the answer will be when a match is made. We often set **The Answer Constant Text** to the word **Match** or **Found** when we only care to know which rows matched. Otherwise we set the **Answer Option** to **The Source Cell Contents** when we wish to see the actual string that matched.

As with many string comparison functions, you can set a case sensitivity option. When this option is set to **Case Sensitive** then all cells are compared using the original strings found in those cells. When this option is set to **CASE INSENSITIVE** then all cells (both **Source** and **Search**) are first temporarily converted to Upper Case and then the matching criterion is applied.

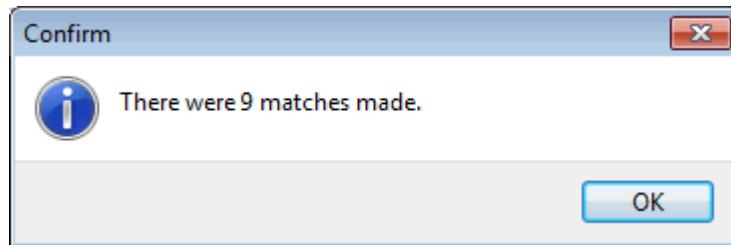
The Match Criterion must be one of two choices. A match will be considered to have been made when either the **Source** cell is the same as the **Search** cell, or a match will be considered to have been made when the **Source** cell is a substring of the **Search** cell. The replacement action differs from that of a text editor that normally only replaces a found substring, not an entire word, when the substring is found within a word. This is a significant aspect of this special search and match function that differs a little from what you might think of as a search and replace function. We call it a search and match function because we are not making a replacement in the cell where the match occurs. The answer is actually placed in a different column.

When this function executes, the program will run down the selected cells in the **Source Column**. It will use each **Source Column** cell as the search text. It will look anew at all the selected cells in the **Search Column**. If and when a match is found, it is the cell in the **Answer Column**, but in the same row as the matched **Search** cell, that gets filled with either **The Answer Constant Text** or with the **Source** cell text.

As an example, consider the grid below in which we have a standard list of 30 names and addresses in columns 1 through 6. We also have attached a blank **Answer Column** in column 7. **The Source Column** is column 8. The row information can be considered consistent and connected across columns 1-7 which comprise the main block. The cells in column 8 are unrelated row-wise to the main block in columns 1-7. Column 8 should simply be considered as an extra column that may have either fewer rows or more rows than the main block. In the grid below, Column 8 has fewer row entries than the main block has row entries.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
>	Title	Search Items:	Street Address	City	State	ZIP Code	The Answer:	Source Items:
1	Mrs.	Brantley, Alma Ruth	1109 Sunset Blvd.	Los Angeles	CA	90021		Washington
2		Bratton, Jay Norton	23234 9th St.	Los Angeles	CA	92323		Strong
3		Bremer, Jeff Bruce	1212 Nineteenth Street	San Diego	CA	93432		Gray
4	Mr.	Brendon, William Charles	6938 West 77th Street	New York	NY	10014		Alvarez
5		Chasin, Andrea Janice	2424 Culver Blvd.	Chicago	IL	21121		Scott
6		Chou, Cindy Cayla	3717 Bagley Ave.	Santa Monica	CA	90401		Hogan
7		Cummings, Dorothy Ellie	1919 Broadway Blvd.	Chicago	IL	23231		Grant
8	Mr.	Doe, John Frederick	12345 Any Street	New York	NY	10001		Guzman
9	Mrs.	Doe, Mary Jane	12345 Any Street	New York	NY	10001		Davenport
10		Foudy, Brian Jon	2024 Holt Ave.	Los Angeles	CA	93458		Branch
11		Garcia, Gilbert Lyle	12120 Washington Place	San Gabriel	CA	90032		Allen
12		Garcia, Jose Ferro	11921 Malibu St.	Santa Monica	CA	90410		Johnson
13	Dr.	Gelf, Layla Kay	3333 Ventura Blvd.	Encino	CA	93421		Vinson
14		Hollister, John Jeff	2790 Club Drive	Chicago	IL	22213		Cummings
15		Humphrey, Robert Allan	1407 Brockton Ave.	Riverside	CA	93481		Spence
16	Mr.	Johnson, Joe Brian	54321 Elm Street	Salt Lake City	UT	84119		Gates
17	Mr.	Langston, Donald Bob	7418 89th Avenue	Chicago	IL	23121		Sears
18		Lucas, Peter Scott	3576 Ocean Ave.	St. Louis	MO	42123		Head
19		Mackey, Jean Claudia	717 Manchester	Salt Lake City	UT	81143		Andrews
20		Madden, Steve Terry	1337 Saltair Ave.	Los Angeles	CA	98743		Garcia
21	Dr.	Madril, Francis Benjamin	234 Sherman Way	Santa Monica	CA	90342		Hardin
22		Martinez, Pedro Alan	1905 Capri Drive	Santa Monica	CA	90532		Bennett
23	Mr.	Morley, Michael Don	219 Banyon Street	Santa Monica	CA	92323		Doe
24		Moy, Peter Michael	1700 Decker Street	Chicago	IL	22212		McLean
25	Mrs.	Scott, Verna Ethel	2323 Pier Street	Long Beach	CA	91122		Franklin
26	Mrs.	Silton, Nancy Fran	2616 Victoria Ave.	Chicago	IL	21321		
27	Mr.	Simons, David John	2616 Venice Ave.	San Francisco	CA	93413		
28		Stern, John Jay	2221 Hilltree Street	Los Angeles	CA	90342		
29		Vogel, Andrew Richard	2417 Hill Street	San Mateo	CA	90121		
30	President	Washington, Jorge Edward	1600 Pennsylvania Avenue NW	Washington	DC	20500		

If we set all the controls as in the dialog shown two pages previous, including different row ranges for the **Search** and **Source** columns, then when we execute this function we will first see the message:



The new grid will appear as shown next:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
>	Title	Search Items:	Street Address	City	State	ZIP Code	The Answer:	Source Items:
1	Mrs.	Brantley, Alma Ruth	1109 Sunset Blvd.	Los Angeles	CA	90021		Washington
2		Bratton, Jay Norton	23234 9th St.	Los Angeles	CA	92323		Strong
3		Bremer, Jeff Bruce	1212 Nineteenth Street	San Diego	CA	93432		Gray
4	Mr.	Brendon, William Charles	6938 West 77th Street	New York	NY	10014		Alvarez
5		Chasin, Andrea Janice	2424 Culver Blvd.	Chicago	IL	21121		Scott
6		Chou, Cindy Cayla	3717 Bagley Ave.	Santa Monica	CA	90401		Hogan
7		Cummings, Dorothy Ellie	1919 Broadway Blvd.	Chicago	IL	23231	Cummings	Grant
8	Mr.	Doe, John Frederick	12345 Any Street	New York	NY	10001	Doe	Guzman
9	Mrs.	Doe, Mary Jane	12345 Any Street	New York	NY	10001	Doe	Davenport
10		Foudy, Brian Jon	2024 Holt Ave.	Los Angeles	CA	93458		Branch
11		Garcia, Gilbert Lyle	12120 Washington Place	San Gabriel	CA	90032	Garcia	Allen
12		Garcia, Jose Ferro	11921 Malibu St.	Santa Monica	CA	90410	Garcia	Johnson
13	Dr.	Gelf, Layla Kay	3333 Ventura Blvd.	Encino	CA	93421		Vinson
14		Hollister, John Jeff	2790 Club Drive	Chicago	IL	22213		Cummings
15		Humphrey, Robert Allan	1407 Brockton Ave.	Riverside	CA	93481		Spence
16	Mr.	Johnson, Joe Brian	54321 Elm Street	Salt Lake City	UT	84119	Johnson	Gates
17	Mr.	Langston, Donald Bob	7418 89th Avenue	Chicago	IL	23121		Sears
18		Lucas, Peter Scott	3576 Ocean Ave.	St. Louis	MO	42123	Scott	Head
19		Mackey, Jean Claudia	717 Manchester	Salt Lake City	UT	81143		Andrews
20		Madden, Steve Terry	1337 Saltair Ave.	Los Angeles	CA	98743		Garcia
21	Dr.	Madril, Francis Benjamin	234 Sherman Way	Santa Monica	CA	90342		Hardin
22		Martinez, Pedro Alan	1905 Capri Drive	Santa Monica	CA	90532		Bennett
23	Mr.	Morley, Michael Don	219 Banyon Street	Santa Monica	CA	92323		Doe
24		Moy, Peter Michael	1700 Decker Street	Chicago	IL	22212		McLean
25	Mrs.	Scott, Verna Ethel	2323 Pier Street	Long Beach	CA	91122	Scott	Franklin
26	Mrs.	Silton, Nancy Fran	2616 Victoria Ave.	Chicago	IL	21321		
27	Mr.	Simons, David John	2616 Venice Ave.	San Francisco	CA	93413		
28		Stern, John Jay	2221 Hilltree Street	Los Angeles	CA	90342		
29		Vogel, Andrew Richard	2417 Hill Street	San Mateo	CA	90121		
30	President	Washington, Jorge Edward	1600 Pennsylvania Avenue NW	Washington	DC	20500	Washington	

In the above grid in Column 7 we can discern that the names Doe and Garcia and Scott appeared twice each in the previous grid in Column 2. This means that of the 9 replacements that were made, 6 were for names that appeared more than once in Column 2 that was the **Search Column**. The remaining 3 names that were found to be matches by this operation were for Cummings and Johnson and Washington. Note that Scott appeared in Column 2 as both a middle name in Row 18 and as a last name in Row 25.

Although this example is simple and somewhat contrived, it allows us to determine which names in Column 8 also appear anywhere in Column 2, because those names appear as answers in Column 7, the **Answer Column**. We should also note that we deliberately cleared the **Answer Column** just before we executed this function.

Another way of thinking about this example is that we also know which names in Column 2 did NOT also appear in Column 8 because we can see the blank cells in the **Answer Column**. If we wanted, we could continue by filtering out the rows that contain a blank, or NOT, in Column 7. See **Columns | Filter On a Column...** to further proceed to do this.

We generally recommend you create a new blank **Answer Column** as we did in the above example. We also suggest that you think of the **Answer Column** as being part of the main block that contains the **Search Column**. It is only the **Source Column** that should be considered as a column distinct from the main block.

If you set the **Prompting Option** to **Prompt For Each Match** then when you run the previous example, you will see several instances of messages similar to the following. For each match that occurs you should see the highlighted cell in the **Search** column grid, and in the row that corresponds to the match.



You must select one of the four buttons to continue. If you press the **Cancel** button then the entire function will prematurely finish immediately. This would be the same as answering **No** for this and all remaining matches. If you press the button **Yes to This and All Remaining Matches**, then the program will no longer prompt you for permission for any further matches that will be handled automatically. Of course, if you press either the **Yes** or **No** buttons then the program will continue with a prompt for the next match, if there is one, after it processes your input.

There are at least three closely related functions that occur in other parts of the program. One of those functions is under the **Rows** menu and is called **Filter Rows With Matching Cells**. What makes that function fundamentally different is that the two columns used in filtering rows contain information that matches on a row by row basis, whereas the current function is usually applied using two columns whose cells are not tied together because they are in the same row.

Another function is the set difference function. If all we wanted was a list of names in Column 2 that are not also names in Column 8, we could have applied the set difference function. However, the result would not have been such that we could use the block in columns 1-6 to determine which rows in that block contained names not in Column 8 because the set difference answer rows would not match up with the rows in the block in columns 1-7. Using the current function as we did in the above example allows us to see the names that were found as matches and it allows us to see the rows where the names that were not found remained blank.

The function to perform **Two-Column Search and Match** is almost the same as the **Referential Integrity** function under the **Validation Menu**. The main difference is in how the **Answer Column** or the **Report Column** gets filled in.

Which function would be most appropriate for your application is something only you can determine.

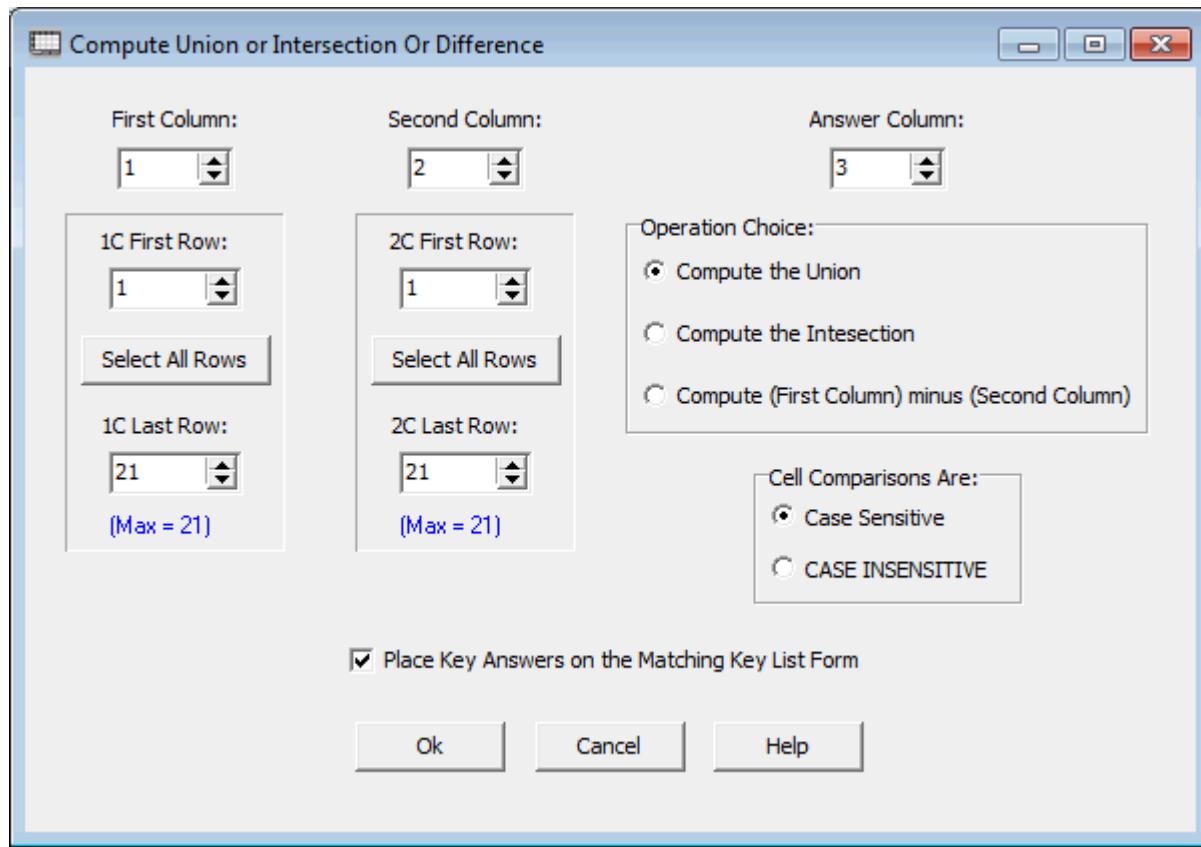
We should make one last comment regarding the **Source** column. While it is not a requirement, it is a good idea to insure that the cells in the **Source** column are unique. When they are not unique, then it will be possible for the program to report that the number of matches made is more than there are cells in the **Answer** column. When you see such a report you may think something is definitely wrong when in fact such a report is ok.

The reason this could happen is that when the **Source** column has duplicate cells, then the same cells in the **Answer** column will be over-written but replaced with the same content multiple times. Generally speaking this is not what you want to have happen unless you are prepared that it may happen. The best way to avoid this is to first check that the selected **Source** cells are unique. Just select all and only the **Source** cells and then execute the function **Uniqueness | Report Duplicate Cells In a Block....** It is a waste of time and memory to have duplicate cells in the **Source** column.

Unions and Intersections and Differences

The **CSV Editor** program has three set-type functions that operate on two columns of source cells, with customized selected consecutive rows for each source column. The operations are for the union or intersection or the difference between the two source column sets of cells. A third output column is used to deposit the answer set of entries.

To perform a set operation, select the menu item **Columns | Union or Intersection or Subtraction of Two Columns...** and you should see a dialog similar to the following:



The top three controls are used to set the three columns needed for the operation. These three column numbers, **First Column**, **Second Column**, and **Answer Column**, need to be distinct. This of course means your grid must contain at least three columns or you won't be able to bring up the dialog.

There is a separate set of controls for specifying the group of consecutive rows for each of the two source columns, **First Column** and the **Second Column** (also labeled as 1C and 2C). You don't have to select all the rows, and in some cases you will want to specify a different range of rows for the **First Column** as compared to the range of rows for the **Second Column**. Next, you should select the **Operation Choice**. Be aware that unions and intersections don't care about the order of the **First Column** and the **Second Column** numbers, but when you compute a difference, or subtract one column from another, then the order of the two column numbers is critical. We always compute a set difference by selecting all the cells in the **First Column** that are NOT in the **Second Column**. To commute, or force the other order, just swap the **First Column** and **Second Column** numbers whenever they are out of order.

The next selection you need to make is for the case sensitivity option. Comparisons can be **Case Sensitive** or **CASE INSENSITIVE**. Comparisons are only used to find matching entries in the two source columns. Case is not applied at all when the answers are placed in the **Answer Column**. The case of the answer entries will be whatever the case is in the source columns. For intersections and differences, the answer case will be that found in the **First Column**. For set unions, many answer entries will have the case of those entries that are in the **First Column**, while only those entries in the **Second Column** that are not in the **First Column** will have the case of the **Second Column**, when they are placed in the **Answer Column**.

The checkbox with the caption **Place Key Answers on the Matching Key List Form** is used to save you time from having to save the answer column in a separate file and then re-load that single column in the grid on the form used to **Export Matching Key Rows**. In general we recommend leaving this checked, even if you don't later decide to export rows that have matching key data. You should also see the function under the **Rows** menu with the title **Export Matching Key Rows....**

When you click the **Ok** button, the program will compute the answer which is always another set of cell entries. The answer that is computed gets placed in consecutive rows in the **Answer Column**, and the answer cells always start with the first data row in the grid. In fact, the entire **Answer Column** is cleared to all empty cells before the answer is computed. Thus you may always want to first insert a blank **Answer Column** so you don't erase any existing columns with data that you want to keep.

While the program is computing, you should notice a counter number in the lower left corner of the **Status Bar**. This counter is used to give you some idea of how long it will take for the current computation to finish. When you have thousands of rows, it can take a minute or more of computation time to compute the answer. In any case, the number will count down to zero.

In the case of a union operation, the program may try to add more rows to the existing grid. This kind of grid expansion is intended to be automatic, but the program will not exceed the maximum number of rows that are allowed for any grid. You might want to always first manually expand your grid to accommodate any possible union of the two source columns. Such a manual expansion by you will avoid an incorrect answer when the maximum number of grid rows is exceeded because you should know how many rows you will be limited to. For intersections and differences no grid expansion will ever be needed.

The grid on the next page shows examples for all three kinds of set operations. In this grid, Column 1 and Column 2 contain the original set of cells used for the operations. We selected rows 1 through 21 inclusive to work with as the two sets of source cell rows.

The union operation answer is what occupies Column 3. In this case we can see the program computed the union by first repeating all 21 entries in Column 1 as the first part of the answer. Then the program found 11 additional entries in Column 2 that were not already in Column 1 and it added those 11 entries to the end of Column 3. The grid was automatically expanded with 11 more additional rows. The ordering of the answer cells in the **Answer Column** is not important, but you can change the order after the answer is computed.

The intersection answer is what occupies Column 4. The order of the answer cells is not important, but the algorithm simply ran down the rows in Column 1 and if it found the same entry anywhere in the selected rows of Column 2, then it put that cell entry in Column 4. Because the Column 1 entries were already sorted by size, increasing from smallest to largest, the same is true in the answer column, Column 4.

The set difference answer is what occupies Column 5. The ordering of the answer cells is not important, but the algorithm simply ran down the rows in Column 1 and if it did not find the same entry anywhere in the selected rows of Column 2, then it put that cell entry in Column 5. Because the Column 1 entries were already sorted by size, increasing from smallest to largest, the same is true in the answer column, Column 5.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Original First Column:	Original Second Column:	Union Answer:	Intersection Answer:	First - Second Answer:
1	1960	1989	1960	1960	1962
2	1961	1978	1961	1961	1963
3	1962	1992	1962	1964	1966
4	1963	1975	1963	1965	1967
5	1964	1999	1964	1970	1968
6	1965	1982	1965	1971	1969
7	1966	1954	1966	1973	1972
8	1967	1957	1967	1975	1974
9	1968	1965	1968	1977	1976
10	1969	1973	1969	1978	1979
11	1970	1985	1970		1980
12	1971	1958	1971		
13	1972	1970	1972		
14	1973	1952	1973		
15	1974	1953	1974		
16	1975	1960	1975		
17	1976	1996	1976		
18	1977	1964	1977		
19	1978	1971	1978		
20	1979	1961	1979		
21	1980	1977	1980		
22			1989		
23			1992		
24			1999		
25			1982		
26			1954		
27			1957		
28			1985		
29			1958		
30			1952		
31			1953		
32			1996		

We should mention that we don't check if the selected groups of cells in either the **First Column** or the **Second Column** constitute an actual set. If either of those cell groups contain any repeated entries, then those groups will not be actual sets. However, we do check that the **Answer Column** contains unique entries, so the **Answer Column** cells should constitute a real set without duplications. In some cases you may want to first verify you are starting with unique cells for the two input columns. You can do this using a uniqueness check under the **Uniqueness** menu to **Report Duplicate Cells In a Block**.

Although the **Answer Column** in the above example is sorted, if you need to change the order of the cells in the **Answer Column**, we suggest you first select those cells and then use either **Columns | Sort In Place (Column Sublist)...** or **Blocks | Block Scramble/Rotate...** to re-order the cells within the **Answer Column**.

A function related to the set difference function is one we have named a Search and Match function. To learn about this other related function see also **Columns | Two-Column Search and Match....** Sometimes using this Search and Match function gives a more useful answer than just computing a set difference, but it all depends on the nature of your data.

Editing Multiple Columns At Once

The **CSV Editor** program has a special mode that allows you to edit multiple columns at once. This mode is useful whenever you spend a lot of time horizontally scrolling to edit the fields in the rightmost columns of any **CSV** file. Using the special multiple column edit mode helps avoid most, if not all, of the horizontal scrolling. You can bring up this special mode by pressing function key **F8** or you can click the button in the toolbar that appears as:



This mode is especially useful when the first line in your **CSV** file is a header line. In that case, the Header titles will appear as the label titles for the individual edit boxes. The program will automatically create and fill in up to 36 columns of data, all at once. The next image shows an example of a **CSV** file that has 25 columns. This data is related to sheets of labels made by various manufacturers.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10
>	Manufacturer Name	Manufacturer Number	Description	Shape	Number of Colu	Number of Row	Label Width	Label Height	Space Between	Space Between
1	Avery Metric	C91149T	Business Card	Rectangular	2	4	3.58	2.17	0.32	0.35
2	Avery Metric	C9146	Photo Label	Rectangular	1	1	8.27	11.69	0	0
3	Avery Metric	C9151	Photo Label	Rectangular	5	5	1.18	1.57	0.3	0.4
4	Avery Metric	C9167	Photo Paper	Rectangular	1	1	7.86	11.38	0	0
5	Avery Metric	C9169	Photo Label	Rectangular	2	2	5.47	3.9	0	0.1
6	Avery Metric	C9269	Other Label	Rectangular	2	2	3.54	4.72	0.4	0.67
7	Avery Metric	C9351	1/2 Top Fold	Rectangular	1	2	8.27	5.85	0	0
8	Avery Metric	C9351	1/2 Side Fold	Rectangular	2	1	5.85	8.27	0	0
9	Avery Metric	C9352	1/2 Side Fold	Rectangular	2	2	3.25	4.75	0	0.19
10	Avery Metric	C9352	1/2 Top Fold	Rectangular	2	2	4.75	3.25	0.19	0
11	Avery Metric	C9353	Postcard Portrait	Rectangular	1	2	4.35	5.77	0.5	0
12	Avery Metric	C9353	Postcard	Rectangular	1	2	5.77	4.35	0	0.5
13	Avery Metric	C9354	Business Card	Rectangular	2	4	3.17	2	0.18	0.33

The main screen above only shows about the first 10 columns of data. If you were doing a lot of editing, you might spend a significant amount of time horizontally scrolling to bring the rightmost and last 15 columns into view. For the above example data file, if you were to press function key **F8** you would bring up the multiple columns edit mode and you would see a dialog like that shown on the next page.

In the dialog shown on the next page we can see all the data for one row, all at once, in one window. In this example there are 25 individual edit boxes or edit fields. The blue text that appears as **Row 1** in the top of the dialog simply tells you what row of the original table you are editing. This dialog allows you to edit one row of data at a time.

When the multiple columns editor dialog is opened, you will find the **CSV** data appears in groups of edit boxes that are arranged with at most 12 edit fields going down vertically, and at most 3 edit boxes going across horizontally. The reading order of the edit boxes should be from top to bottom, somewhat like you would read the 3 columns of a newspaper. If there is a 12th edit box it will appear at the bottom of the 1st column. If there is a 13th edit box, it will appear at the top of the 2nd column. The bottom of the 2nd column would be the 24th edit box. The top of the 3rd column will always be the 25th edit box if your file has that much data. If you had 36 or more columns of **CSV** data, you would find the edit box for the 36th item would be at the bottom right of the entire dialog.

Edit Multiple Columns In One View

		Row 1		
Post and Move Down	Post and Move Up	Exit Without Posting		
Manufacturer Name Column 1 Avery Metric		Color Name Column 13 White		Page Description Column 25 A4 (21 x 29.7 cm)
Manufacturer Number Column 2 C91149T		Red Value Column 14 255		
Description Column 3 Business Card		Green Value Column 15 255		
Shape Column 4 Rectangular		Blue Value Column 16 255		
Number of Columns Column 5 2		Finish Column 17 plain		
Number of Rows Column 6 4		Ink Jet Column 18 y		
Label Width Column 7 3.58		Laser Jet Column 19 y		
Label Height Column 8 2.17		Roll Column 20 n		
Space Between Columns Column 9 0.32		Weather Proof Column 21 n		
Space Between Rows Column 10 0.35		Material Column 22 paper		
Top Margin Column 11 0.98		Sheet Width Column 23 8.27		
Left Margin Column 12 0.59		Sheet Height Column 24 11.69		

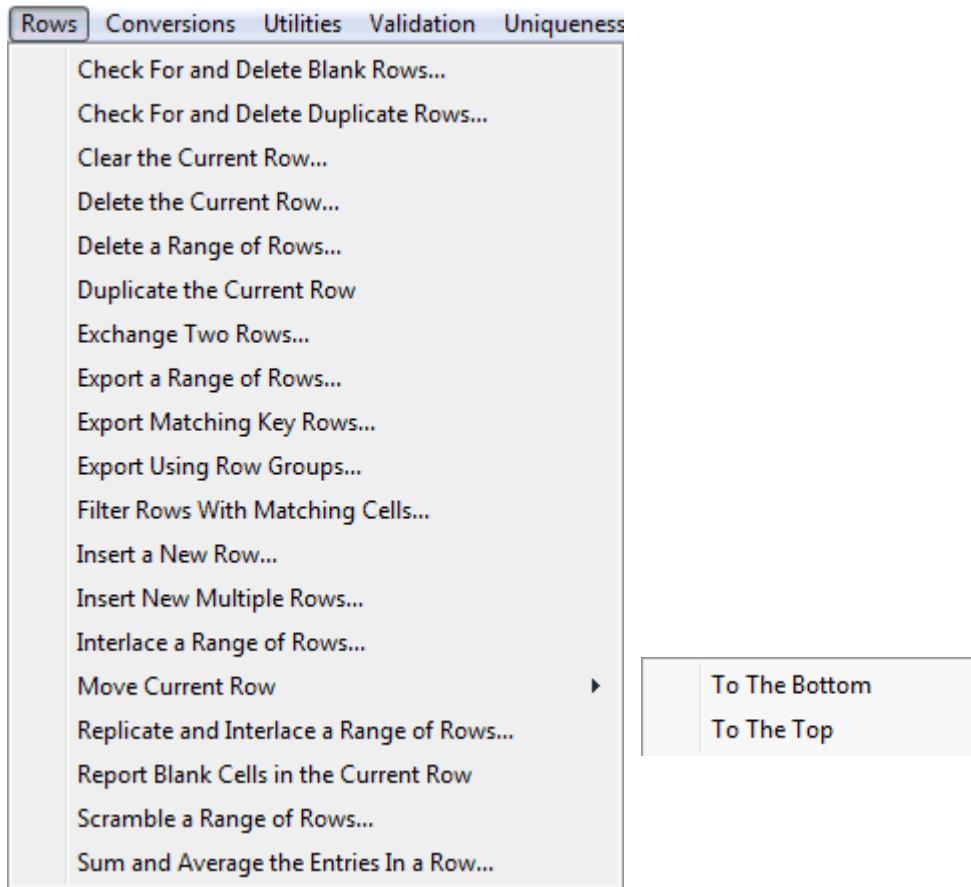
All 25 of the individual edit boxes have been labeled using the Header information that describes the columns of data. **Row 1** is shown above as the row being edited. This row number changes automatically, but it always tells you what row you are currently editing. After editing the data for any row you would normally press the **Post and Move Down** button to save that data and to automatically move down to the next row. The next row of data will automatically fill the fields of this form and the blue row number will be updated.

After editing a row you must Post that row to save your editing changes back in the original grid. After you finish editing several consecutive rows you would press the button **Exit Without Posting** to exit this special mode.

This special mode only loads up to the first 36 columns of data. If your **CSV** file has more than 36 columns, you could use this mode to edit the first 36 columns and then you will just have to use the regular edit mode of the program to edit the 37th column and all other columns past the 36th column. Of course, most files will have less than 36 columns, and when that is the case, you will see fewer than 36 edit box controls populate the above dialog. Thus you will only see as many edit boxes as there columns in the original **CSV** file, up to the first 36 columns of data.

The Rows Menu

The **Rows** menu contains functions that apply to working on rows. The choices are:



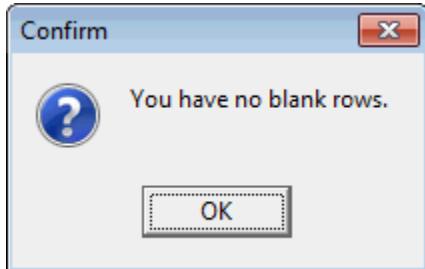
You might note several of these functions are available under the **Columns** menu where they apply to columns. Thus many of the things you can do to rows can also be done to columns. This set of functions provides some limited symmetry between row and column commands.

Keep in mind that every row can also be considered as a block. Some blocks are also just rows. So whenever you are looking for a function, that function could be under the **Rows** menu or the **Blocks** menu. If you don't find the function you need under the **Rows** menu, then you might try looking under the **Blocks** menu, and vice versa.

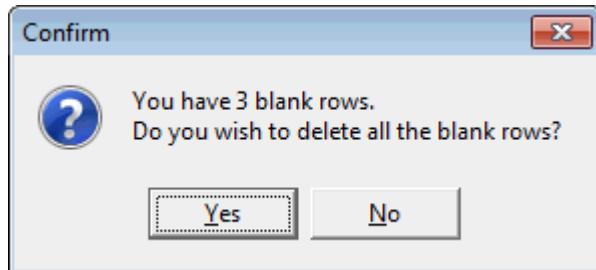
Check For and Delete Blank Rows

If you select the menu item **Rows | Check For and Delete Blank Rows...** you will see one of two messages.

The first message occurs after the program checks and finds no blank rows.



On the other hand, if at least one blank row exists you will see a message similar to the following:

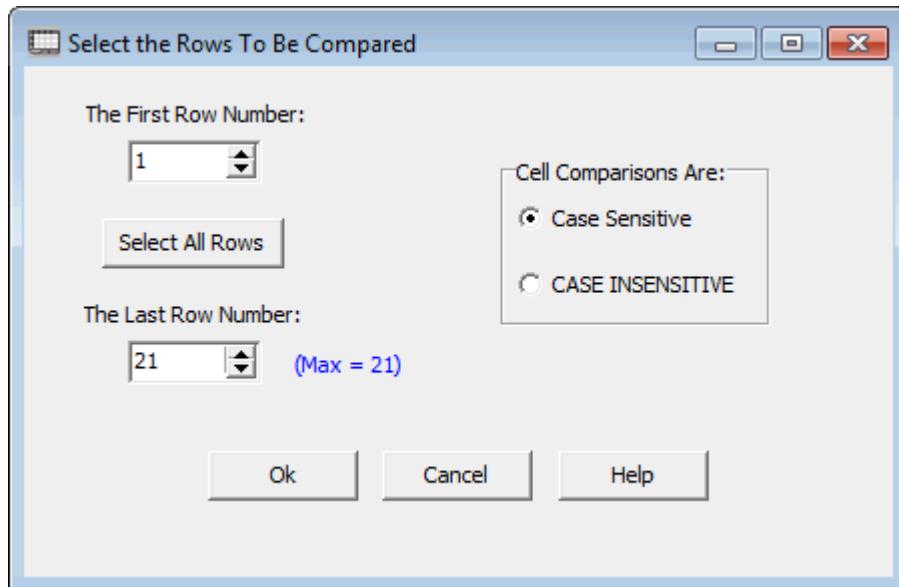


If you click the **Yes** button, the program will then try to delete all the blank rows. However, the program never deletes all the rows in the grid. So only in the rare case that the entire grid is blank, you can be left with exactly one blank row.

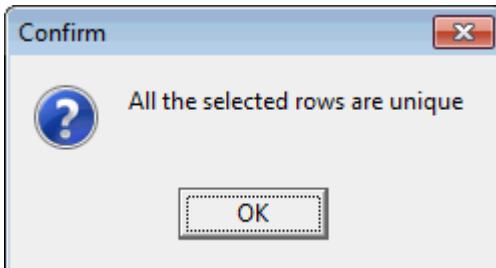
Of course if you click the **No** button then no further action is taken.

Deleting Duplicate Rows

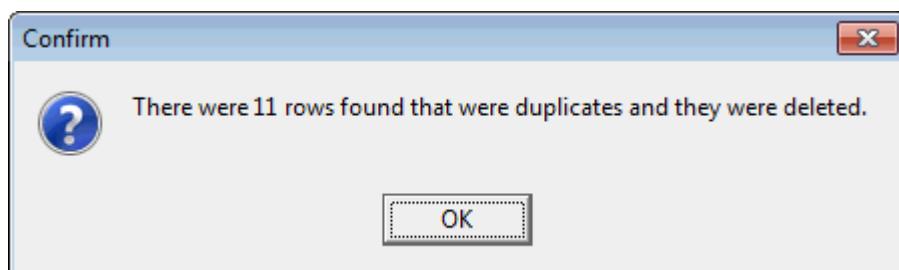
You can delete all duplicate rows in the current grid by first selecting **Rows | Check For and Delete Duplicate Rows....** This will bring up the dialog in which you can select a range of rows and you can set a case sensitivity option. Only duplicate rows that are wasting space will be deleted, while all original first rows remain unchanged. If you just want to know how many pairs of duplicate rows you have, without deleting those that are duplicates, then you should select the menu item **Uniqueness | Report Duplicate Rows In the Grid....**



When you click the **Ok** button the program will automatically search for and delete any duplicate rows that are found within the range you selected. You will then see one of two messages.



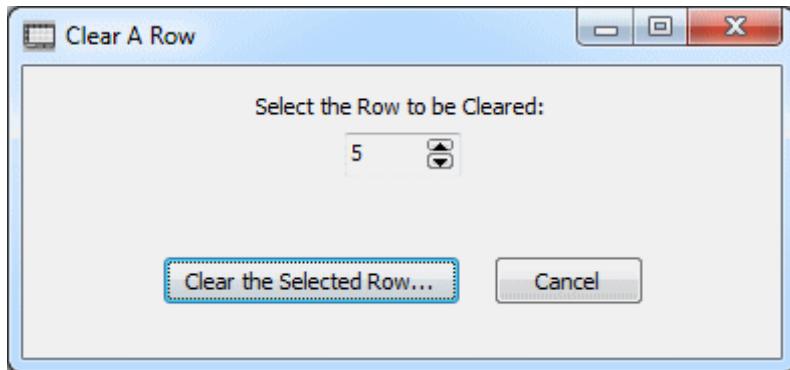
or you may see a message like:



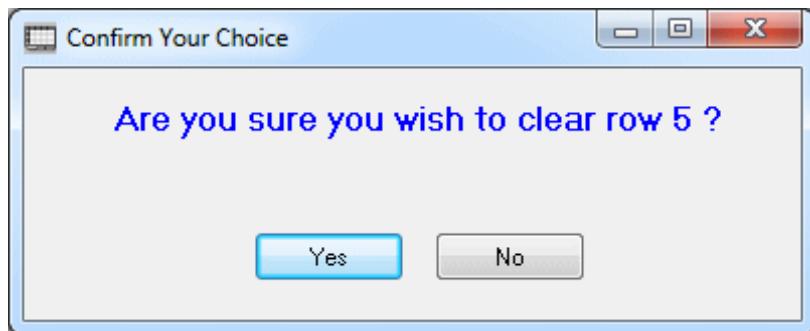
This menu item is duplicated under the **Uniqueness** menu.

Clearing the Current Row

If you leave the selection in any row of a column you can clear that row by making empty cells in all columns. Just select the menu item **Rows | Clear the Current Row...** and you will be given a chance to select any row to be cleared, but the default value should already be filled in for you.



If you click the button to **Clear the Selected Row** then the program will ask you to confirm your choice:



If you click **Yes**, the program will empty each cell in the selected row. If you click **No**, the program will ignore the command to clear the row.

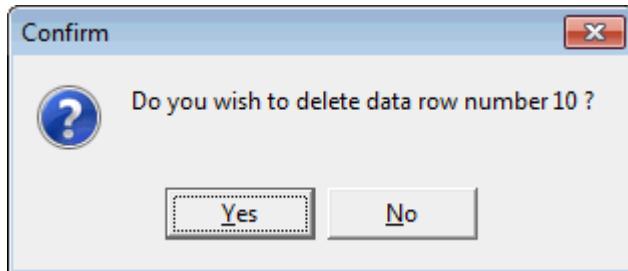
An alternative method to clear the entries in one or more rows is to first select all the entries in the rows to be cleared. Then press the delete key **DEL** on your keyboard to clear all those selected cells.

Clearing a row is not the same as deleting a row. Clearing a row does not remove the row from the grid. It just blanks the entries in the row but otherwise leaves the row in its current position.

If you want to clear a large block of rows or cells that you can't conveniently select with the keyboard, then use the command **Blocks | Block Formatting...** and define the block in the dialog that appears. Then choose **General Strings** as the cell data type and further choose the **General String Formatting** option as **Clear Each Cell**.

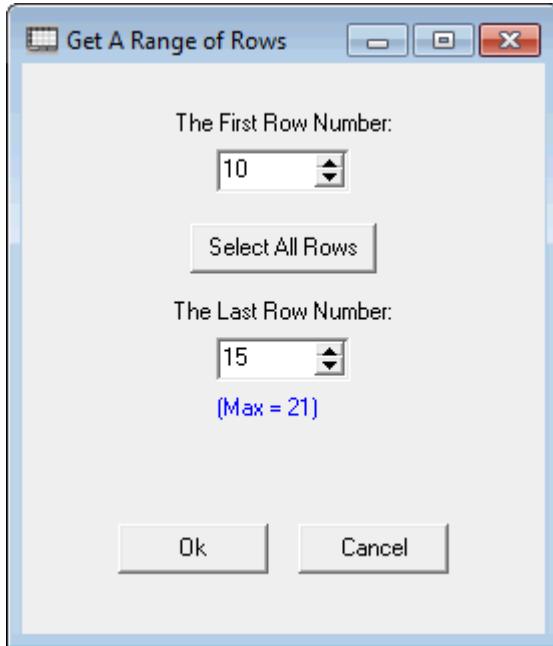
Deleting a Single Row or a Range of Rows

You can delete a single row by first selecting any cell in that row and then you can click the button  in the toolbar, or you can select the menu item **Rows | Delete the Current Row...**. When you do this you may be prompted by:

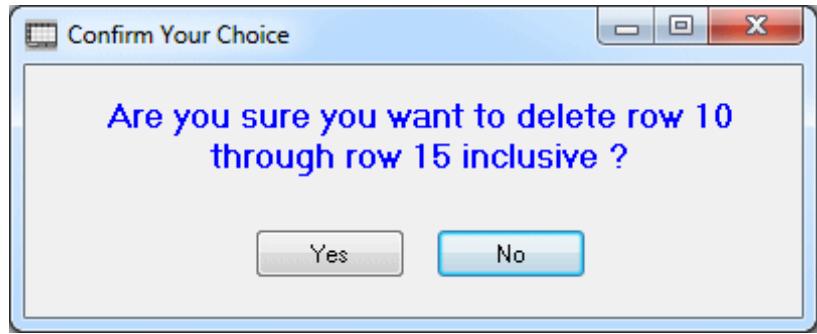


Whether you see the above prompt or not depends on an option that is set in the **Options** menu that has the caption **Prompt for Permission on Row Delete**. If this menu item is not checked then the row is simply immediately deleted without you being given a chance to change your mind. If necessary, you could immediately click the Undo button in the toolbar to correct the potential mistake.

If you know you want to delete many consecutive rows in one step, then you can select **Rows | Delete a Range of Rows...** and you will be prompted to select a range of rows:



If you continue you will be given one last chance to change your mind.



All the rows in the entire grid get renumbered whenever you delete one or more rows.

If you delete a sufficient number of rows so that only 1 row of data is left, then the program will automatically turn off the option to view the first row as a header row. Think of pressing the toolbar button as only making a request to turn the header row display on. The header row view option can only be turned on if you have at least two data rows in the current grid.

Duplicating the Current Row

You can duplicate the current row by clicking the button  in the toolbar or you can select the menu item **Rows | Duplicate the Current Row**.

When you do this the action that takes place is automatic. A new row is created immediately under the current row and the contents in all columns will be filled with the entries of the previous row. All rows including the new one will get renumbered as needed. Of course before the current row is duplicated, all rows, if any, below the current row will get moved down to make room for the new row.

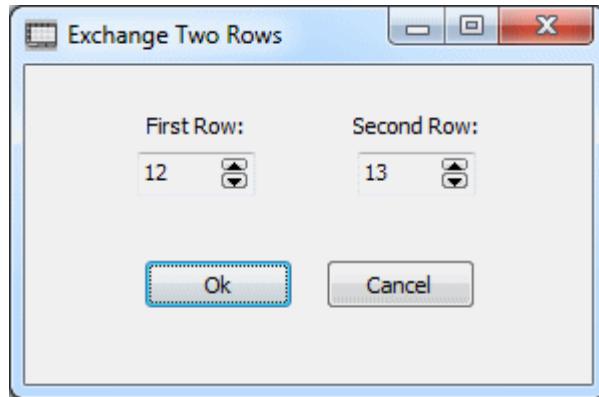
After the new row is created, if you decide to change your mind you can click the undo  button in the toolbar to get back to where you started.

Because we assume you know what you are doing when you duplicate a single row, the program does not check whether such an insertion would violate the normal 50,000 maximum row limit. Instead, the program will always add the extra row no matter how many rows you currently have.

If you want to replicate a row many times, then rather than trying to duplicate it using the toolbar button you should instead use the command **Blocks | Block Replicate...**. Using this option you can define the block as the single row you want replicate, and you can tell the program exactly how many times you want to perform the replication. If needed, you can also just replicate the row until you fill all rows that remain in the grid.

Exchanging Two Rows

You can exchange any two rows by selecting the menu item **Rows | Exchange Two Rows...**. You will then see a dialog that allows you to select the two rows to be exchanged:

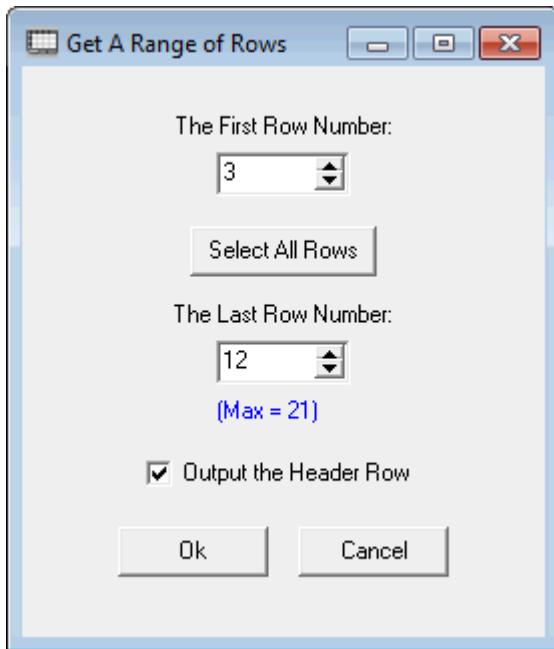


When you click the **Ok** button, the program will swap the contents of the two rows. There is no need to renumber any rows because only the row contents get exchanged. The row numbers remain unchanged.

If you want to move a given row only a few positions away from its current row position, then you can accomplish that by simply making a series of adjacent row exchanges. If you need to move a row more than a few rows away from its current position, then another technique is to use a row insert at the final destination position, followed by a row exchange using one of the rows as the original row to be moved, followed by a row deletion that deletes the blank row from that original position.

Exporting a Range of Rows

There may be times when you need to export a range of consecutive rows from a **CSV** file. Just select the menu item **Rows | Export a Range of Rows...** and you will be prompted by:



The checkbox to **Output the Header Row** may appear disabled and grayed out, unless you have the header row turned on. When the header row is turned on, then you have the option to output the header row or not when your selected rows are output.

You should enter the desired row range and then click the **Ok** button to continue. You will then be prompted by a **Save As** dialog in which you should name the new **CSV** file that will be saved. You can navigate to any directory on your computer and save the file. If necessary, you will have to give permission to overwrite any existing file. The new saved output file will be formatted as a **CSV** file.

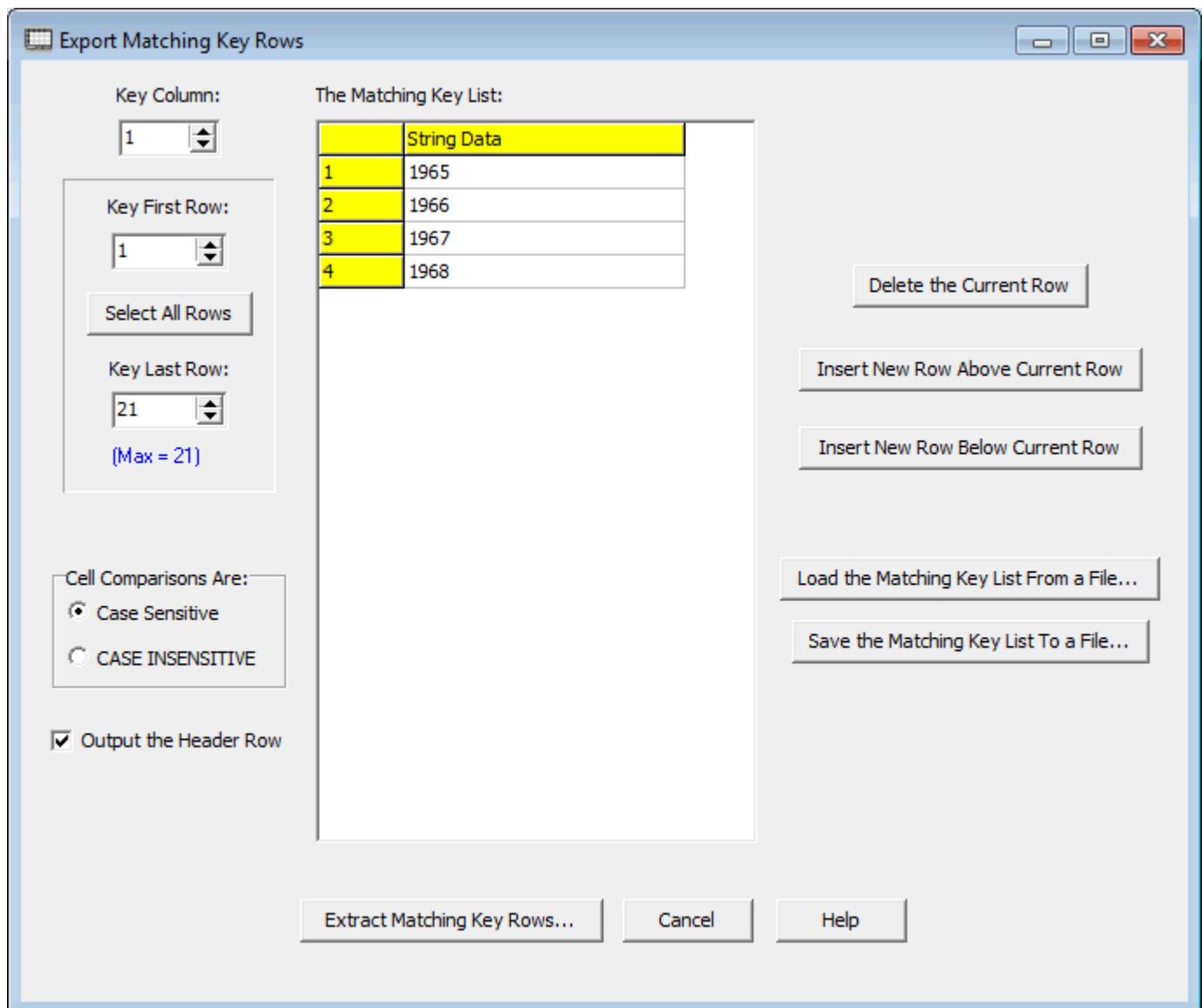
As a special note, if the first row in the grid is a header row, and you want to export that row as a data row, then you could click the **H+** button in the toolbar to turn off the option that highlights the first row as a header line. Once that first row is not highlighted in yellow, it can be exported as a regular data row. An alternative to using the toolbar button would be to go into the **View** menu and toggle the corresponding menu item. In either case, after exporting, you could turn the option to display the first row as a header line back on. Clicking either the **H-** button or the **H+** button in the toolbar is the fastest and easiest way to turn the header row display either on or off.

Exporting Matching Key Rows

After you perform a set operation such as an intersection or a union or a difference of two key columns, you may wish to extract only those rows from one or two tables that have a key element that matches an entry in the answer column to that set operation. For such a set operation, you might first save the key answer column as the only column in what we will call the key grid. Using this function is faster and easier than trying to extract the needed columns one at a time using the function under **Columns | Copy Matching Data....**

Unrelated to a set operation, you may just want to extract a certain set of rows from a grid where you manually enter the matching data into a key grid as if the data were key data.

In either of the above two cases, you should first open the grid that contains the rows of data that will be selectively and specially exported. Then select the menu item **Rows | Export Matching Key Rows...** and you will see a dialog similar to the following:

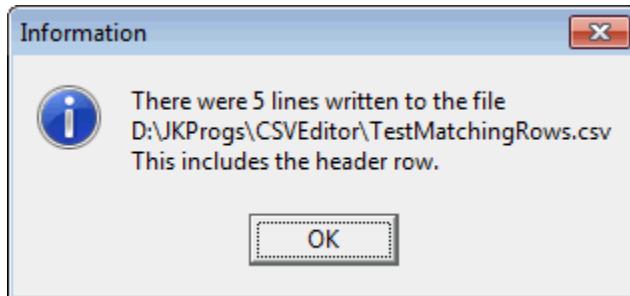


In this dialog, the left-most controls are used to define the key column in the table that is open, and to define a range of rows to which the extraction test will be applied. In most applications you will want to select all rows in the opened table, but in some special cases you might want to restrict the row range of the open table. You can also set an option related to whether cell comparisons are case sensitive or not. The checkbox to **Output the Header Row** may be disabled and grayed out, unless you have the header row turned on. Otherwise this checkbox will be enabled and you will have the option to turn it off and not output the header row.

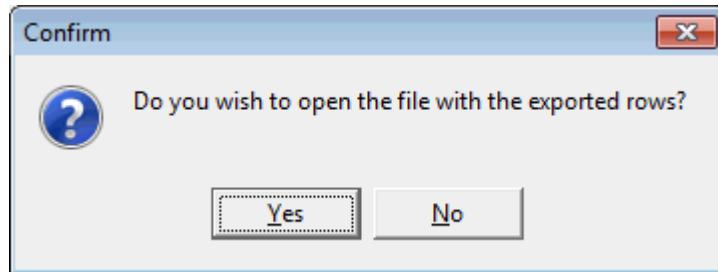
The main middle grid is to contain the set of key values. This grid will be automatically populated for you if you set a checkbox to do so in the form used to compute a set operation. Otherwise, you can manually enter those key values in this grid or more normally you will load those values from the file that has the answer column to the set operation. The other controls on the far right are used to help you edit, load, or save the **Matching Key List**.

After the **Matching Key List** grid is populated, you can click the button with the caption **Extract Matching Key Rows...** and the program will prompt you for a filename for where the extracted rows should be saved.

What the program then does is march down the **Matching Key List**, and it will look for each next key value in the key column in the currently opened grid. If the key value is found in any row (assuming the row number is within the selected range), that complete row from the currently opened grid will be saved in the export output file. You will then see a message that indicates the number of rows that were exported. If the currently opened grid has a header row, that header row will also be written to the output file as its first line. If the file to be created already exists you will first be asked if you would like to overwrite the existing file. In any case, when the process finishes creating the output file you should see an information message like:



You will then be given the opportunity to load the file with the exported rows by responding to the next message.



As a very simple example, suppose you load the Academy Awards grid that appears as shown on the next page and assume you entered the four values as shown in **The Matching Key List** grid shown on the previous page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

Then when the matching rows are exported and you choose to open that file you should see the grid:

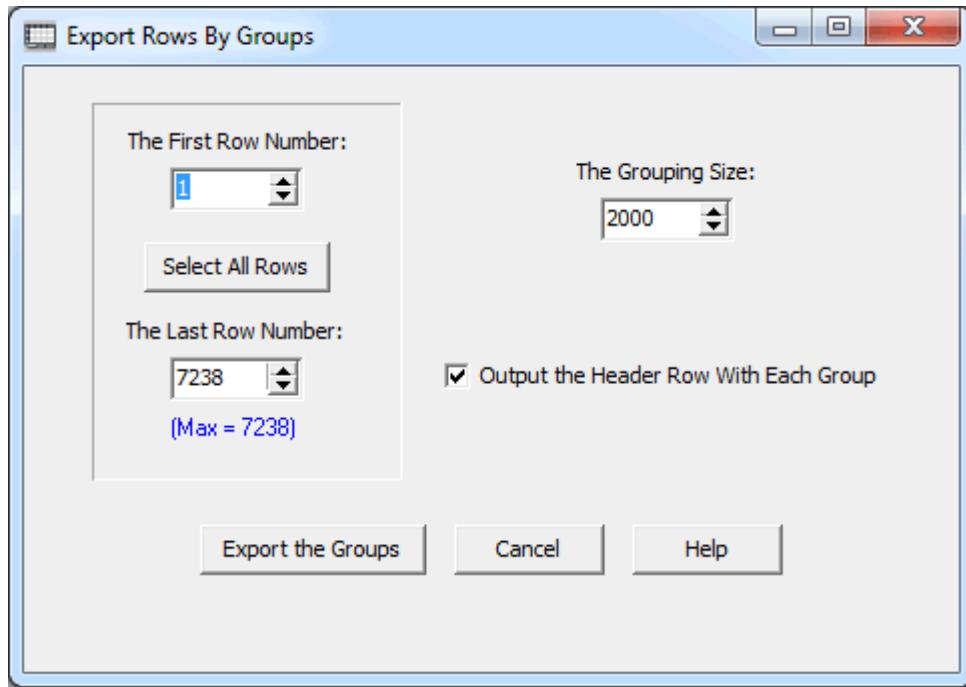
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
2	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
3	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
4	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T

In this example, the years 1965-1968 were the key data that was found in Column 1 in the Academy Awards grid. This explains why only the four rows with these year values were extracted from the original table.

You should find using this function **Export Matching Key Rows** is faster and easier than using the function under the **Utilities** menu that is called **Copy Matching Data**. The **Copy Matching Data** function accomplishes the same thing, but it works using a single grid and it copies only one column at a time.

Exporting Groups of Rows

An alternative to **Exporting a Range of Rows** is to automatically export rows using groups of rows where you can define the group size. This means you can split up a very large file into groups of smaller numbers of rows. To get started select **Rows | Export Using Row Groups...** and you should see the following dialog.



The first group of controls in this dialog allow you to select a range of rows for which the entire function will be applied. Most of the time you will probably want to select all the rows possible, but for those special times when you need finer control, you can set an applicable row range that limits the output.

The next main choice is for **The Grouping Size** where the default value is 2000. This means the program will try to export groups of rows, using 2000 rows at a time until the entire range of rows has been output. The last group of rows may have less than **The Grouping Size** rows in that group. You can set **The Grouping Size** to any positive value that you need, up to a maximum of 50,000.

The final option is what you want to do with any Header row information, assuming you have a Header row that is currently displayed. When you check the checkbox, **Output the Header Row With Each Group**, the program will write Header information for all columns with each group of rows that is exported. However, if the header row is turned off when you open this dialog, then this checkbox will be inactive and grayed out.

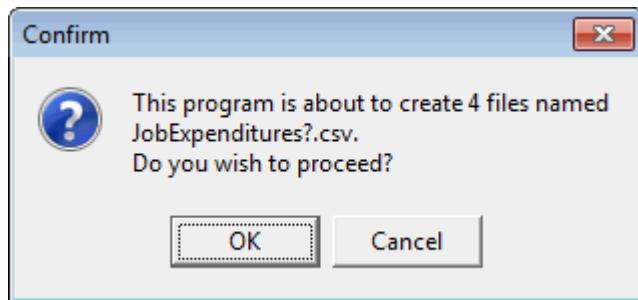
The final point to understand about this function is that it will write the output files in groups, where the output filenames are numbered beginning with the number 1 and continuing through the last group. The output filenames are based on the primary filename of the currently opened file at the start of the operation. You will not be prompted to enter any starting filename, nor will you be asked for permission to overwrite any existing files, so be careful before you apply this function.

As an example, suppose you have a grid with **7,238** rows and you set **The Grouping Size** as **2,000**. Then if the currently opened file is named **JobExpenditures.csv**, the program will create the following four new output files.

JobExpenditures1.csv
JobExpenditures2.csv
JobExpenditures3.csv
JobExpenditures4.csv

The first three files will contain **2,000** rows each (plus any header row information as selected) and the fourth and last file would contain **1,238** rows. Note that $7,238 = 3*(2,000) + 1,238$.

When you press the button with the caption **Export the Groups** you should first see a preliminary message like:

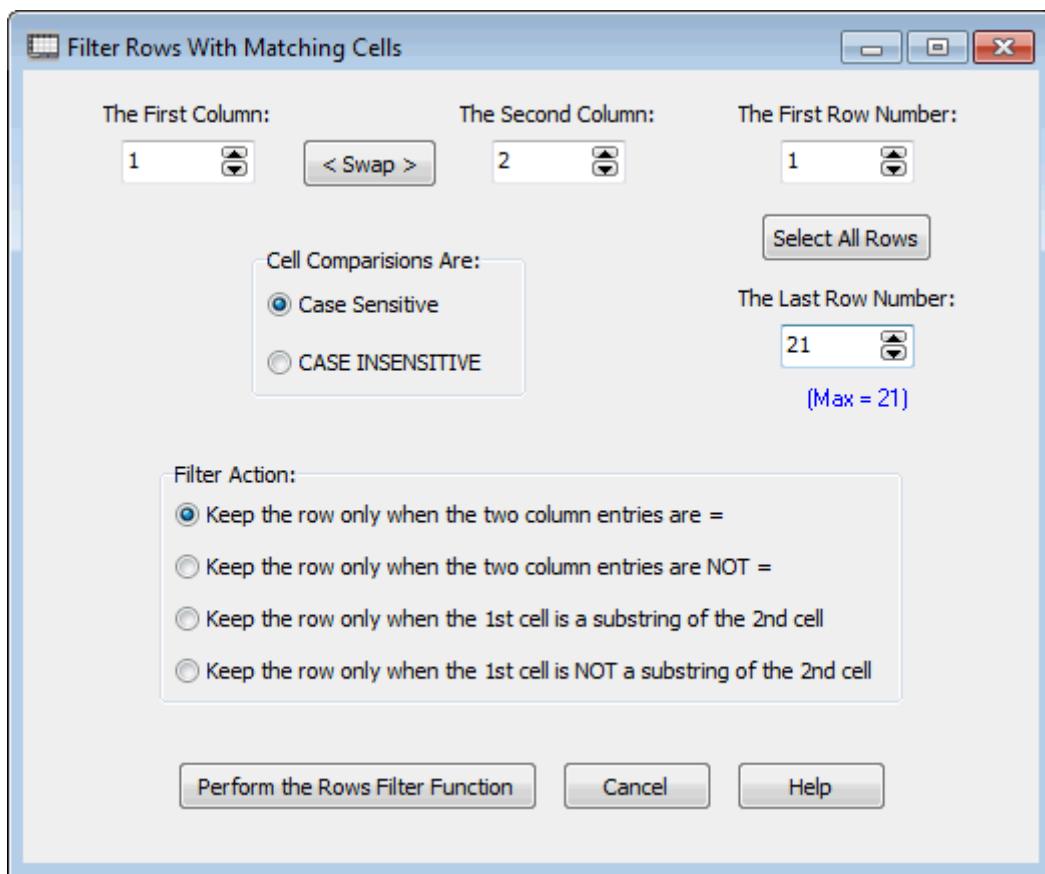


where the actual filenames (like **JobExpenditures?.csv** in this example) will depend on the file named by your current grid.

The purpose of this prompt is to tell you how many files will be created and to give you an indication of what the series of filenames should be. If you click the **Cancel** button then the entire operation will be aborted. Otherwise, the program will automatically create the file series and it will automatically overwrite any existing files that have a matching filename. The ? symbol in the sample filename will get replaced by each next consecutive integer, starting from 1.

Filter Rows With Matching Cells

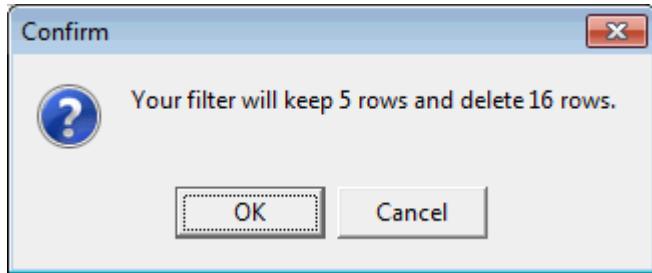
It is possible to filter the rows of a grid by comparing the cells in two different columns. Filtering in this case requires choosing exactly one of four different matching criteria. The **Filter Action** involves tests for either equality or substrings. The choices make it easy to quickly select an opposite type of criterion. To perform this operation, select the menu **Rows | Filter Rows With Matching Cells...** and you should bring up the following dialog:



You need to enter the two column numbers for the cells you will want to compare. The order of these two column numbers is normally not significant when the **Filter Action** is one of the first two choices, so they can be any two columns. However, if you choose a **Filter Action** that involves substrings, then you will want to specify **The First Column** as the one that contains the substrings. Specify **The Second Column** as the one that contains the strings into which you will be looking for a substring. Use the **< Swap >** button to exchange the two column numbers when they are out of order. You can select a range of rows to be filtered. Only the rows within the range you specify will be subject to being kept or deleted, depending on how you set the **Filter Action**. All other grid rows that are not within the specified range will remain unchanged, although their row positions may move up when any rows above them are deleted.

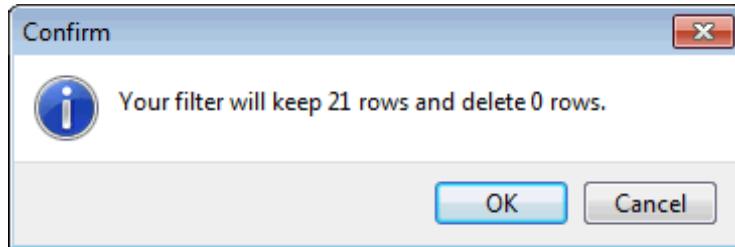
There is a case sensitivity option that determines whether the two cell entries will be considered equal or not or determines whether the first is a substring of the second. Internally, both cells will be temporarily converted to upper case when the **CASE INSENSITIVE** option is selected. Otherwise the cell contents are used as is.

When you click the button to **Perform the Rows Filter Function**, you may see any one of several preliminary confirmation messages as shown next.



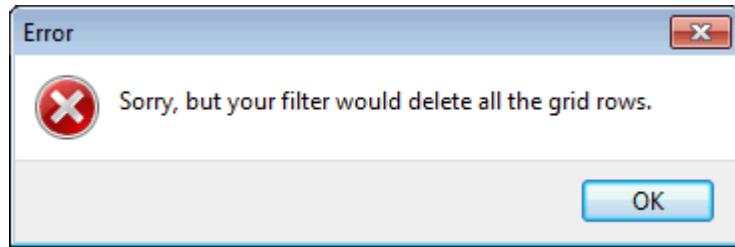
If you press the **Cancel** button then the entire function will be aborted. This means the whole point of showing you the previous confirmation message is to give you a chance to either continue and actually perform the operation, or you can cancel the operation. The choice is yours. If you click the **Ok** button the filter function will be performed. The filter function works by marching down the rows you have selected and it compares the two cells on a row by row basis when it decides whether to delete the selected row or not.

A different extreme case of a confirmation message would be one like:



In this case you are at least informed that the filter will be keeping all the rows and deleting no rows.

A different kind of error message occurs when you see



This error message only occurs when have selected all the rows in the grid, and the filter function would have the effect of deleting all those same rows. Essentially this error message prevents you from ever having an empty grid as the result of deleting some rows. We assume all opened grids always have at least one row. That row may have all empty cells, but at least one row should always exist in any opened grid. Note that you won't see this message even when the filter would delete all the selected rows, as long as not all the grid rows were selected in the first place.

As a somewhat contrived example, consider a list of the 51 states in the United States, including the District of Columbia. The list contains the 2-letter abbreviation in the first column and the fully spelled out state name in the second column as shown on the left below. The question we want to ask is how many and which states are such that their 2-letter abbreviation is contained anywhere within the name of the state?

<	Column 1	Column 2
>	Abbreviation:	State Name:
1	AL	Alabama
2	AK	Alaska
3	AZ	Arizona
4	AR	Arkansas
5	CA	California
6	CO	Colorado
7	CT	Connecticut
8	DE	Delaware
9	DC	District Of Columbia
10	FL	Florida
11	GA	Georgia
12	HI	Hawaii
13	ID	Idaho
14	IL	Illinois
15	IN	Indiana
16	IA	Iowa
17	KS	Kansas
18	KY	Kentucky
19	LA	Louisiana
20	ME	Maine
21	MD	Maryland
22	MA	Massachusetts
23	MI	Michigan
24	MN	Minnesota
25	MS	Mississippi
26	MO	Missouri
27	MT	Montana
28	NE	Nebraska
29	NV	Nevada
30	NH	New Hampshire
31	NJ	New Jersey
32	NM	New Mexico
33	NY	New York
34	NC	North Carolina
35	ND	North Dakota
36	OH	Ohio
37	OK	Oklahoma
38	OR	Oregon
39	PA	Pennsylvania
40	RI	Rhode Island

<	Column 1	Column 2
>	Abbreviation:	State Name:
1	AL	Alabama
2	AR	Arkansas
3	CA	California
4	CO	Colorado
5	CT	Connecticut
6	DE	Delaware
7	FL	Florida
8	ID	Idaho
9	IL	Illinois
10	IN	Indiana
11	KY	Kentucky
12	MA	Massachusetts
13	MI	Michigan
14	NE	Nebraska
15	OH	Ohio
16	OK	Oklahoma
17	OR	Oregon
18	UT	Utah
19	WA	Washington
20	WI	Wisconsin
21	WY	Wyoming

<	Column 1	Column 2
>	Abbreviation:	State Name:
1	AK	Alaska
2	AZ	Arizona
3	DC	District Of Columbia
4	GA	Georgia
5	HI	Hawaii
6	IA	Iowa
7	KS	Kansas
8	LA	Louisiana
9	ME	Maine
10	MD	Maryland
11	MN	Minnesota
12	MS	Mississippi
13	MO	Missouri
14	MT	Montana
15	NV	Nevada
16	NH	New Hampshire
17	NJ	New Jersey
18	NM	New Mexico
19	NY	New York
20	NC	North Carolina
21	ND	North Dakota
22	PA	Pennsylvania
23	RI	Rhode Island
24	SC	South Carolina
25	SD	South Dakota
26	TN	Tennessee
27	TX	Texas
28	VT	Vermont
29	VA	Virginia
30	WV	West Virginia

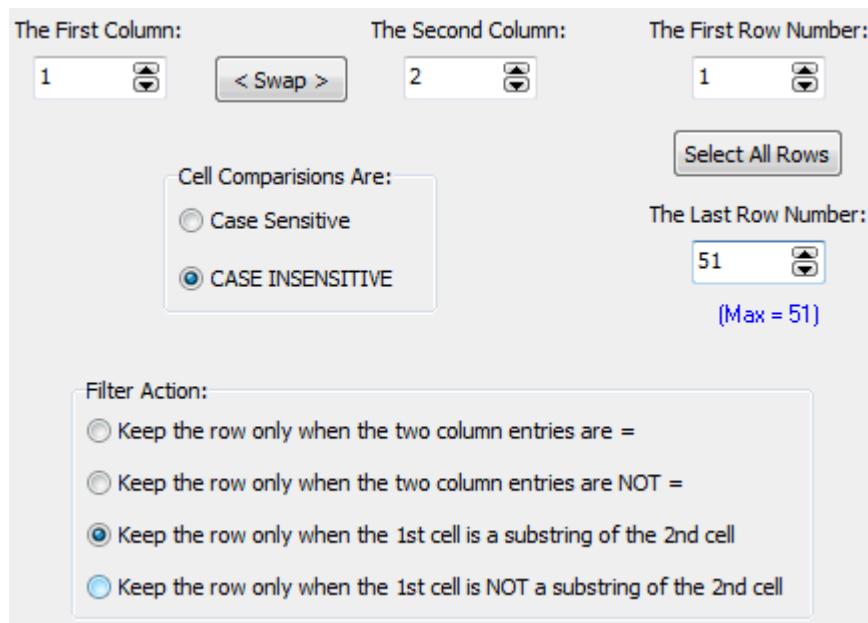
There are three lists shown on the previous page.

The left-most list is the original list of all the states, although only the first 40 show on the page.

The middle list shows the 21 states that have the property that their 2-letter abbreviation is contained within the name of the state. In fact, almost all of these 21 states are such that the 2-letter abbreviation is at the start of the state name. However, **Connecticut** and **Kentucky** are two examples where this is not the case. Can you spot the **CT** and **KY** in the respective state names?

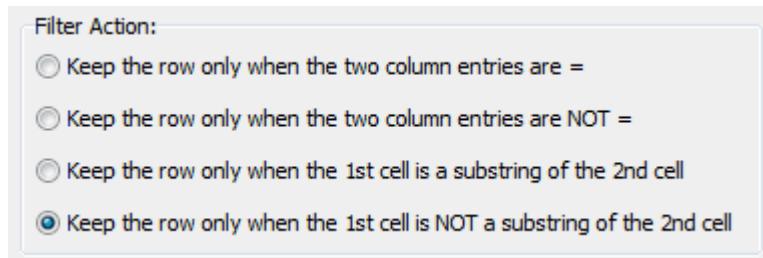
The third or right-most list shows the 30 states where the 2-letter abbreviation does not appear anywhere as part of the state name.

We created the middle list by starting with the original list on the left and we set the controls in the dialog as:



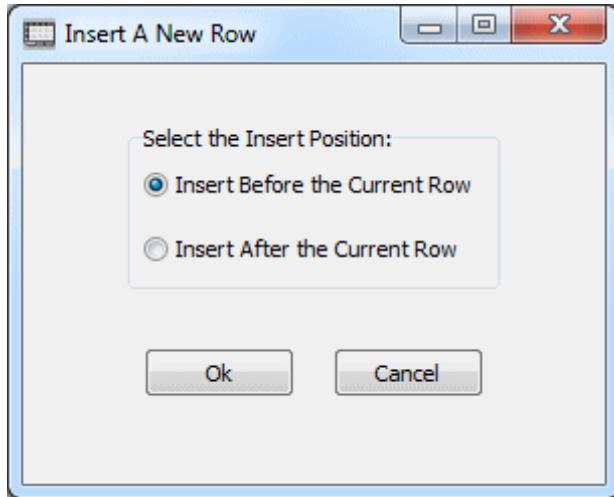
In particular note that we chose the **CASE INSENSITIVE** option.

When we created the right-most list, we also started with the original list and we set the controls as shown above, except we changed the **Filter Action** to the last choice.



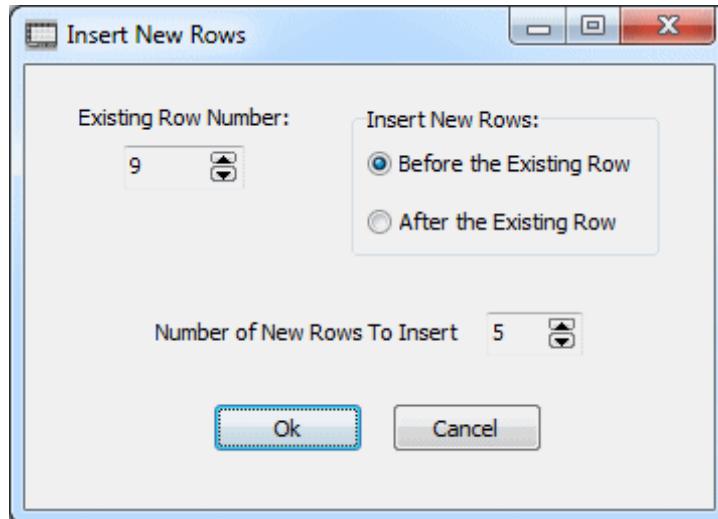
Inserting a New Row or Multiple Rows

If you need to insert a single row you should first select any entry in the nearest old row and then select the menu item **Rows | Insert a New Row...** or you can just click the button  in the toolbar. You will be prompted by the dialog that appears as:



If you click the **Ok** button the program will make a new blank row and it will renumber the other rows as needed.

To insert more than one row, select the menu item **Rows | Insert New Multiple Rows...** and then you should see a different dialog:



Interlacing a Range of Rows

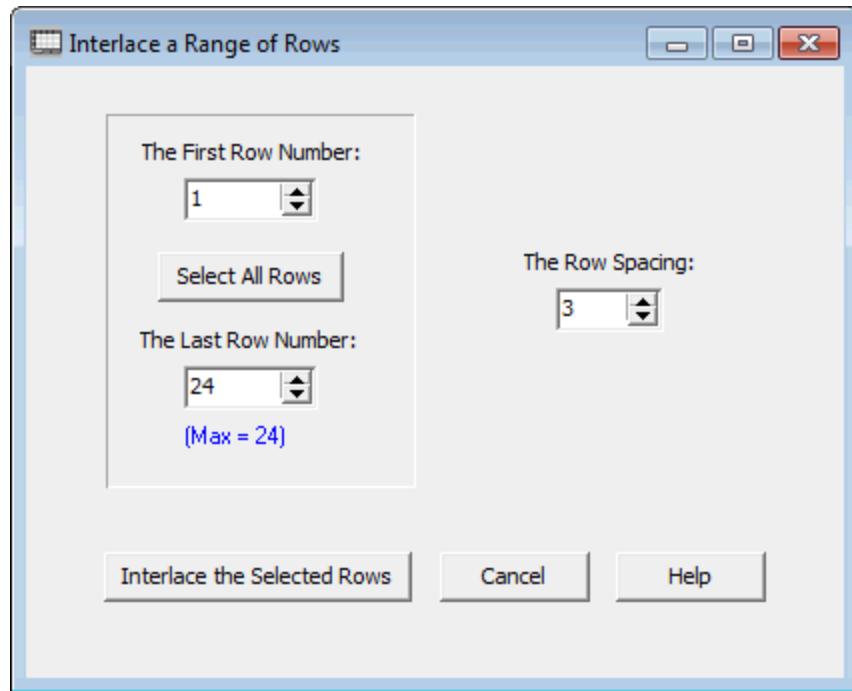
A menu item under the **Rows** menu with the title **Interlace a Range of Rows...** is used to specially re-order and re-group the rows of a grid. To understand what interlacing really does, assume you have a grid with 24 rows that appears as:

<	Column 1
1 >	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24

In the above grid we have entered the numbers 1-24 in order in those same rows that are numbered by the first yellow grid column.

If you select all these rows first, and then select **Rows | Interlace a Range of Rows...** you can fill out the dialog box that is shown on the following page.

The Row Spacing value must be a proper divisor of the number of selected rows.



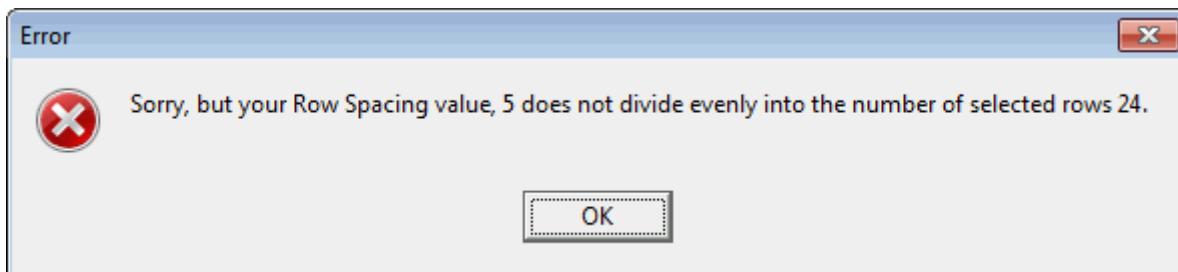
If you fill out the dialog as shown above, then when you click the button with the caption **Interlace the Selected Rows** you should see the grid on the previous page change to the following:

<	Column 1
1 >	1
2	4
3	7
4	10
5	13
6	16
7	19
8	22
9	2
10	5
11	8
12	11
13	14
14	17
15	20
16	23
17	3
18	6
19	9
20	12
21	15
22	18
23	21
24	24

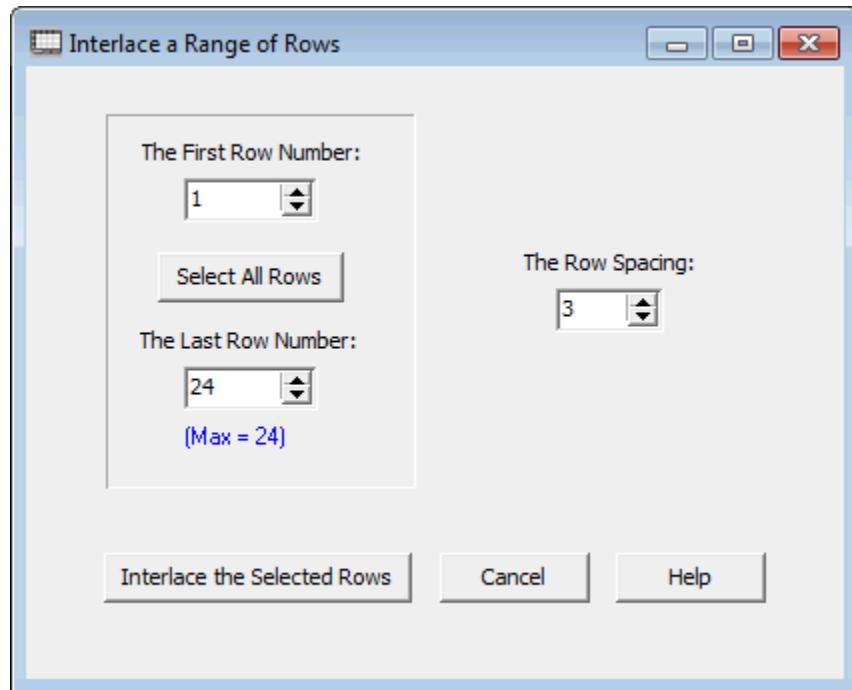
By comparing the grids on the two previous pages you can begin to understand how the rows have been re-ordered. The interlacing effect is to select 8 rows at a time, but every 3 rows are selected in order. Thus the first eight rows have the row number pattern 1 4 7 10 13 16 19 22. Each next row number is 3 more than the previous row number, starting with row number 1. The next group of 8 rows begins with the number 2 and the row numbers are 2 5 8 11 14 17 20 23 where the row numbers again all differ by 3. The last group of 8 rows starts at 3 and is 3 6 9 12 15 18 21 24 and all these differ by 3.

You should now begin to understand the meaning of the term **The Row Spacing**. The **Row Spacing** value must always divide evenly into the number of rows that are selected for interlacing. In fact it must be a proper divisor that is greater than 1 and less than the number of selected rows. In this example, because $24 \div 3 = 8$, there are 3 groups of 8 rows where the rows in each group are spaced three rows apart.

If you try to enter a **Row Spacing** value that does not divide evenly into the number of selected rows, you will generate an error message like the following:



Next we show a more practical example of performing interlacing. The grid on the following page is the Academy Awards grid in which we have repeated three rows of information, 8 times, making a total grid size of 24 rows. Now if we setup the **Interlace a Range of Rows** dialog as shown next,



then when we apply interlacing, the grid below will change to the grid shown on the following page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
3	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
4	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
5	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
6	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
7	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
8	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
9	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
10	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
11	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
12	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
13	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
14	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
15	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
16	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
17	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
18	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
19	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
20	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
21	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
22	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
23	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
24	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
10	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
11	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
12	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
13	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
14	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
15	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
16	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
17	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
18	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
19	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
20	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
21	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
22	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
23	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
24	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T

We can see the effect of interlacing is to re-group the rows 8 at time with consecutive years running from 1960 through 1967 inclusive. The same information is now repeated 3 times in groups of 8 rows.

De-Interlacing Rows

We will finish our discussion of interlacing by discussing the inverse process that can be called de-interlacing. What is most peculiar is that we don't need to have a separate de-interlacing function because interlacing can be undone by selecting the same set of consecutive rows, but using the number that results when we divide the number of rows by the **Row Spacing** value.

In all of the examples we have given so far, we selected 24 rows and we set the **Row Spacing** value to 3. We previously noted that $24 \div 3 = 8$. But it is just as true that $24 \div 8 = 3$. Now the big surprise is that if we start with the grid on the left as shown on the next page, then all we need do is use a **Row Spacing** value of 8 to reverse the interlacing effect.

<	Column 1
1 >	1
2	4
3	7
4	10
5	13
6	16
7	19
8	22
9	2
10	5
11	8
12	11
13	14
14	17
15	20
16	23
17	3
18	6
19	9
20	12
21	15
22	18
23	21
24	24

<	Column 1
1 >	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24

The left grid changes to the right grid → when the **Row Spacing** value used is 8 instead of 3.

If you had a grid with 25 consecutive rows, and if you applied interlacing with a **Row Spacing** value of 5, you would get back to where you started if you applied interlacing twice. The second time would actually perform the inverse of interlacing that could be considered as de-interlacing.

Thus which of the two values you might use for **Row Spacing** has an inverse effect. In the example with 24 rows, the numbers 3 and 8 make a pair that can be described as a divisor and a quotient.

Other pairs that are divisors and quotients for the number 24 might be 4 and 6 or 2 and 12. One of these numbers determines the row spacing which is also the number of groups, and the other number can determine how many rows are in a group.

If you switch the roles of the matching divisor and quotient pair of numbers, you can generate the inverse effect that is de-interlacing.

To better illustrate the relationships between interlacing and de-interlacing we will go back to an example we gave earlier where we did **Block-Column Reshaping**. The first grid on the left below contains the GPS coordinates and the state locations of various cities in the world. The list is in one tall single column where each city name is followed by 3 lines of information related to that city. For this example the list has 1032 rows that contain the information for 258 cities. Note that $1032 \div 4 = 258$. In this example, the divisor number is 4 and the quotient number is 258.

If we perform interlacing with a **Row Spacing** value of 4, the single column will change to the middle grid below in which the effect of interlacing is to separate out the 258 city names first, followed by the 258 latitude coordinates, followed by the 258 longitude coordinates, followed by the 258 state names. In other words, there are 4 groups with 258 rows in each group.

The second and third grids below are really part of the same grid. The second or middle grid shows the first 20 rows that are city names. The third grid on the right shows the last 20 rows of the same middle grid single column where you can see the last 20 location state names or country names.

If we were to perform interlacing on the third grid, with a **Row Spacing** value of 258, the effect would be to interlace all the city information and put it back the way it started out as the single column on the left with 1032 rows where each city name is followed by 3 pieces of information related to that city. So even though we would call this another example of using the interlacing function, with the value of 258 instead of 4, the real effect would be the inverse of interlacing and that would be de-interlacing.

<	Column 1
1 >	Adelaide
2	-34.921971
3	138.581543
4	Australia
5	Akron
6	41.08
7	-81.52
8	Ohio
9	Albuquerque
10	35.12
11	-106.62
12	New Mexico
13	Alexandria
14	38.82
15	-77.09
16	Virginia
17	Amarillo
18	35.20
19	-101.82
20	Texas

<	Column 1
1 >	Adelaide
2	Akron
3	Albuquerque
4	Alexandria
5	Amarillo
6	Anaheim
7	Anchorage
8	Arlington
9	Arlington
10	Atlanta
11	Auckland
12	Augusta-Richmond
13	Aurora
14	Aurora
15	Austin
16	Baghdad
17	Bakersfield
18	Baltimore
19	Bangkok
20	Barcelona

<	Column 1
1013	Arizona
1014	Japan
1015	Ohio
1016	Canada
1017	California
1018	Libya
1019	Arizona
1020	Oklahoma
1021	Washington
1022	Canada
1023	Virginia
1024	Russia
1025	Michigan
1026	Poland
1027	District of Columbia
1028	Utah
1029	Kansas
1030	North Carolina
1031	Massachusetts
1032	New York

Move Current Row to the Bottom or the Top

Selecting the menu item **Rows | Move Current Row ►** gives you a two-submenu choice as to position:



You can move the current row to the bottom of the grid or you can move it to the top of the grid. No prompting is needed for this operation. You can immediately undo this by clicking the **Undo** button in the toolbar.

This action is often used when you find bad data rows in a grid. By moving the bad rows to the bottom or the top you can separate the good from the bad. After that, you can concentrate on either group. Of course any other criteria can be applied to place a set of rows at the extremes of the grid.

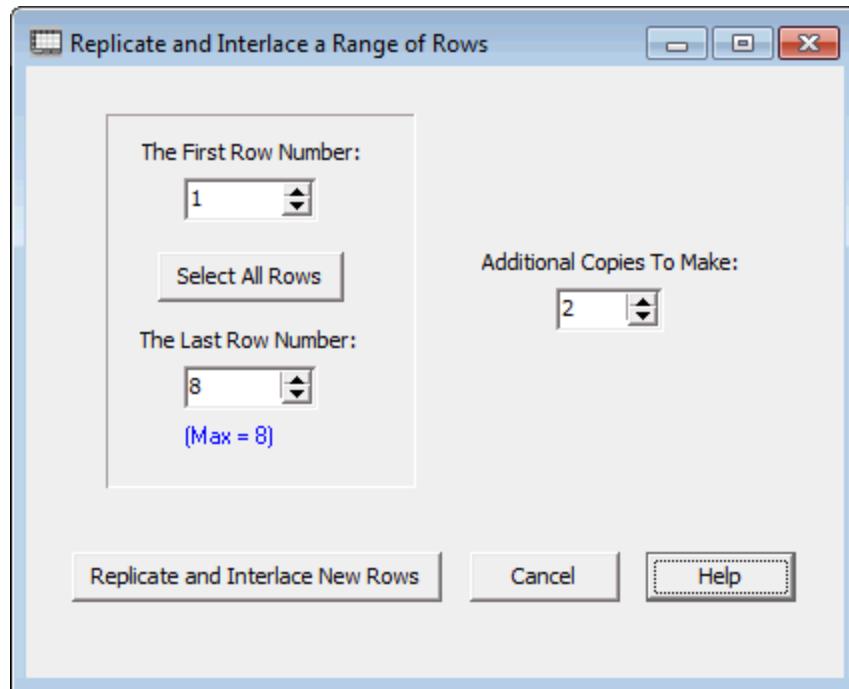
Replicate and Interlace a Range of Rows

You can select **Rows | Replicate and Interlace a Range of Rows...** to have the program make a number of copies of a range of rows and at the same time interlace the new rows with the selected range of rows. The end result will be a series of rows that repeat the same data, usually in a small number of groups of rows.

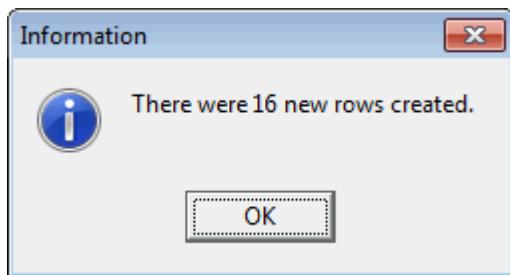
As an explanatory example, suppose we open the **Academy Awards** grid, but we limit the grid to only 8 distinct rows of original data. The original grid we will work on might initially look like the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T

We then select the menu **Rows | Replicate and Interlace a Range of Rows...** and we should see the following dialog:



If we leave the parameters in the dialog as shown above, then after we click the button to **Replicate and Interlace New Rows** then we will first see a message telling us how many new grid rows needed to be created.



The following shows the resulting new grid where the rows have all been interlaced.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
3	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
4	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
5	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
6	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
7	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
8	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
9	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
10	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
11	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
12	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
13	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
14	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
15	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
16	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
17	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
18	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
19	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
20	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
21	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
22	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
23	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
24	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T

Note how each data row is now repeated 3 consecutive times (we made 2 additional copies of the original data), and the new rows are interlaced into the existing grid. The result is 8 groups of 3 repeated rows making 24 total rows of data. One case where this replicating and interlacing is needed is when the **CSV Editor** program is used to make data files for multiple copies of labels that will eventually get printed on sheets of labels.

Before applying this function you can select cells that cover an original range of rows. This will select the rows that will be duplicated so you should not have to make any adjustments to the range of rows when you open the setup dialog for this function. In the example on the previous page we selected the first column before we opened the dialog.

We should make some additional comments regarding how the new data rows get replicated. In fact, we generally assume the selected rows in your grid are all distinct rows, although this is not a requirement. Your rows may also be sorted by any particular order you wish the rows to have. Such a sorted order will be maintained in the grouped duplicate rows that are the result of applying this function.

First, you need to know that the whole operation will need space to insert the replicated data. If the grid is not sufficiently large to accommodate the additional rows of new data, then the grid will automatically be expanded, up to the 50,000 row limit. Second, the replications start getting copied in the first row following the last row of the selected range of rows. This may mean that some existing data can be overwritten by the new data copies unless you first insert a sufficient number of blank rows before you apply this function. Third, this function does not change and does not use any Header row information. If you have a Header row it will remain unchanged and unused by this function.

The number of copies you make will usually be relatively small, perhaps only 2 or 3, with perhaps 5 or more being unusual. In any case, the number of additional copies you can create is limited to at most 99. The smallest number of copies you can make is only 1. In the above example we made 2 additional copies which meant the rows repeated 3 times because the original row data is always used as the first row in each group of the repeated data rows. To double the selected rows you should make only 1 additional copy. To quadruple the selected rows you make only 3 additional copies. We can summarize by stating that the number of additional copies you should make will always be 1 less than the final number of duplicate rows you expect each group of duplicate rows to have.

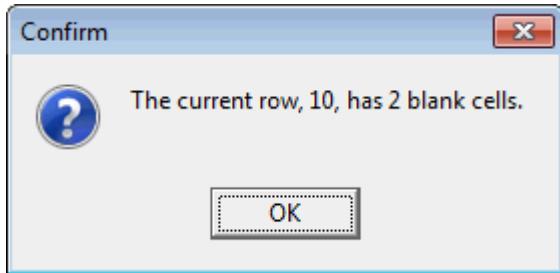
If any new rows need to be added to the grid, then there is a remote possibility that the 50,000 grid row limit may be exceeded. If that is the case, you will first see a warning dialog that will give you the choice of creating an unusually large number of rows, or you will be able to abort the entire operation.

If you want to replicate a range of rows (or a single row), any number of times, but don't want the interlacing effect, then you might consider using the Block Replication function under the menu **Blocks | Block Replicate....**

Report Blank Cells in the Current Row

You can select **Rows | Report Blank Cells in the Current Row...** to have the program count and report the number of blank cells that are in the current row.

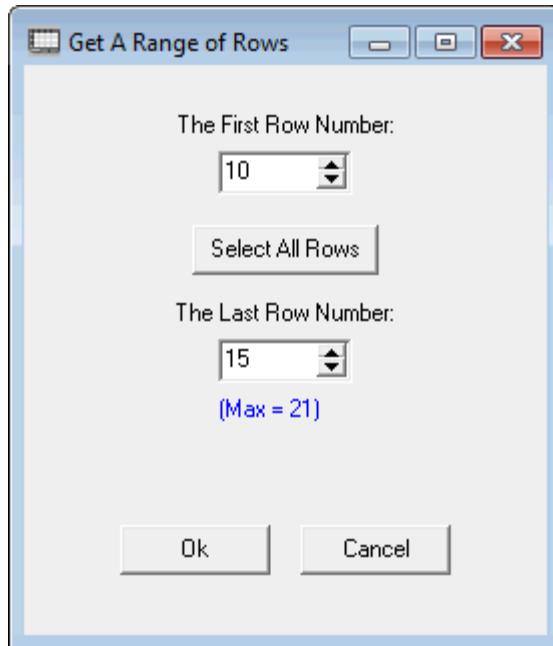
A typical report might appear like the following:



Before applying this function you should click on any cell in the row whose entries you want to check; this will select that row as the current row.

Scrambling a Range of Rows

When you execute the menu item **Rows | Scramble a Range of Rows...** you will first see the dialog to select the range of consecutive rows to be scrambled.



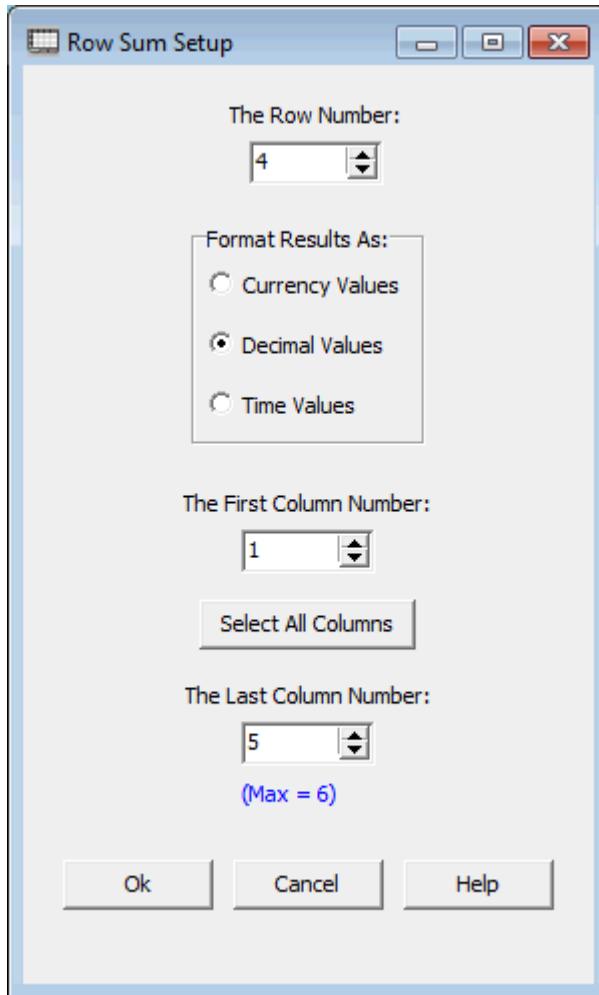
If you click **Cancel**, this function just immediately aborts.

If you click **Ok**, you may see a message telling you to wait while the grid is being scrambled (actually sorted). This will be true when you select a large range of rows. In any case, when the sort finishes the selected set of rows will be scrambled. This means the ordering of the selected rows is now completely random compared to what it was before the scrambling.

Internally this function works by first adding a new last column to the grid, then it fills the selected rows in that column with random numbers. It finishes by sorting the selected rows in the last column and then it deletes the last column. If your grid already has the 200 maximum of columns when you try to apply this function, then you will see an error message that warns you that you cannot apply this function.

Summing the Entries In a Row

The function to sum and average the entries in a row works exactly analogous to the column sum function previously discussed. The only difference is that you work in a single row and sum the entries in a contiguous set of columns, where those columns run from left to right. In other words, the first column number is always smaller than the last column number.

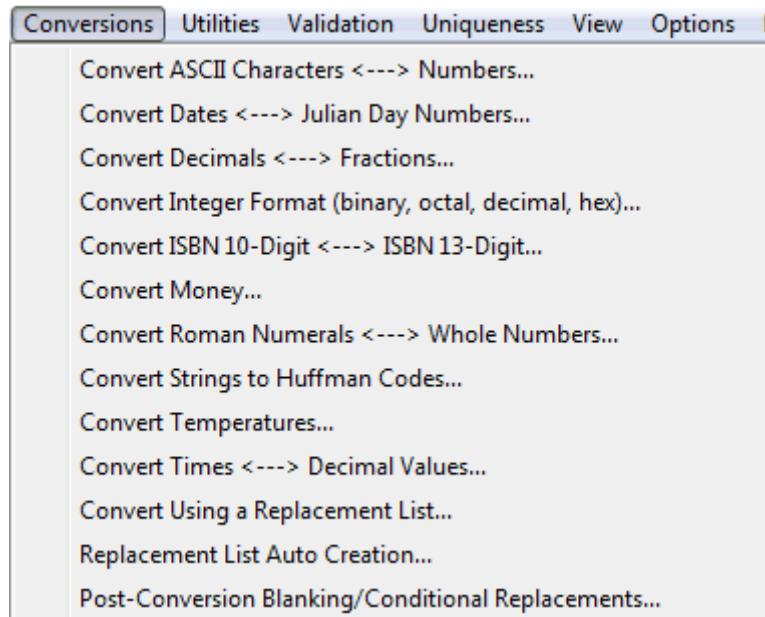


As with summing the entries in a column, the sum answer is also left on the Windows clipboard. If you want to insert the sum answer into another grid cell, just select that other cell for editing and then type **CTRL+V** on your keyboard to paste the sum answer into that cell. Note also that the result depends on the nature of the numbers contained in the grid, but those numbers can be either currency values, or decimal values, or elapsed time values.

If you need to perform a more detailed statistical analysis of your data then you should consider the functionality that is available for performing **Linear Regression**. Select **Utilities | Linear Regression Report...** and when the dialog opens click the **Help** button.

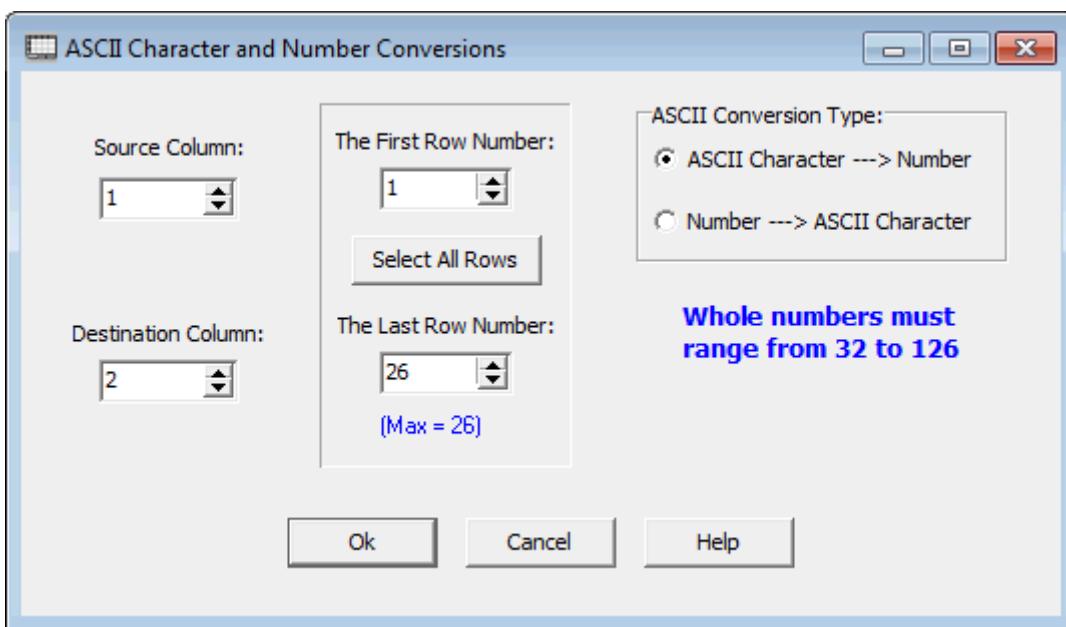
The Conversions Menu

The **CSV Editor** program has a menu devoted to converting data from one form to another. The current **Conversions** menu has thirteen somewhat non-traditional or unusual submenu items.



Conversions With ASCII Characters and Numbers

When you select the menu item **Conversions | Convert ASCII Characters <--> Numbers...** you will bring up the following dialog box that is used to convert between decimal integers and ASCII characters. The whole subject of how characters are represented inside a computer is technical and has an interesting history. The history is also related to the telegraph, where characters were first represented as a series of dots and dashes. In current practice, if you ever hear of something called Unicode characters then you will be on the verge of topics related to this function. The Unicode character scheme is a newer extension of another scheme that is referred to as the ASCII scheme. ASCII stands for American Standard Code for Information Interchange. In any case, the dialog that can be used to convert between whole numbers and single ASCII characters is the following.



As with most of the dialogs for conversions, the above dialog is used to define a **Source Column** and a **Destination Column**, and a set of consecutive rows. For this ASCII Conversion type of function, the source and destination columns can actually be the same column.

The above dialog is used to define a set of consecutive rows in a **Source Column**. Those corresponding cells in the **Destination Column** will automatically be converted to either a numeric form or a character form. Before we give examples of using the above dialog, we will first discuss how characters are represented as numbers inside a computer.

If you type the three letters **ABC** on your computer keyboard, somewhere inside the computer will be stored the three whole numbers **65**, **66**, and **67** that correspond to these three characters. In fact, whenever you type any character from a computer keyboard, that character will be represented by a whole number that is in the range between **32** and **126** inclusive. The space character corresponds to the number **32** and the tilde character **~** corresponds to the number **126**.

There is a scheme that all modern computers use to represent characters in computer memory. We will refer to this scheme using a special table we call the ASCII table. The ASCII table is very technical in nature, but we show it below so you can see for yourself how all characters that are typed on a computer keyboard can be converted to integers in the range between **32** and **126**. Incidentally, the row and column headers for this table and the interior red numbers can all be used to quickly convert any integer between and among any of the popular binary, decimal and hexadecimal formats.

ASCII Table	0000 0	0001 1	0010 2	0011 3	0100 4	0101 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
0000 0	0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL	8 BS	9 HT	10 LF	11 VT	12 FF	13 CR	14 SO	15 SI
0001 1	16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB	24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
0010 2	32 space	33 !	34 "	35 #	36 \$	37 %	38 &	39 '	40 (41)	42 *	43 +	44 ,	45 -	46 .	47 /
0011 3	48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7	56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
0100 4	64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G	72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
0101 5	80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W	88 X	89 Y	90 Z	91 [92 \	93]	94 ^	95 _
0110 6	96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g	104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
0111 7	112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w	120 x	121 y	122 z	123 {	124 	125 }	126 ~	127 DEL
1000 8	128 €	129 ,	130 „	131 ƒ	132 „	133 …	134 †	135 ‡	136 ^	137 „	138 Š	139 „	140 Œ	141 Ž	142 ž	143 č
1001 9	144 ‘	145 ’	146 „	147 „	148 •	149 –	150 –	151 –	152 ~	153 „	154 š	155 „	156 œ	157 ž	158 č	159 ѣ
1010 A	160 ı	161 ı̄	162 ç	163 ȝ	164 ȝ̄	165 ‐	166 ‐̄	167 ‐̄̄	168 ‐̄̄̄	169 ‐̄̄̄̄	170 ‐̄̄̄̄̄	171 ‐̄̄̄̄̄̄	172 ‐̄̄̄̄̄̄̄	173 ‐̄̄̄̄̄̄̄̄	174 ‐̄̄̄̄̄̄̄̄̄	175 ‐̄̄̄̄̄̄̄̄̄̄
1011 B	176 °	177 ±	178 ²	179 ³	180 ⁴	181 µ	182 ¶	183 ·	184 ·̄	185 ·̄̄	186 ·̄̄̄	187 ·̄̄̄̄	188 »	189 ¼	190 ¾	191 ጀ
1100 C	192 À	193 Á	194 Â	195 Ã	196 Ä	197 Å	198 È	199 É	200 Ù	201 Ó	202 Ô	203 Û	204 Í	205 Î	206 Ӯ	207 Ӯ
1101 D	208 Ђ	209 Њ	210 Ѡ	211 Ӯ	212 Ӯ	213 Ӯ	214 Ӯ	215 Ӯ	216 Ӯ	217 Ӯ	218 Ӯ	219 Ӯ	220 Ӯ	221 Ӯ	222 Ӯ	223 Ӯ
1110 E	224 à	225 á	226 â	227 ã	228 ä	229 å	230 æ	231 ç	232 è	233 é	234 ê	235 ë	236 ù	237 í	238 î	239 Ӯ
1111 F	240 ð	241 ñ	242 ò	243 ó	244 ö	245 ȫ	246 ÷	247 ø	248 ø̄	249 ø̄̄	250 ø̄̄̄	251 ø̄̄̄̄	252 ø̄̄̄̄̄	253 ø̄̄̄̄̄̄	254 þ	255 ӱ

The red numbers in this table are the whole numbers that we are referring to when we talk about any decimal integer that corresponds to a character. If you read the red numbers from **65** to **79** that are all in the same row in the above table you can quickly understand that these numbers correspond to the capital letters from A to O inclusive. The capital letters from P through Z continue in the next row and correspond to the numbers from **80** to **90**.

Each row in the above table has 16 entries and the table as a whole has 16 rows. $16 \times 16 = 256$. The red numbers in the table run from **0** to **255** and thus are associated with 256 distinct values. These are the values associated with one byte of data that consists of 8 consecutive bits.

When reading the table we usually start with the row label and follow that with the column label. The ASCII character **A** has a decimal value of **65** but this is easily seen to be the same as hexadecimal **41** (row **4** column **1**) and that is the same as the 8-bit sequence 01000001 where we read hexadecimal **4** as binary 0100 and we read hexadecimal **1** as binary 0001 and we just concatenate the two groups of 4 binary bits to get an 8-bit value.

The space character and other special characters are in the row with the red numbers from **32** to **47**. If you are expecting to see some kind of logical arrangement here, don't. Although it is true that both upper and lower case alphabetical letters are arranged in order, other characters like the punctuation characters may appear in a somewhat random and unrelated order.

Interestingly enough, whenever you type on your keyboard a single numeric digit in the range from **0** through **9**, a computer stores the corresponding character as a whole number in the range from **48** to **57**. The decimal point character corresponds to the whole number **46**. The comma character corresponds to the whole number **44**. In practice, when you type a sequence of characters for a money value like **\$8,356.42** the computer will first enter each typed character as its red number in the above ASCII table before it converts the entire string you type into another number format for performing computations with numbers. All we mean to say here is that internally computers use numbers in many different formats. By the way, the **\$** character corresponds to the whole number **36**.

The red numbers in the above table range from **0** to **255**. We should state however, that the so-called characters in the first two rows are not really characters at all and have many special uses in other systems. Also, the numbers in the table from **127** to **255** have other special uses and it is probably unfair to associate these values with actual characters. The **CSV Editor** program only uses the values from **32** to **126** exclusively. Anything outside of this range is subject to multiple interpretations and because of this misinterpretations are often available.

The characters we use can be interpreted in the Unicode system using the **UTF-8** transformation rules. If you are not familiar with Unicode or **UTF-8** then we highly recommend you read the entire help topic in this help file with the title **Custom Read-Write Parameters**. You could also just select **File | Custom File Read...** and then click the **Help** button in the dialog that appears. However, you should normally never use our custom file read and write operations until you fully understand what is described in that other help topic. You don't need to know anything about Unicode or **UTF-8** to use the functions for converting between ASCII characters and integer numbers.

If you ever need to refer to the above table when you are using the **CSV Editor** program, the quickest way to do so is with the **Help** menu item **Show the ASCII Table of Characters....** As we have already noted, there are entries in the above table that don't correspond to characters you ordinarily type on an English keyboard and see in English documents. Such entries may have purposes in other computer programs and systems such as in using foreign languages with foreign language keyboards. We won't discuss other special purposes here.

Now that we have established some necessary background, we can show a couple of examples that involve ASCII characters and number conversions.

Suppose you have a grid column with 26 rows in which you want to put the letters from A-Z. Although you could type these letters one at a time, a faster way to do it is to just select the 26 rows, then perform a **Block Fill** using Automatic Numbers that start with the number 65.

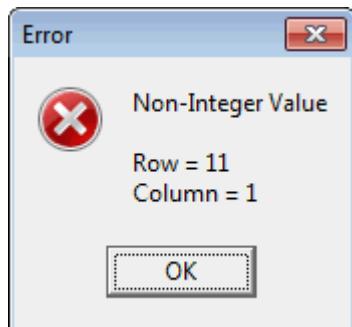
Your source column will then look like Column 1 shown below on the left. Then if you select the entries in this column and open the above dialog and check the **ASCII Conversion Type** radio button to do **Number ---> ASCII Character** and click **Ok** you should see the destination Column 2 change to what we show next on the right.

<	Column 1	
1 >	65	
2	66	
3	67	
4	68	
5	69	
6	70	
7	71	
8	72	
9	73	
10	74	
11	75	
12	76	
13	77	
14	78	
15	79	
16	80	
17	81	
18	82	
19	83	
20	84	
21	85	
22	86	
23	87	
24	88	
25	89	
26	90	

<	Column 1	Column 2
1 >	65	A
2	66	B
3	67	C
4	68	D
5	69	E
6	70	F
7	71	G
8	72	H
9	73	I
10	74	J
11	75	K
12	76	L
13	77	M
14	78	N
15	79	O
16	80	P
17	81	Q
18	82	R
19	83	S
20	84	T
21	85	U
22	86	V
23	87	W
24	88	X
25	89	Y
26	90	Z

This is the fastest way to make consecutive characters in a given column without doing any real typing.

What if a cell contains a number that is not in the range from **32** to **126**? In that case the program will display an error message (assuming this is the first error) and leave the destination cell alone. If a cell in your selected source column does not represent a valid number then you will see an error message as shown below. If any error occurs you should correct the improper cell entry and then try again.



At the end of the conversion process, if you have two or more errors you will see a message that gives the total count of all errors. Only the first error found will display a message like that shown above that identifies the improper cell.

Next, we want to give an example that performs a conversion in the other direction. Before we do that we explain that whenever you convert a grid cell to an ASCII whole number value, the program will only use the first character in the cell. Second, if a cell is empty then that cell is simply ignored.

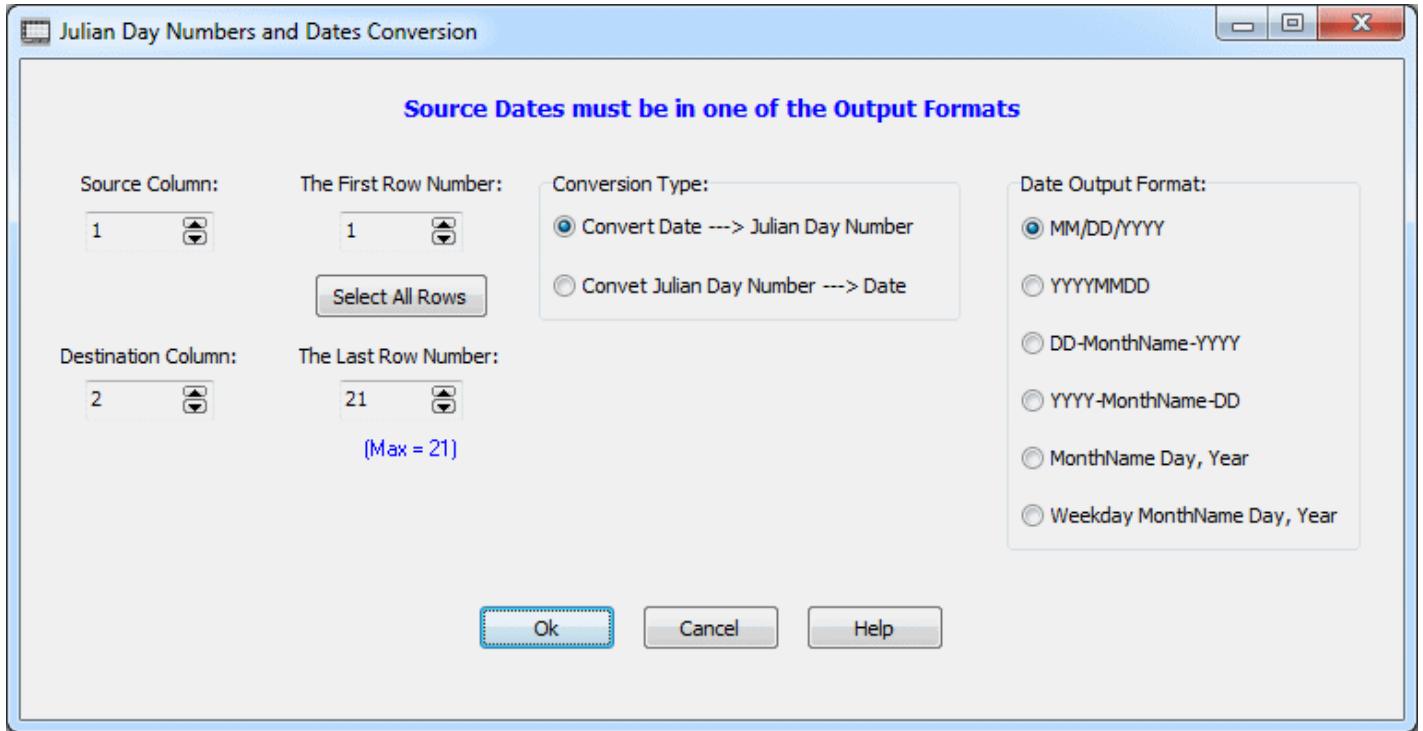
Our final example is shown by the two columns below. The column on the left contains names of Academy Awards winners. If this column is selected and you open the conversion dialog and choose the **ASCII Conversion Type** radio button with the caption **ASCII Character ----> Number** then after you click **Ok** you should see the destination column change to what is shown below on the right. In this example we chose column 6 as the destination column. Only the first character in the name gets used when the whole cell is converted to a single number.

Column 3	Column 6
Best Actress	Tagged
Elizabeth Taylor	69
Sophia Loren	83
Anne Bancroft	65
Patricia Neal	80
Julie Andrews	74
Julie Christie	74
Elizabeth Taylor	69
Katherine Hepburn	75
Katherine Hepburn	75
Maggie Smith	77
Glenda Jackson	71
Jane Fonda	74
Liza Minelli	76
Glenda Jackson	71
Ellen Burstyn	69
Louise Fletcher	76
Faye Dunaway	70
Diane Keaton	68
Jane Fonda	74
Sally Field	83
Sissy Spacek	83

Having the ability to convert between numbers and ASCII characters may be rarely used, but when it is needed, it is a very valuable function to have.

Conversions With Julian Day Numbers and Dates

The function that deals with date conversions is used to convert between dates and Julian Day Numbers. When you select the menu item **Conversions | Convert Dates <--> Julian Day Numbers...** you will bring up a dialog box like the following.



You must choose two essential columns. The **Source Column** is assumed to contain the values that will be converted. The **Destination Column** is the column that will contain the new computed values.

In converting from dates to Julian Day Numbers, we assume each **Source Column** date cell is in one of the six possible formats that are shown as **Date Output Format**. These are the only date formats that should be used when the **Source Column** contains dates. You don't actually choose a particular format for source dates. The **Date Output Format** radio buttons can be used to choose any one of six different output formats when we convert from Julian Day Numbers back into dates. Thus the **Date Output Format** selects only 1 possible output, where as source dates are not restricted to any particular format as long as they are one of the six possibilities.

When converting dates, it is possible that your **Source Column** may contain an invalid date. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first invalid date. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

Julian Day Numbers are special real numbers that are used to count days from a time way in the past like January 1, 4713 B.C. We denote these numbers using the abbreviation **JDN** and we point out that most calculations involving dates are greatly simplified using **JDNs**.

For example, given a particular year, if you tried to determine the date that is 3650 days past February 1 for that year, you would probably find this is not an easy problem. One of the main difficulties would involve determining how to count February 29, or not, depending on whether any intervening year was a leap year. Many modern financial calculations depend on the ability to determine the exact number of days between any two dates. In fact, the number 3650 might represent the number of days in a 10-year loan.

JDNs are also used by astronomers, but we won't concern ourselves with any real technical details. We only point out that **JDNs** extend from noon on one day to noon on the next day. This rule avoids having a **JDN** change during the night when most astronomers work. We also state that **JDNs** have nothing to do with the so-called Julian Calendar. We won't discuss calendars at all, primarily because they can also be complicated.

In modern times, we can state that the **JDN** of the day stretching from noon September 11 to noon September 12, 2001 is the integer 2,452,164.

The following grid shows some random dates in Column 1 and the corresponding **JDNs** for those dates are in Column 2.

<	Column 1	Column 2
>	Random Date:	Julian Day Number:
1	10/03/2002	2452551
2	11/05/2001	2452219
3	05/17/2003	2452777
4	01/09/2010	2455206
5	01/25/2005	2453396
6	06/13/2016	2457553
7	05/09/2008	2454596
8	05/25/2005	2453516
9	03/05/2016	2457453
10	02/20/2003	2452691
11	01/17/2013	2456310
12	06/28/2011	2455741
13	12/05/2018	2458458
14	03/27/2003	2452726
15	11/18/2017	2458076
16	04/22/2011	2455674
17	10/22/2014	2456953
18	05/17/2014	2456795
19	01/10/2011	2455572
20	05/04/2006	2453860

In practice, **JDNs** are used to perform all kinds of calculations that involve the counting of days. We will give one example in which we start with the first **JDN** given in the above list. We simply make a new list that is a sequence of **JDNs** that are all 17 days apart. These are in Column 1 in the list below.

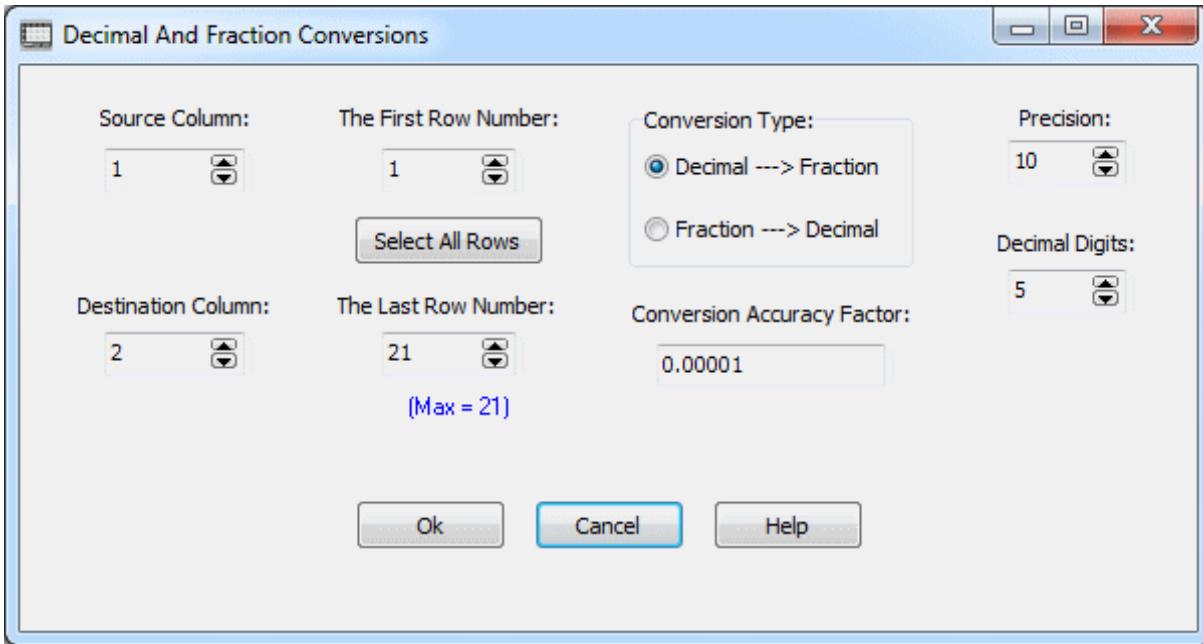
In Column 2 we have converted these **JDNs** back into dates that all have the same date output format.

<	Column 1	Column 2
>	Julian Day Number:	Converted Date:
1	2452551	Thursday October 03, 2002
2	2452568	Sunday October 20, 2002
3	2452585	Wednesday November 06, 2002
4	2452602	Saturday November 23, 2002
5	2452619	Tuesday December 10, 2002
6	2452636	Friday December 27, 2002
7	2452653	Monday January 13, 2003
8	2452670	Thursday January 30, 2003
9	2452687	Sunday February 16, 2003
10	2452704	Wednesday March 05, 2003
11	2452721	Saturday March 22, 2003
12	2452738	Tuesday April 08, 2003
13	2452755	Friday April 25, 2003
14	2452772	Monday May 12, 2003
15	2452789	Thursday May 29, 2003
16	2452806	Sunday June 15, 2003
17	2452823	Wednesday July 02, 2003
18	2452840	Saturday July 19, 2003
19	2452857	Tuesday August 05, 2003
20	2452874	Friday August 22, 2003

One last technical point is that when we read a **JDN** from a grid cell, we compute the integer part of the absolute value of that number. Thus you can use **JDNs** with fractional or decimal parts, but those parts will be ignored. No **JDN** is allowed to be a negative number in our system. In fact, the only proper **JDNs** that we work with are all in the range between 1,721,060 (=01/01/0000) and 5,373,484 (=12/31/9999). A **JDN** with a decimal part might represent a date with a time, in which the decimal part denotes a fractional part of a 24-hour day. Such real numbers can represent time accurately to the nearest second, or even better. However, the **CSV Editor** program doesn't use decimal times with **JDNs**.

Conversions Between Decimals and Fractions

When you select the menu item **Conversions | Convert Decimals <---> Fractions...** you will see the following dialog box.



The main controls in this dialog are the two that select a **Source Column** and a **Destination Column**. In other words, when you convert, you must have a source string and you must produce another destination string. The purpose of the **Source Column** and the **Destination Column** is to allow you identify where the source numbers are and to identify where what you will compute must go.

The **Conversion Type** radio buttons only give you two choices. You must either convert from a Decimal to a Fraction or you must convert from a Fraction to a Decimal. Let's first discuss what we mean by these types of conversions.

We assume most people know that if you are given fraction like $1/3$, this fraction can be converted to a repeating decimal value like $0.\overline{3}333333333$. The length that we write for the decimal depends on the precision that we like to limit our decimal values to have. In fact, converting fractions to decimals is relatively easy because essentially all that is required is to perform a simple division operation to get the decimal result.

When most fractions get converted to decimals, the result is usually a repeating decimal, as in the example just given. Only very special fractions like $1/4$, and $3/1000$, result in terminating decimals like 0.25 and 0.003 . The secret here is that when the fraction denominator has prime factors that consist only of the prime numbers **2** or **5** then the resulting fraction is a terminating decimal.

Less well-known is that when we are given a non-repeating and essentially a non-terminating decimal value like $\pi = 3.14159265\cdots$, this decimal can be approximated by infinitely many fractions. In the table below we list the first nine fractions that can be described as the very best fractions that approximate the value of π .

<	Column 1	Column 2
>	Fraction:	Divided Decimal:
1	3/1	3.0000000000000000
2	22/7	3.142857142857143
3	333/106	3.141509433962264
4	355/113	3.141592920353982
5	103993/33102	3.141592653011903
6	104348/33215	3.141592653921421
7	208341/66317	3.141592653467437
8	312689/99532	3.141592653618937
9	833719/265381	3.141592653581078

When you study this table you will begin to understand that larger fractions are required to make more accurate representations of the true value for π . The fraction **22/7** is accurate to only two decimal places while the fraction **104348/33215** is accurate to about nine decimal places.

In the dialog on the previous page, the purpose of the **Accuracy Factor** is to help determine the best fraction that approximates a given decimal, without returning fractions whose numerator or denominator or both, are too large to be practical. In general, the algorithm we use to convert a decimal to a fraction will keep generating fractions with larger numerators and denominators, until we stop when the difference between the divided fraction and the evaluation decimal is less in absolute value than the accuracy factor number. If you select smaller accuracy factors then the program will return larger fractions.

When converting either way, it is possible that your **Source Column** may contain an invalid string for a number. That number could be either an invalid decimal value or an invalid fraction value. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first invalid number. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

It is only when you convert fractions to decimals that the program will use the two values denoted by the labels **Precision** and **Decimal Digits**. The **Precision** value basically tells how many total significant digits to limit the value (minimum is 2; default is 10; maximum is 18). The **Decimal Digits** value tells how many digits to write after the decimal point (minimum is 0; default is 5; maximum is 15). Decimal values that are computed for output get written using the **Precision** and **Decimal Digits** values.

When we converted the fractions to decimals as shown in the above table, we set the **Precision** to 18 and we set the **Decimal Digits** to 15. In general, when decimals are converted to fractions, the **Precision** and **Decimal Digits** values are not used. Only the **Accuracy Factor** is used.

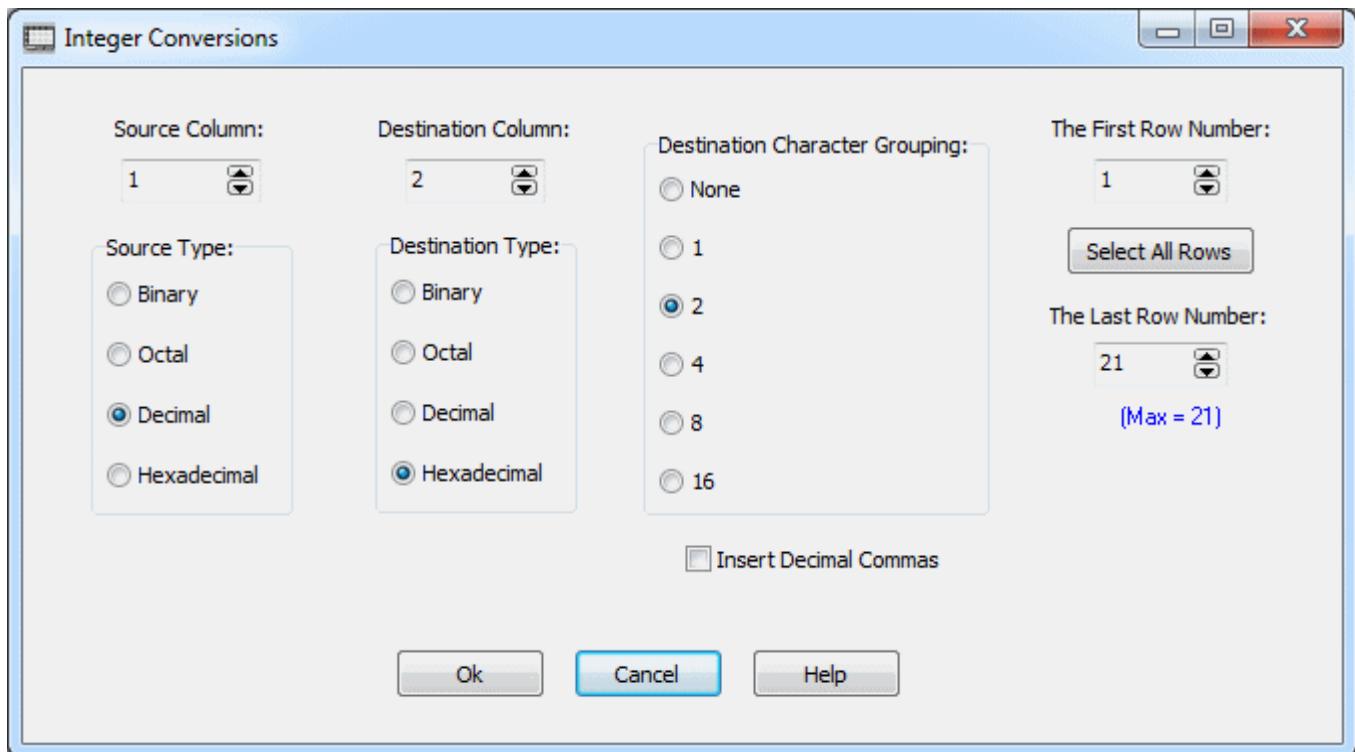
The example table shown on next page converted the decimals shown in Column 3 to the fractions shown in Column 4. In this example we used an **Accuracy Factor = 0.000001**.

Column 3	Column 4
Decimal Value:	Approximate Fraction:
0.125	1/8
7.5	15/2
0.00234	3/1282
1.414213562	3363/2378
1.732050808	5042/2911
5	5/1
1.875	15/8
0.333333333	1/3
3.230769231	42/13

In the above table we can see that the number **5** in the sixth row was converted to the fraction **5/1** even though **5** was given in a string form that does not contain any decimal point. This just means whole numbers will be treated as if they contained a decimal point.

Conversions With Integers

When you select the menu item **Conversions | Convert Integer Format (binary, octal, decimal, hex)...** you will bring up the following dialog box that is used to convert between various forms of nonnegative integers, sometimes thought of as logical integers. The four standard forms have the names **Binary**, **Octal**, **Decimal**, and **Hexadecimal**. These integer formats are popular with programmers, but sometimes you may have data in one of these formats that needs to be converted to another of these formats.



The controls in this dialog are similar to those found in other conversion dialogs. What is new is selecting a **Source Type** and a **Destination Type** that describe the types of integers that are expected in the **Source Column** and the **Destination Column**. Usually these two types will be different, but they can be the same if all you need to do is reformat the result.

There are two other controls that are new in this dialog. The **Destination Character Grouping** has to do with grouping the digits in the formatted output. The given choices can result in something different than what you might expect, depending on the **Destination Type**. As an example, if the **Destination Type** is **Octal**, then the grouping will only be either **None** or **3**. In other words, if you choose anything other than **None**, then result will still be **3**, even though **3** does not appear as a radio button.

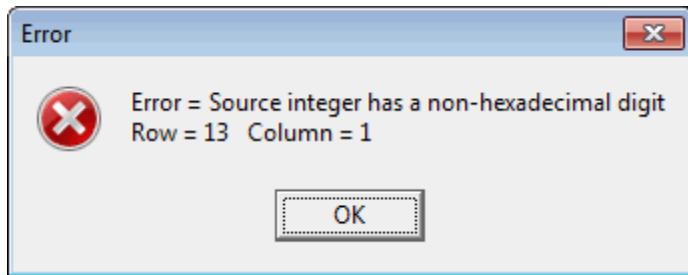
As another example, when the **Destination Type** is **Decimal**, then the value **None** is the only value used and the only other formatting option is the checkbox for whether or not you want to insert commas in the output. When the **Destination Type** is **Binary**, then the grouping will only be either **None** or **4** or **8** or **16**. This means the choices **1** and **2** are not really used and default to **None**. All of these special exceptions are applied so that you can only get really meaningful output.

The columns shown below give examples of the various conversions. The first column contains random integers. The header row explains the type of output for each different column. We just give a sampling of different formats. We used Column 1 as the **Source Column** for all these example conversions.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	The Integer:	Binary Format Grouped 4:	Octal Format:	Hex Format Grouped 4:	Decimals with Commas:	Hex Format Grouped 2:
1	36482591	10 0010 1100 1010 1110 0001 1111	213 127 037	22C AE1F	36,482,591	2 2C AE 1F
2	255	1111 1111	377	FF	255	FF
3	21504038	1 0100 1000 0010 0000 0010 0110	122 020 046	148 2026	21,504,038	1 48 20 26
4	51152384	11 0000 1100 1000 0110 0000 0000	303 103 000	30C 8600	51,152,384	3 0C 86 00
5	2048	1000 0000 0000	4 000	800	2,048	8 00
6	27424077	1 1010 0010 0111 0101 0100 1101	150 472 515	1A2 754D	27,424,077	1 A2 75 4D
7	46502060	10 1100 0101 1001 0000 1010 1100	261 310 254	2C5 90AC	46,502,060	2 C5 90 AC
8	65535	1111 1111 1111 1111	177 777	FFFF	65,535	FF FF
9	134217728	1000 0000 0000 0000 0000 0000 0000	1 000 000 000	800 0000	134,217,728	8 00 00 00
10	65536	1 0000 0000 0000 0000	200 000	1 0000	65,536	1 00 00
11	34354610	10 0000 1100 0011 0101 1011 0010	203 032 662	20C 35B2	34,354,610	2 0C 35 B2
12	68719476736	1 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	1 000 000 000 000	10 0000 0000	68,719,476,736	10 00 00 00 00 00

When the program performs any of these conversions, the first thing it does is remove all blank and comma characters from any source value. It also converts all characters into UPPER case, this being done primarily for the benefit of hexadecimal digits. Next each digit is checked to insure only digits of the source type are present. Finally, the string value is converted to another string, using the base type for that source, and the type of grouping for the output.

It is possible that a source value may not represent a valid integer, given the expected source format. When that is the case, you may see an error message similar to the following that indicates the exact position of the cell and also indicates why the error occurred. In this example the expected source format was hexadecimal.

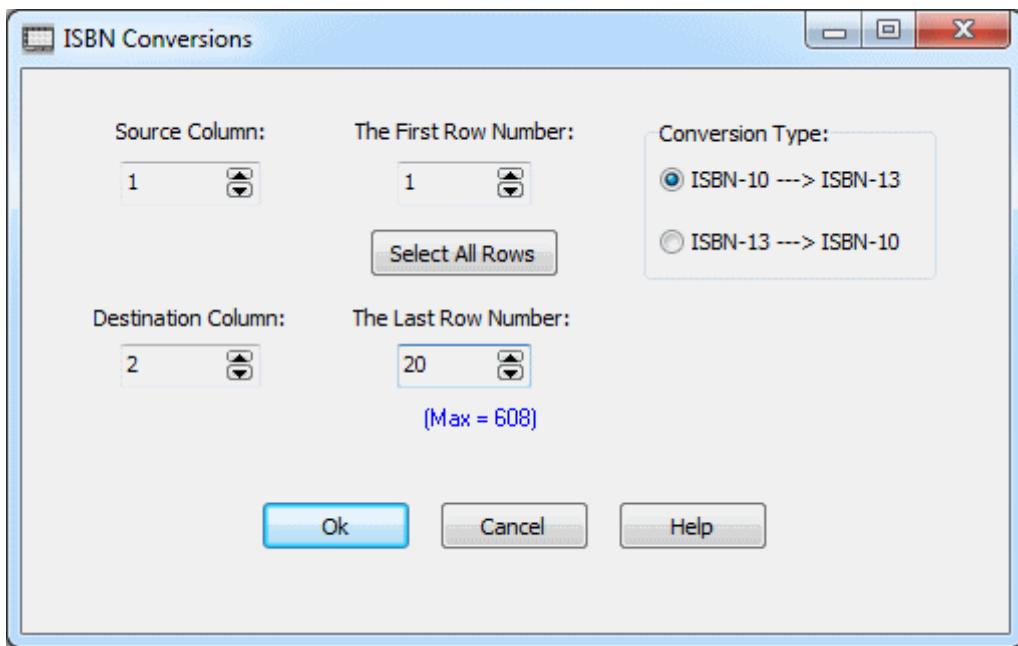


The program only stops with an error message for the first cell, if any, that is found to be improper. After you read the error message and click the **Ok** button, the program will continue processing all remaining rows but it won't show another error message. The program will write a full error message in the **Destination Column** for any and all source rows that contain any source error. Thus you should correct all **Source Column** cells with a corresponding error message in the **Destination Column**, and then perform the conversion again. Keep converting until you get a clean result without any error messages in the **Destination Column**.

Conversions With ISBN Numbers

There are two popular ISBN formats. ISBN stands for International Standard Book Number and virtually every book published in the United States is stamped with an ISBN number. The original format is now called ISBN-10 and the newer format is called ISBN-13. ISBN numbers may also appear with barcodes.

When you select the menu item **Conversions | Convert ISBN-10 Digit <----> ISBN 13-Digit...** you will see the following dialog box.



The main controls in this dialog are the two that select a **Source Column** and a **Destination Column**. In other words, when you convert, you must have a source string and you must produce another destination string. The purpose of the **Source Column** and the **Destination Column** is to allow you identify where the source numbers are and to identify where what you will compute must go.

The **Conversion Type** radio buttons allow you to select the direction you wish to go. You can convert from ISBN-10 to ISBN-13 or you can convert from ISBN-13 to ISBN-10.

The last set of controls identify a row range.

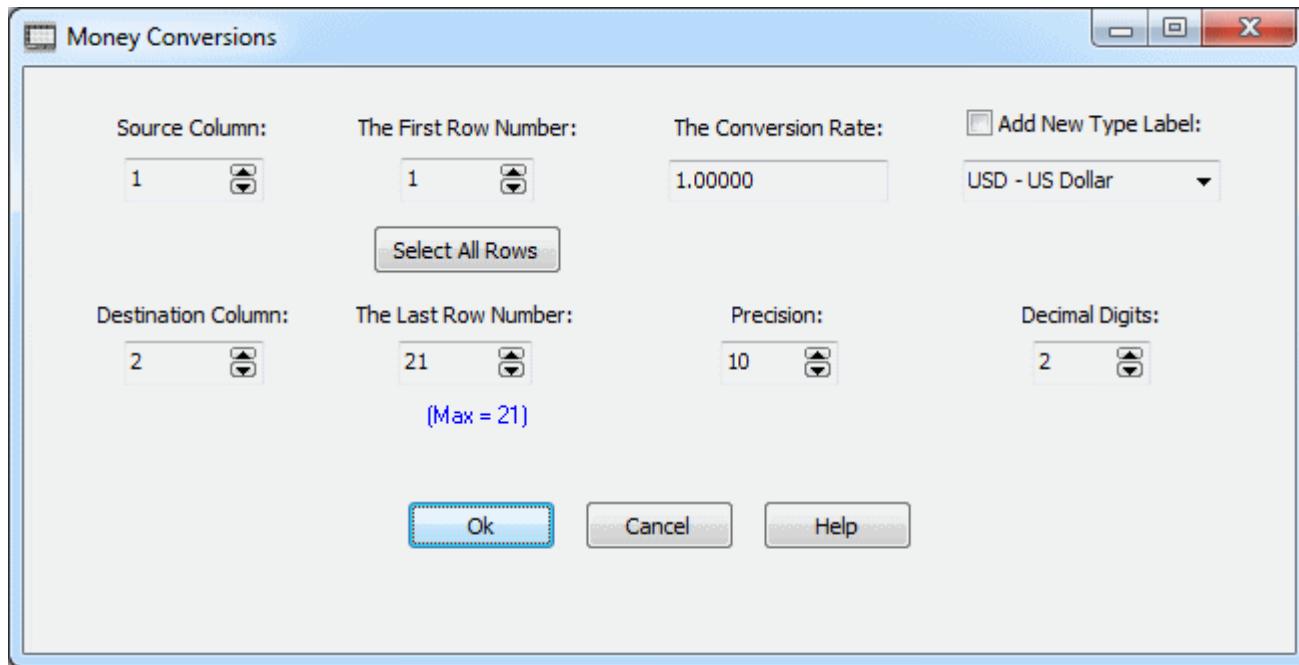
An example grid showing two different columns of ISBN numbers is shown on the next page.

<	Column 1	Column 2
>	Old ISBN 10 Numbers:	New ISBN 13 Numbers:
1	0-387-97562-7	978-0-387-97562-7
2	0-387-97606-1	978-0-387-97606-8
3	1-56592-098-3	978-1-56592-098-9
4	0-471-53656-8	978-0-471-53656-7
5	1-55828-390-0	978-1-55828-390-9
6	1-883577-25-1	978-1-883577-25-4
7	0-672-30499-5	978-0-672-30499-6
8	1-55615-679-0	978-1-55615-679-3
9	0-89588-644-3	978-0-89588-644-6
10	0-7821-1576-7	978-0-7821-1576-5

When converting ISBN numbers, it is possible that your **Source Column** may contain an invalid ISBN. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first bad ISBN. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

Conversions With Money

When you select the menu item **Conversions | Convert Money...** you will see the following dialog box.



The main controls in this dialog are the two that select a **Source Column** and a **Destination Column**. In other words, when you convert, you must have a source string and you must produce another destination string. The purpose of the **Source Column** and the **Destination Column** is to allow you identify where the source numbers are and to identify where what you will compute must go.

The **Conversion Rate** number is used to multiply the source money value to compute the destination money value. This number could also be called the rate multiplier. You must always type in current rate multiplier if want the result to be accurate for today's conversion rate or for whatever other rate is needed. You can always find the current money conversion rate somewhere on the Internet, when you apply particular currencies.

You don't choose a source money type because the program only assumes the source values are some kind of decimal values. The drop down list box allows you to select a destination type of money. When this list is expanded it appears as:



While not every currency is represented in the list on the previous page, the most popular currencies are in the list.

The only purpose in choosing an item from this list is so you can label the output in the **Destination Column**. To actually label the output you must also check the checkbox that says **Add New Type Label**. All the labeling does is add the 3-letter abbreviation to the numerical value that is otherwise the only output. If this checkbox is not checked then no 3-letter abbreviation will be added to the numerical output.

A hidden feature of this drop-down list is that if you need a currency label that is not in this list, you can key in your own 3-letter abbreviation and title. Only the first 3 letters you type will actually be used for the label.

You can set a range of rows and you can set the display format. The controls for **Precision** and **Decimal Digits** work the same as they do for **Block Formatting**. Here the default number of **Decimal Digits** or decimal places is just 2.

An example grid showing two different columns of money is the following. In this example we converted from US dollars to Japanese Yen and we labeled the output result. The conversion rate number was 93.3800 which may seem high compared to European rate values, but this is an accurate rate for converting to Yen.

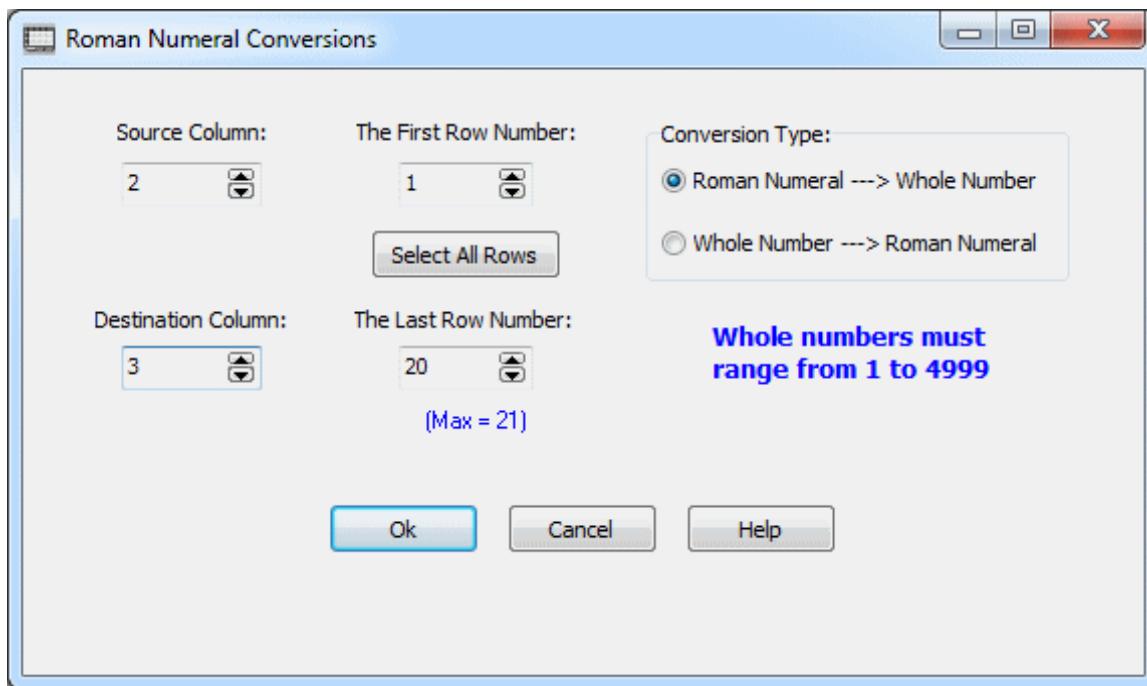
<	Column 1	Column 2
>	US Dollars	Japanese Yen
1	\$56.11	5239.55 JPY
2	\$56.57	5282.51 JPY
3	\$33.48	3126.36 JPY
4	\$78.27	7308.85 JPY
5	\$58.69	5480.47 JPY
6	\$42.43	3962.11 JPY
7	\$71.25	6653.33 JPY

When converting money, it is possible that your **Source Column** may contain an invalid number. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first invalid number. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

You might also note that what we have accomplished here could also have been done by applying a Mathematical Expression by choosing **Blocks | Block Math Expression....** But in doing so you would have to write the correct corresponding formula and you might have to also duplicate the source column values before proceeding. Using the above dialog under the **Conversions** menu is faster and easier.

Conversions With Roman Numerals

Ok, we have to admit that conversions involving Roman Numerals may not be a hot topic at the top of everyone's wish list. But if you ever had to do such conversions it would be nice to have it automated. When you select the menu item **Conversions | Convert Roman Numerals <---> Whole Numbers...** you will see the following dialog box.



The main controls in this dialog are the two that select a **Source Column** and a **Destination Column**. In other words, when you convert, you must have a source string and you must produce another destination string. The purpose of the **Source Column** and the **Destination Column** is to allow you identify where the source numbers are and to identify where what you will compute must go.

The **Conversion Type** radio buttons only give you two choices. You must either convert from Roman Numerals to Whole Numbers or you must convert from Whole Numbers to Roman Numerals. Note the range of whole numbers is limited to those in the range from 1 to 4999.

The next page shows two example grids. In the taller grid on the left, Columns 1 and 2 are paired and columns 3 and 4 are paired. Column 1 has the consecutive whole numbers from 1 to 30 while Column 3 has numbers that are 3-digits long and almost appear to be random. Actually, they are multiples of 17 that have been added to 100. In any case these examples should give you a good flavor of various kinds of Roman Numerals.

In the shorter example grid on the right we show the Roman Numerals that would be associated with the years for some of the Academy Awards films that won Best Picture between 1960 and 1980. Perhaps the last time you saw a Roman Numeral was when you viewed a movie trailer.

<	Column 1	Column 2	Column 3	Column 4
1 >	1	I	100	C
2	2	II	117	CXVII
3	3	III	134	CXXXIV
4	4	IV	151	CLI
5	5	V	168	CLXVIII
6	6	VI	185	CLXXXV
7	7	VII	202	CCII
8	8	VIII	219	CCIX
9	9	IX	236	CCXXXVI
10	10	X	253	CCLIII
11	11	XI	270	CCLXX
12	12	XII	287	CCLXXXVII
13	13	XIII	304	CCCIV
14	14	XIV	321	CCCXI
15	15	XV	338	CCCXXXVIII
16	16	XVI	355	CCCLV
17	17	XVII	372	CCCLXXII
18	18	XVIII	389	CCCLXXXIX
19	19	XIX	406	CDVI
20	20	XX	423	CDXXIII
21	21	XXI	440	CDXL
22	22	XXII	457	CDLVII
23	23	XXIII	474	CDLXIV
24	24	XXIV	491	CDXCI
25	25	XXV	508	DVIII
26	26	XXVI	525	DXV
27	27	XXVII	542	DXLII
28	28	XXVIII	559	DLIX
29	29	XXIX	576	DLXXVI
30	30	XXX	593	DXCIII

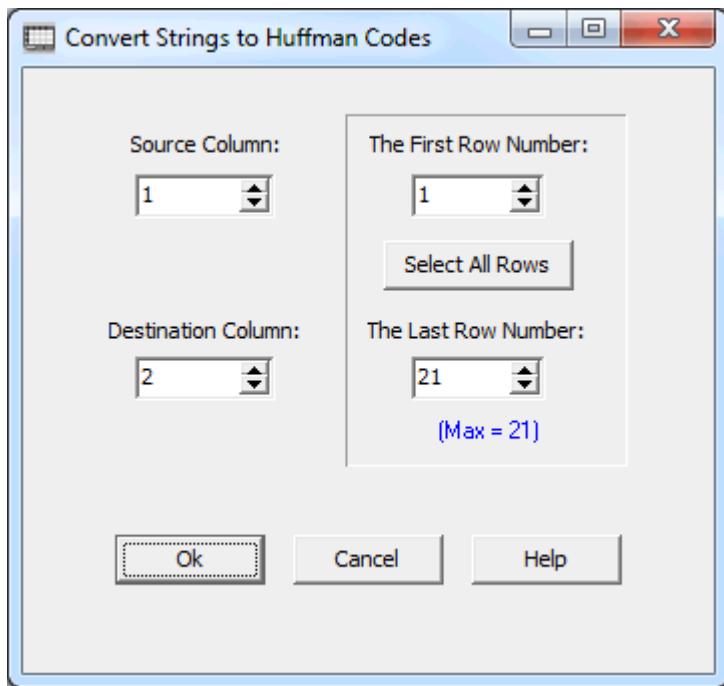
<	Column 1	Column 2	Column 3
>	Year	Best Picture	Roman N.
1	1960	The Apartment	MCMLX
2	1961	West Side Story	MCMLXI
3	1962	Lawrence of Arabia	MCMLXII
4	1963	Tom Jones	MCMLXIII
5	1964	My Fair Lady	MCMLXIV
6	1965	The Sound of Music	MCMLXV
7	1966	A Man For All Seasons	MCMLXVI
8	1967	In The Heat Of The Night	MCMLXVII
9	1968	Oliver	MCMLXVIII
10	1969	Midnight Cowboy	MCMLXIX
11	1970	Patton	MCMLXX
12	1971	The French Connection	MCMLXXI
13	1972	The Godfather	MCMLXXII
14	1973	The Sting	MCMLXXIII
15	1974	The Godfather Part II	MCMLXXIV
16	1975	One Flew Over The Cuckoos Nest	MCMLXXV
17	1976	Rocky	MCMLXXVI
18	1977	Annie Hall	MCMLXXVII
19	1978	The Deer Hunter	MCMLXXVIII
20	1979	Kramer vs Kramer	MCMLXXIX
21	1980	Ordinary People	MCMLXXX

When converting Roman Numerals, it is possible that your **Source Column** may contain an invalid number. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first invalid number. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

You might also note that what we have accomplished here is unique! You could not do all these Roman Numeral conversions by trying to apply a Mathematical Expression.

Huffman Coding

This program can perform a special operation known as **Huffman Coding**, which is a process where you start with a given set of elements and attempt to develop a set of minimal binary codes that can be associated with those elements. The total length of the created binary codes will be a minimum. In our implementation, the elements are assumed to occupy a range of rows, all within one column. To get started, just select the menu item **Conversions | Convert Strings to Huffman Codes...** and you will see a dialog like the following:



All you need do is choose the **Source Column** and the **Destination Column** and choose the range of rows. When you click the **Ok** button the program will do the rest and fill the chosen rows with binary codes, but in the **Destination Column**. Each element will be assigned its own unique binary code.

For our first example we will use the grid that appears as shown on the next page. In this example, the first column contains single letters (including spaces) that are from the message sent by a telegraph from Samuel B. Morse on May 24, 1844 between Washington, D.C. and Baltimore, Maryland. The message was:

WHAT HATH GOD WROUGHT

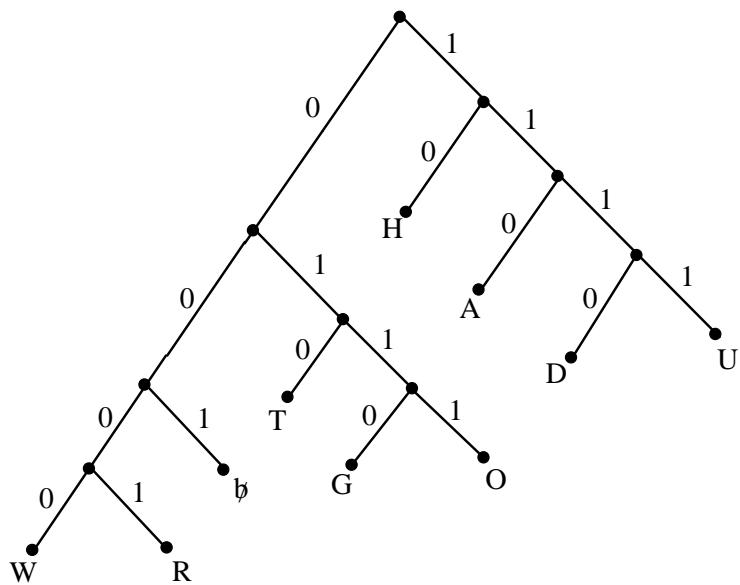
There are 21 letters total (including 3 space characters) in this message and the idea behind a Huffman Coding scheme is to create a series of variable length bit codes, where each letter will be assigned a unique code, and at the same time the least number of total bits will be used when all the letters get replaced by those codes.

The resulting coding scheme is shown on the next page. The grid on the left is the initial grid. The grid on the right is the resulting grid after we clicked the **Ok** button to fill in the second column with the Huffman binary codes. Read the first column from top to bottom to see the original message in the two grids.

<	Column 1	Column 2
>	The Letter:	The Huffman Code:
1	W	
2	H	
3	A	
4	T	
5		
6	H	
7	A	
8	T	
9	H	
10		
11	G	
12	O	
13	D	
14		
15	W	
16	R	
17	O	
18	U	
19	G	
20	H	
21	T	

<	Column 1	Column 2
>	The Letter:	The Huffman Code:
1	W	0000
2	H	10
3	A	110
4	T	010
5		001
6	H	10
7	A	110
8	T	010
9	H	10
10		001
11	G	0110
12	O	0111
13	D	1110
14		001
15	W	0000
16	R	0001
17	O	0111
18	U	1111
19	G	0110
20	H	10
21	T	010

All of the information contained in the above grid on the right can also be shown in a tree structure. In the next figure, the symbol \flat is used to denote the space character. The small black dots are called the nodes of the tree. Note that characters appear only at the leaf nodes. Left bearing branches are labeled with 0's while right bearing branches are labeled with 1's.



This same information can also be shown in a short table like the following. Below we can see there are only 10 distinct letters or characters. Because $2^3 = 8$ is not large enough for 10 items, the first power of 2 that is larger than 10 is $2^4 = 16$. This helps explain why any Huffman code that represents the given message must use at least 4 bits for some of its letters.

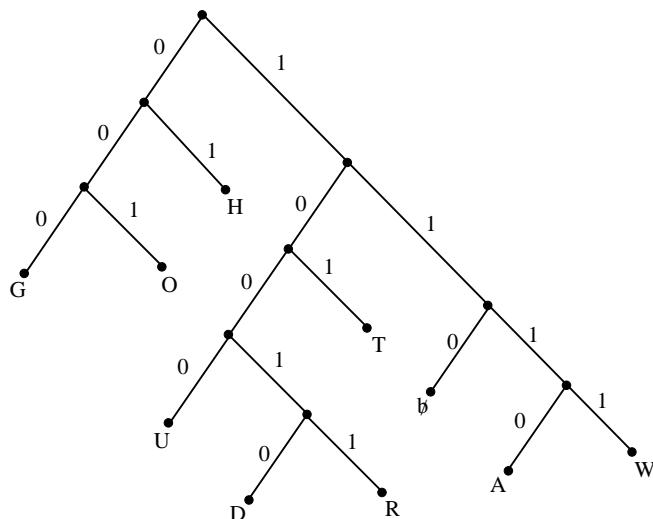
	Column 1	Column 2
>	The Letter:	The Huffman Code:
1	W	0000
2	H	10
3	A	110
4	T	010
5		001
6	G	0110
7	O	0111
8	D	1110
9	R	0001
10	U	1111

The letter H appears the most number of times and for this reason H was assigned the shortest code that consists of 2 bits. The letter R occurs only once and it was assigned one of the longest codes that is 4 bits. To read the codes from the tree structure shown on the previous page, all you need do is start at the top of the tree. Read bits going down until you reach a desired letter that is always at a tree leaf node. The string of bits that you read are the binary code for that letter. The binary code for each letter should be the same as in the table's Column 2.

If you string all the bits together, in one large string of bits, the entire message would be encoded using the 68 bits shown below. It is this total length, 68, that is minimized by the Huffman code for the entire message.

W H A T b H A T H b G O D b W R O U G H T
0000 10 110 010 001 10 110 010 10 001 0110 0111 1110 001 0000 0001 0111 1111 0110 10 010

It can be proven mathematically that there is no other coding scheme that is more efficient than one like this. We should mention however, that this same message could be encoded using a different Huffman tree like the next one shown below.



All this means is that Huffman trees are not unique, but this last tree also requires exactly the same number of bits, namely 68, to encode the entire message. So for all practical purposes the two trees we have shown are in a sense equivalent.

Huffman coding works best when your data is made up of just a few characters that repeat a large number of times, such as all the letters in this sentence.

If we encode the last sentence above (using the red characters and blank spaces), we would find the following table shows the encoding of all the unique characters. There are 155 total characters in the sentence, but without repetitions, there are only 26 unique characters. The next grid shows the Huffman codes for those unique letters. The space character appears the most often and is given the shortest code, 01, that is only two bits long. The letter e is the next most frequently appearing letter and its code, 001, is three bits long.

<	Column 1	Column 2
1 >	H	0000000
2	u	10000
3	f	11000
4	m	00010
5	a	1010
6	n	10110
7		01
8	c	11010
9	o	11100
10	d	100010
11	i	11011
12	g	110010
13	w	100011
14	r	10111
15	k	0001100
16	s	1111
17	b	000111
18	e	001
19	t	1001
20	h	11101
21	y	1100110
22	p	000001
23	j	1100111
24	l	00001
25	.	0001101
26	.	0000001

If you used fixed length 8-bit bytes for each of the 155 characters in the above example red-letter sentence, then that sentence would occupy 1,240 total bits. Using variable length Huffman bit codes instead of the 8-bit bytes, this sentence would occupy only 643 bits. Thus Huffman codes can be used to compress data, and for this example sentence we would save about 48% of the total required space. For most English text that is compressed using Huffman coding, the typical savings is about 30%.

In general, Huffman codes do not have to be applied to individual letters. Huffman codes can be applied to any kind of data, but it is best applied when parts of that data repeats.

The last example we will give is an encoding for the distinct words in the rhyme:

**As I was going to Saint Ives I met a man with seven wives,
each wife had seven sacks, each sack had seven cats,
each cat had seven kits: kits, cats, sacks and wives,
how many were going to Saint Ives?**

The above version of this rhyme has only 25 distinct words and 41 total words. The Huffman Coding for the distinct words in this rhyme appears as follows. The word **seven** occurs the most number of times and that is why this word has been assigned the smallest number of bits for any word, which is three bits, namely 001.

<	Column 1	Column 2
>	The Words:	The Huffman Code:
1	As	00000
2	I	10000
3	was	01000
4	going	10100
5	to	1100
6	Saint	10110
7	Ives	0110
8	met	01110
9	a	101110
10	man	11010
11	with	101010
12	seven	001
13	wives	0101
14	each	1110
15	wife	10010
16	had	1111
17	sacks	0001
18	sack	101011
19	cats	10001
20	cat	11011
21	kits	10011
22	and	101111
23	how	01111
24	many	01001
25	were	00001

The algorithm to perform Huffman coding begins by analyzing the data and tabulating frequency counts for the individual elements. The Huffman tree building algorithm can be briefly described with five easy steps.

- (1) Create an initial list of elements (subtrees) and their frequencies.
- (2) If the subtree list contains only one tree item then you are done!
- (3) Otherwise, remove from the list any two subtrees with the two smallest frequencies and make them children of a new combined subtree whose frequency is the sum of the two child frequencies.
- (4) Add the new combined subtree to the list. (An option is to keep the list sorted by frequency.)
- (5) Go back to step (2).

The nature of the algorithm is that the elements with the largest frequencies get assigned the shortest codes, while the elements with the smallest frequencies get assigned the longest codes. Each Huffman tree depends on how you setup or order the initial list of elements, and it further depends on how you handle ties when searching and/or sorting the elements to find the two smallest frequencies. Further details are beyond the scope of this help file. Internally the program creates tree structures, similar to the two trees shown above.

The real purpose of the Huffman Coding function in the **CSV Editor** program is to just show the Huffman codes that are produced. The **CSV Editor** program does not really do anything useful with the Huffman codes it produces. For example, it does not store them in a more useable or compressed file format. The **CSV Editor** program does not do file compression, but it does allow you to see the codes it produces and it does allow you to create codes for any kind of string data that is contained in a single column of a **CSV** file.

The next page shows 10 possible stages of building a Huffman tree for the letters of the message **WHAT HATH GOD WROUGHT**. The number of stages is always the same as the number of distinct items. In the figure on the next page, the circled numbers are the frequencies of the subtrees that appear just below the circled numbers.

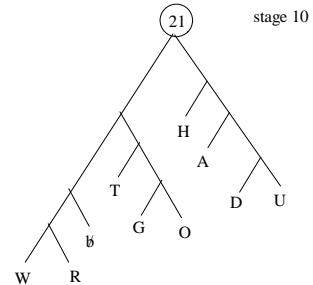
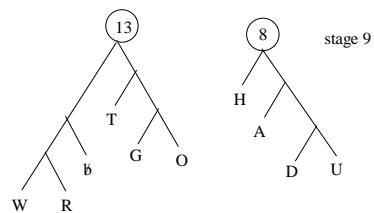
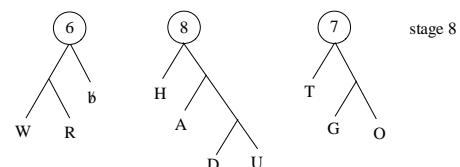
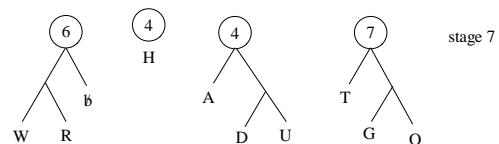
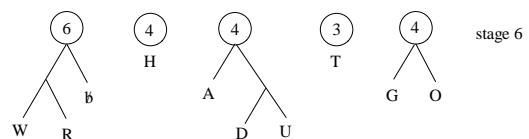
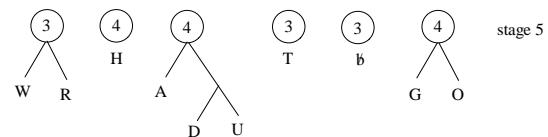
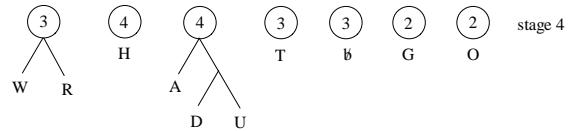
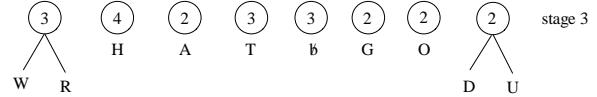
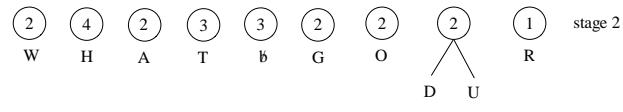
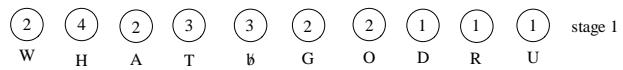
The next figure may help you better understand how you might build or construct a Huffman tree after you have determined the frequencies of the initial elements. Note how the frequency of each newly created subtree is the sum of the two frequencies from the two elements chosen with the smallest frequencies in the previous stage.

In this example we do not keep the list sorted by frequency. Nor do we explain how we handle ties when searching for the two smallest frequencies. The letters listed in the first stage are in the order in which they first appear in the message. The order of the elements is not relevant to the implementation of the algorithm. Just be aware that different initial orders produce very different looking, but otherwise equivalent Huffman trees, in terms of minimizing the total number of bits.

Incidentally, if you had any grid with a sufficiently large number of rows that was mostly empty, except for the cell in the first row and the third column, then you could apply the following string expression to separate out the individual letters in the message. Imagine the message is in the cell **C(1,3)** where the message has 500 letters. Then you could select the block in column 1 in the first 500 rows, and apply the following string expression to that block.

SubString(C(1,3), I, 1)

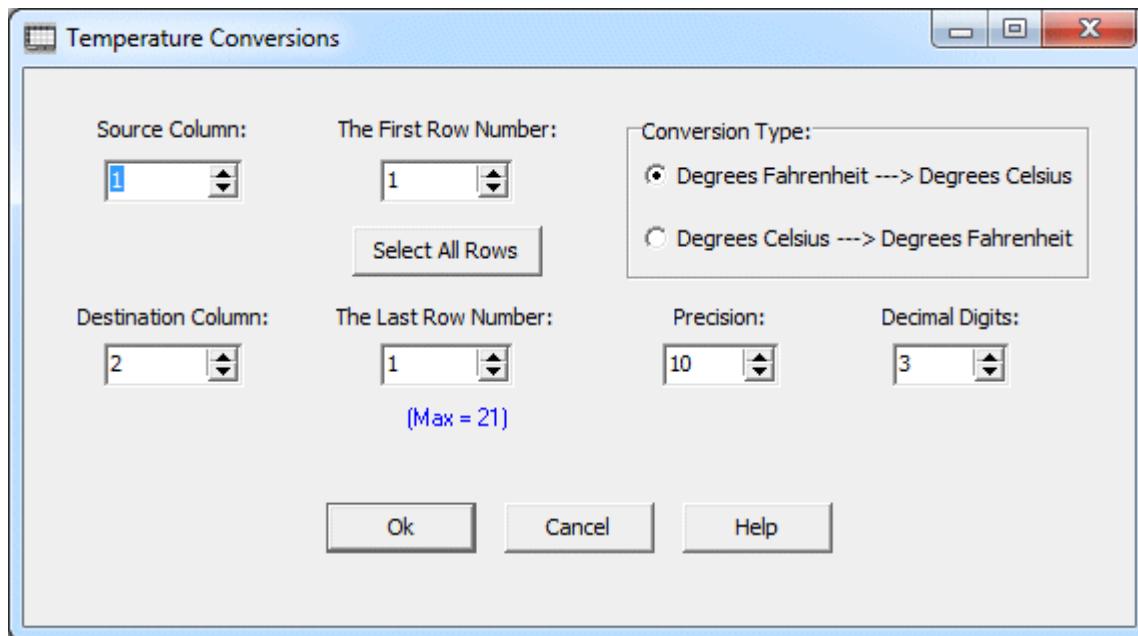
This is a nice example of using a simple but powerful string expression. This is all that would be required to get all the single letters of the message to occupy the first 500 rows in Column 1 of the grid. Then you could apply Huffman coding where Column 2 would be the destination column. You could also apply the function to delete duplicate rows (see **Rows | Check For and Delete Duplicate Rows...**) to a get a list with unique elements. We used both the **Substring** function and in some cases deleted duplicate rows to create all of the tables that are in this help topic.



Conversions With Temperatures

The two most popular temperature scales have been given the names Fahrenheit and Celsius.

When you select the menu item **Conversions | Convert Temperatures...** you will see the following dialog box.



The main controls in this dialog are the two that select a **Source Column** and a **Destination Column**. In other words, when you convert, you must have a source string and you must produce another destination string. The purpose of the **Source Column** and the **Destination Column** is to allow you identify where the source numbers are and to identify where what you will compute must go.

The **Conversion Type** radio buttons allow you to select the direction you wish to go. You can convert from Fahrenheit to Celsius or from Celsius to Fahrenheit. You can set a range of rows and you can set the display format. The controls for **Precision** and **Decimal Digits** work the same as they do for **Block Formatting**. Here the default number of **Decimal Digits** or decimal places is just 3.

An example grid showing two different columns of Temperature numbers is the following.

<	Column 1	Column 2
>	Fahrenheit	Celsius
1	32.000	0.000
2	212.000	100.000
3	60.000	15.556
4	80.000	26.667

When converting temperatures, it is possible that your **Source Column** may contain an invalid number. When that is the case, the program will put the word **Error** in the corresponding row in the **Destination Column**. The program will process all rows inserting the word **Error** as needed. After that, you will see an error message telling you the **Source Column** cell that contains the first invalid number. The error message is only shown for the first row found that has an error. After viewing the message you should fix all the source cells and then try to re-convert those rows that contain the word **Error** in the **Destination Column**.

You might also note that what we have accomplished here could also have been done by applying a Mathematical Expression by choosing **Blocks | Block Math Expression...**. But in doing so you would have to write the correct corresponding formula and you might also want to duplicate the source column values. Using the above dialog under the **Conversions** menu is faster and easier.

Conversions Between Times and Decimals

The menu item **Conversions | Convert Times <---> Decimal Values...** is used to convert between strings that represent times and strings that represent decimal values. For the **CSV Editor** program, all times should initially be considered to consist of an integer number of **Hours**, **Minutes**, and **Seconds**. This is a broad but general concept. In general, these integers can be any large integers as long as they are not negative. However, when times get formatted or converted, then we reduce **Seconds** so they are in the range from 0 to 59. We do this by casting out values of 60 while we simultaneously increase the number of **Minutes** by 1 each time we cast out a value of 60. In a similar fashion we reduce **Minutes** so they are also in the range from 0 to 59 and we simultaneously increase the number of hours as necessary.

When times are formatted as strings, we have choices of 12-hour and 24-hour and elapsed time formats. Both the 12-hour and 24-hour formats further reduce the integer number of **Hours** to an appropriate range by throwing away hours outside the corresponding ranges. The 12-hour format also appends an AM or a PM indicator to the string that represents the time.

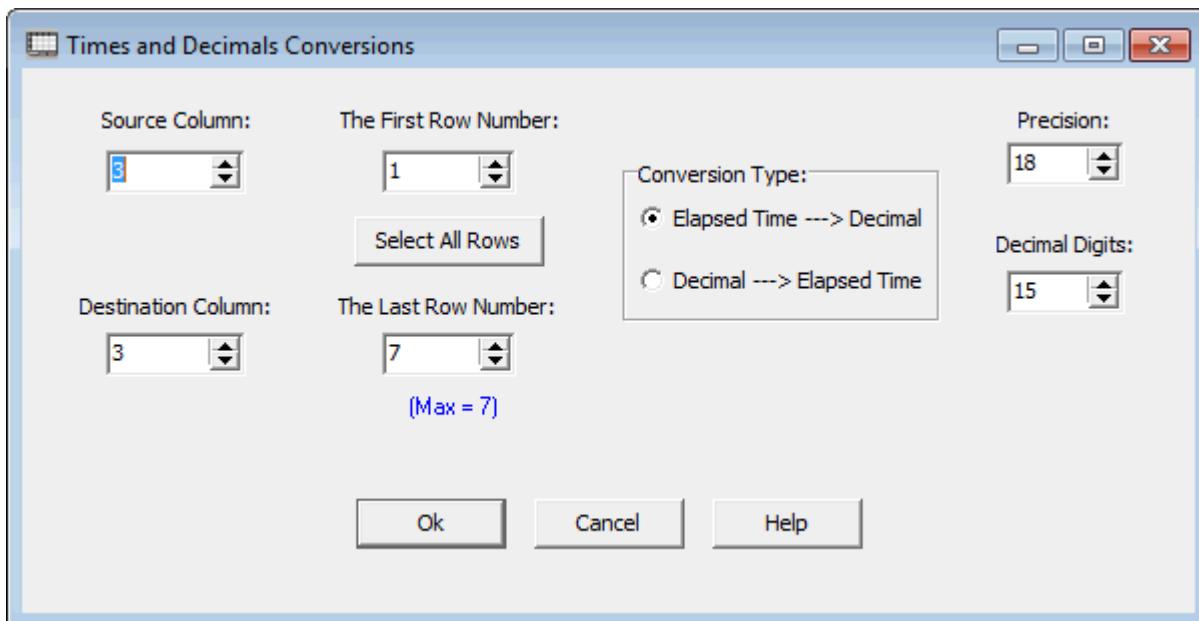
To understand how conversions get performed, we use the equation:

$$\text{Decimal} = \text{Hours}/24 + \text{Minutes}/(24 \cdot 60) + \text{Seconds}/(24 \cdot 60 \cdot 60)$$

This equation gets applied after all **Hours** and **Minutes** and **Seconds** have been appropriately reduced for an intended output format. We recommend formatting times before performing conversions to decimals.

When converting from decimals to elapsed times, we apply the above equation by first determining the integer number of **Hours** and **Minutes** and **Seconds** from the equation. For elapsed times, the number of **Hours** can be large (i.e., much larger than 24).

When you select the menu **Conversions | Convert Times <---> Decimal Values...** you will see the following dialog.



The above dialog has all the standard controls for doing a typical conversion. You should first set the **Source** and **Destination** columns. Then select a range of rows. Finish by selecting a **Conversion Type**. The values used for the **Precision** and **Decimal** digits will almost always be left with their default values.

It is important to understand that when converting from decimals to times, the resulting time strings will be formatted as elapsed times only. If you prefer to have times in the 12-hour or 24-hour formats then you should further perform a Block Formatting function to change the output values.

When you click the **Ok** button to perform a conversion, the program may place an Error message in the **Destination** column whenever it finds an inappropriate input value. Only the first Error message is actually displayed, although all rows get processed. So more than one row may have an Error message.

The following two grids show the before and after results of converting from various Times to Decimals.

<	Column 1	Column 2
>	Times:	Decimal Value:
1	02:03:04 AM	
2	02:05:06 PM	
3	14:00:00	
4	0	
5	14:00:00 PM	
6	11:00:00 PM	
7	12345	
8	59:159:180	
9	124:40:38	
10	23:59:59	
11	24:60:60	
12	0:0:0	
13	12:00:00 AM	
14	12:00:00 PM	
15	23:59:60	
16	11:59:60 AM	
17	06:00:00 AM	
18	06:00:00 PM	
19	24:00:00	
20	48:00:00	
21	72:00:00	
22	96:00:00	
23	01:02:03x	
24	-5	

<	Column 1	Column 2
>	Times:	Decimal Value:
1	02:03:04 AM	0.085462962962963
2	02:05:06 PM	0.5868750000000000
3	14:00:00	0.5833333333333333
4	0	0.0000000000000000
5	14:00:00 PM	0.5833333333333333
6	11:00:00 PM	0.9583333333333333
7	12345	514.3750000000000000
8	59:159:180	2.5708333333333333
9	124:40:38	5.194884259259259
10	23:59:59	0.999988425925926
11	24:60:60	1.0423611111111111
12	0:0:0	0.0000000000000000
13	12:00:00 AM	0.0000000000000000
14	12:00:00 PM	0.5000000000000000
15	23:59:60	1.0000000000000000
16	11:59:60 AM	0.5000000000000000
17	06:00:00 AM	0.2500000000000000
18	06:00:00 PM	0.7500000000000000
19	24:00:00	1.0000000000000000
20	48:00:00	2.0000000000000000
21	72:00:00	3.0000000000000000
22	96:00:00	4.0000000000000000
23	01:02:03x	Error, invalid elapsed time value.
24	-5	Error, invalid elapsed time value.

Note how the last two rows generated error messages. The x character in row 23 is the cause of the first error. Negative numbers are not allowed for times, and this explains the second error in row 24.

The following grid shows converting from decimals back to times. The decimal values are in Column 1. The three possible converted time formats are in Column 2 (elapsed) and Column 3 (24-Hour) and Column 4 (12-Hour). This table was created in two steps. First we converted the decimals in Column 1 to the elapsed times in Column 2. Then we manually reformatted the times in Column 2 to create columns 3 and 4. Just remember that decimal conversions always assume elapsed times are input or output in the conversion process.

<	Column 1	Column 2	Column 3	Column 4
>	Decimal Value:	Elapsed Time:	24-Hour Format:	12-Hour Format:
1	0.085462962962963	2:03:04	02:03:04	02:03:04 AM
2	0.5868750000000000	14:05:06	14:05:06	02:05:06 PM
3	0.5833333333333333	13:00:00	13:00:00	01:00:00 PM
4	0.0000000000000000	0:00:00	00:00:00	12:00:00 AM
5	0.5833333333333333	13:00:00	13:00:00	01:00:00 PM
6	0.9583333333333333	22:00:00	22:00:00	10:00:00 PM
7	514.3750000000000000	12345:00:00	09:00:00	09:00:00 AM
8	2.5708333333333333	61:42:00	13:42:00	01:42:00 PM
9	5.194884259259259	124:40:38	04:40:38	04:40:38 AM
10	0.9999988425925926	23:59:59	23:59:59	11:59:59 PM
11	1.0423611111111111	25:01:00	01:01:00	01:01:00 AM
12	0.0000000000000000	0:00:00	00:00:00	12:00:00 AM
13	0.0000000000000000	0:00:00	00:00:00	12:00:00 AM
14	0.5000000000000000	12:00:00	12:00:00	12:00:00 PM
15	1.0000000000000000	24:00:00	00:00:00	12:00:00 AM
16	0.5000000000000000	12:00:00	12:00:00	12:00:00 PM
17	0.2500000000000000	6:00:00	06:00:00	06:00:00 AM
18	0.7500000000000000	18:00:00	18:00:00	06:00:00 PM
19	1.0000000000000000	24:00:00	00:00:00	12:00:00 AM
20	2.0000000000000000	48:00:00	00:00:00	12:00:00 AM
21	3.0000000000000000	72:00:00	00:00:00	12:00:00 AM
22	4.0000000000000000	96:00:00	00:00:00	12:00:00 AM

Look at rows 16 through 18 and note how decimal values 0.5 and 0.25 and 0.75 make for $\frac{1}{2}$ a day, $\frac{1}{4}$ of a day, and $\frac{3}{4}$ of a day. Each quarter of a day corresponds to a 6 hour time period, starting from midnight.

Note how the values in Column 1 in rows 19 through 22 make for four distinct elapsed times in Column 2, but they all make the same zero time 00:00:00 value in Column 3 and they also make the same 12:00:00 AM time in Column 4.

We should point out that there are really two different uses of time. When most people use time, it is usually to represent a particular time in a day. For example, if you were to schedule a train ride from San Francisco to Boston you might have a departing event time that is 10:15 AM. This of course is a particular time of day.

A different kind of time value is what we call a duration time or an elapsed time. If the duration of your train trip was 47 hours and 30 minutes, then that use of time would be an elapsed time whose purpose is to describe how many hours and minutes you would be riding on the train. Such an elapsed time would require going beyond the normal 24 hour limit of a specific time of day that we call an event time. You can't have 47 hours in any daytime event value, but you could have that many hours in an elapsed time value.

So durations or elapsed times can differ significantly from times that specify when an event occurs. We should also point out that we don't measure or use times that have more accuracy than 1 second. So if you just think of having any number of nonnegative hours, with minutes and seconds in the range from 0-59 you should be Ok. There are applications like timing events in the Olympics or scientific applications where the time resolution needs to be within a hundredth or a thousandth or an even smaller fraction of a second. We don't accommodate applications of time that require such high resolutions. However, more accurate times should probably be written using only ordinary decimals anyway.

Also, we don't associate or mix dates with times. If you are used to using a spreadsheet that allows a specific time event to also specify a date, then you need to know that we don't encourage this. We would recommend you separate your data into two columns, a date column and a time column. Then you can handle times as either a duration or when an event occurs during a day.

It is probably conceptually simplest to think of one of our time values as being something given like a military time, but where the number of hours can be 24 or larger. Every 24-hour or 12-hour event time can be considered as an elapsed time, but not always vice versa. When elapsed times are converted to 24- or 12-hour event times then any hours over 24 or 12 should be reduced accordingly.

One final peculiar thought. If you see a time written as 14:22:36 you can't determine whether it is an event time or an elapsed time because it could be either. You can infer that it is in the 24-hour format, but it could represent an elapsed time. A time written as 08:32:26 is either a 24-hour formatted event time or it is an elapsed time. Only times written as 11:29:57 AM or 04:45:13 PM must be interpreted as event times, not elapsed times. Times that are formatted as 12-hour time values should always be interpreted as event times and not elapsed times.

Conversions Using a Replacement List

Imagine you have a list of electronic items like that shown below in which we have deliberately duplicated the second column data in the fourth column. The problem we would like to solve is this. The old part numbers in **Column 2** need to be replaced with new part numbers that already exist in another database system. We will replace the copied part numbers in **Column 4** with the new part numbers. Prior to making the replacements, we changed the **Column 4** header with the new title, **New Part Number**:

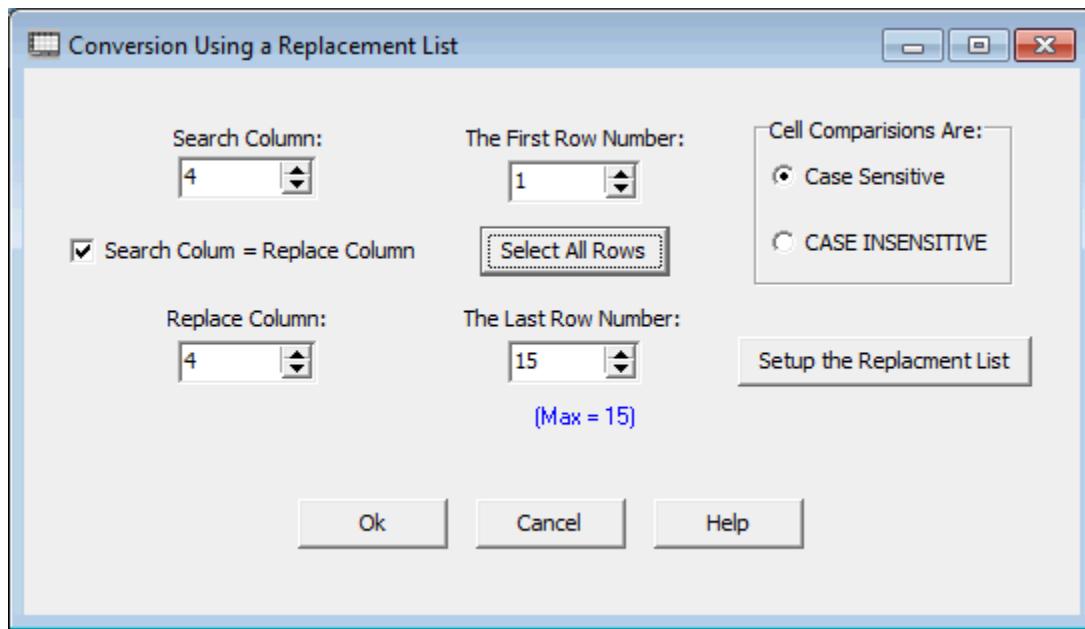
<	Column 1	Column 2	Column 3	Column 4
>	Item Description:	Old Part Number:	Price:	New Part Number:
1	Fluke Multimeter	8845A	\$1,455.00	8845A
2	Tektronix Oscilloscope	DP04VID	\$868.00	DP04VID
3	Power Sensor	799271	\$59.00	799271
4	RF Cable Assembly	4532-C-60	\$25.89	4532-C-60
5	Test Lead	12972	\$23.10	12972
6	Amphenol Connector	10SL-3P	\$10.95	10SL-3P
7	Panel Mount Connector	480-721	\$3.30	480-721
8	Terminal Strip	TS7591	\$4.79	TS7591
9	Insulated Grip	IG-832	\$0.50	IG-832
10	Jacketed Video Cable	762-VC	\$42.79	762-VC
11	Tensioning Tool	EV07	\$300.00	EV07
12	Frame Rack	E4DSRA	\$29.00	E4DSRA
13	DIN Power Rail	PS-1505	\$12.95	PS-1505
14	Micro Switch	GLKMC03	\$33.45	GLKMC03
15	Photoelectric Sensor	NPN-734	\$91.57	NPN-734

The best we can do is to make the conversion using what we call a **Replacement List**. A **Replacement List** is just another CSV file in which new and old values are lined up row by row. Imagine we create and fill in a list like the following.

The Replacement List:		
	Original Data	Replacement Data
1	8845A	VT3901
2	DP04VID	VT3995
3	799271	VT4672
4	4532-C-60	VT8217
5	12972	VT8281
6	10SL-3P	VT0293
7	480-721	VT8880
8	TS7591	VT2313
9	IG-832	VT8492
10	762-VC	VT9383
11	EV07	VT4442
12	E4DSRA	VT2928
13	PS-1505	VT0092
14	GLKMC03	VT8847
15	NPN-734	VT3215

Here we are assuming the new system of part numbers is unrelated in any automatic or programmatic way to the old system. In other words, there is no simple single formula, be it a mathematical formula or a string expression, that we can apply that will do the job automatically for all rows in the table. That is why we will make the conversion using a **Replacement List** like that shown above. If we could make the conversion more automatic by using some kind of an expression, we would prefer to do it that way, but when we can't do that, then we need to use the **Replacement List** method.

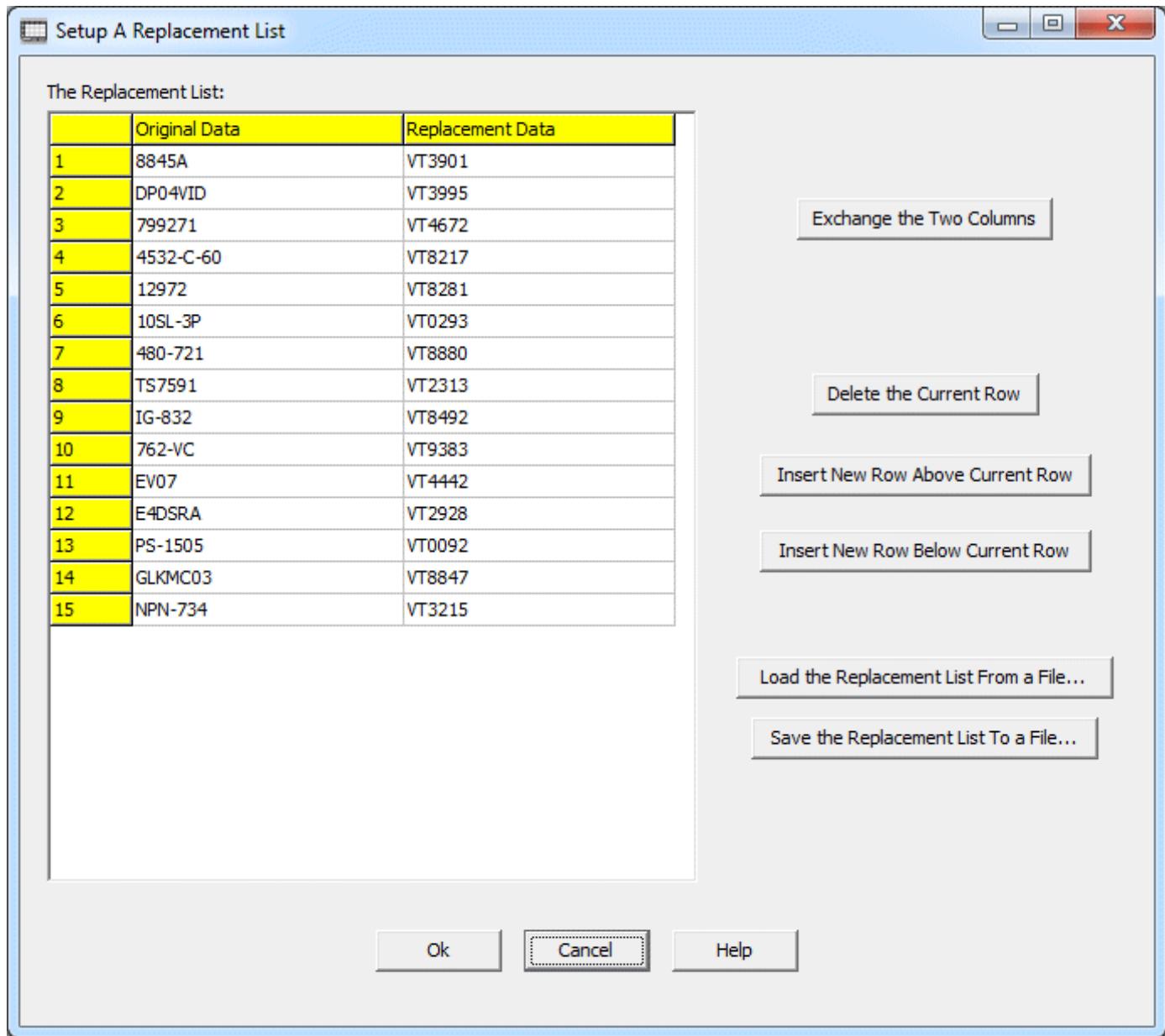
To perform the conversion, we first select all rows in **Column 4** in the original string grid, and then we choose the menu item **Conversions | Convert Using a Replacement List...** to bring the following dialog box.



The first thing we do in this dialog is click the button with the caption **Setup the Replacement List**. This brings up another dialog in which we can make and edit the actual **Replacement List** as already shown above.

Making a **Replacement List** is similar to making a **Pick List**. The only real difference is that a **Replacement List** has two columns in which every entry in each row in the first column corresponds to the entry in the same row in the second column. Thus on the previous page we have labeled the first column as the **Original Data** and we have labeled the second column as the **Replacement Data**. For our parts number problem we can assume the old part numbers are in the first column while the new part numbers are in the second column, all shown on the next page. All replacement lists are just 2-column CSV files.

When the conversion function executes, the program will scan the selected rows and read each cell value in the **Search Column** in the main grid. The program continues by trying to find the cell value anywhere in the entire first column of the **Replacement List**. If found, the cell value in the original main grid, but in the **Replace Column**, gets replaced by the value in the second column in the found row in the **Replacement List** grid. This kind of functionality is similar to a database join operation where the replacement data replaces the key column data. Only one column at a time is joined by this replacement operation. Usually the **Search Column** = the **Replace Column**, but these two column numbers can be different when you wish to make the replacement in a column different from the **Search Column**. When the checkbox is checked, changing either the **Search** or the **Replace** column number automatically changes the other number; otherwise the two column numbers will act independent of each other.



In this example we only have 15 rows of data, but in the real world you might have 15,000 rows. That will work just fine as long as each row in the **Replacement List** makes a pair of corresponding items.

In the above dialog the first column contains the old data while the second column contains the new data. If your pairings need to be reversed, you can just click the button that says **Exchange The Two Columns**. In any case, you will usually want to save your **Replacement List** in a CSV file before continuing because doing so allows you to use the same list again, or as many times as might ever be needed in the future.

When you are done creating and saving your **Replacement List** you can click the **Ok** button to close down this dialog. You should then see the dialog shown on the previous page. You should set all the parameters as needed for the conversion. This includes selecting the **Search** and **Replace** column numbers, and selecting a range of rows and selecting the case sensitivity option for how cell comparisons are made.

When everything is set as needed, you should click the **Ok** button in the dialog and the program will make the desired replacements in the **Replace** column in the original grid. After performing the replacements you will see a message that confirms exactly how many replacements were actually made. For our parts number example the message appears as:



For the part numbers example, the new main grid will only change **Column 4** to look as follows:

<	Column 1	Column 2	Column 3	Column 4
>	Item Description:	Old Part Number:	Price:	New Part Number:
1	Fluke Multimeter	8845A	\$1,455.00	VT3901
2	Tektronix Oscilloscope	DP04VID	\$868.00	VT3995
3	Power Sensor	799271	\$59.00	VT4672
4	RF Cable Assembly	4532-C-60	\$25.89	VT8217
5	Test Lead	12972	\$23.10	VT8281
6	Amphenol Connector	10SL-3P	\$10.95	VT0293
7	Panel Mount Connector	480-721	\$3.30	VT8880
8	Terminal Strip	TS7591	\$4.79	VT2313
9	Insulated Grip	IG-832	\$0.50	VT8492
10	Jacketed Video Cable	762-VC	\$42.79	VT9383
11	Tensioning Tool	EV07	\$300.00	VT4442
12	Frame Rack	E4DSRA	\$29.00	VT2928
13	DIN Power Rail	PS-1505	\$12.95	VT0092
14	Micro Switch	GLKMC03	\$33.45	VT8847
15	Photoelectric Sensor	NPN-734	\$91.57	VT3215

We left the old part numbers in **Column 2** just so you could compare the results with when we started this problem. If we no longer need the old part numbers, then to finish this example, we should swap **Column 2** with **Column 4** and then we should delete that **Column 4** with the old part numbers. If we still needed the old part numbers we would just leave them where they are in **Column 2**.

You may have noticed that conversions using a **Replacement List** are almost identical to a special case of using a **Block Fill** function where you choose the button to **Setup US State Names and Abbreviations**. With this particular **Block Fill** function, you can convert from a fully spelled out US State Name to a 2-letter abbreviation, or vice versa. In fact, our list of **US State Names** and their abbreviations can be used as a **Replacement List** because that list has the essential property that each row contains a pair of matching items. All **Replacement Lists** are just pairs of matching items, but they can now be used to perform a specialized but fully automated search and replacement function for any grid.

We should make one more comment about using replacement lists for converting data. In some applications you may have all your rows in a table identified by a single key column. But you may wish to synchronize those rows by adding or appending several data columns from another table that also has the same kind of a key column with similar key data. This is similar to a join operation, in database parlance. You can only join one column at a time, but using successive steps you can essentially join two grids together as long as those grids share a common key column.

Perhaps a specific example will better show what we are talking about. Consider the following two tables of data. The first table on the left contains a simple list of names and Social Security Numbers. The table on the right has a different number of rows, but one of the key columns in that table also has Social Security numbers. Now the operation we would like to perform is to almost merge the second table into the first, except the tables don't have the same number of rows and the rows are not sorted nor do they appear in any particular order in either table. We want to essentially insert or append the columns from the second table into the first table, but synchronize the rows and information as we do so, using the key column information from the second table.

Because the second table has fewer rows than the first table, we won't be able to make replacements using all the rows in the first table. But we will append to the first table columns using as much information as we can from the second table. We will append one column at a time from the second table, repeating the same steps four times for each of the four columns that contain the data to be appended. In the process we will first duplicate and then replace the Social Security numbers.

	Column 1	Column 2
<	Name:	SS Number:
1	Paula Boone	601-12-5515
2	Camila Gray	405-26-8423
3	Sheila Meadows	395-31-2131
4	Clint Burnis	802-47-2761
5	Edith Leblanc	624-47-9928
6	Chris Rich	476-61-2002
7	Lydia Ramirez	738-60-9874
8	Benjamin Tanner	774-04-6277
9	James Russell	283-49-7186
10	Tony Walker	399-63-4919
11	Harold Ryan	571-78-8986
12	Lucas Tate	618-56-7242
13	Frank Perkins	632-85-2733
14	Wanda Avila	620-76-9563
15	Anthony Logan	684-21-9560
16	Jessica Cardenas	876-25-3643

	Column 1	Column 2	Column 3	Column 4	Column 5
<	SS Number:	Work Status:	Age:	Marital Status:	# of Children:
1	876-25-3643	Working	27	Married	3
2	774-04-6277	Retired	71	Divorced	2
3	405-26-8423	Retired	38	Married	1
4	618-56-7242	Retired	77	Single	0
5	738-60-9874	Retired	51	Single	3
6	395-31-2131	Working	36	Divorced	2
7	399-63-4919	Retired	61	Single	3
8	476-61-2002	Retired	45	Married	0
9	571-78-8986	Retired	48	Single	2
10	684-21-9560	Retired	64	Divorced	3
11	601-12-5515	Working	72	Married	1
12	632-85-2733	Retired	33	Married	2
13	624-47-9928	Working	56	Married	3

The way to proceed is to first duplicate Column 2 in the first table as a new Column 3 in that same table. In the second table we would extract Column 1 and Column 2 to make our first replacement list. Just remember that all replacement lists are just 2-column **CSV** files. We show the results so far with the original table on the left and with the first replacement list on the right.

	Column 1	Column 2	Column 3
>	Name:	SS Number:	SS Number:
1	Paula Boone	601-12-5515	601-12-5515
2	Camila Gray	405-26-8423	405-26-8423
3	Sheila Meadows	395-31-2131	395-31-2131
4	Clint Burris	802-47-2761	802-47-2761
5	Edith Leblanc	624-47-9928	624-47-9928
6	Chris Rich	476-61-2002	476-61-2002
7	Lydia Ramirez	738-60-9874	738-60-9874
8	Benjamin Tanner	774-04-6277	774-04-6277
9	James Russell	283-49-7186	283-49-7186
10	Tony Walker	399-63-4919	399-63-4919
11	Harold Ryan	571-78-8986	571-78-8986
12	Lucas Tate	618-56-7242	618-56-7242
13	Frank Perkins	632-85-2733	632-85-2733
14	Wanda Avila	620-76-9563	620-76-9563
15	Anthony Logan	684-21-9560	684-21-9560
16	Jessica Cardenas	876-25-3643	876-25-3643

	Column 1	Column 2
>	SS Number:	Work Status:
1	876-25-3643	Working
2	774-04-6277	Retired
3	405-26-8423	Retired
4	618-56-7242	Retired
5	738-60-9874	Retired
6	395-31-2131	Working
7	399-63-4919	Retired
8	476-61-2002	Retired
9	571-78-8986	Retired
10	684-21-9560	Retired
11	601-12-5515	Working
12	632-85-2733	Retired
13	624-47-9928	Working

Then we open the table on the left and perform a **Convert Using a Replacement List**, using the table on the right as the replacement list. The first data table will change to the following:

	Column 1	Column 2	Column 3
>	Name:	SS Number:	Work Status:
1	Paula Boone	601-12-5515	Working
2	Camila Gray	405-26-8423	Retired
3	Sheila Meadows	395-31-2131	Working
4	Clint Burris	802-47-2761	802-47-2761
5	Edith Leblanc	624-47-9928	Working
6	Chris Rich	476-61-2002	Retired
7	Lydia Ramirez	738-60-9874	Retired
8	Benjamin Tanner	774-04-6277	Retired
9	James Russell	283-49-7186	283-49-7186
10	Tony Walker	399-63-4919	Retired
11	Harold Ryan	571-78-8986	Retired
12	Lucas Tate	618-56-7242	Retired
13	Frank Perkins	632-85-2733	Retired
14	Wanda Avila	620-76-9563	620-76-9563
15	Anthony Logan	684-21-9560	Retired
16	Jessica Cardenas	876-25-3643	Working

There are three rows that still contain Social Security numbers in Column 3, but we can manually blank those entries because we don't have any information for those three rows from the second data table.

To continue the example, we would operate on the previous table by duplicating Column 2 as a new Column 4. We would also extract Column 1 and Column 3 from the second data table to make a second replacement list. Just before making the replacements in Column 4 in the table on the left, the results would appear as:

<	Column 1	Column 2	Column 3	Column 4
>	Name:	SS Number:	Work Status:	SS Number:
1	Paula Boone	601-12-5515	Working	601-12-5515
2	Camila Gray	405-26-8423	Retired	405-26-8423
3	Sheila Meadows	395-31-2131	Working	395-31-2131
4	Clint Burris	802-47-2761		802-47-2761
5	Edith Leblanc	624-47-9928	Working	624-47-9928
6	Chris Rich	476-61-2002	Retired	476-61-2002
7	Lydia Ramirez	738-60-9874	Retired	738-60-9874
8	Benjamin Tanner	774-04-6277	Retired	774-04-6277
9	James Russell	283-49-7186		283-49-7186
10	Tony Walker	399-63-4919	Retired	399-63-4919
11	Harold Ryan	571-78-8986	Retired	571-78-8986
12	Lucas Tate	618-56-7242	Retired	618-56-7242
13	Frank Perkins	632-85-2733	Retired	632-85-2733
14	Wanda Avila	620-76-9563		620-76-9563
15	Anthony Logan	684-21-9560	Retired	684-21-9560
16	Jessica Cardenas	876-25-3643	Working	876-25-3643

<	Column 1	Column 2
>	SS Number:	Age:
1	876-25-3643	27
2	774-04-6277	71
3	405-26-8423	38
4	618-56-7242	77
5	738-60-9874	51
6	395-31-2131	36
7	399-63-4919	61
8	476-61-2002	45
9	571-78-8986	48
10	684-21-9560	64
11	601-12-5515	72
12	632-85-2733	33
13	624-47-9928	56

Then after the replacements are made in Column 4 we would see the new first data table as:

<	Column 1	Column 2	Column 3	Column 4
>	Name:	SS Number:	Work Status:	Age:
1	Paula Boone	601-12-5515	Working	72
2	Camila Gray	405-26-8423	Retired	38
3	Sheila Meadows	395-31-2131	Working	36
4	Clint Burris	802-47-2761		802-47-2761
5	Edith Leblanc	624-47-9928	Working	56
6	Chris Rich	476-61-2002	Retired	45
7	Lydia Ramirez	738-60-9874	Retired	51
8	Benjamin Tanner	774-04-6277	Retired	71
9	James Russell	283-49-7186		283-49-7186
10	Tony Walker	399-63-4919	Retired	61
11	Harold Ryan	571-78-8986	Retired	48
12	Lucas Tate	618-56-7242	Retired	77
13	Frank Perkins	632-85-2733	Retired	33
14	Wanda Avila	620-76-9563		620-76-9563
15	Anthony Logan	684-21-9560	Retired	64
16	Jessica Cardenas	876-25-3643	Working	27

By now you should be able to guess how we are going to proceed. We would duplicate Column 2 as a new Column 5 and we would setup a new replacement list by extracting columns 1 and 4 from the second data table.

	Column 1	Column 2	Column 3	Column 4	Column 5
<	Name:	SS Number:	Work Status:	Age:	SS Number:
1	Paula Boone	601-12-5515	Working	72	601-12-5515
2	Camila Gray	405-26-8423	Retired	38	405-26-8423
3	Sheila Meadows	395-31-2131	Working	36	395-31-2131
4	Clint Burris	802-47-2761			802-47-2761
5	Edith Leblanc	624-47-9928	Working	56	624-47-9928
6	Chris Rich	476-61-2002	Retired	45	476-61-2002
7	Lydia Ramirez	738-60-9874	Retired	51	738-60-9874
8	Benjamin Tanner	774-04-6277	Retired	71	774-04-6277
9	James Russell	283-49-7186			283-49-7186
10	Tony Walker	399-63-4919	Retired	61	399-63-4919
11	Harold Ryan	571-78-8986	Retired	48	571-78-8986
12	Lucas Tate	618-56-7242	Retired	77	618-56-7242
13	Frank Perkins	632-85-2733	Retired	33	632-85-2733
14	Wanda Avila	620-76-9563			620-76-9563
15	Anthony Logan	684-21-9560	Retired	64	684-21-9560
16	Jessica Cardenas	876-25-3643	Working	27	876-25-3643

<	Column 1	Column 2
>	SS Number:	Marital Status:
1	876-25-3643	Married
2	774-04-6277	Divorced
3	405-26-8423	Married
4	618-56-7242	Single
5	738-60-9874	Single
6	395-31-2131	Divorced
7	399-63-4919	Single
8	476-61-2002	Married
9	571-78-8986	Single
10	684-21-9560	Divorced
11	601-12-5515	Married
12	632-85-2733	Married
13	624-47-9928	Married

The above shows the data table on the left with the replacement list on the right. After replacing Column 5 using the replacement list we would see our next data table as:

	Column 1	Column 2	Column 3	Column 4	Column 5
<	Name:	SS Number:	Work Status:	Age:	Marital Status:
1	Paula Boone	601-12-5515	Working	72	Married
2	Camila Gray	405-26-8423	Retired	38	Married
3	Sheila Meadows	395-31-2131	Working	36	Divorced
4	Clint Burris	802-47-2761			802-47-2761
5	Edith Leblanc	624-47-9928	Working	56	Married
6	Chris Rich	476-61-2002	Retired	45	Married
7	Lydia Ramirez	738-60-9874	Retired	51	Single
8	Benjamin Tanner	774-04-6277	Retired	71	Divorced
9	James Russell	283-49-7186			283-49-7186
10	Tony Walker	399-63-4919	Retired	61	Single
11	Harold Ryan	571-78-8986	Retired	48	Single
12	Lucas Tate	618-56-7242	Retired	77	Single
13	Frank Perkins	632-85-2733	Retired	33	Married
14	Wanda Avila	620-76-9563			620-76-9563
15	Anthony Logan	684-21-9560	Retired	64	Divorced
16	Jessica Cardenas	876-25-3643	Working	27	Married

Ok, you should be able to figure out the last replacement on your own. We would duplicate Column 2 into a new Column 6 and then perform a replacement on that column after setting up the final replacement list containing the number of children. We just show the final result on the next page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Name:	SS Number:	Work Status:	Age:	Marital Status:	# of Children:
1	Paula Boone	601-12-5515	Working	72	Married	1
2	Camila Gray	405-26-8423	Retired	38	Married	1
3	Sheila Meadows	395-31-2131	Working	36	Divorced	2
4	Clint Burris	802-47-2761				
5	Edith Leblanc	624-47-9928	Working	56	Married	3
6	Chris Rich	476-61-2002	Retired	45	Married	0
7	Lydia Ramirez	738-60-9874	Retired	51	Single	3
8	Benjamin Tanner	774-04-6277	Retired	71	Divorced	2
9	James Russell	283-49-7186				
10	Tony Walker	399-63-4919	Retired	61	Single	3
11	Harold Ryan	571-78-8986	Retired	48	Single	2
12	Lucas Tate	618-56-7242	Retired	77	Single	0
13	Frank Perkins	632-85-2733	Retired	33	Married	2
14	Wanda Avila	620-76-9563				
15	Anthony Logan	684-21-9560	Retired	64	Divorced	3
16	Jessica Cardenas	876-25-3643	Working	27	Married	3

The above grid shows the final result. We have added four additional columns of data to our original first data table. We have also blanked out those entries where no replacements were made. If you now read the original two data tables and compare with our final table you should appreciate what we just accomplished.

If the second data table would have contained the same or even more rows than the first data table, and if every key entry in the first table appeared in some row in the second table, then there would be no blank entries in the final result. In other words, every row would have all new entries filled in with appropriate data.

This example of **Converting Using a Replacement List** to add columns and synchronizing key data and rows is actually very powerful. If you had a table with 16,000 rows instead of 16 rows you would really appreciate the results. Although we did a lot of separate operations, everything we did to create new tables was automatic, using existing columns in existing tables. There is no free lunch, but everything was very straightforward and the final result saved us a tremendous amount of manual labor that would have been subject to numerous copying and pasting and typing errors had we been foolish enough to try to do it manually.

This whole last example could have been done in one step using the command **File | Merge a CSV File Using a Common Key Column....** See the related help topic for how to make the process involving multiple columns even more efficient. We gave the above examples more for understanding than for actual practical use.

It is also possible to do conversions using replacement lists when the database key consists of more than one column. In that case you would first have to combine all the key columns into one column. Do this for both the original table and the table whose columns contain the data entries that are to be appended. Then perform the conversion as usual and then you could delete the single combined key columns from both the original table and the table used for appending.

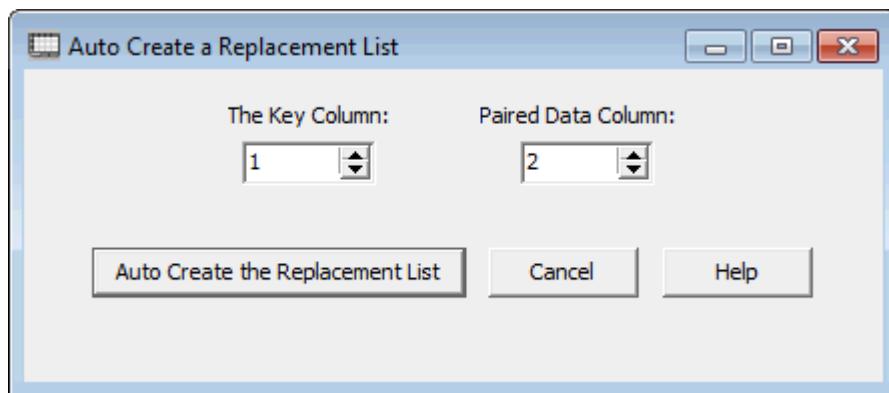
See also the help topic under the **Columns** menu that is titled **Copying Matching Data**. Another related topic is under the **File** menu, **Merging CSV Files With Common Key Columns**. These topics are closely related to performing a join operation on database tables.

Automatic Creation of a Replacement List

The previous help topic explains how to perform conversions using replacement lists. When you read that topic you will quickly learn that creating a replacement list requires a little preparation. In fact, in the last part of the previous help topic we showed an example in which we used key columns four times to manually save four different replacement lists in 4 distinct files.

Each time we opened another **CSV** data file and then we exported a pair of columns, the key column and the paired data column. As part of the column extraction we saved each replacement list in another **CSV** file and later we manually re-loaded those replacement lists each time we worked on another column. While this process is straightforward, we can short-cut the exporting and saving and re-loading steps.

Whenever you need to create a replacement list, you don't have to manually extract two columns to save the list and then manually reload it later. All you need do is open the **CSV** file that contains the key data and the replacement data (we also call this the paired data because it should be paired with the key data) and then select the menu item **Conversions | Replacement List Auto Creation...** and you will see the dialog:



All you need do with this dialog is to select **The Key Column** and select the **Paired Data Column** and click the button with the caption **Auto Create the Replacement List**.

Then what happens is the two chosen columns from the current grid will be automatically loaded into the two columns in the **Replacement List** dialog. To see the **Replacement List** dialog you must choose **Conversions | Convert Using a Replacement List...** and once the first dialog opens click the button with the caption **Setup the Replacement List** and you will be in the **Replacement List** dialog where you should see the most recent list that was created. Such a list isn't saved to disk, unless you do it from the **Replacement List** dialog.

As an example, imagine the current grid holds the **CSV** file with the Academy Awards data that appears as shown on the next page.

CSV Editor - AcademyAwards.csv

File Blocks Columns Rows Conversions Utilities Validation Uniqueness View Options Help

H+ ⌂ ⌄ A

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:	Tagged:
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton	T
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection	T
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather	T
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting	T
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II	T
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest	T
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky	T
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall	T
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter	T
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer	T
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People	T

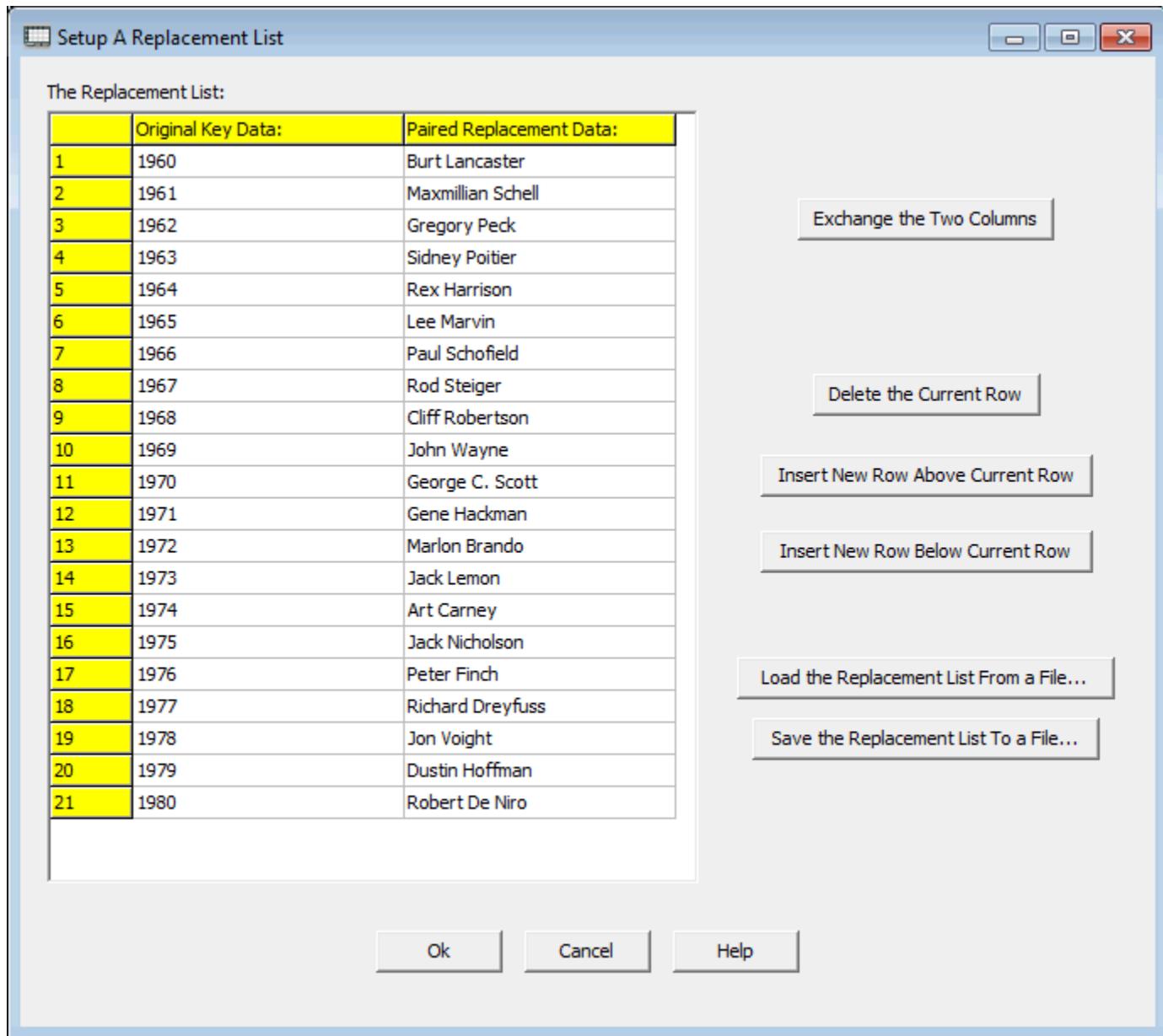
Row 1 Column 1

While this file is open, if we selected **Conversions | Replacement List Auto Creation...**, we would see the dialog box on the previous page. If we then chose Column 1 as **The Key Column** and if we chose Column 2 as the **Paired Data Column** and we clicked the button with the caption **Auto Create the Replacement List** we would just see the dialog close down and we would see the message flash

Replacement List Created

and we would be staring at the Academy Awards data.

However, if we then selected **Conversions | Convert Using a Replacement List...** and subsequently clicked the button with the caption **Setup the Replacement List** we would see the list that was automatically created and loaded as shown on the next page.



Above we can see that columns 1 and 2 have been filled in with data from the original Academy Awards grid. In this case we are using the year numbers as the key column data.

The loading of this list was done automatically and we would avoid having to manually extract the two columns to a new **CSV** file and we would avoid having to manually **Load the Replacement List From a File**.

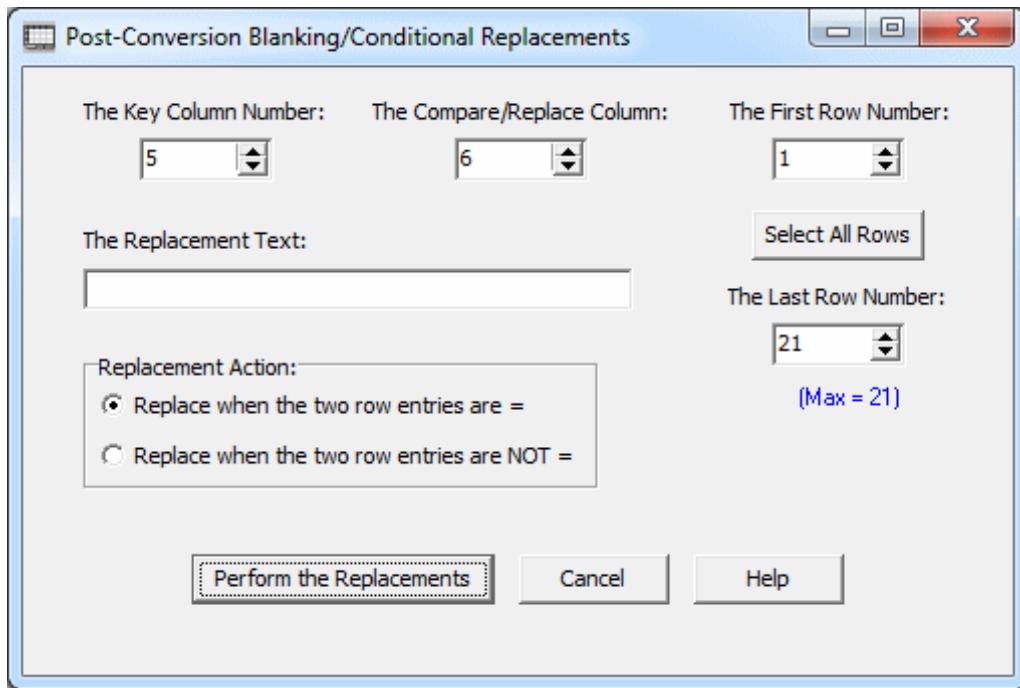
Incidentally, while you are viewing any replacement list, you might want to save it for future use. Otherwise, if you created the replacement list by extracting columns from an existing **CSV** file then your list should already be saved in a file. Auto creating a replacement list doesn't automatically save it to disk. Thus you may want to always use the button above labeled as **Save the Replacement List To a File...**

Post-Conversion Blanking/Conditional Replacements

Immediately after you perform a conversion using a replacement list, you can perform a special editing operation that we call **Post-Conversion Blanking**. This function can also be considered as making **Conditional Replacements**. Normally this function is applied to clean up those rows for which the conversion using a replacement list did not replace the key data because the key data was not found in the replacement list. However, other more general **Conditional Replacements** are also possible.

When used for **Post-Conversion Blanking**, this special function locates all the rows where the key data entries were not replaced, and it replaces those key values with blank text or with any other replacement text that you enter in the dialog below. The two main columns used with this function are called the **Key Column** and the **Compare/Replace Column**. When the contents in the two columns in a given row are compared, then the entry in that row in the **Compare/Replace Column** is automatically replaced by **The Replacement Text** string, depending on the **Replacement Action** that is chosen.

When you select the menu item **Conversions | Post-Conversion Blanking/Conditional Replacements...** you will see:



The three main controls define **The Key Column Number** and define **The Compare/Replace Column** and define the **Replacement Action**. The default **Replacement Text** is blank. The remaining controls are just for selecting a range of rows. **The Compare/Replace Column** is the column that will change when the two column entries are compared in each selected row.

Once you understand the purpose of this function it should be easy to apply. However, this function can be used without having to perform any kind of a conversion first. When performing true conditional replacements then the **Replacement Text** will probably not be blank.

We will give two examples of using this dialog. First, imagine you have an original grid that appears as shown below on the left. To the right of the grid we show a special 2-column replacement list. We are going to apply a regular conversion using the replacement list where we will be changing the entries in Column 4 in the original grid that has SS Numbers in Column 4. Those SS Numbers are the key values that will be replaced with the age numbers that are in **The Replacement List** shown on the right.

<	Column 1	Column 2	Column 3	Column 4
>	Name:	SS Number:	Work Status:	SS Number:
1	Paula Boone	601-12-5515	Working	601-12-5515
2	Camila Gray	405-26-8423	Retired	405-26-8423
3	Sheila Meadows	395-31-2131	Working	395-31-2131
4	Clint Burris	802-47-2761		802-47-2761
5	Edith Leblanc	624-47-9928	Working	624-47-9928
6	Chris Rich	476-61-2002	Retired	476-61-2002
7	Lydia Ramirez	738-60-9874	Retired	738-60-9874
8	Benjamin Tanner	774-04-6277	Retired	774-04-6277
9	James Russell	283-49-7186		283-49-7186
10	Tony Walker	399-63-4919	Retired	399-63-4919
11	Harold Ryan	571-78-8986	Retired	571-78-8986
12	Lucas Tate	618-56-7242	Retired	618-56-7242
13	Frank Perkins	632-85-2733	Retired	632-85-2733
14	Wanda Avila	620-76-9563		620-76-9563
15	Anthony Logan	684-21-9560	Retired	684-21-9560
16	Jessica Cardenas	876-25-3643	Working	876-25-3643

The Replacement List:

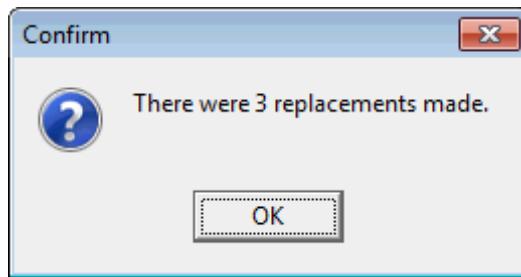
	Original Data	Replacement Data
1	876-25-3643	27
2	774-04-6277	71
3	405-26-8423	38
4	618-56-7242	77
5	738-60-9874	51
6	395-31-2131	36
7	399-63-4919	61
8	476-61-2002	45
9	571-78-8986	48
10	684-21-9560	64
11	601-12-5515	72
12	632-85-2733	33
13	624-47-9928	56

After the normal conversion is made using the function under **Conversions | Convert Using a Replacement List**, the new grid appears as:

<	Column 1	Column 2	Column 3	Column 4
>	Name:	SS Number:	Work Status:	Age:
1	Paula Boone	601-12-5515	Working	72
2	Camila Gray	405-26-8423	Retired	38
3	Sheila Meadows	395-31-2131	Working	36
4	Clint Burris	802-47-2761		802-47-2761
5	Edith Leblanc	624-47-9928	Working	56
6	Chris Rich	476-61-2002	Retired	45
7	Lydia Ramirez	738-60-9874	Retired	51
8	Benjamin Tanner	774-04-6277	Retired	71
9	James Russell	283-49-7186		283-49-7186
10	Tony Walker	399-63-4919	Retired	61
11	Harold Ryan	571-78-8986	Retired	48
12	Lucas Tate	618-56-7242	Retired	77
13	Frank Perkins	632-85-2733	Retired	33
14	Wanda Avila	620-76-9563		620-76-9563
15	Anthony Logan	684-21-9560	Retired	64
16	Jessica Cardenas	876-25-3643	Working	27

Now it is to this last grid that we will apply the **Post-Conversion Blanking** function to clean up the SS Numbers that remain in Column 4, that are not age numbers. We select Column 2 as the **Key Column**, we select Column 4 as the **Compare/Replace Column**, and we operate on all rows. We also leave **The Replacement Text** blank and set the **Replacement Action** to perform the replacement when the two row entries are equal.

The first message we see tells how many replacements were actually made.



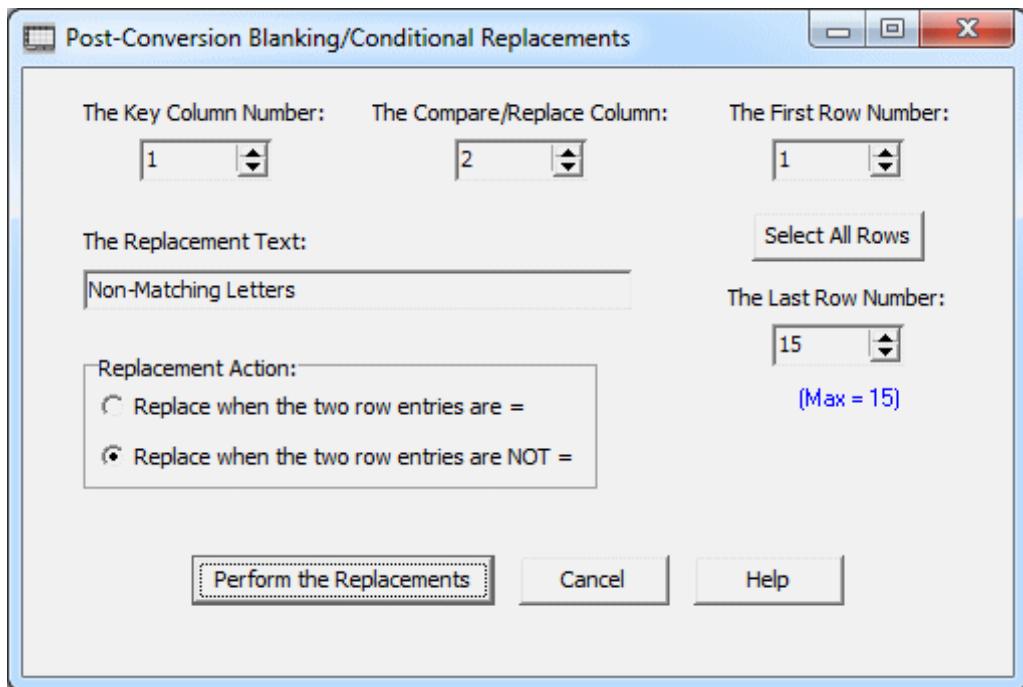
When we inspect the new Column 4 we can see that only the rows that originally had duplicate SS Numbers in columns 2 and 4 have been changed so those rows now contain blank entries in Column 4. This example demonstrates a typical use of the **Post-Conversion Blanking** function. We blanked those three rows that did not have a matching SS Number in the replacement list when the conversion was performed.

<	Column 1	Column 2	Column 3	Column 4
>	Name:	SS Number:	Work Status:	Age:
1	Paula Boone	601-12-5515	Working	72
2	Camila Gray	405-26-8423	Retired	38
3	Sheila Meadows	395-31-2131	Working	36
4	Clint Burris	802-47-2761		
5	Edith Leblanc	624-47-9928	Working	56
6	Chris Rich	476-61-2002	Retired	45
7	Lydia Ramirez	738-60-9874	Retired	51
8	Benjamin Tanner	774-04-6277	Retired	71
9	James Russell	283-49-7186		
10	Tony Walker	399-63-4919	Retired	61
11	Harold Ryan	571-78-8986	Retired	48
12	Lucas Tate	618-56-7242	Retired	77
13	Frank Perkins	632-85-2733	Retired	33
14	Wanda Avila	620-76-9563		
15	Anthony Logan	684-21-9560	Retired	64
16	Jessica Cardenas	876-25-3643	Working	27

Now for a non-traditional true **Conditional Replacement** example. Consider the grid shown below in which columns 1 and 2 have various combinations of the letters T and F that might represent True and False values. Both columns contain a random mixture of these two values in their rows.

<	Column 1	Column 2
1 >	T	T
2	T	F
3	T	T
4	F	F
5	T	T
6	T	F
7	F	T
8	F	T
9	F	F
10	F	F
11	F	T
12	T	F
13	F	T
14	T	T
15	F	F

Next we open and setup the **Post-Conversion Blanking/Conditional Replacements** dialog to contain the following values. In this case we have **Non-Matching Letters** for **The Replacement Text** and we have set the **Replacement Action** to perform the replacement when the two row entries are NOT equal.



After we click the button to **Perform the Replacements**, the new grid will appear as shown below on the right. The grid on the left is the original grid. When you compare the two grids you can determine Column 2 in the left grid has changed to what appears in the right grid Column 2, but only where the letters did not match. The rows that did match were left unchanged.

<	Column 1	Column 2
1 >	I	T
2	T	F
3	T	T
4	F	F
5	T	T
6	T	F
7	F	T
8	F	T
9	F	F
10	F	F
11	F	T
12	T	F
13	F	T
14	T	T
15	F	F

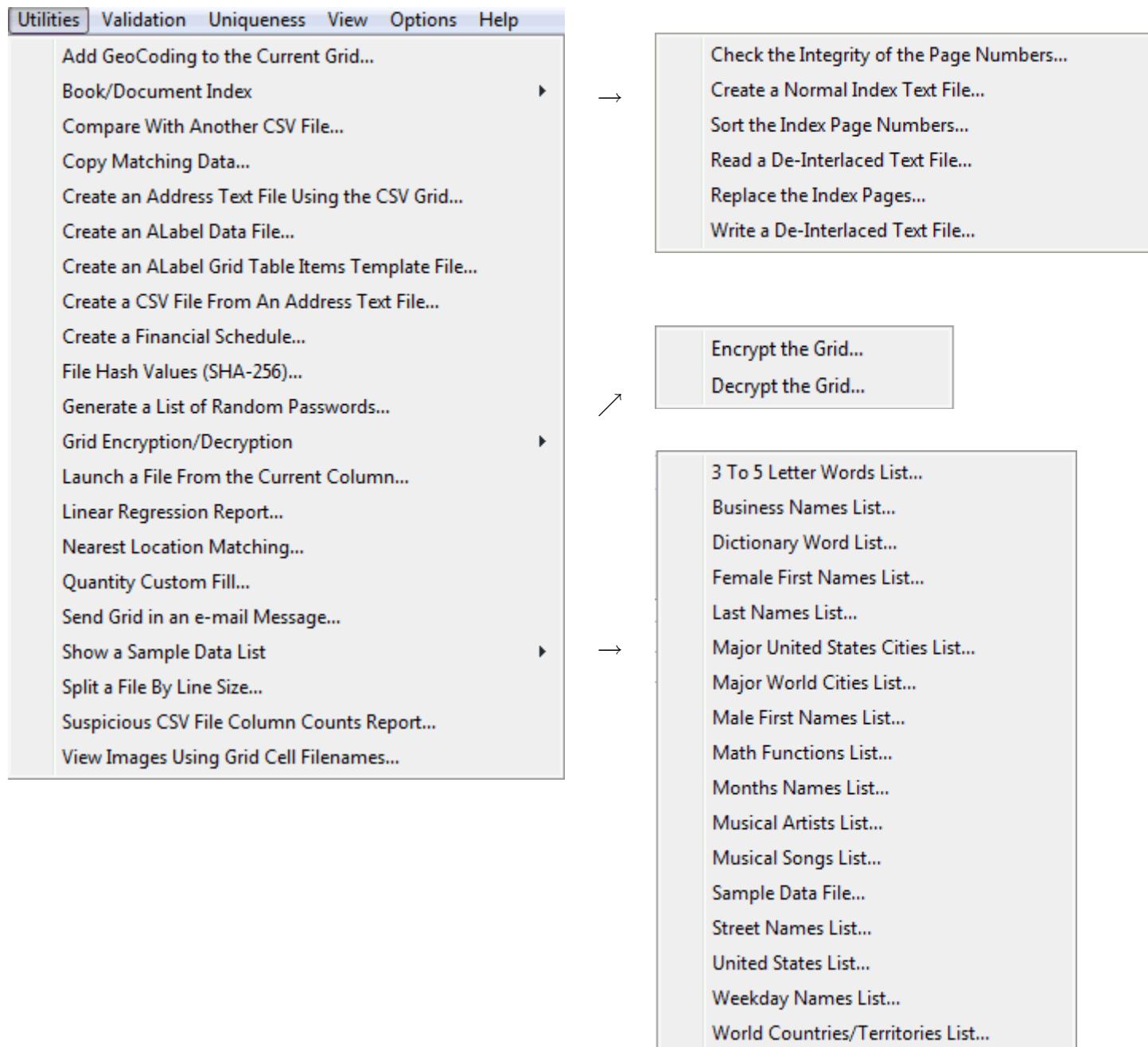
<	Column 1	Column 2
1 >	T	T
2	T	Non-Matching Letters
3	T	T
4	F	F
5	T	T
6	T	Non-Matching Letters
7	F	Non-Matching Letters
8	F	Non-Matching Letters
9	F	F
10	F	F
11	F	Non-Matching Letters
12	T	Non-Matching Letters
13	F	Non-Matching Letters
14	T	T
15	F	F

The action in this second example is best thought of as performing a conditional test on each row in the first grid. The conditional test is whether the two row entries are not the same, in that original grid. When the two entries are different, then the text entry in Column 2 was changed to **Non-Matching Letters**. This is **The Replacement Text** that was entered in the dialog as shown on the previous page.

The Utilities Menu

The **Utilities** menu contains functions that can be considered part of a suite of applications related to working with **CSV** files. In some cases the **CSV** files represent entire applications and in other cases the files are just one table that is part of a larger database set of tables. As a whole, these functions all perform utilitarian types of actions.

In some cases these functions might have been placed under the **File** menu because they may read or write external files, but then the **File** menu would appear substantially overloaded. Thus we created another menu and named this the **Utilities** menu.



Adding Latitude and Longitude Information To a Grid

One of the functions under the **Utilities** menu gives you the ability to add what is called **GeoCoding** information to a **CSV** grid. Before you do this you should have already opened a **CSV** file that has at least three columns of standard address information. As an example, consider the grid that appears as:

	Column 1	Column 2	Column 3	Column 4	Column 5
<	Street Address:	City:	State:	Latitude:	Longitude:
1	13425 Washington Blvd.	Culver City	CA		
2	922 Gayley Ave.	Los Angeles	CA		
3	9149 South Sepulveda Blvd.	Westchester	CA		
4	9245 Venice Blvd.	Los Angeles	CA		

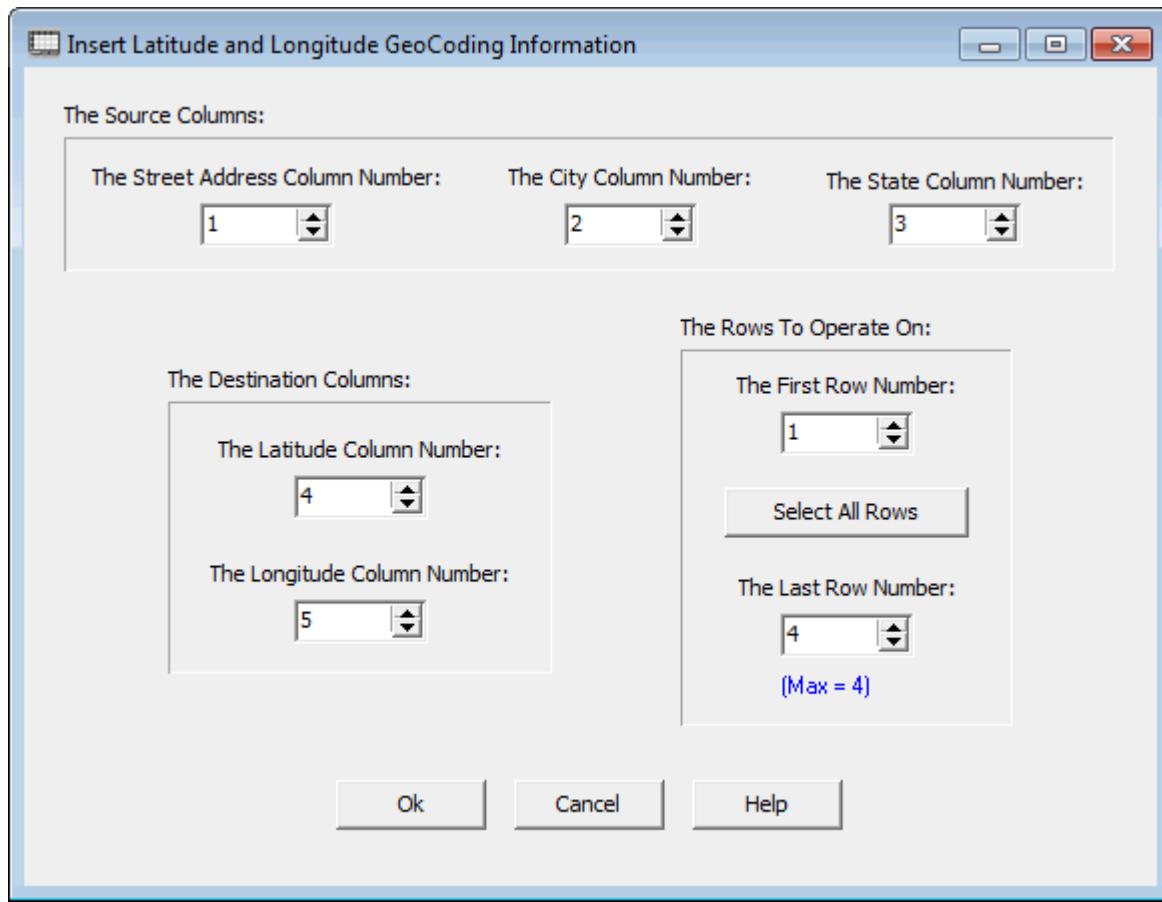
This example grid contains a list of four real addresses of **In & Out Hamburger Drive-In** locations in the Los Angeles area of southern California. The original list only had three columns, so we added two additional columns that we labeled as **Latitude:** and **Longitude:**. These two terms are also known as **GPS** coordinates.

Now if you wanted to get the **GPS** coordinates for these four businesses, you could open your browser and use **Google Maps** to find and click on each location to get its **GPS** coordinates. Then you could manually type in those two coordinates in the corresponding grid cells. This is time consuming and involves a lot of manual labor, even for just four locations.

Fortunately, the **CSV Editor** program has an automatic function that allows you to fill in the last two columns without doing any manual lookup or manual typing. The only real requirement for a successful operation is that any and all addresses you use must be real existing addresses. If any address is determined to be spurious then empty strings will be inserted in the two **GPS** grid cells.

We use **Google Maps** to perform the lookup of the **GPS** coordinates for each address. The **GPS** coordinates are simply the Latitude and Longitude that **Google** determines for the given address. These values are usually returned with 8 decimal places of accuracy. However, you should not expect to make more than about 2,500 **GPS** coordinate requests in any one 24 hour period. **Google** does not want users to abuse their free system, so they limit how much use you can make of it.

To get started, open the **Utilities** menu item that says **Add GeoCoding to the Current Grid...** This will cause the following dialog to appear:



You need to identify three source columns that represent the **Street Address**, and the **City Name**, and the **State Name**. These source columns can be any three different columns in your grid. They don't have to be consecutive columns and they can be in any order. Use the controls in the dialog to set each column number.

You also need to set the two column numbers for the **Latitude** and **Longitude** columns. These two columns don't have to be adjacent to each other; they can be any two columns in any order. They don't have to be the last two columns. Last, select a range of consecutive rows that you will apply the **GeoCoding** information to. When you click the **Ok** button, the program will automatically access the **Internet** and it will try to fill in the two **GPS** coordinates for you. For the above example grid the following shows the result.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Street Address:	City:	State:	Latitude:	Longitude:
1	13425 Washington Blvd.	Culver City	CA	33.9917070	-118.4460640
2	922 Gayley Ave.	Los Angeles	CA	34.063090	-118.4479960
3	9149 South Sepulveda Blvd.	Westchester	CA	33.9537110	-118.3963480
4	9245 Venice Blvd.	Los Angeles	CA	34.02629790	-118.39419110

When you try to fill a large number of rows with latitudes and longitudes it may happen that some rows are skipped in the process. We don't have control over this because **Google Maps** is responsible for the skipping. We can only suggest that you sort your rows by the latitude column and then choose the range of rows that are blank and make another try at filling those rows. You may have to do this more than once until all the rows are appropriately filled. If after several tries you still have rows that remain blank then the more probable culprit is that your addresses are improper. In this case we suggest you change the address information.

To help matters, we only look up three addresses every 3 seconds, so you can expect it take over a minute to fill 100 addresses. Just be patient as you will see a series of popup messages every three seconds that say "**Waiting on Google...**". Then when everything is finished you will see one final message that says "**Finished Google Maps Geocoding...**".

See also the function under **Utilities | Nearest Location Matching...** that makes use of **GPS** coordinates.

Reading a De-Interlaced Text File

This help topic describes how to create an index for any book or document. Other help topics deal with how to manipulate a book index CSV grid, and they precisely describe the requirements for such a grid, but they don't explain the easiest way to create such a grid in the first place. We have heard experienced book publishers describe their philosophies about what should go in a book index, and there are many word processors that assist in making book indexes. But it is rare to have someone tell you how simple it is to make a first crack at a book index. We will next describe a very simple set of tasks that comprise three stages.

In the first stage to create a book index, all you need do is start on page 1 of a book. Scan the page and if you see an item on that page that you think should be in the book index, say it is the word **cache**, then write the page number and the text item in the following order in a simple text editor.

```
1 cache
```

Then go to page 2 in the book and scan that page for anything that should be in the index. If nothing on page 2 strikes your fancy then skip that page and go to page 3. Let's assume on page 3 you find two items for the index that are the word **differentiation** and the phrase **implicit definition**. Then add a new line to your index where you separate the two items on page 3 with a semi-colon. Your index should now appear as:

```
1 cache
3 differentiation; implicit definition
```

Believe it or not, everything we have described up to this point is the simplest of algorithms to create a book index in its first stage. Scan each next page and determine if there are any items on the page that should be in the index. If there are no items then skip that page. Write the page number first, and if there is more than one item, write the items separated by a semi-colon character. After you scan about 25 pages your first stage initial book index might look similar to the following example. This is what we refer to as the De-Interlaced form.

```
1 cache
3 differentiation; implicit definition
4 analytical engine; von Neumann
5 analog device; kernel;
6 associative law; binary arithmetic
8 calculus; hexadecimal; Fortran
9 integrator; NOR gate; cache
10 Fortran; index register; numerical analysis
11 erasable PROM; bias
12 branch instruction; GOTO
13 joystick; monomorphism; modulo
15 fourth generation
18 stepwise refinement; kernel; hexadecimal
19 screen dump; nonlinear regression
20 NOR gate; identifier; Fortran
21 idempotent law; content-addressable
22 context free; condition number;
23 reverse Polish; bias; optical storage
25 keypunch; kernel
```

Creating a book index isn't any more exotic than this! The previous statement about semi-colons assumes no phrase in your index will contain the semi-colon character. What is shown on the previous page is precisely what we call a De-Interlaced book index. It is a sorted list of page numbers, with all the items for a page on one line, and with multiple items in a line separated by semi-colons.

If you save these lines in a text file and select the menu item **Utilities | Book/Document Index | Read a De-Interlaced Text File...** you will be prompted to open that text file. Immediately save the new grid as directed. Then you should see a grid that appears as:

<	Column 1	Column 2	Column 3	Column 4
1 >	analog device	5		
2	analytical engine	4		
3	associative law	6		
4	bias	11	23	
5	binary arithmetic	6		
6	branch instruction	12		
7	cache	1	9	
8	calculus	8		
9	condition number	22		
10	content-addressable	21		
11	context free	22		
12	differentiation	3		
13	erasable PROM	11		
14	Fortran	8	10	20
15	fourth generation	15		
16	GOTO	12		
17	hexadecimal	8	18	
18	idempotent law	21		
19	identifier	20		
20	implicit definition	3		
21	index register	10		
22	integraph	9		
23	joystick	13		
24	kernel	5	18	25
25	keypunch	25		
26	modulo	13		
27	monomorphism	13		
28	nonlinear regression	19		
29	NOR gate	9	20	
30	numerical analysis	10		
31	optical storage	23		
32	reverse Polish	23		
33	screen dump	19		
34	stepwise refinement	18		
35	von Neumann	4		

Voila! It's a Book Index in grid form! This is a nice example of a new book index grid.

Before you select the menu item to **Read a De-Interlaced Text File...** you want to make sure the text file you open has the format shown two pages earlier. The reason the De-Interlaced format is so important is that it shows what a book index looks like at birth. Our function that reads the De-Interlaced format automatically separates out the text items and the page numbers into distinct columns and it throws away the semi-colon characters. It performs a case insensitive ascending sort on the text items in Column 1 and finally it makes the page numbers sorted, ascending within each grid row. These are the tasks that no human should ever have to do manually!

The two functions **Read a De-Interlaced Text File** and **Write a De-Interlaced Text File** turn out to be nearly inverse functions of each other. Later we will explain the *nearly* part of the previous sentence.

We can summarize all the book index related functions that are under the **Utilities** menu. If you are creating a book index from scratch, then follow the directions at the start of this help topic and first write and save the de-interlaced lines in a text file. Then choose **Utilities | Book/Document Index | Read a De-Interlaced Text File...** to read that file and create the first draft of the book index grid. Be sure to always immediately save that grid as directed. The grid looks like a real book index, but is in grid form.

Later, if you need to change the index because of editing pages (page insertions, deletions, and moves), then choose the menu item **Utilities | Book/Document Index | Replace The Index Pages....** Remember, the reason you want to mostly keep your book index in grid form is because of all the automatic functionality we provide to make book index replacements easy. You could also do some minor manual editing or tweaking of the index.

After all the editing changes are complete, then you should be prepared for the final stage. Just select the menu item **Utilities | Book/Document Index | Create a Normal Index Text File....** This step creates the form of the book index that is normal text that can be copied and pasted into a word processor. This text will normally form the last pages in your book or document. This is the last step needed to get to the final useable form of your book index. Of course, after you paste the text into your word processor you may make some further touch ups within your word processor. We will discuss some other custom index formatting options later.

This help topic has shown the three stages a book index goes through before it is complete.

Stage 1: Manually create the De-Interlaced form of the index.

This stage is normally done only once and can then be dispensed with.

Stage 2: Convert the De-Interlaced form into the grid form.

The grid form is used to edit the index and to make automatic page replacements as you edit your book or document. Save the grid form and keep backup copies of it because you never know when you may update your book or document.

Stage 3: Convert the grid form to the normal text form.

The normal text form is what can be pasted into a word processor.

The next page shows the last stage for our example. You should note that letters not used as starting letters in Column 1 of the grid do not appear here either. The grid should normally be kept sorted on Column 1 with a case insensitive sort. The next page shows the normal text form that is automatically created from the grid form.

A
analog device 5
analytical engine 4
associative law 6

B
bias 11,23
binary arithmetic 6
branch instruction 12

C
cache 1,9
calculus 8
condition number 22
content-addressable 21
context free 22

D
differentiation 3

E
erasable PROM 11

F
Fortran 8,10,20
fourth generation 15

G
GOTO 12

H
hexadecimal 8,18

I
idempotent law 21
identifier 20
implicit definition 3
index register 10
integraph 9

J
joystick 13

K
kernel 5,18,25
keypunch 25

M
modulo 13
monomorphism 13

N
nonlinear regression 19
NOR gate 9,20
numerical analysis 10

O
optical storage 23

R
reverse Polish 23

S
screen dump 19
stepwise refinement 18

V
von Neumann 4

Performing Book Index Replacements

In addition to helping you create an index for a book, this program has a special function that allows you to update all the page numbers for a book, after you insert or delete or move pages in the book. This involves performing replacements of page numbers that are in an existing book index grid that needs to be updated. This job deserves to be automated and even works if you just need to move some existing pages without inserting or deleting any pages. We will use the grid shown below to illustrate how this function works.

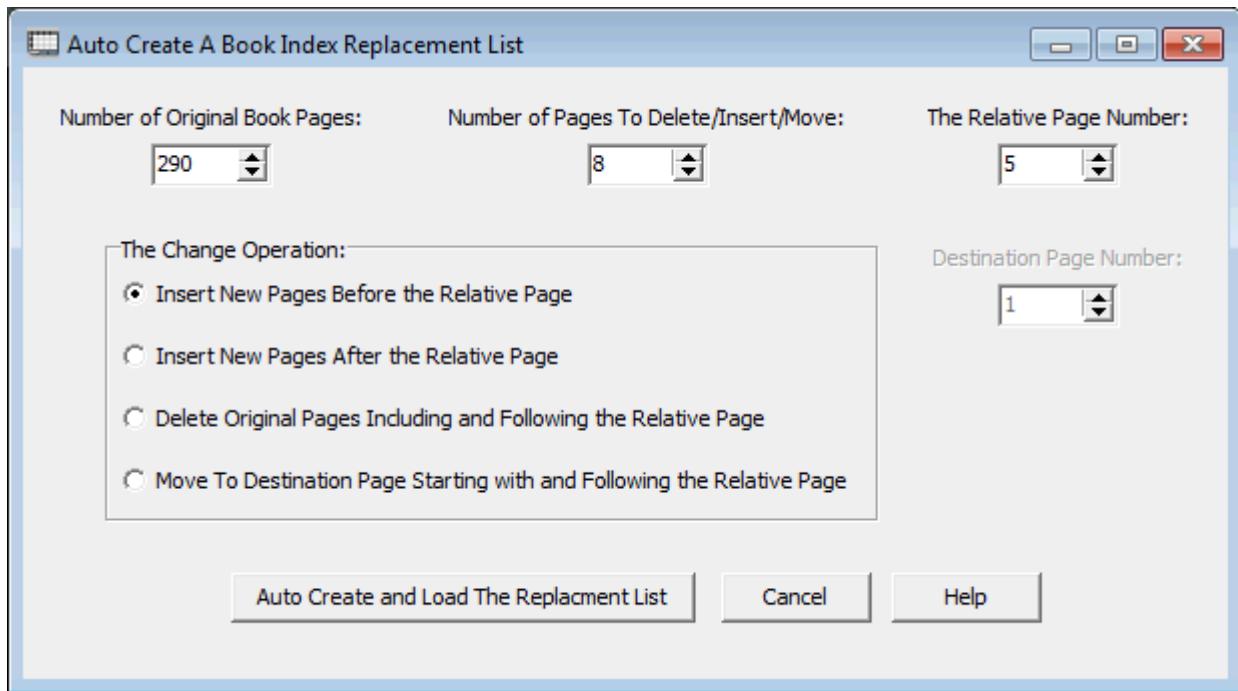
<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Index Term:	Occurrence 1:	Occurrence 2:	Occurrence 3:	Occurrence 4:
1	Academy Awards grid	3	14	23	33
2	accuracy factor	174			
3	address text file	20	21		
4	Adobe Reader	1			
5	ALabel program	1	25		
6	base path option	261			
7	basic limitations	10			
8	binary	176			
9	blank cells	133	153	165	
10	blank column removal	113			
11	Boolean data	7	81	218	
12	bottom	164			
13	Celsius	89	194		
14	clearing a column	115			
15	clearing a row	155			
16	column character extents	259			
17	column display widths	257			
18	column report	30			
19	data types	7			
20	date data	7	180	220	
21	date formats	180	221		
22	date sequence	57			
23	decimal	176			
24	decimal digits	79	174	194	

Imagine we have the above index list as an original book index grid. Further assume that after we have typed in all the page numbers in the above grid that we insert 8 additional new pages to the book, just after the original page number 5. This means most, but not all, of the above page numbers need to be increased by 8. If we had additional index entries on any of those 8 pages we would also need to manually add entries to the grid.

The **CSV Editor** program has a special function that will allow us to easily update all the page numbers in the above grid. First, you will need to know how many total pages are in the book and you will need to know how many new pages need to be inserted, or you need to know how many existing pages will be deleted. Finally you need to know where you want the insertions or deletions to be made. Armed with this knowledge you should first open the grid that contains the existing or original book index information. Such a grid should be like that shown above. Then continue as described in the next help topic.

Auto Create A Book Index Replacement List

You first want to load an existing book index CSV file. You need to know how you will change the existing grid by either adding some new pages or deleting or just moving some existing pages. After the grid pages are automatically updated, then you can manually add new entries to the grid or you can manually edit existing entries. But first you should automatically update the pages in the grid as a result of adding or deleting or moving pages in your book. When you are ready to automatically update the pages, select the menu item **Utilities | Book/Document Index | Replace the Index Pages....** This will bring up a dialog like that shown below, after the program performs an integrity check on the existing grid. If any errors are found this function aborts.



You should first enter the **Number of Original Book Pages** in the first control. Then set the **Number of Pages To Delete/Insert/Move**. Then enter **The Relative Page Number**. **The Relative Page Number** is simply a page number related to where the change operation will take place. When inserting **Before the Relative Page**, the page before the relative page will be the last page number that won't change. When inserting **After the Relative Page**, the relative page will be the last page number that won't change. When deleting pages, the relative page will be the first page that gets deleted. When moving pages, **The Relative Page Number** is the first page in the existing set of consecutive pages to be moved.

The **Destination Page Number** is only used when moving existing pages without inserting or deleting any pages. It is the number of the first page in the move block, AFTER the move is made. Determining the **Destination Page Number** is more carefully explained below. This control will be inactive and grayed-out unless you choose to make a **Change Operation** that is a **Move** operation. Finally make the appropriate choice for **The Change Operation**.

Then click the button with the caption **Auto Create and Load The Replacements List** and you should then see a replacement list that is automatically created for you. Correctly filling out the above dialog is all that is required for the program to make the page replacement list.

For our first example, the list will appear as shown below. Here we are inserting 8 new book pages, starting after page 5. So the first actual page number change will occur with old page number 6 that will change to the new page number 14. All remaining pages after 6 will also change by adding 8 to the original page number.

Setup A Replacement List

The Replacement List:

	Original Data	Replacement Data
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	14
7	7	15
8	8	16
9	9	17
10	10	18
11	11	19
12	12	20
13	13	21
14	14	22
15	15	23
16	16	24
17	17	25
18	18	26
19	19	27
20	20	28
21	21	29
22	22	30

Exchange the Two Columns

Delete the Current Row

Insert New Row Above Current Row

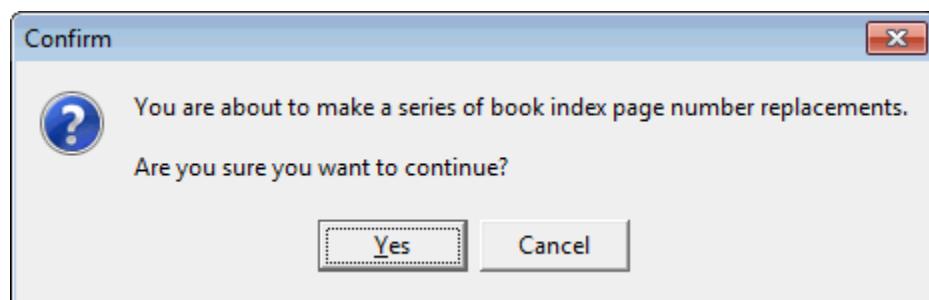
Insert New Row Below Current Row

Load the Replacement List From a File...

Save the Replacement List To a File...

Ok Cancel Help

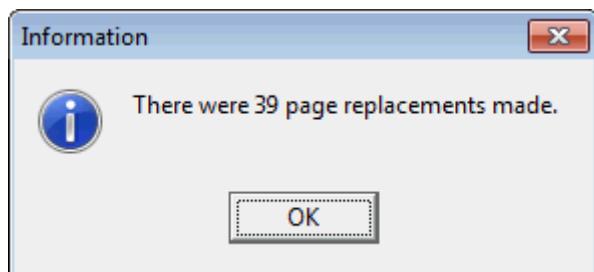
This special 2-column grid is going to be used to automatically make all the page replacements. When you click the **Ok** button the program will first prompt you with a preliminary confirmation message:



If you click the **Cancel** button then the entire operation will be aborted.

Otherwise, if you click the **Yes** button the program will automatically update all the page numbers in the current book index grid.

You should then see a message like the following telling you how many page replacements were actually made.

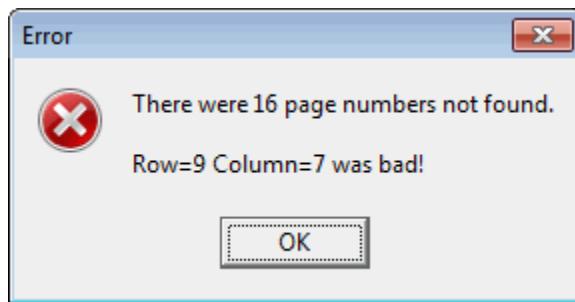
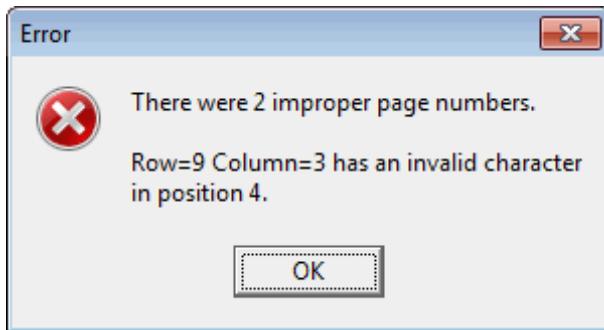


For this particular example the new grid should appear as shown below. Compare with the grid shown three pages earlier in this help file.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Index Term:	Occurrence 1:	Occurrence 2:	Occurrence 3:	Occurrence 4:
1	Academy Awards grid	3	22	31	41
2	accuracy factor	182			
3	address text file	28	29		
4	Adobe Reader	1			
5	ALabel program	1	33		
6	base path option	269			
7	basic limitations	18			
8	binary	184			
9	blank cells	141	161	173	
10	blank column removal	121			
11	Boolean data	15	89	226	
12	bottom	172			
13	Celsius	97	202		
14	clearing a column	123			
15	clearing a row	163			
16	column character extents	267			
17	column display widths	265			
18	column report	38			
19	data types	15			
20	date data	15	188	228	
21	date formats	188	229		
22	date sequence	65			
23	decimal	184			
24	decimal digits	87	182	202	

Errors can occur when the replacements are made. You should note that the book index replacement function is very special and differs from a normal string replacement function that only acts in one column. Book index replacements happen in multiple columns, from Column 2 on.

In any case, when the program is unable to locate one or more page numbers in your currently opened grid you may see an error message like the following. Only the last bad cell is reported, but in case of any error, you can undo the operation and fix the grid and re-try the operation. An error message similar to either of the following might appear:



The first kind of error message will abort the operation, while the second error message might appear after some pages have been processed, but there were page numbers found that were out of range.

You may see other informational types of messages. When you delete pages, the program may delete one or more columns or rows from the opened book index grid. Such deletions are only made when any row or any column in the original grid is left holding only empty entries. Thus such deletions should be natural, but we like to tell you exactly how many rows or columns, if any, were automatically deleted. Of course, no such deletions will even be attempted if you are inserting new pages or moving a set of consecutive existing pages.

Next we show an example replacement list where the operation is to delete 4 pages from a 100-page book, starting at page 12.

The Replacement List:		
	Original Data	Replacement Data
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
11	11	11
12	12	Remove
13	13	Remove
14	14	Remove
15	15	Remove
16	16	12
17	17	13
18	18	14
19	19	15
20	20	16

You should note that those page numbers that have been designated to be deleted have the word **Remove** in the 2nd column of the replacement list. All page numbers following the last **Remove** are the same numbers in the 1st column, but they are subtracted by 4. In the 2nd column, pages 1-11 remain unchanged from their column 1 counterparts because none of the pages change until we get to page 12, the first page to be deleted.

The **Remove** word is not actually used as replacement data, but it is used instead as an instruction that gets executed as part of the entire replacement function action. We show the word **Remove** in replacements lists like the above list just so you can know what pages will be removed or deleted.

What really happens internally is that deleted page cells get replaced with empty cells (not with the word **Remove**), and empty cells within a given row get moved to the right-most column in that row while at the same time all the non-empty entries to the right of an empty cell get moved left by one column. Then as part of a last clean-up operation, all empty cells that can be removed as entire rows or as entire columns will be deleted from the book index grid. Thus the size of a book index grid can sometimes shrink as a result of choosing to delete some pages. Such shrinkage only occurs when an entire row or an entire column contains all empty cells after all the page replacements get made. Such shrinkage may be rare, but it can happen, and is entirely dependent on the nature of the data contained in the rows and columns of your book index grid.

The example below shows the before and after grids for a book index. The book had 100 pages before we deleted 4 pages, starting at page 12. Column 1 consists of random last names without any real meaning.

The original book index before deleting 4 pages:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1 >	Allen	11	12	51		
2	Alvarez	4	5	72		
3	Anderson	5	6	98		
4	Andrews	19	20	63		
5	Branch	10	11	47		
6	Cummings	14	15	16		
7	Davenport	9	10	31		
8	Gates	16	17	67		
9	Grant	7	8	10		
10	Gray	3	4	84		
11	Guzman	8	9	16		
12	Haney	20	21	22		
13	Head	12	18	49		
14	Hogan	6	7	64		
15	Kennedy	12	13	17		
16	Rosa	1	12	79	88	
17	Sears	17	18	92		
18	Spence	15	16	17		
19	Strong	2	3	9	13	14
20	Vinson	12	13	14		

The new book index after deleting 4 pages:

<	Column 1	Column 2	Column 3	Column 4
1 >	Allen	11	47	
2	Alvarez	4	5	68
3	Anderson	5	6	94
4	Andrews	15	16	59
5	Branch	10	11	43
6	Cummings	12		
7	Davenport	9	10	27
8	Gates	12	13	63
9	Grant	7	8	10
10	Gray	3	4	80
11	Guzman	8	9	12
12	Haney	16	17	18
13	Head	14	45	
14	Hogan	6	7	60
15	Kennedy	13		
16	Rosa	1	75	84
17	Sears	13	14	88
18	Spence	12	13	
19	Strong	2	3	9

When this example was run, there were a total of 63 index page replacements made. There were two columns that got deleted and there was 1 row that got deleted.

The row that got deleted was the last row with the name Vinson. The Vinson row was deleted because all three of its original page references were for pages that got deleted, leaving no remaining references in that row.

The reason columns 5 and 6 got deleted was because of the action that occurred in rows 16 and 19.

In row 16 page 12 got deleted and that moved the 88 number left into column 4.

In row 19 both the numbers 13 and 14 were for pages that got deleted, so these two numbers were deleted from columns 5 and 6.

We previously indicated that after you automatically update a book index grid you may need to manually insert any new entries that are associated with any new pages just inserted. You might also want to add some new entries to existing pages. You always manually add new entries after you automatically update any grid as a result of inserting or deleting or moving pages.

We will finish this help topic by showing two examples of replacement grids where we simply move an existing set of consecutive pages. Moving pages never creates any new pages and never deletes any existing pages. When moving pages, the **Destination Page Number** is directly related to the **Relative Page Number**. Just add or subtract the number of pages the block will move over or move across, from the **Relative Page Number**. For example, subtract 5 when the block moves towards the beginning by 5 pages, or add 5 if the block moves towards the end, by 5 pages. Page moves are tricky to completely understand and are perhaps even more subtle when you have to program them. Taking a little extra time to study the next two examples will be worth your time if you really want to master all the complications associated with page moves.

If we had a 40-page book and decided to move pages 34 through 37 inclusive to occupy page positions 6 pages earlier in the book, the setup dialog should first look like that shown below. We also show the new replacement list. In this example pages 1-18 don't show, but like pages 1-27, none of those pages would change. All page changes occur between pages 28 and 37. Note that the last three pages, 38-40 would also not change. Note that $28 = 34 - 6$.

Number of Original Book Pages:	Number of Pages To Delete/Insert/Move:	The Relative Page Number:
40	4	34
The Change Operation: <input type="radio"/> Insert New Pages Before the Relative Page <input type="radio"/> Insert New Pages After the Relative Page <input type="radio"/> Delete Original Pages Including and Following the Relative Page <input checked="" type="radio"/> Move To Destination Page Starting with and Following the Relative Page		
		Destination Page Number: 28

	Original Data	Replacement Data
19	19	19
20	20	20
21	21	21
22	22	22
23	23	23
24	24	24
25	25	25
26	26	26
27	27	27
28	28	32
29	29	33
30	30	34
31	31	35
32	32	36
33	33	37
34	34	28
35	35	29
36	36	30
37	37	31
38	38	38
39	39	39
40	40	40

The above settings define the set of consecutive pages to be moved. The number of pages to be moved is 4 and the set of pages starts with the **Relative Page Number**, namely page 34. This is how we know pages 34, 35, 36, and 37 are the only pages we are trying to move. Other page numbers automatically move into the positions vacated by the original page numbers when those pages are moved elsewhere.

We can try to make more clear what needs to happen in the previous example by using the table below.

Needed Action:	Original Page #:	New Page #:
No change.	1 ⋮ 27	1 ⋮ 27
Add 4 to the original page numbers because these pages move down by 4.	28 ⋮ 33	32 ⋮ 37
Number the same as the Destination pages. Subtract 6 because these pages move forward by 6.	34 ⋮ 37	28 ⋮ 31
No change.	38 ⋮ 40	38 ⋮ 40

If we had a 40-page book and decided to move 7 pages, namely 12 through 18 inclusive, to occupy page positions 10 pages later in the book, the setup dialog should first look like that shown below. For this example we show the needed actions table before we show the replacement list. Note that $22 = 12 + 10$.

Number of Original Book Pages:	Number of Pages To Delete/Insert/Move:	The Relative Page Number:
<input type="text" value="40"/>	<input type="text" value="7"/>	<input type="text" value="12"/>
The Change Operation: <input type="radio"/> Insert New Pages Before the Relative Page <input type="radio"/> Insert New Pages After the Relative Page <input type="radio"/> Delete Original Pages Including and Following the Relative Page <input checked="" type="radio"/> Move To Destination Page Starting with and Following the Relative Page		Destination Page Number:
		<input type="text" value="22"/>

Needed Action:	Original Page #:	New Page #:
No change.	1 ⋮ 11	1 ⋮ 11
Number the same as the Destination pages. Add 10 because these pages move down by 10.	12 ⋮ 18	22 ⋮ 28
Subtract 7 from the original page numbers because these pages move up by 7.	19 ⋮ 28	12 ⋮ 21
No change.	29 ⋮ 40	29 ⋮ 40

We also show the new replacement list below. In this example pages 1-11 don't change. All page changes occur between pages 12 and 28. Note that the remaining pages, 29-40 would also not change.

	Original Data	Replacement Data
11	11	11
12	12	22
13	13	23
14	14	24
15	15	25
16	16	26
17	17	27
18	18	28
19	19	12
20	20	13
21	21	14
22	22	15
23	23	16
24	24	17
25	25	18
26	26	19
27	27	20
28	28	21
29	29	29
30	30	30
31	31	31
32	32	32

In the previous example you might note that page 22 is 10 pages later than page 12. The number of pages to be moved is 7, and those original pages are precisely pages 12 through 18 inclusive.

We think it is important that you always review the entire replacement list before you proceed. In fact, it is probably a good idea to first make a backup of any existing book index grid before you try to make any book index page replacements. Of course we provide an Undo button, but having backups is always worthwhile whenever something in your setup is wrong, but you don't realize that until later.

When performing a **Move To Destination Page...** operation, if you should accidentally set parameters that are improper, then you will first see a warning message and the entire operation will be aborted. You need to make sure the number of pages to move together with both the relative and destination page numbers do not sum to make a page value outside of the number of current pages in the book.

Probably the trickiest part of entering the parameters for a **Move** operation is setting the **Move To Destination Page**. This page number is the number of the first page in the move block, AFTER the move is made. It is the same as the new page number of the first page in the move block, AFTER the block has been moved. When the block moves towards the end of the document, you have to take into account that other pages will move forward and this may require some careful thinking! When the block moves towards the beginning of the document, the calculation is more direct.

We need to make one more comment about a **Move** operation. The program will make a report on the integrity of the index after the **Move** page replacements have all been made. It is possible that the integrity check will fail due to the page numbers not being properly sorted within one or more rows within the index. If you should see any error message after you perform a **Move**, you should not panic. Instead, you should select the menu item **Utilities | Book/Document Index | Sort the Index Page Numbers...** and select all rows. Then you should get a normal confirmation message that the grid is now properly sorted.

If your grid still has errors, perhaps other than sorting errors, then manually correct the index by performing an integrity check that will more specifically tell you the position and reason for each error that is found. Keep performing integrity checks and making manual corrections until there are no more remaining errors.

Working With Book/Document Indexes

Sometimes a CSV file contains a grid that is used as an index for a book. The grid in this case has a very simple format in which the first column contains the text of the index items or terms and the remaining columns contain the book page numbers for the items. The grid can have any number of columns (at least 2 are required), but needs no more than the largest number of page references for any one item. Having a header row for the grid is optional.

The index terms in Column 1 should be sorted alphabetically with a case insensitive sort. Each index term, when upper-cased, should normally begin with a first letter in the range from A-Z. For each row, when an index term has only one or just a few page references, then all remaining columns to the right of the last used page reference must be empty. A typical book index grid should look like the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Index Term:	Occurrence 1:	Occurrence 2:	Occurrence 3:	Occurrence 4:
1	Academy Awards grid	3	14	23	33
2	accuracy factor	174			
3	address text file	20	21		
4	Adobe Reader	1			
5	ALabel program	1	25		
6	base path option	261			
7	basic limitations	10			
8	binary	176			
9	blank cells	133	153	165	
10	blank column removal	113			
11	Boolean data	7	81	218	
12	bottom	164			
13	Celsius	89	194		
14	clearing a column	115			
15	clearing a row	155			
16	column character extents	259			
17	column display widths	257			
18	column report	30			
19	data types	7			
20	date data	7	180	220	
21	date formats	180	221		

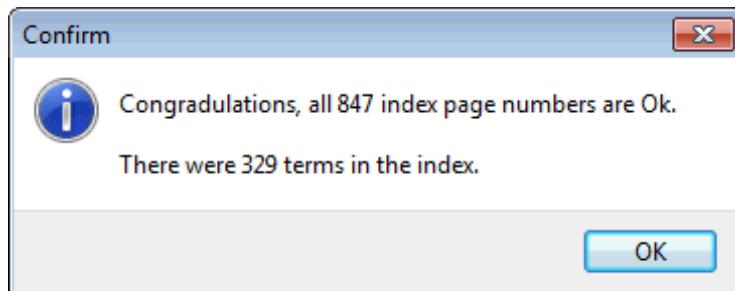
In the above grid, the cells that look blank are in fact empty. This is the main requirement of a book index grid. We only show 5 columns in the above grid, but a book index grid can have any number of columns. Except for the first column, the remaining cells must contain page numbers or they must be blank. (positive integers only; an option is to attach a single dash character - immediately after some integers)

The first row in this example grid has four page number references in Column 2 through Column 5 inclusive. The second row has only one page reference, so columns 3, 4, and 5 in that row must be empty. The third row has two page references followed by two empty cells.

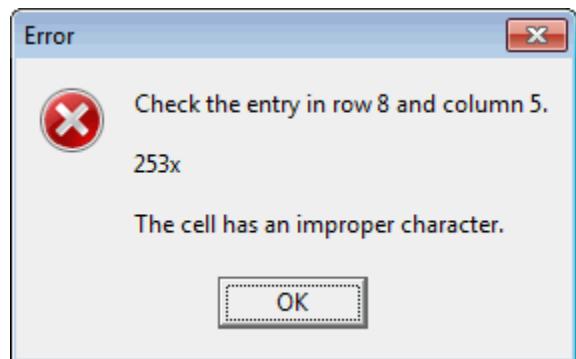
When you execute the function **Utilities | Book/Document Index | Check the Integrity of the Page Numbers...**, the program assumes you have already opened a book or document index CSV file, similar to the one shown on the previous page.

This particular function will verify that all your page numbers are in a proper format. On a row by row basis, the program will check that the index page numbers are proper and that they are properly sorted within each row. The response to making this function is usually only a simple message.

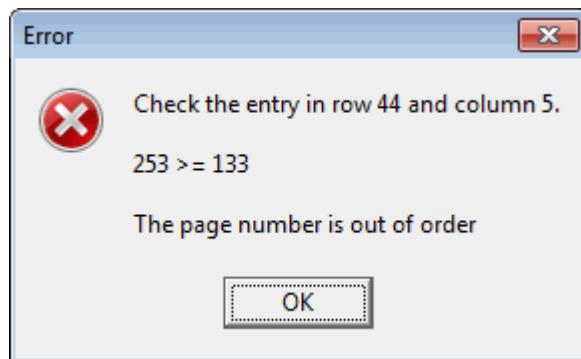
If everything checks out Ok you will see a confirmation message like the following:



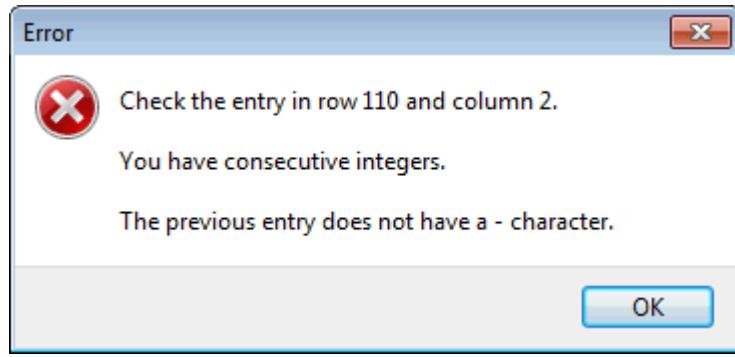
If any cell contains either an improper character or if a page number is out of order in a row, or if two page numbers are consecutive integers and the first one is not followed by a dash character, then you will see one of the following error messages.



or



or

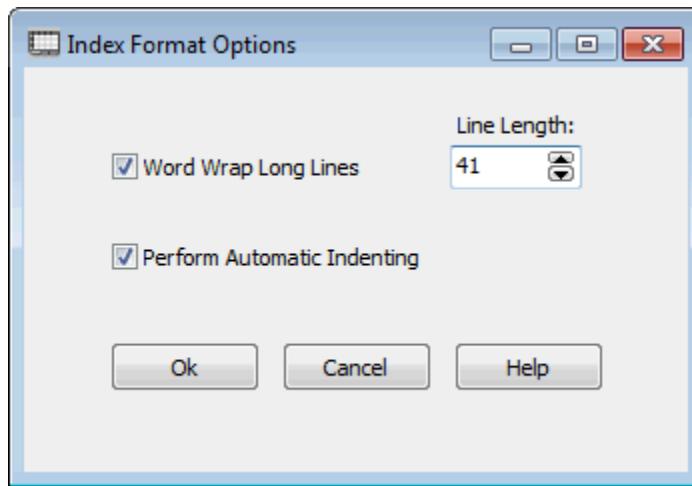


The program stops checking at the first error it finds, so you need to fix each error and then repeat the integrity check until no more errors are found.

Running an integrity check at least insures your index is properly formatted. The program always performs an integrity check before making any kind of a report. If errors are found, you will need to keep fixing the errors one at a time until no more errors are found.

Index Formatting Options

After an index has passed an integrity check, you may want to create a normal book/index text file. To do this select **Utilities | Book/Document Index | Create a Normal Index Text File...**. You will first see the following dialog.



You can perform word wrapping of long line where you can specify the line length. We usually set this option when our index will consist of two columns on a page.

A different option is to have the program try to **Perform Automatic Indenting**. This option aids when you try to make indented subheadings. All this means is that when the first word in a following line is the same as the first word in a previous line, then the following line will be indented, and will be written without that first word. As a simple example of indenting, the topics under the word **string** below appear indented two spaces.

```
street names 46,252,323
string
    data 9,46,146-149,348
    expressions 103-113,397
    length function 106,109-110
    reverse function 106
    substring function 105-113
    token function 111-113
    subdirectories 57,69
```

We should acknowledge that our automatic indenting function supports only one level of indenting. If you need more than one indenting level, then we suggest you include full headings that you later manually edit in your word processor. As an example, you can see the subheadings and indentings we manually made for the index at the end of this document. Doing this manually gives you exact control over how your index looks.

After you click the **Ok** button in the above dialog, you will be prompted to save a text file that you should name. After the program saves the text file, it should automatically open that text file so you can see it. A normal book/document index text file, without automatic indenting, looks like the following.

```
A
Academy Awards grid 3,14,23,33
accuracy factor 174
address text file 20,21
Adobe Reader 1
ALabel program 1,25

B
base path option 261
basic limitations 10
binary 176
blank cells 133,153,165
blank column removal 113
Boolean data 7,81,218
bottom 164

C
Celsius 89,194
clearing a column 115
clearing a row 155
column character extents 259
column display widths 257
column report 30

D
data types 7
date data 7,180,220
date formats 180,221
```

Of course this example output is deliberately short. A real book index text file might have hundreds of index entries. Note how the program has automatically inserted single alphabet letters **A B C D** and has separated the letter groups by one blank line in the text file.

You should expect that the text file is intended to be copied into a text editor for further editing and use. This function saves you from having to enter the letters and having to group the items, and it saves you from typing in all the commas that separate the page numbers.

When this function executes, its last step should be to automatically open the text file that was created so you can inspect the results. If you look at the very last few pages of this help file, you will see an index that was created automatically using both word-wrap and automatic indenting. We re-formatted the text in our word processor to make two columns of information on each index page. It is normal to have to do some manual editing of the normal index text that this program automatically creates. Our automatic indenting and formatting may not always be exactly what you need or want, but it should be sufficiently close that only minor manual tweaking remains.

A special format option for a book index grid is to indicate a range of pages like **220-227**. In this case one column would contain the item **220-** and the next column would contain **227**. The dash character **-** can only be used if the following cell holds a connecting page number.

Instead of separating these items with a comma, whenever the program sees a page number immediately followed by a dash character then no comma is output, and the dash character will connect the two entries as if they were one entry. Except for the special dash character, each page entry should consist of a single page number positive integer and nothing else.

The above example showed how to make a normal book/document index. A different kind of report can be made by selecting the menu **Utilities | Book/Document Index | Write a De-Interlaced Text File...**. This function will first make an integrity check behind the scenes, and if that is Ok, it will prompt you to save the output in a text file that you name. As an example of what de-interlacing the page numbers means, consider the text lines shown below.

This non-realistic de-interlaced report is from a different grid than the previous example. However, these few lines should be enough for you to understand the meaning of de-interlacing.

```
1 Adobe Reader; ALabel program
3 Academy Awards grid
7 Boolean data; data types; date data
10 basic limitations
14 Academy Awards grid
17 GeoCoding; Google Maps; GPS coordinates; latitude; longitude
21 address text file
23 ALabel data file; field names; keyname
25 ALabel program
30 column report
33 Academy Awards grid
57 date sequence
79 decimal digits
81 Boolean data
84 month range; starting seed; year range
113 blank column removal
115 clearing a column
155 clearing a row
164 bottom
165 blank cells
174 accuracy factor; decimal digits
```

The above de-interlaced report lists each used page number only once, and for that page number it shows the index text entries on that page, separated by semi-colon characters. In the example above, only pages **1**, **7**, **17**, **23**, **84**, and **174** contain multiple entries in their rows.

This example de-interlacing report does not list page numbers that did not appear in the original book index grid, and this explains why not all page numbers are present.

The report is such that the page numbers are sorted in ascending order. Also, going across in a row, the text entries are also sorted within each row, wherever you see multiple entries in a row separated by semi-colon characters.

You should now have a better idea of the meaning and usefulness of a de-interlaced book index report. Such a report allows you to inspect/analyze any book index on a page by page basis. It also forms the basis for how you might originally create a book index.

In the case where two adjacent page number entries appear like **220-** and **227** in a book/document index grid row, and you have selected **Utilities | Book/Document Index | Write a De-Interlaced Text File...**, then the dash character will be noted and the report will contain output lines for both page **220** and page **227**, and also all intervening pages between **220** and **227** will appear in the de-interlace report.

When you read a de-interlaced text file, the program will automatically determine when a range of consecutive pages like **220-227** appear and the program will insert the dash character for you and it will determine the largest range of consecutive pages, across which a given item appears. This is a nice custom feature to have.

One of the reasons reading and writing a de-interlaced file are not always exact inverses of one another is because it is possible for one index text entry to be a substring of another index text entry. When that is the case the entry that is the substring may not appear as a separate line in the grid when the de-interlaced file is read. A special attribute of any index is where no text entry is a substring of another text entry. This can be tested for by selecting the command **Uniqueness | Report Substring Occurrences In a Column...** where the **Source Column** is the first column in an index grid. Be careful to create and then later delete a **Report Column** when you are finished with this function.

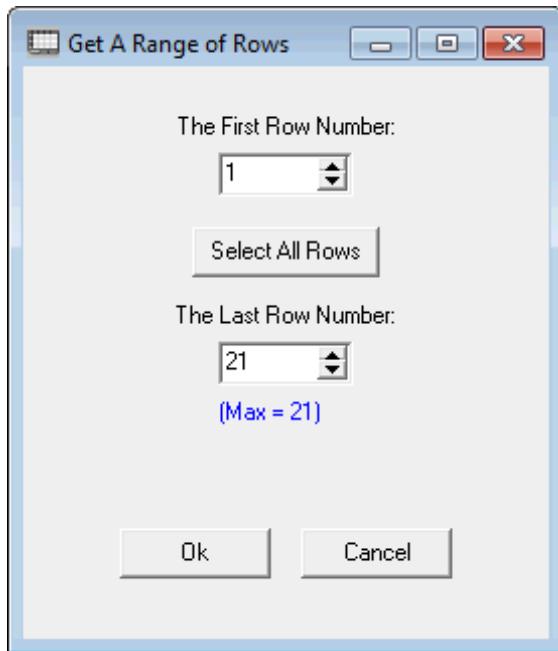
We also assume you are making indexes for books or documents that tend to be more technical in nature as opposed to books that deal with history/literature/philosophy where there may be a need for more concepts to be included in the index.

Concepts can span multiple pages and normally require human analysis of the document. Although professional indexers might say that no writer should ever make their own indexes, we disagree. We think you can use our program to make your own index. Any index is better than no index.

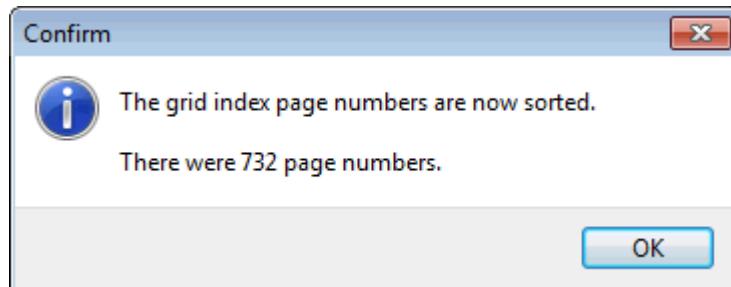
While our indexing functions are very basic, they are useful if your word processor does not provide any indexing capabilities. Our indexing functions are effective for documents that seldom have complete page insertions and deletions after the initial index is complete. Documents can be edited with many small changes and many small changes can be made to the index without requiring a full renumbering of most of the pages.

Sorting Book Index Page Numbers

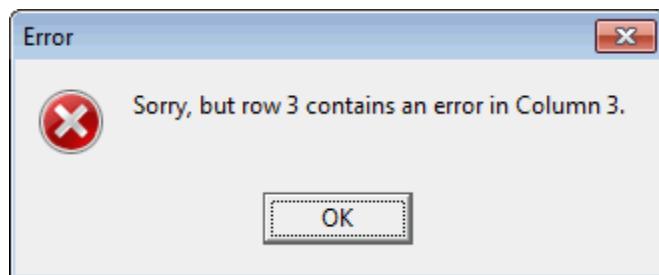
For those times when you have already opened a book index grid, and you would like to insure the page numbers for that book index are properly sorted, you can select the menu item **Utilities | Book/Document Index | Sort the Index Page Numbers...** and you will first see a standard dialog for selecting a range of rows from the current grid.



Usually you will select all the rows, but you have the option of sorting any particular range of rows. When you click the **Ok** button you should normally see a confirmation message like:



However, you may see an error message if any cells in the book index grid don't contain proper page number entries.



The following is an example book index grid whose page numbers are not sorted. In the format shown below, this grid would not be considered as a valid book index grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Index Term:	Occurrence 1:	Occurrence 2:	Occurrence 3:	Occurrence 4:	Occurrence 5:
1	Academy Awards grid			49		
2	accuracy factor		113	8	159	201
3	address text file		238	30		
4	Adobe Reader	288	35		206	171
5	ALabel program		140	252	104	32
6	base path option	166	93		216	142
7	basic limitations	63	15		230	169
8	binary				85	116
9	blank cells		74	157	20	135
10	blank column removal	253	13	236	14	
11	Boolean data	31	17			
12	bottom	225		153	192	265
13	Celsius	40	22	16	140	238
14	clearing a column			19	90	
15	clearing a row	38	220	8	273	152
16	column character extents	177	82		57	
17	column display widths	105	25			2
18	column report	236		38		
19	data types	11	138		146	18
20	date data	3	242		281	201
21	date formats		221	41		
22	date sequence	53	21		118	103
23	decimal	10	47	74		243
24	decimal digits	146		10	178	121

When you perform a sort operation, the program will first scan all the page number entries and it will remove any blank characters from those cells. Next, for each selected row the program will move all empty cells to the right-most positions within the row. Next, the program will check the remaining non-empty cells to insure they contain only valid page numbers.

Note that in some cases you may have a cell with a page number entry like **152-** where the page number reference ends with a dash character. The program will temporarily ignore any trailing dash characters at the end of a page number while it sorts the page numbers within the row.

For the above example grid, we next show the result of sorting the page numbers for all 24 rows in the grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Index Term:	Occurrence 1:	Occurrence 2:	Occurrence 3:	Occurrence 4:	Occurrence 5:
1	Academy Awards grid	49				
2	accuracy factor	8	113	159	201	
3	address text file	30	238			
4	Adobe Reader	35	171	206	288	
5	ALabel program	32	104	140	252	
6	base path option	93	142	166	216	
7	basic limitations	15	63	169	230	
8	binary	85	116			
9	blank cells	20	74	135	157	
10	blank column removal	13	14	236	253	
11	Boolean data	17	31			
12	bottom	153	192	225	265	
13	Celsius	16	22	40	140	238
14	clearing a column	19	90			
15	clearing a row	8	38	152	220	273
16	column character extents	57	82	177		
17	column display widths	2	25	105		
18	column report	38	236			
19	data types	11	18	138	146	
20	date data	3	201	242	281	
21	date formats	41	221			
22	date sequence	21	53	103	118	
23	decimal	10	47	74	243	
24	decimal digits	10	121	146	178	

Although this example did not contain any cells with dash characters, the result of the sort would be considered a valid book index grid.

By comparing the grid on this page with the grid on the previous page, you can see the before and after effects of sorting the page numbers. The page numbers increase as you read from left-to-right across in each row and all empty cells appear after the last used page number in each row.

Comparing Two CSV Files

Another function under the **Utilities** menu lets you compare the currently loaded **CSV** file with another **CSV** file. When you select the menu item, **Utilities | Compare With Another CSV File...**, the program will prompt you to select the other **CSV** file. The program will then make the comparison and report the results by creating a text file named **ComparisonReport.txt** that is stored in the base directory of the **CSV Editor** program.

That report file will automatically be opened for reading. You will then see on a line by line basis, each field difference between the two **CSV** files. You will also see some miscellaneous information about the compared **CSV** file. The currently loaded **CSV** file will remain loaded and unchanged by executing this **CSV** file comparison function.

An example comparison report is shown in blue text below. In this example report we made a test file that had one more row than the loaded **CSV** file and it had a particular row with fewer columns than in its other rows, and we changed several random items. This example report should give you an idea of the kind of information that gets reported when a **CSV** file comparison is made. To avoid having unusually large report files, we stop reporting differences after approximately 5,000 report lines have been output. In such cases, the report will not show all differences, but we assume after 5,000 differences you won't really care about any remaining ones.

In general, we recommend you only try to compare two **CSV** files that have the same number of rows and where most of those rows match. Otherwise, the report can be very long because every field or cell item will likely be mismatched. In cases where the compared file has more columns than the loaded file, only column mismatches within the column space of the loaded **CSV** file will be reported. This means extra columns in the compared file are essentially ignored.

If the compared file has fewer columns than the loaded file, then the extra columns in the loaded file will not be reported as mismatches. If the compared file has more rows than the loaded file, then those extra rows will not report any mismatches as they are also essentially ignored, but the report will state that there were extra rows found. This means all mismatches that are reported lie within the region common to both files. Think of this region as the intersection of the two **CSV** files in terms of row and column counts. If the two files are identical in terms of row and column sizes and all fields or cells match, then the report will note that fact. However, two such files can still differ in terms of internal **CSV** field formatting, so they may still not be byte for byte identical.

The following is an example of a very short **CSV** file comparison report:

```
The compared file is: D:\JKProgs\CSVEditor\Test.csv
Row 10 column 3 difference: @testing@Katherine Hepburn@
Row 11 column 1 difference: @199@1969@
Row 11 column 2 difference: @John Payne@John Wayne@
Row 11 column 3 difference: @Maggie Mith@Maggie Smith@
Row 11 column 4 difference: @Jack Schlesinger@John Schlesinger@
Row 11 column 5 difference: @Midwife Cowboy@Midnight Cowboy@
There were 23 rows read from the compared file.
The compared file had more rows (23) than the loaded file rows (22).
The compared file had more columns (6) than the loaded file columns (5).
The compared file had 5 minimum columns in row 1.
The compared file had 6 maximum columns in row 23.
```

Copying Matching Data

The operation to copy matching data is normally performed after you complete a set operation like finding the intersection or difference of two sets of column cells. This is one of the more intricate operations you can perform with the **CSV Editor** program, so it requires a little explanation before you may grasp its real significance. However, this operation can save you countless hours of manually searching and replacing data, so when you need to apply it, you should find it to be a real time saver.

Essentially this operation is used to fill in other columns with data where the columns were originally tied to the set data, but the columns were columns other than those used to compute a set intersection or a set difference. We will give one practical example. The list below contains mostly technical books.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	ISBN-13:	Book Title:	Book Author:	Publisher	Date Received:
1	978-0-596-80552-4	JavaScript The Definitive Guide	Flanagan, David	O'Reilly & Associates	7/6/2011
2	978-1-59327-286-9	The Book of CSS3	Gastón, Peter	No Starch Press	9/10/2011
3	978-0-691-15265-3	A Mathematical Nature Walk	Adam, John A.	Princeton University Press	9/28/2011
4	978-0-262-51668-6	Internet Alley HT in Tyson's Corner 1945-2005	Ceruzzi, Paul	MIT Press	10/3/2011
5	978-1-4129-1314-0	The Tao of Statistics	Keller, Dana K.	Sage Publications	10/25/2011
6	978-0-321-78442-1	Introducing HTML 5 2nd Edition	Lawson, Bruce; Sharp, Remy	New Riders Publishing	10/25/2011
7	978-0-8174-3939-2	Understanding Exposure	Peterson, Bryan	Crown Publishers	1/13/2012
8	978-0-13-322495-5	Floating Point Computation	Sterbenz, Pat	Prentice Hall	1/14/2012
9	978-0-8176-4372-0	Elementary Functions Algorithms & Imp.	Muller, Jean-Michel	Birkhäuser	1/24/2012
10	978-0-8176-4704-9	Handbook of Floating Point Arithmetic	Muller, Jean-Michel	Birkhäuser	1/24/2012
11	978-0-19-923400-4	Oxford Dictionary of Computing 6th Edition	Oxford University Press	Oxford University Press	1/27/2012
12	978-1-449-30468-3	The Information Diet	Johnson, Clay	O'Reilly & Associates	1/29/2012
13	978-0-691-14039-1	Matrix Mathematics	Bernstein, Dennis S.	Princeton University Press	2/15/2012
14	978-0-19-374044-4	Algorithmic Puzzles	Levitin, Anat; Levitin, Maria	Oxford University Press	2/15/2012
15	978-0-691-14714-7	9 Algorithms That Changed The Future	MacCormick, John	Princeton University Press	2/15/2012
16	978-0-486-47883-8	An Intro to Functional Programming through LC	Michaelson, Greg	Dover Publications	2/23/2012
17	978-0-521-51644-0	A Computational Introduction to Number Theory	Shoup, Victor	Cambridge University Press	2/25/2012
18	978-0-465-01775-1	Professor Stewart's Hoard of Math Treasures	Stewart, Ian	Basic Books	2/27/2012
19	978-0-486-47417-5	A Book of Abstract Algebra, 2nd Edition	Pinter, Charles C.	Dover Publications	3/2/2012
20	978-0-471-11709-4	Applied Cryptography	Schneier, Bruce	John Wiley & Sons	4/12/2012
21	978-0-8218-4418-2	Finite Fields and Applications	Mullen, Gary L.	American Mathematical Society	4/14/2012
22	978-0-691-15270-7	In Pursuit of the Traveling Salesman	Cook, William J.	Princeton University Press	7/6/2012
23	978-0-691-14342-2	The Irrationals	Havil, Julian	Princeton University Press	7/17/2012
24	978-0-321-62930-2	What is a p-value is it anyway?	Vickers, Andrew	Addison-Wesley	8/21/2012
25	978-0-674-05755-5	Measurement	Lockhart, Paul	Belknap Press	9/13/2012
26	978-1-118-46446-5	Raspberry Pi User Guide	Upton, Eben; Hockenberry, Gareth	Wiley Computer Publishing	9/25/2012
27	978-0-307-72095-5	Makers The New Industrial Revolution	Anderson, Chris	Crown Publishers	10/4/2012
28	978-1-59420-411-1	The Signal and the Noise	Silver, Nate	The Penguin Press	10/4/2012
29	978-1-59184-492-1	Automate This	Steiner, Christopher	Portfolio/Penguin	10/4/2012
30	978-0-984-72511-3	Race Against The Machine	Brynjolfsson, Erik; McAfee, Andrew	Digital Frontier Press	10/5/2012
31	978-1-118-20413-9	Windows 8 Secrets	Thurrott, Paul; Rivera, Rafael	Wiley Computer Publishing	10/11/2012
32	978-0-321-88691-0	Canon EOS Rebel T4i/650D	Revell, Jeff	Peach Pit Press	10/15/2012
33	978-1-285-42457-6	Canon EOS Rebel T4i/650D	Busch, David	Cengage Learning	10/23/2012
34	978-0-07-180783-8	Programming the Raspberry Pi	Monk, Simon	McGraw-Hill	11/19/2012
35	978-0-470-40765-3	Be Expert With Map & Compass	Kjellstrom, Björn; Elgin, Carina	John Wiley & Sons	11/23/2012
36	978-1-56881-721-7	Unexpected Expectations	Wapner, Leonard	CRC Press	12/8/2012
37	978-0-691-14892-2	Heavenly Mathematics	Van Brummelen, Glen	Princeton University Press	12/27/2012
38	978-0-7484-0304-2	Map Projections A Reference Manual	Snyder, John; Bugayevskiy, Lev	CRC Press	12/28/2012
39	978-0-691-15271-4	Henri Poincaré A Scientific Biography	Gray, Jeremy	Princeton University Press	1/15/2013
40	978-0-321-81958-1	Lightroom 4	Kelby, Scott	New Riders Publishing	1/15/2013
41	978-0-97109011-8	UTM Using Your GPS With UTM	Carnes, John	MapTools	1/22/2013
42	978-0-8218-3255-4	Portraits of the Earth	Freeman, Timothy G.	AMS Chelsea Publishing Series	1/22/2013
43	978-0-13-612824-3	Goode's World Atlas	Veregin, Howard	Prentice Hall	2/1/2013
44	978-0-9560030-7-2	Adobe Photoshop Lightroom 4 The Missing FAQ	Bampton, Victoria	Lightroom Queen Publishing	2/13/2013
45	978-0-06-206024-2	The Inventor's Dilemma	Christensen, Clayton	Harper & Row Publishers	3/6/2013

Now imagine that you have another shorter list of ISBN numbers that is something like:

<	Column 1
>	ISBN-13:
1	978-1-285-42457-6
2	978-0-691-15265-3
3	978-1-59327-286-9
4	978-0-321-78442-1
5	978-0-06-206024-2
6	978-0-262-51668-6
7	978-1-449-30468-3
8	978-0-9710901-1-8
9	978-0-691-14039-1
10	978-0-691-14892-2
11	978-0-321-88691-0
12	978-0-8174-3939-2
13	978-0-8176-4372-0
14	978-0-13-612824-3
15	978-0-07-180783-8
16	978-0-691-14342-2
17	978-0-465-01775-1
18	978-0-521-51644-0
19	978-1-118-20413-9
20	978-0-307-72095-5
21	978-0-8218-3255-4
22	978-1-59420-411-1
23	978-0-486-47883-8
24	978-1-59184-492-1

Given these two grids we would like to create a list of the books that are in the first grid, but not in the second grid.

We should note that the ISBN numbers in the two lists are in very different orders or arrangements. Neither arrangement is necessarily sorted, even though ISBN numbers constitute key data that uniquely identifies a given book. Also note that the two columns of ISBN numbers have very different lengths.

The easiest way to solve the set difference problem is to first open the large grid and append the short list as a 6th additional column with that first list. You could choose **File | Import/Append Additional Columns From Another CSV File....**

Then we append a blank 7th column that is used to compute the answer to the set difference of Column 1 minus Column 6. When that set difference is computed we are left with a grid like the one shown on the following page.

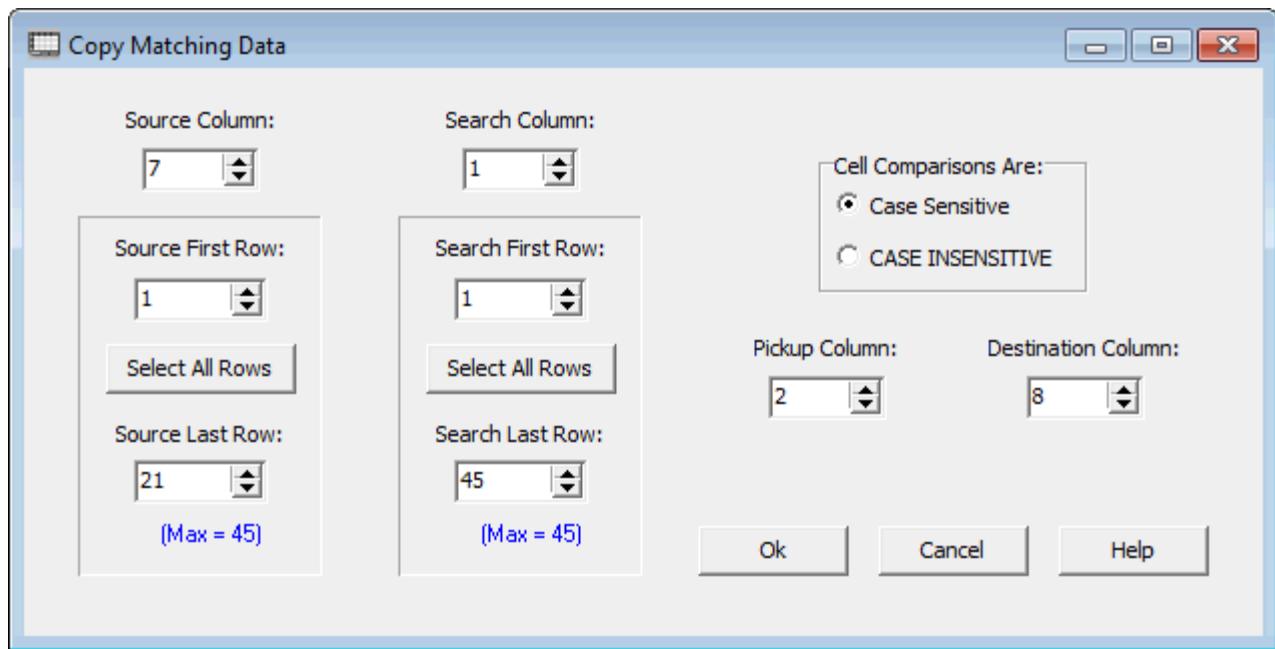
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
>	ISBN-13:	Book Title:	Book Author:	Publisher	Date Received:	ISBN-13:	
1	978-0-596-80552-4	JavaScript The Definitive Guide	Flanagan, David	O'Reilly & Associates	7/6/2011	978-1-285-42457-6	978-0-596-80552-4
2	978-1-59327-286-9	The Book of CSS3	Gastón, Peter	No Starch Press	9/10/2011	978-0-691-15285-3	978-1-4129-1314-0
3	978-0-691-15265-3	A Mathematical Nature Walk	Adam, John A.	Princeton University Press	9/28/2011	978-1-59327-286-9	978-0-13-322495-5
4	978-0-262-51668-6	Internet Alley HT in Tyson's Corner 1945-2005	Ceruzzi, Paul	MIT Press	10/3/2011	978-0-321-78442-1	978-0-8176-4704-9
5	978-1-4129-1314-0	The Tao of Statistics	Keller, Dana K.	Sage Publications	10/25/2011	978-0-06-206024-2	978-0-19-923400-4
6	978-0-321-78442-1	Introducing HTML 5 2nd Edition	Lawson, Bruce; Sharp, Remy	New Riders Publishing	10/25/2011	978-0-262-51668-6	978-0-19-974044-4
7	978-0-8174-3939-2	Understanding Exposure	Peterson, Bryan	Crown Publishers	1/13/2012	978-1-449-30468-3	978-0-691-14714-7
8	978-0-13-322495-5	Floating Point Computation	Sterbenz, Pat	Prentice Hall	1/14/2012	978-0-9710901-1-8	978-0-486-47417-5
9	978-0-8176-4372-0	Elementary Functions Algorithms & Imp.	Muller, Jean-Michel	Birkhäuser	1/24/2012	978-0-691-14039-1	978-0-471-11709-4
10	978-0-8176-4704-9	Handbook of Floating Point Arithmetic	Muller, Jean-Michel	Birkhäuser	1/24/2012	978-0-691-14892-2	978-0-8218-4418-2
11	978-0-19-923400-4	Oxford Dictionary of Computing 6th Edition	Oxford University Press	Oxford University Press	1/27/2012	978-0-321-88691-0	978-0-691-15270-7
12	978-1-449-30468-3	The Information Diet	Johnson, Clay	O'Reilly & Associates	1/29/2012	978-0-8174-3939-2	978-0-321-62930-2
13	978-0-691-14039-1	Matrix Mathematics	Bernstein, Dennis S.	Princeton University Press	2/15/2012	978-0-8176-4372-0	978-0-674-05755-5
14	978-0-19-974044-4	Algorithmic Puzzles	Levitin, Anay; Levitin, Maria	Oxford University Press	2/15/2012	978-0-13-612824-3	978-1-118-46446-5
15	978-0-691-14714-7	9 Algorithms That Changed The Future	MacCormick, John	Princeton University Press	2/15/2012	978-0-07-180783-8	978-0-984-72511-3
16	978-0-486-47883-8	An Intro to Functional Programming through LC	Michaelson, Greg	Dover Publications	2/23/2012	978-0-691-14342-2	978-0-470-40765-3
17	978-0-521-51644-0	A Computational Introduction to Number Theory	Shoup, Victor	Cambridge University Press	2/25/2012	978-0-465-01775-1	978-1-56881-721-7
18	978-0-465-01775-1	Professor Stewart's Hoard of Math Treasures	Stewart, Ian	Basic Books	2/27/2012	978-0-521-51644-0	978-0-7484-0304-2
19	978-0-486-47417-5	A Book of Abstract Algebra, 2nd Edition	Pinter, Charles C.	Dover Publications	3/2/2012	978-1-118-20413-9	978-0-691-15271-4
20	978-0-471-11709-4	Applied Cryptography	Schneier, Bruce	John Wiley & Sons	4/12/2012	978-0-307-72095-5	978-0-321-81958-1
21	978-0-8218-4418-2	Finite Fields and Applications	Mullen, Gary L.	American Mathematical Society	4/14/2012	978-0-8218-3255-4	978-0-9560030-7-2
22	978-0-691-15270-7	In Pursuit of the Traveling Salesman	Cook, William J.	Princeton University Press	7/6/2012	978-1-59420-411-1	
23	978-0-691-14342-2	The Irrationals	Havil, Julian	Princeton University Press	7/17/2012	978-0-486-47883-8	
24	978-0-321-62930-2	What is a p-value is it anyway?	Vickers, Andrew	Addison-Wesley	8/21/2012	978-1-59184-492-1	
25	978-0-674-05755-5	Measurement	Lockhart, Paul	Belknap Press	9/13/2012		
26	978-1-118-46446-5	Raspberry Pi User Guide	Upton, Eben; Halcrow, Gareth	Wiley Computer Publishing	9/25/2012		
27	978-0-307-72095-5	Makers The New Industrial Revolution	Anderson, Chris	Crown Publishers	10/4/2012		
28	978-1-59420-411-1	The Signal and the Noise	Silver, Nate	The Penguin Press	10/4/2012		
29	978-1-59184-492-1	Automate This	Steiner, Christopher	Portfolio/Penguin	10/4/2012		
30	978-0-984-72511-3	Race Against The Machine	Brynjolfsson, Erik; McAfee, Andrew	Digital Frontier Press	10/5/2012		
31	978-1-118-20413-9	Windows 8 Secrets	Thurrott, Paul; Rivera, Rafael	Wiley Computer Publishing	10/11/2012		
32	978-0-321-88691-0	Canon EOS Rebel T4i/650D	Revell, Jeff	Peach Pit Press	10/15/2012		
33	978-1-285-42457-6	Canon EOS Rebel T4i/650D	Busch, David	Cengage Learning	10/23/2012		
34	978-0-07-180783-8	Programming the Raspberry Pi	Monk, Simon	McGraw-Hill	11/19/2012		
35	978-0-470-40765-3	Be Expert With Map & Compass	Kjellstrom, Björn; Elgin, Carina	John Wiley & Sons	11/23/2012		
36	978-1-56881-721-7	Unexpected Expectations	Wapner, Leonard	CRC Press	12/8/2012		
37	978-0-691-14892-2	Heavenly Mathematics	Van Brummelen, Glen	Princeton University Press	12/27/2012		
38	978-0-7484-0304-2	Map Projections A Reference Manual	Snyder, John; Bugayevskiy, Lev	CRC Press	12/28/2012		
39	978-0-691-15271-4	Henri Poincaré A Scientific Biography	Gray, Jeremy	Princeton University Press	1/15/2013		
40	978-0-321-81958-1	Lightroom 4	Kelby, Scott	New Riders Publishing	1/15/2013		
41	978-0-9710901-1-8	UTM Using Your GPS With UTM	Carnes, John	MapTools	1/22/2013		
42	978-0-8218-3255-4	Portraits of the Earth	Freeman, Timothy G.	AMS Chelsea Publishing Series	1/22/2013		
43	978-0-13-612824-3	Goode's World Atlas	Veregin, Howard	Prentice Hall	2/1/2013		
44	978-0-9560030-7-2	Adobe Photoshop Lightroom 4 The Missin FAQ	Bampton, Victoria	Lightroom Queen Publishing	2/13/2013		
45	978-0-06-206024-2	The Inventor's Dilemma	Christensen, Clayton	Harper & Row Publishers	3/6/2013		

Now at this point Column 7 contains a list of the ISBN numbers that identify the books in Column 1 that are not in Column 6. So far so good. All we have done is compute a key column containing the key part of the answer.

However, we don't just want the list of ISBN numbers, let's assume we also want to copy the book title and author and publisher information, but on a corresponding row by row basis next to Column 7. This is where we could use a function to copy matching data, say in three additional columns that we will add to the above grid.

We will perform a look-up and copy operation three times, once each for getting the book title, and then the author information, and finally the publisher information.

So we first add a blank 8th column to the above grid and then we execute the function **Utilities | Copy Matching Data...** and we fill out the dialog as shown on the next page.



The **Source Column** is Column 7 because this column contains the key information and it is the column that contains the answer to the set operation. Note the **Source Column** consists of a block of 21 consecutive rows.

The **Search Column** is Column 1 because it is the key column for the block that contains the pickup data. We search down the **Search Column** until we find the matching ISBN number from the **Source Column**. Note the **Search Column** has 45 consecutive rows, different from the 21 rows of the **Source Column**.

The **Pickup Column** is Column 2 because the book title will come from Column 2. By definition the **Pickup Column** contains the data that will be copied to the **Destination Column**.

The **Destination Column** is Column 8 because we will be depositing the book title we find in Column 2 into Column 8. The **Destination Column** is the current selection in the block of answer cells that we are creating.

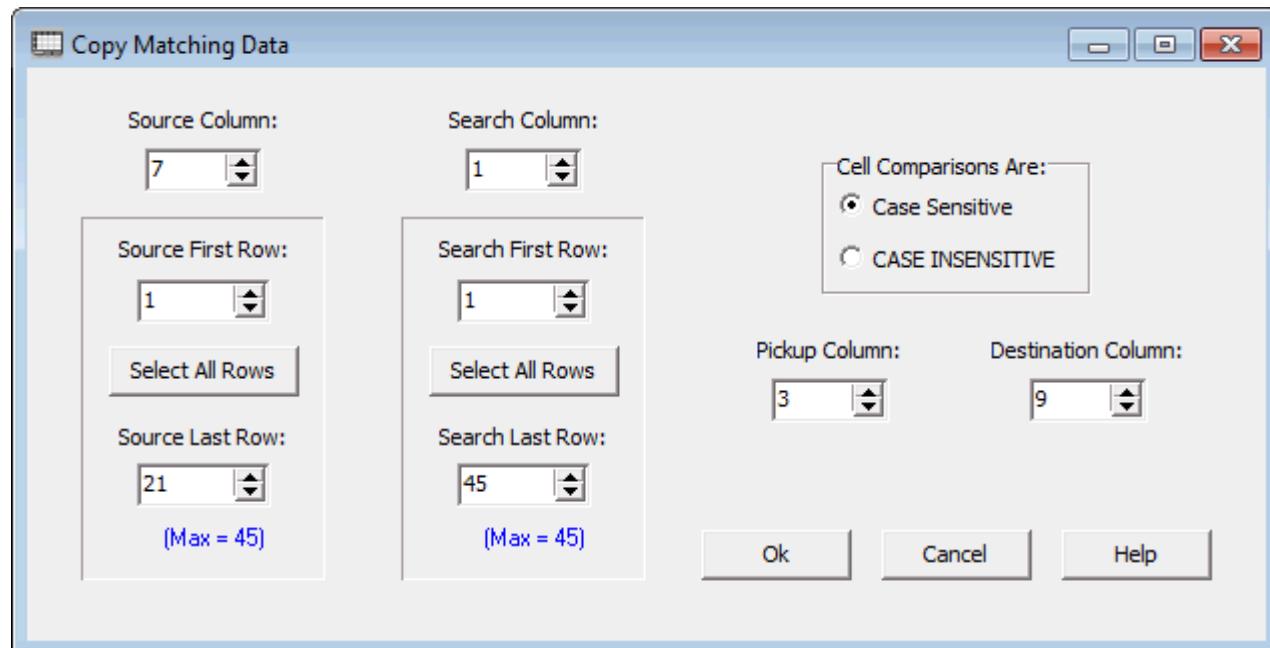
Now the main point to understand is why all this setup information is necessary. It is because the 45 rows in the main block comprising Column 1 through Column 5 inclusive, don't correspond to the block of 21 rows in Column 7. The two sets of rows have entirely different lengths.

Another way of describing what we need to do is the following. We are going to march down the list of 21 ISBN numbers in Column 7 on a row by row basis. We must temporarily remember the row the ISBN comes from in Column 7. For each source ISBN number, we are going to search in Column 1 to locate the row in Column 2 that contains the matching book title for the ISBN. This means we have two row numbers to keep track of, one from Column 7 and one from Column 1. Then we get the book title from Column 2, and we deposit that book title in the remembered row from the ISBN in Column 7, but we place the book title in the corresponding row in the **Destination Column**, Column 8.

So after we execute this first copy operation the block of cells in Column 7 and Column 8 should contain corresponding book information that appears as shown on the next page.

Column 7	Column 8
978-0-596-80552-4	JavaScript The Definitive Guide
978-1-4129-1314-0	The Tao of Statistics
978-0-13-322495-5	Floating Point Computation
978-0-8176-4704-9	Handbook of Floating Point Arithmetic
978-0-19-923400-4	Oxford Dictionary of Computing 6th Edition
978-0-19-974044-4	Algorithmic Puzzles
978-0-691-14714-7	9 Algorithms That Changed The Future
978-0-486-47417-5	A Book of Abstract Algebra, 2nd Edition
978-0-471-11709-4	Applied Cryptography
978-0-8218-4418-2	Finite Fields and Applications
978-0-691-15270-7	In Pursuit of the Traveling Salesman
978-0-321-62930-2	What is a p-value is it anyway?
978-0-674-05755-5	Measurement
978-1-118-46446-5	Raspberry Pi User Guide
978-0-984-72511-3	Race Against The Machine
978-0-470-40765-3	Be Expert With Map & Compass
978-1-56881-721-7	Unexpected Expectations
978-0-7484-0304-2	Map Projections A Reference Manual
978-0-691-15271-4	Henri Poincare A Scientific Biography
978-0-321-81958-1	Lightroom 4
978-0-9560030-7-2	Adobe Photoshop Lightroom 4 The Missing FAQ

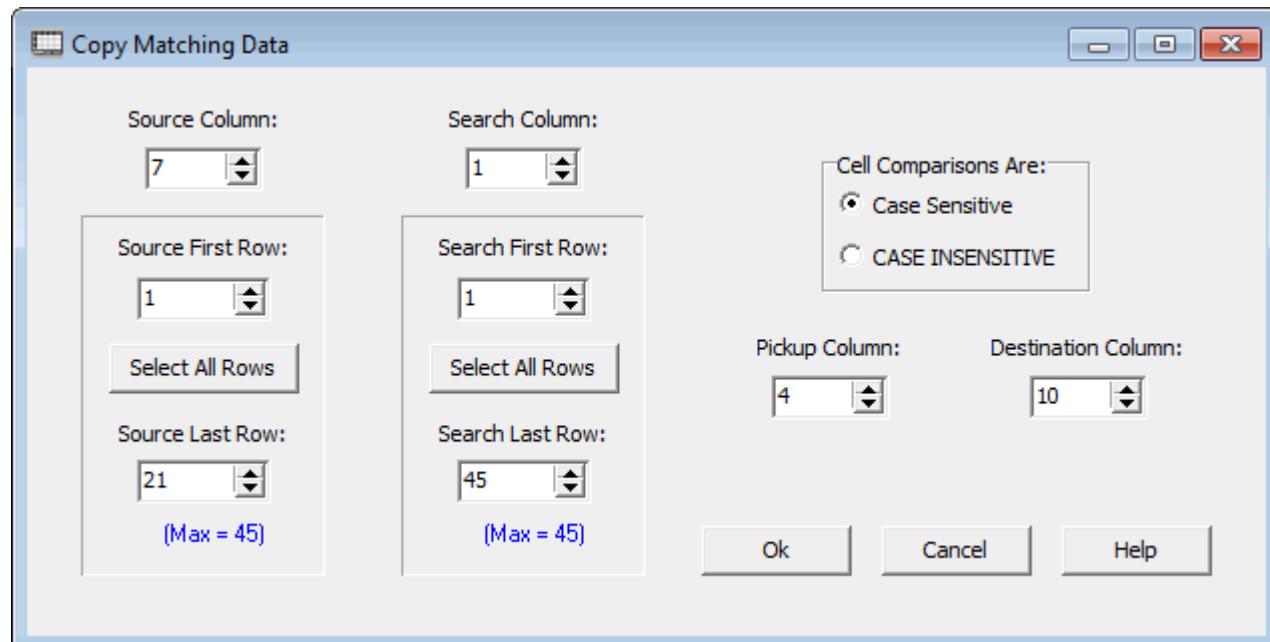
Now to obtain the author information, we add a new 9th blank column and then we perform another **Utilities | Copy Matching Data...** and we fill out the dialog the same as what we did before for the **Source Column** and the **Search Column**, but we change the **Pickup Column** and the **Destination Column**.



The new result has the Author information in Column 9 and the next grid appears as:

Column 7	Column 8	Column 9
978-0-596-80552-4	JavaScript The Definitive Guide	Flanagan, David
978-1-4129-1314-0	The Tao of Statistics	Keller, Dana K.
978-0-13-322495-5	Floating Point Computation	Sterbens, Pat
978-0-8176-4704-9	Handbook of Floating Point Arithmetic	Muller, Jean-Michel
978-0-19-923400-4	Oxford Dictionary of Computing 6th Edition	Oxford University Press
978-0-19-974044-4	Algorithmic Puzzles	Levitin, Anay; Levitin, Maria
978-0-691-14714-7	9 Algorithms That Changed The Future	MacCormick, John
978-0-486-47417-5	A Book of Abstract Algebra, 2nd Edition	Pinter, Charles C.
978-0-471-11709-4	Applied Cryptography	Schneier, Bruce
978-0-8218-4418-2	Finite Fields and Applications	Mullen, Gary L.
978-0-691-15270-7	In Pursuit of the Traveling Salesman	Cook, William J.
978-0-321-62930-2	What is a p-value is it anyway?	Vickers, Andrew
978-0-674-05755-5	Measurement	Lockhart, Paul
978-1-118-46446-5	Raspberry Pi User Guide	Upton, Eben; Helfacree, Gareth
978-0-984-72511-3	Race Against The Machine	Brynjolfsson, Erik; McAfee, Andrew
978-0-470-40765-3	Be Expert With Map & Compass	Kjellstrom, Bjorn; Elgin, Carina
978-1-56881-721-7	Unexpected Expectations	Wapner, Leonard
978-0-7484-0304-2	Map Projections A Reference Manual	Snyder, John; Bugayevskiy, Lev
978-0-691-15271-4	Henri Poincare A Scientific Biography	Gray, Jeremy
978-0-321-81958-1	Lightroom 4	Kelby, Scott
978-0-9560030-7-2	Adobe Photoshop Lightroom 4 The Missing FAQ	Bampton, Victoria

Finally, to obtain the publisher information, we add a new 10th blank column and then we perform another **Utilities | Copy Matching Data...** where we fill out the dialog the same as what we did before for the **Source Column** and the **Search Column**, but we change the **Pickup Column** and the **Destination Column**.



The resulting grid now has the Publisher information in Column 10 and appears as:

Column 7	Column 8	Column 9	Column 10
978-0-596-80552-4	JavaScript The Definitive Guide	Flanagan, David	O'Reilly & Associates
978-1-4129-1314-0	The Tao of Statistics	Keller, Dana K.	Sage Publications
978-0-13-322495-5	Floating Point Computation	Sterbens, Pat	Prentice Hall
978-0-8176-4704-9	Handbook of Floating Point Arithmetic	Muller, Jean-Michel	Birkhauser
978-0-19-923400-4	Oxford Dictionary of Computing 6th Edition	Oxford University Press	Oxford University Press
978-0-19-974044-4	Algorithmic Puzzles	Levitin, Anay; Levitin, Maria	Oxford University Press
978-0-691-14714-7	9 Algorithms That Changed The Future	MacCormick, John	Princeton University Press
978-0-486-47417-5	A Book of Abstract Algebra, 2nd Edition	Pinter, Charles C.	Dover Publications
978-0-471-11709-4	Applied Cryptography	Schneier, Bruce	John Wiley & Sons
978-0-8218-4418-2	Finite Fields and Applications	Mullen, Gary L.	American Mathematical Society
978-0-691-15270-7	In Pursuit of the Traveling Salesman	Cook, William J.	Princeton University Press
978-0-321-62930-2	What is a p-value is it anyway?	Vickers, Andrew	Addison-Wesley
978-0-674-05755-5	Measurement	Lockhart, Paul	Belknap Press
978-1-118-46446-5	Raspberry Pi User Guide	Upton, Eben; Halfacree, Gareth	Wiley Computer Publishing
978-0-984-72511-3	Race Against The Machine	Brynjolfsson, Erik; McAfee, Andrew	Digital Frontier Press
978-0-470-40765-3	Be Expert With Map & Compass	KJellstrom, Bjorn; Elgin, Carina	John Wiley & Sons
978-1-56881-721-7	Unexpected Expectations	Wapner, Leonard	CRC Press
978-0-7484-0304-2	Map Projections A Reference Manual	Snyder, John; Bugayevskiy, Lev	CRC Press
978-0-691-15271-4	Henri Poincare A Scientific Biography	Gray, Jeremy	Princeton University Press
978-0-321-81958-1	Lightroom 4	Kelby, Scott	New Riders Publishing
978-0-9560030-7-2	Adobe Photoshop Lightroom 4 The Missin FAQ	Bampton, Victoria	Lightroom Queen Publishing

At this point we could edit the Header information and extract the block of 21 rows in Column 7 through Column 10 to arrive at the grid that has all the information we needed to derive. The above final answer consists of a block of 21 rows and 4 columns of corresponding book information.

For this example we did not try to get the **Date Received** information that was in Column 5, but if needed, we could perform yet another **Copy Matching Data** operation to copy that information into a new 11th column.

Hopefully this example has provided you with enough information to understand how the **Copy Matching Data** function works. The hardest part is correctly filling out the dialog information.

If you should enter improper setup information in the dialog, you may generate output that shows

KEY DATA NOT FOUND

in some cells in the destination column.

We will now summarize the processes involved. First, we assume we have two tables named **Table 1** and **Table 2** that share a common single key column type of data. **Table 2** need only consist of a single key column where we assume some of the entries in that column are the same entries as exist in the key column of **Table 1**. The problem is to essentially compute either **Table 1** intersect **Table 2**, or to compute **Table 1** minus **Table 2**, where the answer data we want consists of most of the columns from **Table 1** along with the key data. See the figure on the next page. **Table 1** always contains the data columns that you want to pickup.

Table 1

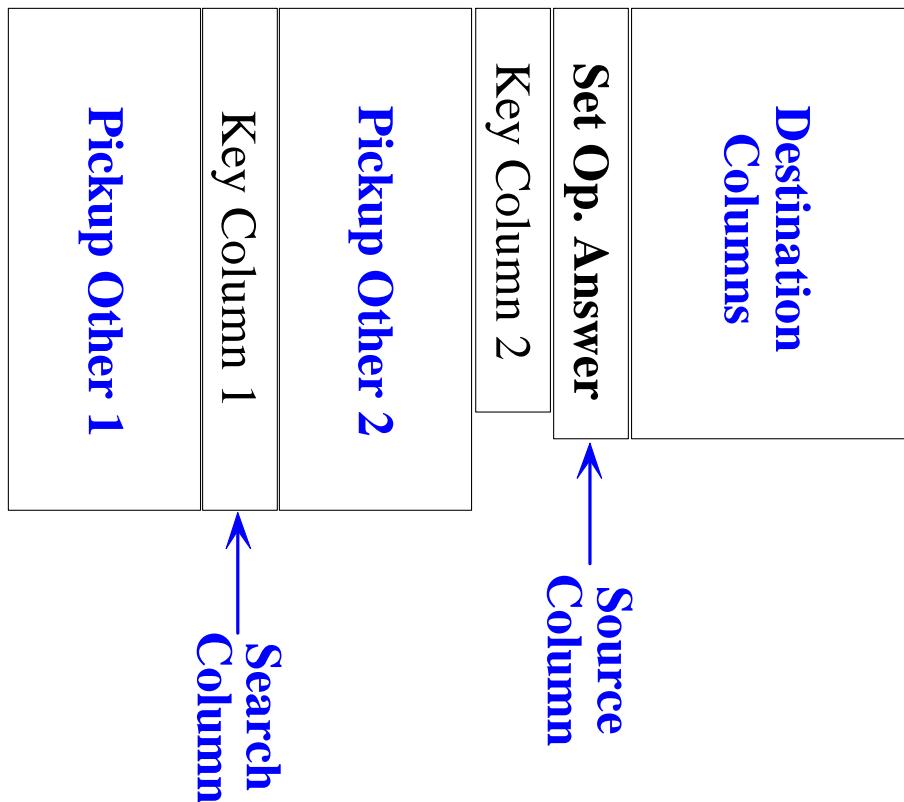
Pickup Other 2
Key Column 1
Pickup Other 1

Table 2

Key Column 2

In practice, we need to first merge the **Table 2** key column with **Table 1** because to compute a set intersection or a set difference requires all the key cells to be in one grid. So the first step is to append the single key column of **Table 2** as an additional column to **Table 1**. After that, we can append several other blank columns to what we now call the **Computation Table**, that is the modified **Table 1**. The first added blank column will be used to store the set operation answer and the remaining blank columns will be considered as **Destination Columns** because they will be used to store copies of the **Pickup data**. So our next view is as follows:

Computation Table



Note the positions of what we now call the **Source Column** and the **Search Column**. The **Source Column** is the same as the **Set Operation Answer** column. The **Search Column** is the key column for the block of cells that contains the pickup data that is to be copied.

Also note that the number of rows in **Key Column 1** and **Key Column 2** and the **Set Operation Answer** column can all be different. After we compute the **Set Operation Answer** we no longer need or use **Key Column 2**.

Now to actually perform the **Copy Matching Data** function, we setup the two different row ranges for the **Source Column** and the **Search Column**. Then we select one **Pickup Column** and select one **Destination Column** at a time and perform the **Copy Matching Data** function multiple times, until all the **Destination Columns** have been filled.

The number of **Destination Columns** will usually be the same as the number of columns in the union of most, if not all, of the **Pickup Columns**. You don't have to pickup all the data from the original **Table 1** grid, but most of the time you will want to pickup most of the **Table 1** columns so as to keep all the related information together.

At the very end of the process we can extract the columns from the **Computation Table** that consist of the **Set Operation Answer Column** and the **Destination Columns** to create what we call the **Answer** grid. Further, we may need to remove any blank rows that remain at the bottom of the **Answer** grid.

You may wish to compare everything we have explained in this help topic with another help topic that discusses how to merge multiple columns from one grid into another grid, but using a key column that is common to both grids. Here we actually manually setup and compute the key column set difference or key column set intersection before we can begin to copy any of the other data columns, and even then, we copy the other data columns one at a time.

Although the intersection operator is commutative, in other words $T1 \cap T2 = T2 \cap T1$, when using the **Copy Matching Data** function, it does make a difference as to which of your two original tables is used to pickup data. Don't think using **Table 1** and **Table 2** is commutative because their usage is different as **Table 1** is the main table to be opened. As described, you always want to make **Table 1** the table that contains the data you want copied, the pickup data. **Table 2** should be the one in which you will be using only the key column to determine the intersection set answer. The choice of which table should be considered as **Table 1** is more clear when the operation is a set difference, because set differences are not commutative.

See also the help topic [Merging CSV Files With Common Key Columns](#) under the **File** menu. Another related topic worthy of reading is [Conversions Using a Replacement List](#) under the **Conversions** menu. These topics are closely related to performing a join operation on database tables. See also [Export Matching Key Rows...](#) under the **Rows** menu. In general, you should find using this latter function is faster and easier than using the function described in this help topic.

Creating an Address Text File Using the CSV Grid

The Utilities menu item that says **Create an Address Text File Using the CSV Grid...** requires that your grid have exactly 11 columns in which the meaning of the data in each column matches the meanings indicated by the second yellow header row shown in the figure below.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Title:	First Name:	Middle Name:	Last Name:	Company Name:	Address Line 1:	Address Line 2:	City:	State:	ZIP Code:	Country:
1	Mr.	John	Michael	Doe		123 Main Street		New York	NY	12345	
2	Mrs.	Mary	Jane	Doe		Apt. #12B	789 Lincoln Blvd.	Los Angeles	CA	90025	
3		The		President	General George Washington	The White House	1600 Pennsylvania Avenue	Washington	DC	00000	USA
4		John	B.	Smith	The John Smith Company	Rancho Business Park	12345 Elm Street	Los Angeles	CA	90025-4951	USA
5	Dr.	Layla		Gelf		Suite #312	3333 Ventura Blvd.	Encino	CA	93421	

This function simply uses the grid data and writes out an address text file similar to that shown below. Each row in the grid represents one person's address information.

Mr. John Michael Doe
123 Main Street
New York, NY 12345

Mrs. Mary Jane Doe
Apt. #12B
789 Lincoln Blvd.
Los Angeles, CA 90025

The President
General George Washington
The White House
1600 Pennsylvania Avenue
Washington, DC 00000
USA

John B. Smith
The John Smith Company
Rancho Business Park
12345 Elm Street
Los Angeles, CA 90025-4951
USA

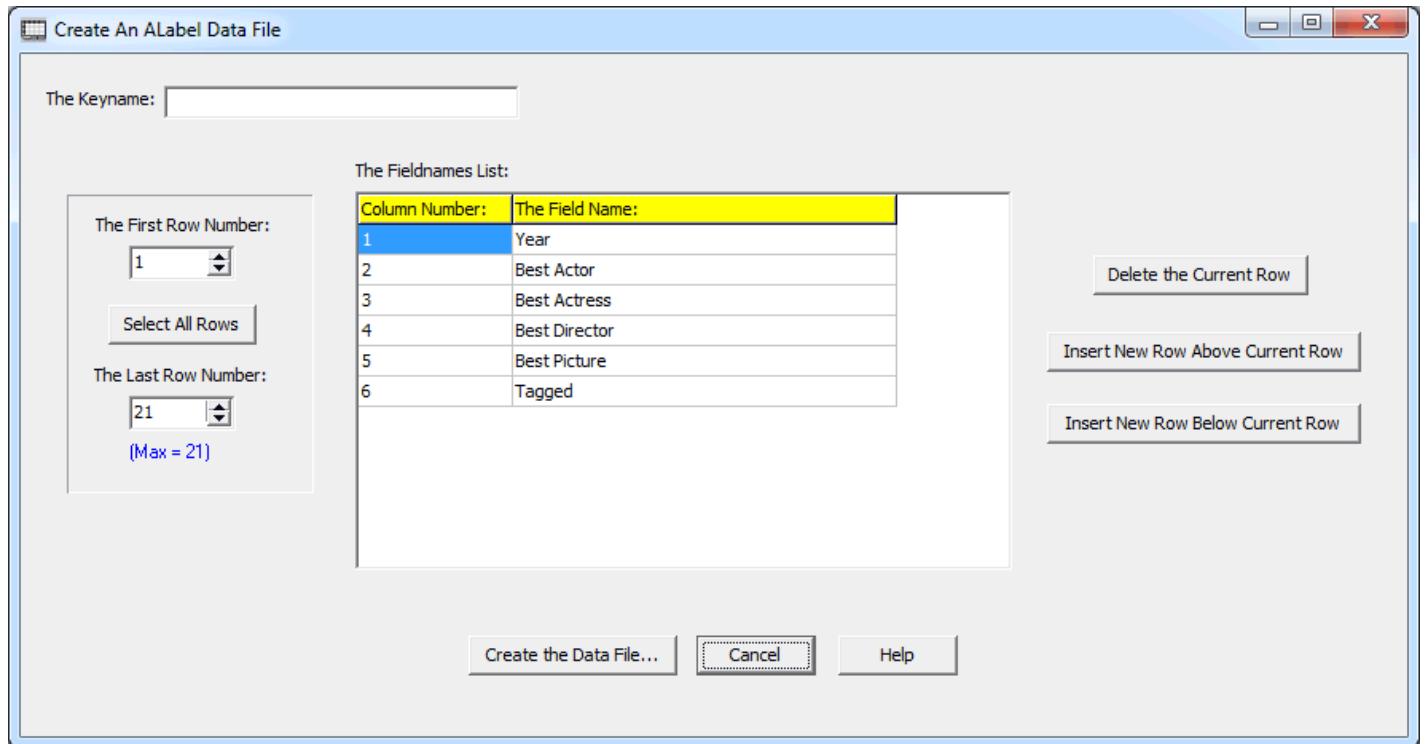
Dr. Layla Gelf
Suite #312
3333 Ventura Blvd.
Encino, CA 93421

Each person must have a First Name and a Last Name and AddressLine1 must not be empty. The City is also required to be non-empty and the State must be a non-empty 2-letter abbreviation. Neither the Title nor the Middle Name nor the Company Name are required. The Country is also optional. AddressLine2 will normally have the street number and street name when AddressLine1 contains other information. Otherwise AddressLine2 will be blank when AddressLine1 contains that information.

Creating an ALabel Data File

The **Utilities** menu contains a special menu item with the title **Create an ALabel Data File...**. To use this function it is assumed you are already familiar with the **ALabel** program and have opened a grid whose columns contain data that conforms to a particular **ALabel** program items template design. If you don't know about items template files for the **ALabel** program then you should not try to use this function.

When you select **Utilities | Create an ALabel Data File...** you will see a dialog similar to the following:



The **Keyname** edit box needs to be filled with an **ALabel** program items template keyname.

The **Fieldnames List** will initially be filled with all the columns and their header titles. In the above example, we opened the Academy Awards CSV example grid as shown on the next page. The program chose all six columns from our standard table and it made the **Field Names** the same as the header titles. If your current grid is not displaying header titles, then the second column in the list will be filled with names that are of the form **fieldnameN** where **N** is the column number. In any case, you can use the three buttons to the right of the listing grid to either delete or add new columns/fieldname pairs. The first column should only contain actual column numbers that are in the range of valid columns for the current grid. You don't have to use all the columns in the grid and you can change **The Field Name** for any particular column as needed.

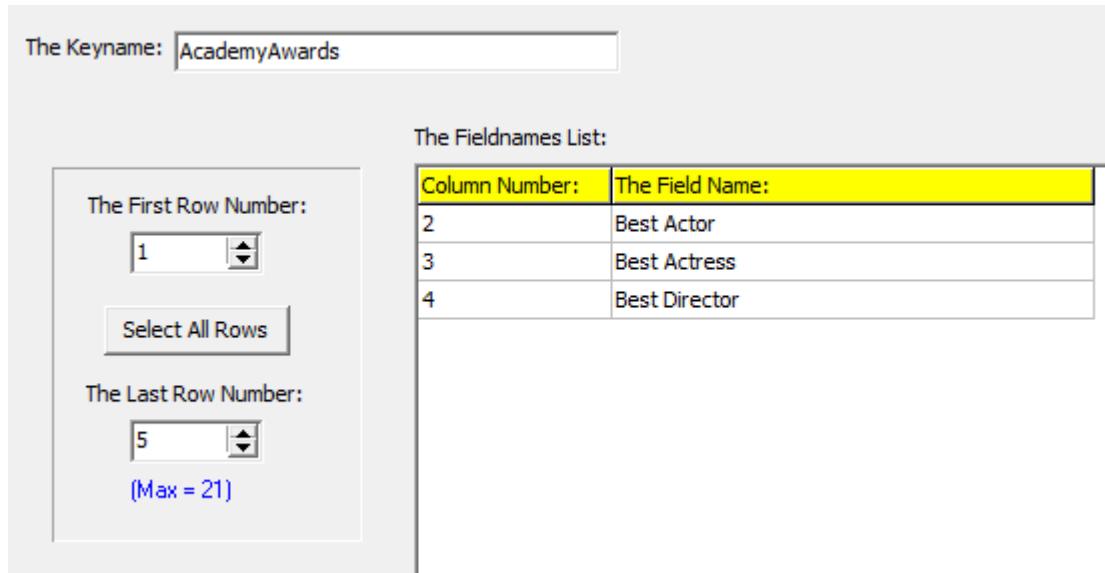
Finally you should select a range of rows from which you intend to print labels. When everything is setup as needed you can click the button with the caption **Create the Data File...**. The program will first verify that **The Keyname** is not empty and it will check that all column numbers are valid and that all **Field Names** are not empty. If any violations are found you will be returned to the above setup dialog for further editing.

Otherwise, if everything is Ok, when you click the button **Create the Data File...**, you will be immediately prompted for a save filename with a standard save dialog. You can navigate to any directory and you can give the **ALabel** data file any name you need. The default name is **KeynameDataFile.txt**. where **Keyname** is whatever keyname you entered.

For an explanatory example, suppose we open the Academy Awards grid that appears as show below:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	Year	Best Actor	Best Actress	Best Director	Best Picture	Tagged
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment	T
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story	T
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia	T
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones	T
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady	T
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music	T
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons	T
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night	T
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver	T
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy	T

When the dialog appears, to setup the creation of an **ALabel** data file, suppose we enter **The Keyname** as **AcademyAwards** and assume we select only three columns (2, 3, and 4). This means we deleted three columns in the dialog. Assume we limit the range of rows to be from 1 to 5. The main controls in the setup dialog would appear as shown below before we create the data file.



When we click the button **Create the Data File...** the output made by this function is a simple text file whose contents should appear as shown on the next page.

Our data file is named **AcademyAwardsDataFile.txt**.

A view of the output data text file is the following:

```
AcademyAwards
Best Actor: Burt Lancaster
Best Actress: Elizabeth Taylor
Best Director: Billy Wilder
Next Label
Best Actor: Maximilian Schell
Best Actress: Sophia Loren
Best Director: Robert Wise
Next Label
Best Actor: Gregory Peck
Best Actress: Anne Bancroft
Best Director: David Lean
Next Label
Best Actor: Sidney Poitier
Best Actress: Patricia Neal
Best Director: Tony Richardson
Next Label
Best Actor: Rex Harrison
Best Actress: Julie Andrews
Best Director: George Cukor
Next Label
```

Note that each **Field Name** is terminated by a colon character and one space character. Then the data follows on the same line. The first line in the **ALabel** data file is the keynote, in this case **AcademyAwards**.

There is data for five labels, where the first label automatically begins right after the first line that is the keynote. Note how the label data corresponds to the data in the table for each of the selected output rows. Each label, including the very last one, is terminated by the line **Next Label**.

When this data file is printed on a sheet of labels, the result might look like what is shown on the next page. Of course the actual output will depend on the **ALabel** template design. We will assume the template design for a typical label looks like:



Actor's Name
Actress' Name
Director's Name

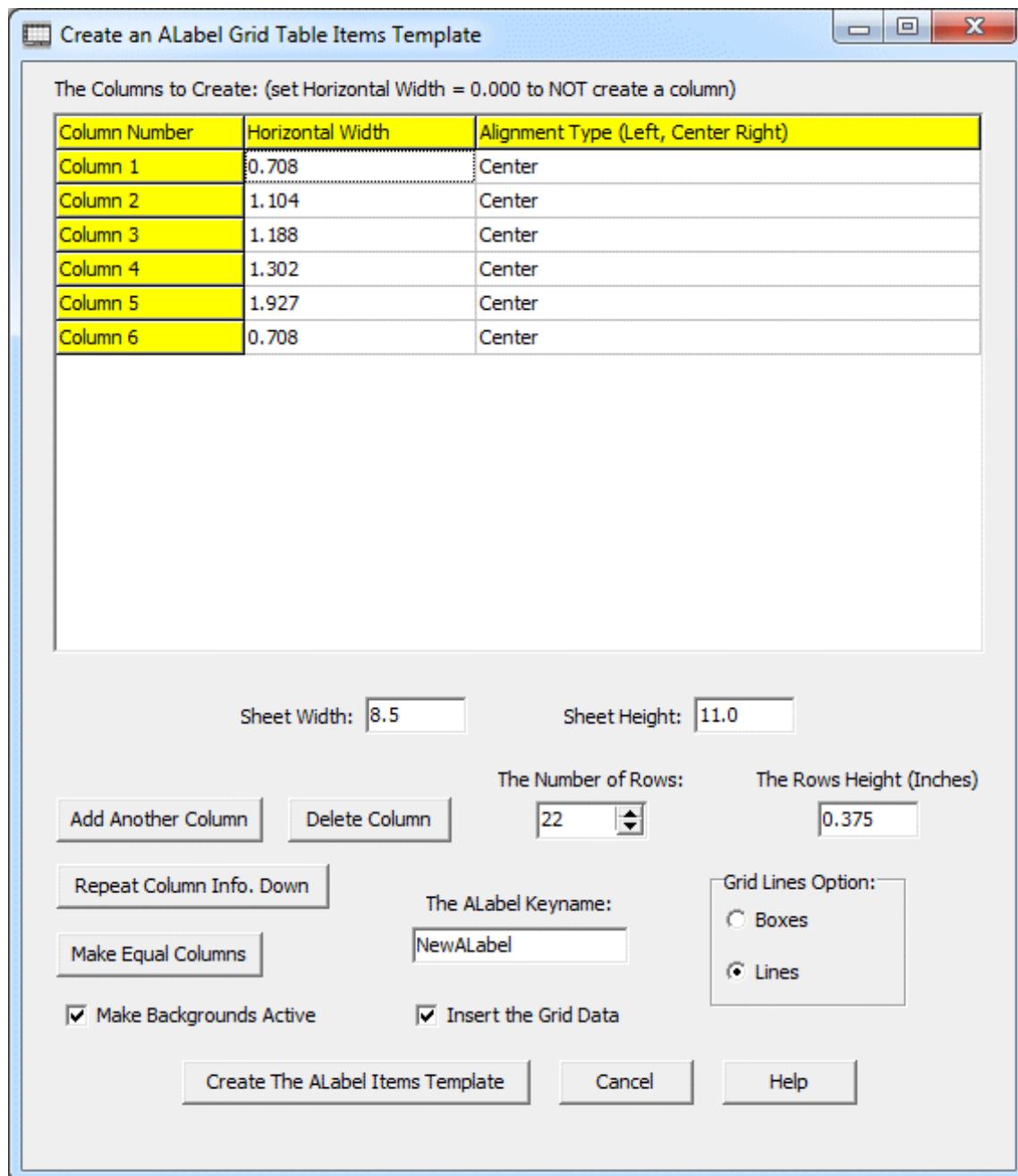
Then when the above data file is actually printed on a sheet of labels, the final output (using black ink) would appear as:

Burt Lancaster
Elizabeth Taylor
Billy Wilder
Maxmillian Schell
Sophia Loren
Robert Wise
Gregory Peck
Anne Bancroft
David Lean
Sidney Poitier
Patricia Neal
Tony Richardson
Rex Harrison
Julie Andrews
George Cukor

You might note that because we restricted the range of rows to be 1-5, the above output is limited to 5 rows. The output matches the data contained in the data file that is shown on the previous page. The format of the labels matches that shown on the previous page for the items template design.

Creating an ALabel Grid Table

If you are not familiar with the **ALabel** program then you can ignore this topic. The **Utilities** menu contains a very special menu item with the title **Create an ALabel Grid Table Items Template File....** When you select this menu item you will see a dialog box similar to the one shown below. In fact, this example table is for the Academy Awards CSV file. This dialog box is used to setup and define a single page of table information for another program that is called the **ALabel** program.



The program will normally create as many columns for this dialog as there are columns in the currently opened **CSV** grid. This explains why there are 6 columns listed because the Academy Awards **CSV** file has 6 columns.

You have the option to change the **Horizontal Width** for any column and you have the option to change the alignment type for that column. If you want to delete a given column, you should first click in the column row for that column and then you can click the button with the caption **Delete Column**. The remaining columns will be renumbered and their row information will move up in the table.

The button with the caption **Repeat Column Info. Down** is used to fill in the column information from the currently selected row to all rows below that row. This can save you a lot of typing when the columns are uniform in size after a given column.

The button with the caption **Make Equal Columns** is used to automatically set the column width of all columns, assuming your sheet has left and right side margins that are 0.25 inches wide each. In other words, the program will subtract 0.5 inches from the **Sheet Width**, and it will equally divide the remaining space to determine the same width for all columns. This button saves you from having to calculate the column width and then replicate that information.

The checkbox with the caption **Make Backgrounds Active** is specific to a feature in the **ALabel** program. All it means is that the resulting cells can be easily given a background color when the items template is edited within the **ALabel** program itself. If this checkbox is turned off, then all cells will not have their background color used. In general we recommend turning this option on, but whether you do this or not, you won't see any difference because the default background color is white anyway.

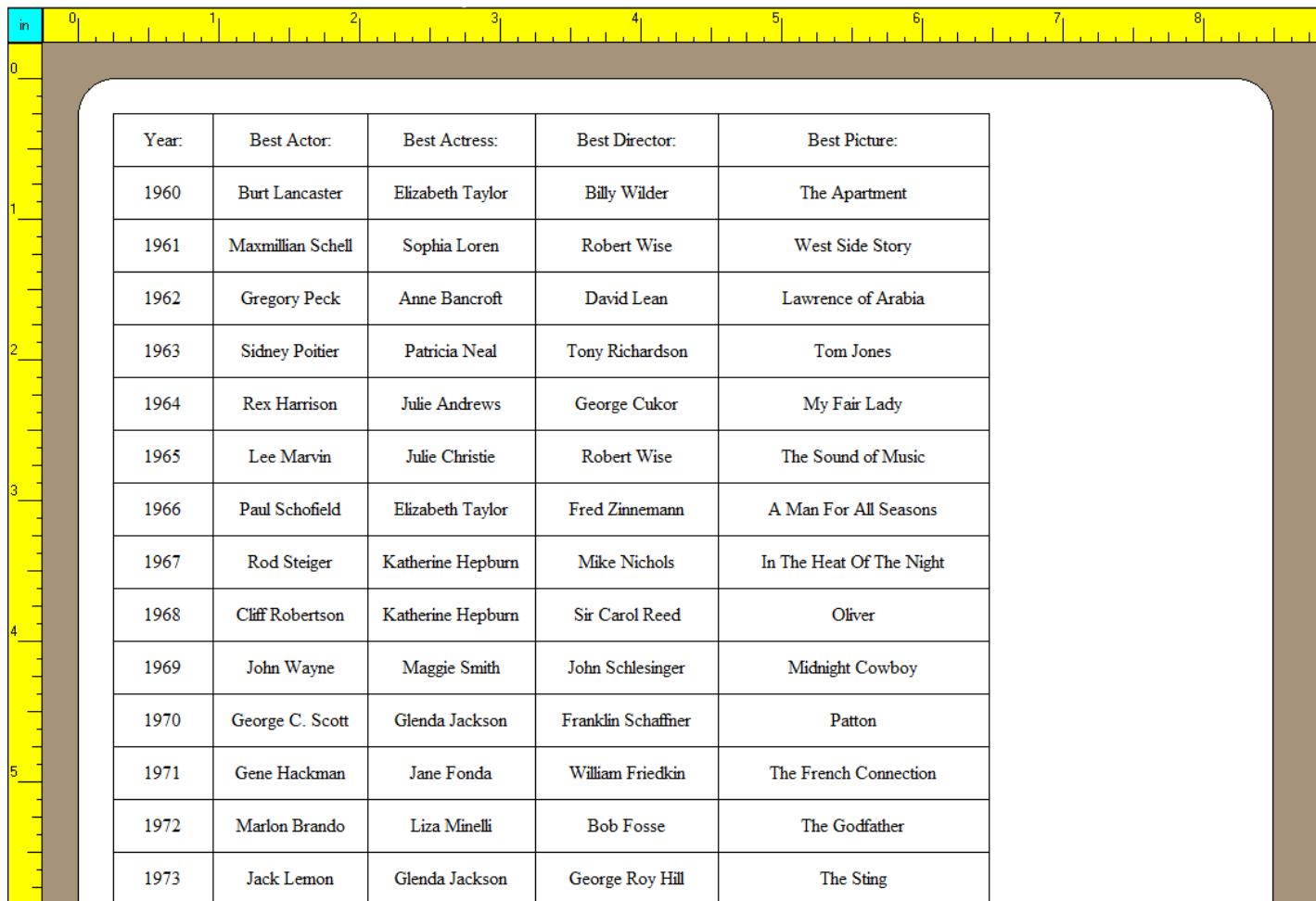
You can set the number of rows you want to create, and you can even change the size of the default sheet that is for a standard sheet of paper. You don't want to create more rows than will fit on the sheet whose size you define.

Note that you can precisely define the width of each column. All rows will have the same height that you can also precisely define.

For the most part you won't see any difference between choosing **Boxes** or **Lines** for the **Grid Lines Option**. However, **Lines** is the default and this choice is more efficient in terms of the number of objects to be created. Using **Boxes** does allow you to later merge entries by combining cells. We chose **Lines** for this example.

When you click the button to **Create The ALabel Items Template**, the program will prompt you for where you would like to save the file it is about to create. For the above example, we created the grid table that appears as shown next. The created file will only be of interest to users of the program named **ALabel**. Before making the example shown below we deleted the **Column 6** information so we only had 5 columns for the Academy Awards **CSV** file.

We won't explain the meaning of the grid for the **ALabel** program because that program has too many technical details. Users already familiar with the **ALabel** program will immediately appreciate the next figure which is taken from that program. The next figure is the result of opening the saved file in the **ALabel** program. Thus you will never see the next figure in the **CSV Editor** program.



A vertical ruler scale is positioned on the left side of the page, showing measurements from 0 to 8 inches. The scale is yellow with black markings and numbers.

Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:
1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment
1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story
1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia
1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones
1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady
1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music
1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons
1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night
1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver
1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy
1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton
1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection
1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather
1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting

What you see above is taken from a program named **ALabel**. This page of information is very special for the **ALabel** program. Our purpose here is only to show the result. If you are familiar with the **ALabel** program then you will understand just how special this table is for that program. Each text string is centered inside what is called a block text object. The entire page is an 8.5x11 inch sheet and it contains a total of 133 objects for the **ALabel** program. This **CSV Editor** program automatically created all 133 objects. No further explanation of these results will be discussed in this help file because that discussion is not relevant to using the **CSV Editor** program.

Creating a CSV File From An Address Text File

The **Utilities** menu contains a special menu item with the title **Create a CSV File From An Address Text File...**

When you execute this function you will be asked to open an **Address Text File**. What we mean by an address text file is an ordinary text file that contains a list of people's names and their addresses. The exact requirements will be described below. A typical address text file might look like the following:

Mr. John Michael Doe
123 Main Street
New York, NY 12345

Mrs. Mary Jane Doe
Apt. #12B
789 Lincoln Blvd.
Los Angeles, CA 90025

The President
General George Washington
The White House
1600 Pennsylvania Avenue
Washington, DC 00000
USA

John B. Smith
The John Smith Company
Rancho Business Park
12345 Elm Street
Los Angeles, CA 90025-4951
USA

Dr. Layla Gelf
Suite #312
3333 Ventura Blvd.
Encino, CA 93421

This file format is very simple. Each person is required to have an address that contains anywhere between 3 and 6 lines of information. Each person is separated from the next person by exactly one blank line in the file. When this file is converted to a **CSV** file the result should look like the following:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
1	Mr.	John	Michael	Doe		123 Main Street		New York	NY	12345	
2	Mrs.	Mary	Jane	Doe		Apt. #12B	789 Lincoln Blvd.	Los Angeles	CA	90025	
3		The		President	General George Washington	The White House	1600 Pennsylvania Avenue	Washington	DC	00000	USA
4		John	B.	Smith	The John Smith Company	Rancho Business Park	12345 Elm Street	Los Angeles	CA	90025-4951	USA
5	Dr.	Layla		Gelf		Suite #312	3333 Ventura Blvd.	Encino	CA	93421	

We can discern that each person will have their information split into 11 distinct columns. If we add header information for the grid we can see the intended meaning of each column below. Note that a few of the data items are blank, and this means each person may not have all 11 columns of information in the address text file.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Title:	First Name:	Middle Name:	Last Name:	Company Name:	Address Line 1:	Address Line 2:	City:	State:	ZIP Code:	Country:
1	Mr.	John	Michael	Doe		123 Main Street		New York	NY	12345	
2	Mrs.	Mary	Jane	Doe		Apt. #12B	789 Lincoln Blvd.	Los Angeles	CA	90025	
3		The		President	General George Washington	The White House	1600 Pennsylvania Avenue	Washington	DC	00000	USA
4		John	B.	Smith	The John Smith Company	Rancho Business Park	12345 Elm Street	Los Angeles	CA	90025-4951	USA
5	Dr.	Layla		Gelf		Suite #312	3333 Ventura Blvd.	Encino	CA	93421	

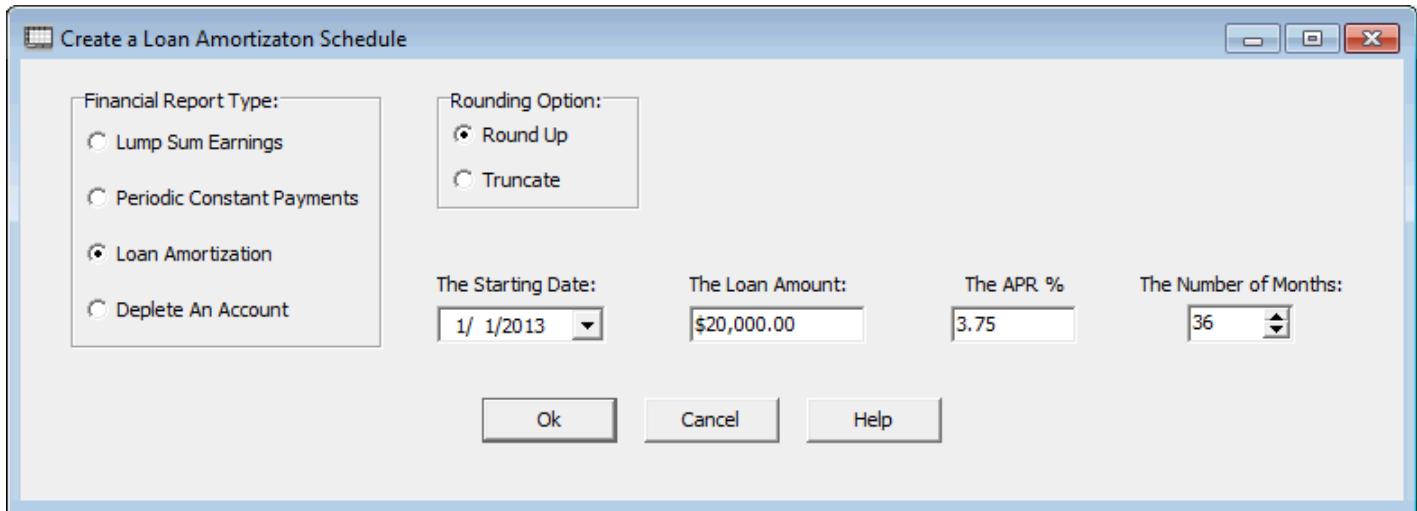
You might also note that both the Title and the name parts for George Washington are not correct. In general, all columns should probably contain no more than 30 characters each, except the State column should probably be limited to 2 characters and the ZIP Code doesn't need to contain more than 10 characters.

In fact, the Title column and First Name and Last Name columns did not translate for George Washington as you might otherwise expect. This means you may have to do some manual editing when an entry has an unusual Title. It is impossible for the program to detect all possible titles together with First and Middle and Last Names, especially when they are distributed across the first two lines of a new address.

Each person must have a First Name and a Last Name and AddressLine1 must not be empty. The City is also required to be non-empty and the State must be a non-empty 2-letter abbreviation. Neither the Title nor the Middle Name nor the Company Name are required. The Country is also optional. AddressLine2 will normally have the street number and street name when AddressLine1 contains other information. Otherwise AddressLine2 will be blank when AddressLine1 contains that information. In an address text file, there should always be a comma and a single space immediately following the City name and before the State abbreviation. Exactly one space character should separate the State and the ZIP code.

Creating a Financial Schedule

There is a menu item under the **Utilities** menu that has the title **Create a Financial Schedule....** This function is used to compute a table of values that are related to any of four different financial schedules. When you select this menu item you will see a setup dialog that should appear similar to:



Loan Amortization Schedule

In the above dialog we have already set the values for what might be a typical car loan. First, we set the **Financial Report Type** to **Loan Amortization**. Second, we set the **Rounding Option** to **Round Up**. This means any time a dollar amount is calculated, the value will be rounded up to the nearest penny. We have set **The Loan Amount** to be \$20,000.00 and we assume the annual interest rate is 3.75% and we assume the loan is a 3-year loan that will last for 36 months. We also set the starting date for the loan to be January 1, 2013 which means the first loan payment won't actually be until February 1, 2013.

When you click the **Ok** button the program will automatically overwrite the existing grid and it will fill a new grid with the values shown on the next page. The program also prompts you to immediately save the new grid in a new file that you name.

There are a few special calculations that we make. To keep things simple, we only assume monthly payments that always occur on the same day of the month. What we denote by **The APR %** is sometimes also known as the nominal annual rate. Precisely defining interest rates can be a subtle mathematical idea, because there are several different kinds of interest rates. There is another rate named the effective rate, but internally we compute using a monthly rate that is what you enter divided by 12 because there are 12 months in a year.

We also round values to the nearest penny each time we calculate a new value. On the next page you may note that the final balance in the last row and in the last column is off by 4 cents. In fact, the final balance is a negative 4 cents. The enclosing parentheses indicate a negative value. It is normal for a loan remaining balance to only be near zero after the last payment. In case you are interested, we compute all the basic financial values using the two formulas:

$$FV = PV(1 + i)^n \quad \text{and} \quad FV = PMT \cdot \left\{ \frac{(1 + i)^n - 1}{i} \right\}$$

PV denotes the present value of the loan and i denotes the periodic interest rate and n denotes the number of compounding time periods (in our application months). FV denotes the future value, and we compute the FV value first, using the first formula, and then we use the same FV value in the second formula to finally compute the constant periodic payment amount PMT . PV and i and n are what you enter in the dialog. i is further converted from a percent to an ordinary decimal value before it is used in either formula.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	PMT #:	The Date:	PMT Amount:	Interest Charged:	Balance Decrease:	Remaining Balance:
1	n = 36	i = 0.003125	PV = \$20,000.00	PMT = \$588.26	FV = \$22,377.52	
2						
3		01/01/2013				\$20,000.00
4	1	02/01/2013	\$588.26	\$62.50	\$525.76	\$19,474.24
5	2	03/01/2013	\$588.26	\$60.86	\$527.40	\$18,946.84
6	3	04/01/2013	\$588.26	\$59.21	\$529.05	\$18,417.79
7	4	05/01/2013	\$588.26	\$57.56	\$530.70	\$17,887.09
8	5	06/01/2013	\$588.26	\$55.90	\$532.36	\$17,354.73
9	6	07/01/2013	\$588.26	\$54.23	\$534.03	\$16,820.70
10	7	08/01/2013	\$588.26	\$52.56	\$535.70	\$16,285.00
11	8	09/01/2013	\$588.26	\$50.89	\$537.37	\$15,747.63
12	9	10/01/2013	\$588.26	\$49.21	\$539.05	\$15,208.58
13	10	11/01/2013	\$588.26	\$47.53	\$540.73	\$14,667.85
14	11	12/01/2013	\$588.26	\$45.84	\$542.42	\$14,125.43
15	12	01/01/2014	\$588.26	\$44.14	\$544.12	\$13,581.31
16	13	02/01/2014	\$588.26	\$42.44	\$545.82	\$13,035.49
17	14	03/01/2014	\$588.26	\$40.74	\$547.52	\$12,487.97
18	15	04/01/2014	\$588.26	\$39.02	\$549.24	\$11,938.73
19	16	05/01/2014	\$588.26	\$37.31	\$550.95	\$11,387.78
20	17	06/01/2014	\$588.26	\$35.59	\$552.67	\$10,835.11
21	18	07/01/2014	\$588.26	\$33.86	\$554.40	\$10,280.71
22	19	08/01/2014	\$588.26	\$32.13	\$556.13	\$9,724.58
23	20	09/01/2014	\$588.26	\$30.39	\$557.87	\$9,166.71
24	21	10/01/2014	\$588.26	\$28.65	\$559.61	\$8,607.10
25	22	11/01/2014	\$588.26	\$26.90	\$561.36	\$8,045.74
26	23	12/01/2014	\$588.26	\$25.14	\$563.12	\$7,482.62
27	24	01/01/2015	\$588.26	\$23.38	\$564.88	\$6,917.74
28	25	02/01/2015	\$588.26	\$21.62	\$566.64	\$6,351.10
29	26	03/01/2015	\$588.26	\$19.85	\$568.41	\$5,782.69
30	27	04/01/2015	\$588.26	\$18.07	\$570.19	\$5,212.50
31	28	05/01/2015	\$588.26	\$16.29	\$571.97	\$4,640.53
32	29	06/01/2015	\$588.26	\$14.50	\$573.76	\$4,066.77
33	30	07/01/2015	\$588.26	\$12.71	\$575.55	\$3,491.22
34	31	08/01/2015	\$588.26	\$10.91	\$577.35	\$2,913.87
35	32	09/01/2015	\$588.26	\$9.11	\$579.15	\$2,334.72
36	33	10/01/2015	\$588.26	\$7.30	\$580.96	\$1,753.76
37	34	11/01/2015	\$588.26	\$5.48	\$582.78	\$1,170.98
38	35	12/01/2015	\$588.26	\$3.66	\$584.60	\$586.38
39	36	01/01/2016	\$588.26	\$1.83	\$586.43	(\$0.04)

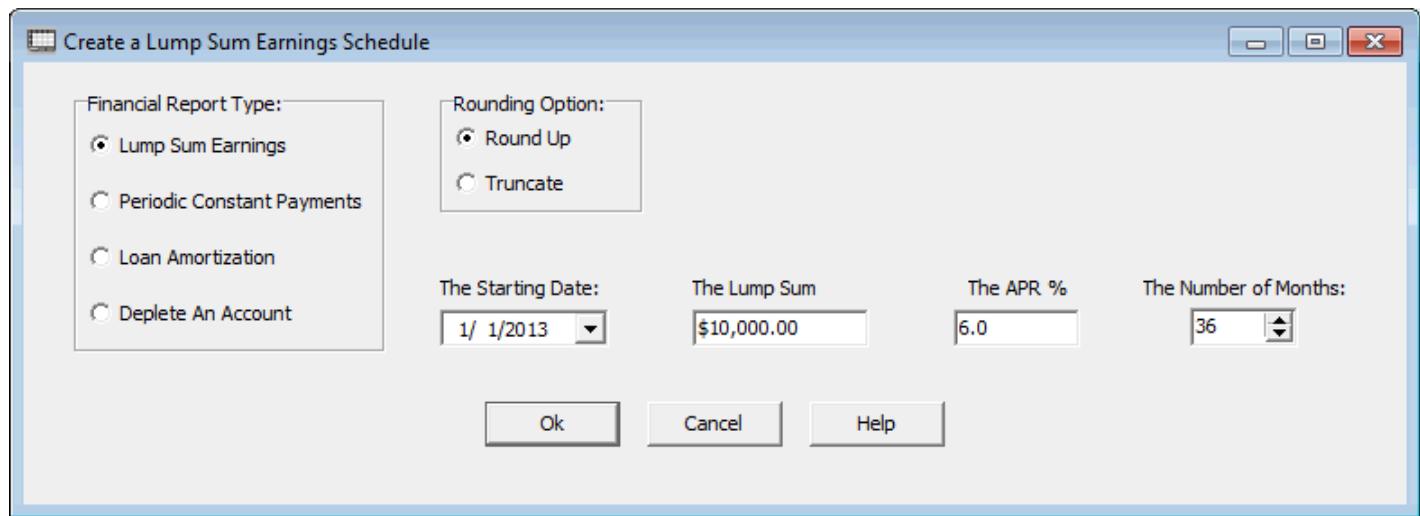
Note that grid row 1 gives some overall information about the loan while grid row 2 is blank. The series of loan payments does not begin until grid row 4 that has the first payment information. Due to different types of rounding, if you generate an amortization schedule using a bank's web tool, you may get slightly different results. Our schedules should closely match what you would get using an HP-12C financial calculator when that calculator shows two decimal places. In other words, we should be accurate to about ± 1 penny.

There are four different kinds of financial schedules you can make. Probably the most typical is a loan amortization schedule for a car loan or a house loan. Usually all such loans involve monthly payments. Next we will discuss the other three types of financial schedules that this program can make.

Lump Sum Interest Earning Schedule

Perhaps the simplest kind of a financial schedule is one in which you deposit a lump sum into an account and you let that account earn a constant interest rate for a number of equally spaced time periods. Let's say you deposit \$10,000.00 in an account that earns a constant 6% per year. Assuming the account earns interest that is compounded monthly, how much will be in the account after 3 years?

To setup this problem you can choose **Utilities | Create a Financial Schedule...** and when the dialog comes up, setup the following values:



Each time you click on a different radio button in the **Financial Report Type** control, the dialog box title and some of the labels will change to make appropriate settings. For example, the edit box above the **Cancel** button had a previous label that said **The Loan Amount**. That label has now changed to show **The Lump Sum**.

We set **The Lump Sum** as \$10,000 and we set the **APR %** to 6.0 and we set **The Number of Months** to 36.

When you click the **Ok** button the program will automatically overwrite the existing grid and it will fill a new grid with the values shown on the next page. The program also prompts you to immediately save the new grid in a new file that you name.

In this example we just let the lump sum of \$10,000 earn compound interest for 36 months. We make no further deposits into the account, other than the initial \$10,000 deposit. We do of course let the account earn interest that is compounded monthly. The resulting schedule shows how the **New Balance** grows month after month.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	PMT #:	The Date:	Earned Interest:	New Balance:	Comment:
1	n = 36	i = 0.005	PV = \$10,000.00	PMT = \$304.22	FV = \$11,966.81
2					
3		01/01/2013		\$10,000.00	
4	1	02/01/2013	\$50.00	\$10,050.00	= \$9,745.78 + \$304.22
5	2	03/01/2013	\$50.25	\$10,100.25	= \$9,490.29 + \$609.96
6	3	04/01/2013	\$50.50	\$10,150.75	= \$9,233.52 + \$917.23
7	4	05/01/2013	\$50.75	\$10,201.50	= \$8,975.46 + \$1,226.04
8	5	06/01/2013	\$51.01	\$10,252.51	= \$8,716.12 + \$1,536.39
9	6	07/01/2013	\$51.26	\$10,303.77	= \$8,455.48 + \$1,848.29
10	7	08/01/2013	\$51.52	\$10,355.29	= \$8,193.54 + \$2,161.75
11	8	09/01/2013	\$51.78	\$10,407.07	= \$7,930.29 + \$2,476.78
12	9	10/01/2013	\$52.04	\$10,459.11	= \$7,665.73 + \$2,793.38
13	10	11/01/2013	\$52.30	\$10,511.41	= \$7,399.84 + \$3,111.57
14	11	12/01/2013	\$52.56	\$10,563.97	= \$7,132.62 + \$3,431.35
15	12	01/01/2014	\$52.82	\$10,616.79	= \$6,864.06 + \$3,752.73
16	13	02/01/2014	\$53.08	\$10,669.87	= \$6,594.16 + \$4,075.71
17	14	03/01/2014	\$53.35	\$10,723.22	= \$6,322.91 + \$4,400.31
18	15	04/01/2014	\$53.62	\$10,776.84	= \$6,050.31 + \$4,726.53
19	16	05/01/2014	\$53.88	\$10,830.72	= \$5,776.34 + \$5,054.38
20	17	06/01/2014	\$54.15	\$10,884.87	= \$5,501.00 + \$5,383.87
21	18	07/01/2014	\$54.42	\$10,939.29	= \$5,224.28 + \$5,715.01
22	19	08/01/2014	\$54.70	\$10,993.99	= \$4,946.18 + \$6,047.81
23	20	09/01/2014	\$54.97	\$11,048.96	= \$4,666.69 + \$6,382.27
24	21	10/01/2014	\$55.24	\$11,104.20	= \$4,385.80 + \$6,718.40
25	22	11/01/2014	\$55.52	\$11,159.72	= \$4,103.51 + \$7,056.21
26	23	12/01/2014	\$55.80	\$11,215.52	= \$3,819.81 + \$7,395.71
27	24	01/01/2015	\$56.08	\$11,271.60	= \$3,534.69 + \$7,736.91
28	25	02/01/2015	\$56.36	\$11,327.96	= \$3,248.15 + \$8,079.81
29	26	03/01/2015	\$56.64	\$11,384.60	= \$2,960.17 + \$8,424.43
30	27	04/01/2015	\$56.92	\$11,441.52	= \$2,670.75 + \$8,770.77
31	28	05/01/2015	\$57.21	\$11,498.73	= \$2,379.89 + \$9,118.84
32	29	06/01/2015	\$57.49	\$11,556.22	= \$2,087.57 + \$9,468.65
33	30	07/01/2015	\$57.78	\$11,614.00	= \$1,793.79 + \$9,820.21
34	31	08/01/2015	\$58.07	\$11,672.07	= \$1,498.54 + \$10,173.53
35	32	09/01/2015	\$58.36	\$11,730.43	= \$1,201.81 + \$10,528.62
36	33	10/01/2015	\$58.65	\$11,789.08	= \$903.60 + \$10,885.48
37	34	11/01/2015	\$58.95	\$11,848.03	= \$603.90 + \$11,244.13
38	35	12/01/2015	\$59.24	\$11,907.27	= \$302.70 + \$11,604.57
39	36	01/01/2016	\$59.54	\$11,966.81	= \$0.00 + \$11,966.81

The comment column is given so you may compare this financial schedule with the one that is made next for depleting an account.

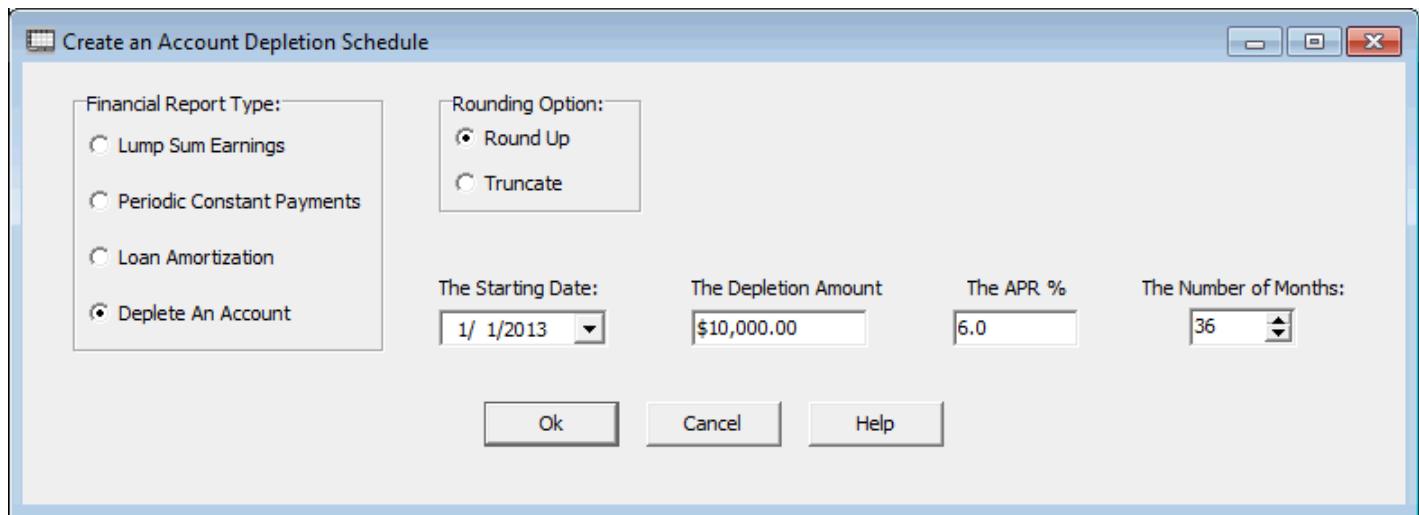
Depleting an Account

The opposite of having a lump sum earn interest is to start with an account that already has a certain amount that can be considered as the starting lump sum. You then make periodic constant withdrawals until the account is depleted. The main variable is the number of months until you want the account to be depleted.

The program will automatically determine for you what constant amount should be withdrawn each month. You just enter **The Depletion Amount** and **The Number of Months** along with the annual interest rate.

Let's assume you have an account that starts with \$10,000.00 and that over the next three years you will withdraw a constant amount every month until the account is essentially empty. As in the previous example, we will assume the account earns 6% per year, but we assume the earned interest is compounded monthly.

You should select **Utilities | Create a Financial Schedule...** and setup the dialog with the following values.



Note we have selected the fourth radio button in the **Financial Report Type** group. The edit box above the **Cancel** button has changed its label so it now appears as: **The Depletion Amount**. Previously that label showed **The Lump Sum**. We set **The Depletion Amount** to \$10,000.00.

When you click the **Ok** button the program will automatically overwrite the existing grid and it will fill a new grid with the values shown on the next page. The program also prompts you to immediately save the new grid in a new file that you name.

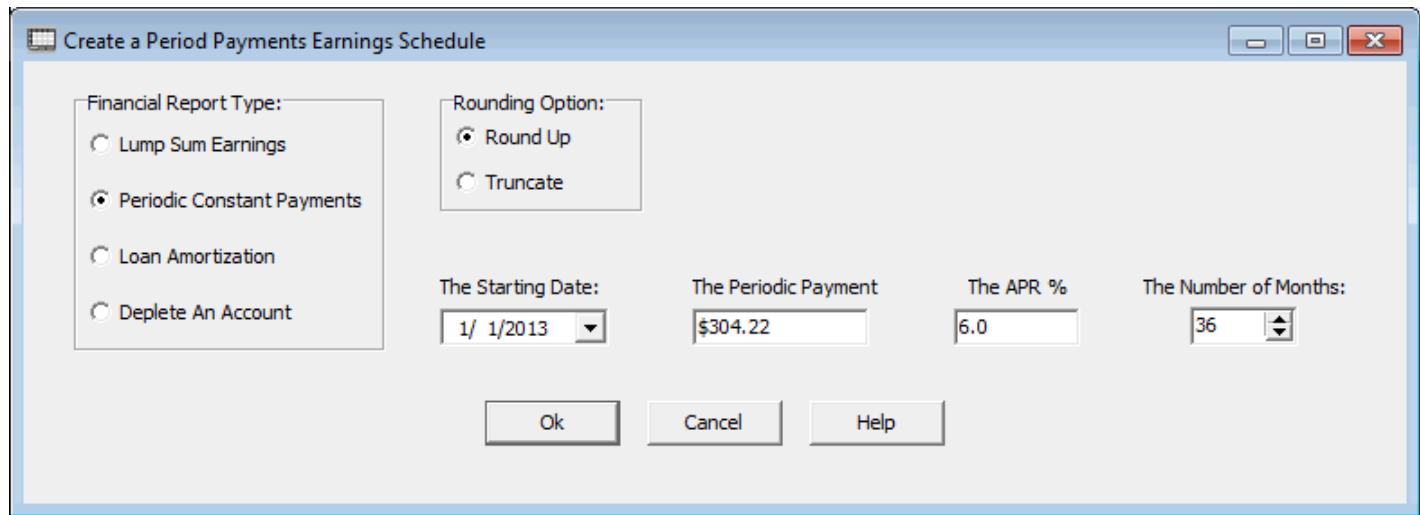
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	PMT #:	The Date:	PMT Amount:	Interest Earned:	Balance Decrease:	Remaining Balance:
1	n = 36	i = 0.005	PV = \$10,000.00	PMT = \$304.22	FV = \$11,966.81	
2						
3		01/01/2013				\$10,000.00
4	1	02/01/2013	\$304.22	\$50.00	\$254.22	\$9,745.78
5	2	03/01/2013	\$304.22	\$48.73	\$255.49	\$9,490.29
6	3	04/01/2013	\$304.22	\$47.45	\$256.77	\$9,233.52
7	4	05/01/2013	\$304.22	\$46.17	\$258.05	\$8,975.47
8	5	06/01/2013	\$304.22	\$44.88	\$259.34	\$8,716.13
9	6	07/01/2013	\$304.22	\$43.58	\$260.64	\$8,455.49
10	7	08/01/2013	\$304.22	\$42.28	\$261.94	\$8,193.55
11	8	09/01/2013	\$304.22	\$40.97	\$263.25	\$7,930.30
12	9	10/01/2013	\$304.22	\$39.65	\$264.57	\$7,665.73
13	10	11/01/2013	\$304.22	\$38.33	\$265.89	\$7,399.84
14	11	12/01/2013	\$304.22	\$37.00	\$267.22	\$7,132.62
15	12	01/01/2014	\$304.22	\$35.66	\$268.56	\$6,864.06
16	13	02/01/2014	\$304.22	\$34.32	\$269.90	\$6,594.16
17	14	03/01/2014	\$304.22	\$32.97	\$271.25	\$6,322.91
18	15	04/01/2014	\$304.22	\$31.61	\$272.61	\$6,050.30
19	16	05/01/2014	\$304.22	\$30.25	\$273.97	\$5,776.33
20	17	06/01/2014	\$304.22	\$28.88	\$275.34	\$5,500.99
21	18	07/01/2014	\$304.22	\$27.50	\$276.72	\$5,224.27
22	19	08/01/2014	\$304.22	\$26.12	\$278.10	\$4,946.17
23	20	09/01/2014	\$304.22	\$24.73	\$279.49	\$4,666.68
24	21	10/01/2014	\$304.22	\$23.33	\$280.89	\$4,385.79
25	22	11/01/2014	\$304.22	\$21.93	\$282.29	\$4,103.50
26	23	12/01/2014	\$304.22	\$20.52	\$283.70	\$3,819.80
27	24	01/01/2015	\$304.22	\$19.10	\$285.12	\$3,534.68
28	25	02/01/2015	\$304.22	\$17.67	\$286.55	\$3,248.13
29	26	03/01/2015	\$304.22	\$16.24	\$287.98	\$2,960.15
30	27	04/01/2015	\$304.22	\$14.80	\$289.42	\$2,670.73
31	28	05/01/2015	\$304.22	\$13.35	\$290.87	\$2,379.86
32	29	06/01/2015	\$304.22	\$11.90	\$292.32	\$2,087.54
33	30	07/01/2015	\$304.22	\$10.44	\$293.78	\$1,793.76
34	31	08/01/2015	\$304.22	\$8.97	\$295.25	\$1,498.51
35	32	09/01/2015	\$304.22	\$7.49	\$296.73	\$1,201.78
36	33	10/01/2015	\$304.22	\$6.01	\$298.21	\$903.57
37	34	11/01/2015	\$304.22	\$4.52	\$299.70	\$603.87
38	35	12/01/2015	\$304.22	\$3.02	\$301.20	\$302.67
39	36	01/01/2016	\$304.22	\$1.51	\$302.71	(\$0.03)

Note that the interest earned each month becomes less and less because as we make the same \$304.22 withdrawal each month, there is less and less money left in the account to earn interest. After the final withdrawal there is essentially nothing left in the account. In fact, in this example, because negative 3 cents is left, your final withdrawal could be \$304.19.

Making a Series of Periodic Constant Payments

Our last example of making a financial schedule is one in which we will make a series of periodic constant payments into an account. We assume there is nothing in the account when we make our first payment. This means we don't earn any new interest until we make the second payment. We assume this account earns 6% annual interest, and we further assume that interest is compounded monthly. We will make constant payments of \$304.22 each month. How much money will be in the account after 3 years?

To find out, we select **Utilities | Create a Financial Schedule...** and we setup the dialog as follows:



Note that we have selected the second radio button for the **Financial Report Type**. Also note that the edit box above the **Cancel** button now has a new label that says: **The Periodic Payment**. In the previous example that label read as **The Starting Amount**. We set **The Periodic Payment** to \$304.22.

When you click the **Ok** button the program will automatically overwrite the existing grid and it will fill a new grid with the values shown on the next page. The program also prompts you to immediately save the new grid in a new file that you name.

By reading the value in the last row and the last column from the grid on the next page we can determine the answer to the question as to how much money can be expected to be in the account after 3 years. That amount is \$11,966.83. This amount is the Future Value FV that also appears in row 1 and column 5.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	PMT #:	The Date:	PMT Amount:	Interest Earned:	Balance Increase:	New Balance:
1	n = 36	i = 0.005	PV = \$10,000.02	PMT = \$304.22	FV = \$11,966.83	
2						
3		01/01/2013				\$0.00
4	1	02/01/2013	\$304.22	\$0.00	\$304.22	\$304.22
5	2	03/01/2013	\$304.22	\$1.52	\$305.74	\$609.96
6	3	04/01/2013	\$304.22	\$3.05	\$307.27	\$917.23
7	4	05/01/2013	\$304.22	\$4.59	\$308.81	\$1,226.04
8	5	06/01/2013	\$304.22	\$6.13	\$310.35	\$1,536.39
9	6	07/01/2013	\$304.22	\$7.68	\$311.90	\$1,848.29
10	7	08/01/2013	\$304.22	\$9.24	\$313.46	\$2,161.75
11	8	09/01/2013	\$304.22	\$10.81	\$315.03	\$2,476.78
12	9	10/01/2013	\$304.22	\$12.38	\$316.60	\$2,793.38
13	10	11/01/2013	\$304.22	\$13.97	\$318.19	\$3,111.57
14	11	12/01/2013	\$304.22	\$15.56	\$319.78	\$3,431.35
15	12	01/01/2014	\$304.22	\$17.16	\$321.38	\$3,752.73
16	13	02/01/2014	\$304.22	\$18.76	\$322.98	\$4,075.71
17	14	03/01/2014	\$304.22	\$20.38	\$324.60	\$4,400.31
18	15	04/01/2014	\$304.22	\$22.00	\$326.22	\$4,726.53
19	16	05/01/2014	\$304.22	\$23.63	\$327.85	\$5,054.38
20	17	06/01/2014	\$304.22	\$25.27	\$329.49	\$5,383.87
21	18	07/01/2014	\$304.22	\$26.92	\$331.14	\$5,715.01
22	19	08/01/2014	\$304.22	\$28.58	\$332.80	\$6,047.81
23	20	09/01/2014	\$304.22	\$30.24	\$334.46	\$6,382.27
24	21	10/01/2014	\$304.22	\$31.91	\$336.13	\$6,718.40
25	22	11/01/2014	\$304.22	\$33.59	\$337.81	\$7,056.21
26	23	12/01/2014	\$304.22	\$35.28	\$339.50	\$7,395.71
27	24	01/01/2015	\$304.22	\$36.98	\$341.20	\$7,736.91
28	25	02/01/2015	\$304.22	\$38.68	\$342.90	\$8,079.81
29	26	03/01/2015	\$304.22	\$40.40	\$344.62	\$8,424.43
30	27	04/01/2015	\$304.22	\$42.12	\$346.34	\$8,770.77
31	28	05/01/2015	\$304.22	\$43.85	\$348.07	\$9,118.84
32	29	06/01/2015	\$304.22	\$45.59	\$349.81	\$9,468.65
33	30	07/01/2015	\$304.22	\$47.34	\$351.56	\$9,820.21
34	31	08/01/2015	\$304.22	\$49.10	\$353.32	\$10,173.53
35	32	09/01/2015	\$304.22	\$50.87	\$355.09	\$10,528.62
36	33	10/01/2015	\$304.22	\$52.64	\$356.86	\$10,885.48
37	34	11/01/2015	\$304.22	\$54.43	\$358.65	\$11,244.13
38	35	12/01/2015	\$304.22	\$56.22	\$360.44	\$11,604.57
39	36	01/01/2016	\$304.22	\$58.02	\$362.24	\$11,966.81

Note that the balance increases by more than \$304.22 each month because the account is constantly earning interest that is compounded. We are making constant payments of \$304.22 but each monthly increase grows more compared to the previous month, due to the compounding where interest is earned on top of other interest earnings.

File Hash Values (SHA-256)

We provide a special utility function that is used to compute a signature or a hash value for any file. The hash value is created using what is called a **Secure Hash Algorithm** that returns 256 bits for the hash value. A 256-bit value can also be thought of as a value that consists of 32 bytes of data, or 64 hexadecimal characters. The formal name of the algorithm is **SHA-256**. This algorithm is supported by the National Institute of Standards and Technology (**NIST**) that is part of the United States Government.

To understand what hash values are and how they are used, we can make an analogy with human fingerprints and human signatures. Human fingerprints as well as human signatures are used to uniquely identify individuals. That is, each human is considered to have a unique fingerprint as well as a unique signature. Well, in the real world it may be possible for two people to have the same fingerprint, but such an occurrence is exceedingly rare. It is also possible for someone else to fake your signature, but a handwriting expert is usually able to determine an imposter. In the same way, hash values are uniquely associated with files. While it is theoretically possible for two files to have the same hash value, that would be nearly impossible and is even more rare than two humans having the same fingerprint or the same signatures.

The advantage of having and using hash values is that compared to real files that can sometimes be very large, a hash value by comparison is relatively very small. Hash values have a couple of other very important properties. One of these is that if a file is changed in some very minor way, the hash value associated with the new changed file will differ radically from the hash value associated with the original file. This means it is also nearly impossible to create or fake a hash value that will match the true hash value associated with a given file.

If you can somehow securely give me a hash value associated with one of your private files, then I can use that hash value to prove that another file given to me by someone else is the same as the file you have. In other words, using the hash value alone I can determine if another file is in fact the same file as the one you have. If a third party tries to tamper with your file by trying to change it, even in a very minor way by altering only one bit, I can determine if the file they present to me is genuinely your file or not.

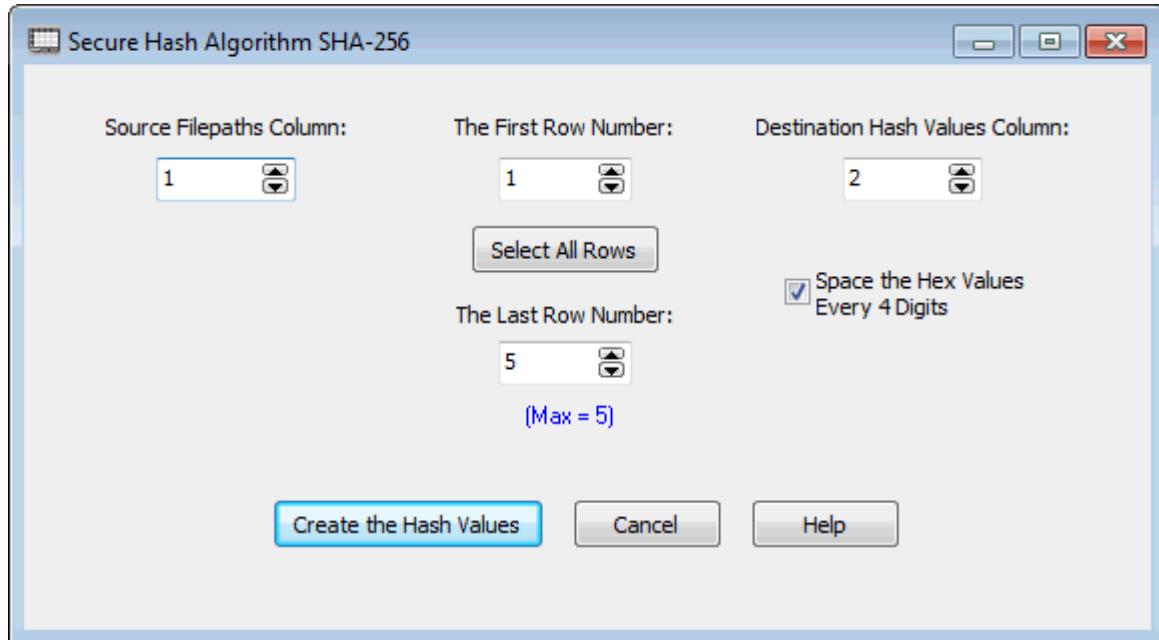
The **SHA-256** algorithm that we use has the following three essential properties.

1. It is computationally infeasible to find (discover or produce or generate) another file that will match a given 256-bit hash value.
2. It is computationally infeasible to find (discover or produce or generate) two different files that will match each other's 256-bit hash values.
3. Any change to an original file (no matter how small) will produce with very high probability a totally different 256-bit hash value.

Using hash values alone, we can prove whether or not two files are identical. Neither file has to be in our possession to do this, as long as we have the true hash values associated with each file. Comparing the hash values is equivalent to comparing the contents of the two files. In this manner we can determine if a file has been changed or tampered with, and this technique forms the basis for what are called electronic signatures. While we don't fully discuss electronic signatures here because they require other cryptographic functions, one essential part is being able to compute hash values that have the above three properties.

To compute the hash values associated with a given collection of files, you would normally select a block of cells within a given grid and use the **Block Fill** function to load the cells in a single column with the complete filepaths for the desired files.

Then you would select the function **Utilities | File Hash Values (SHA-256)...** and when you do this you will see the dialog box shown below.



When using this dialog you should think as if you were performing a conversion function. The source filepaths are expected to be in one column while the destination column will get the computed hash values. You can apply the hash function using a selected range of rows. If any files are non-existent or cannot be opened for reading, the corresponding value in the destination column will contain the word **Error** with a brief explanation as to why the hash value could not be computed.

The next figure below shows an example grid in which the 1st column contains a list of complete filepaths and the 2nd column contains the corresponding hash values. In other words, the filepaths and hash values correspond on a row by row basis. When we ran this example, we selected the filepaths in the five rows of the first column. In the above dialog we checked the box to **Space the Hex Values Every 4 Digits** and that is why the hexadecimal values you see in column 2 contain a space character between every 4 hex digits. Otherwise all the hex digits would appear as one string of 64 hex characters. Clicking the button to **Create the Hash Values** causes the hash values to be computed for the filepaths in the source column selected rows. The computed hash values will get placed in the same selected rows, but in the destination column.

<	Column 1	Column 2
>	The File:	The SHA-256 Hash Value:
1	C:\JKSHA\ABC.txt	BA78 16BF 8F01 CFEA 4141 40DE 5DAE 2223 B003 61A3 9617 7A9C B410 FF61 F200 15AD
2	C:\JKSHA\TheQuickBrownFox.txt	EF53 7F25 C895 BFA7 8252 6529 A9B6 3D97 AA63 1564 D5D7 89C2 B765 448C 8635 FB6C
3	C:\JKSHA\Gettysburg.txt	628A 87D6 6049 18CA 1A9 CC7C 408D B616 9F73 0892 C6C2 AD8E D9B6 7D92 B93F A20A
4	C:\JKSHA\NISTSpecialExample.txt	248D 6A61 D206 38B8 E5C0 2693 0C3E 6039 A33C E459 64FF 2167 F6EC EDD4 19DB 06C1
5	C:\JKSHA\EmptyFile.txt	E3B0 C442 98FC 1C14 9AFB F4C8 996F B924 27AE 41E4 649B 934C A495 991B 7852 B855

We should show the contents of the text files that are listed in the above grid, just in case you wish to test our results with those of another program you might use that also computes 256-bit hash values. The first text file is named **ABC.txt** and its contents consists of exactly three letters, the lower-case letters that appear as:

abc

The second text file is named **TheQuickBrownFox.txt** and its content is exactly one sentence that appears as:

The quick brown fox jumps over the lazy dog.

For the above text file, note the last character in the sentence is the period character. If you were to drop the period character, the **SHA-256** hash value would change to:

D7A8 FBB3 07D7 8094 69CA 9ABC B008 2E4F 8D56 51E4 6D3C DB76 2D02 D0BF 37C9 E592

This is an example of the 3rd property of hash values that says if you slightly change a file you get a totally different hash value. Compare with the second row grid hash value on the previous page.

The third text file is named **Gettysburg.txt** and its contents is the famous speech given by President Abraham Lincoln. This text file contains carriage return and line feed characters that you don't see at the end of each line.

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this.

But, in a larger sense, we can not dedicate--we cannot consecrate--we cannot hallow-- this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note nor long remember, what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us--that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion--that we hereby highly resolve that these dead shall not have died in vain--that this nation, under God, shall have a new birth of freedom--and that government of the people, by the people, and for the people, shall not perish from the earth.

The fourth text file is named **NISTSpecialExample.txt** and its contents are the characters:

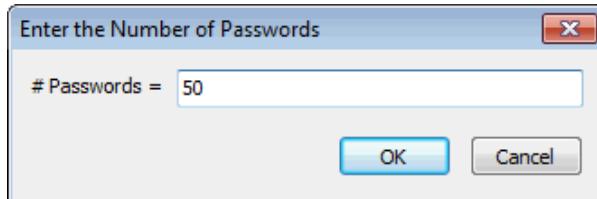
abcdefghijklmnopqrstuvwxyz

The fifth text file is named **EmptyFile.txt** and it is an empty file that contains no characters. This example exists just to show that every file can be given a hash value, even an empty file.

There is one practical restriction on the size of a file that is to be hashed. The file size must be less than or equal to $2,305,843,009,213,693,951 = 2^{61} - 1$ bytes.

Generating a List of Random Passwords

One of our special utility functions can be used to generate a list of random passwords. To make such a list just select the menu **Utilities | Generate a List of Random Passwords...** and the program may first ask if you would like to save the current grid before proceeding. Otherwise the program will prompt you for the number of passwords to be created where the default number is 50. You can enter any number up to 50,000.



After you click **Ok**, the program will immediately prompt you to save the list in a file that you can name.

<	Column 1
>	The Password:
1	OLIOSARS9CHICK6~
2	1\${fendcedersheer
3	agaze2ALP2(BOOMS
4	64ISKIERvireoTOBY
5	plea!#WAUGHslaty
6	8TUSBAblowyDOERSI;
7	GILDmaarsRUGBY&64
8	{vomitBLOOM6VENOM2
9	{guffAILS1SOUND1
10	k[wHELMricinbedew
11	2ULNAS1}ploptoons
12	typpsl2@rahEASTS
13	YEGGbrentDISCS2*8
14	8]7ohmstrodedices
15	OSLANGHAIKUerect:3
16	arksNAIL^orpin8
17	CLASTMISS3&CION6
18	2:9riskPINKOADD
19	coy5riped20VUM<
20	pacts6:\$CULLSarrow
21	hajes2wipes7(stats
22	tokesagy5styed8[
23	lullcaverPALSY)4%
24	9winositupGOLFS37
25	[8LOONSMUCINTORII^
26	2LOCOS34CAPERATED
27	dunkHONES5TOKED2[
28	**AMITYPILUSCAKE%
29	wifly[(2JUBEtroth
30	maud@KOLAliken2{

The program will automatically change the grid to a 1-column **CSV** file. It is important that you name the new file when you are prompted with a standard file save dialog.

The example passwords list shown above has only 30 passwords. Most of the passwords are between 12 and 18 characters in length and contain a mixture of upper and lower case letters as well as a sprinkling of random punctuation characters. The average length is usually more than 16 characters. Most passwords, but not all, usually have at least one numeric digit and at least one punctuation character and also usually have upper and lower case letters. It is not unusual to see some passwords that have no digits or no punctuation characters and that are either all upper case or are all lower case. Variety is the spice of life!

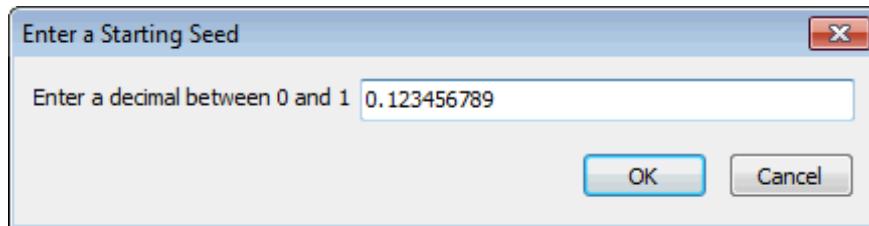
If you only need a few passwords, or if you want even longer passwords, you can manually edit and copy and paste new entries using entries from the list that is created.

Another technique is to create two such lists with the same number of rows and merge them together into one column. Then you can extract substrings from that column. The substrings might have any desired length or even random lengths.

Encrypting the Grid

The **CSV Editor** program allows you to encipher the characters in a grid. Our enciphering algorithm is only intended to make the grid appear unreadable and thus mostly secret. If you need a really secure form of a cipher then you should perform a more secure operation on the **CSV** file itself, not on the characters within the file. In other words, our enciphering is intended to make the contents of a grid completely disguised, but this is not the same as making the contents completely secure.

To encipher the characters in a grid, select the function **Utilities | Grid Encryption/Decryption | Encrypt the Grid...** and you will first be prompted to enter a seed value for a random number generator scheme.



The seed value is always a decimal number between 0 and 1. After you enter a seed, you should write it down on a piece of paper. You will need to use exactly the same seed value when you decrypt the grid.

When you click the **Ok** button, the program will automatically disguise all the characters in the grid.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:
2	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment
3	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story
4	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia
5	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones
6	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady
7	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music
8	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons
9	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night
10	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver
11	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy
12	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton
13	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection
14	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather
15	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting
16	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II
17	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest
18	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky
19	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall
20	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter
21	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer
22	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People

After encrypting the grid shown on the previous page that grid will automatically change and look like the following. Note that we have changed or disguised all the individual characters.

<	Column 1	Column 2	Column 3	Column 4	Column 5
1 >	G.<q	u.zS"LSNq(u.zS"LSq.zz(u.zS"xEq.;SNq(u.zS"IE;S9q(
2	C?4	u9q5"ac^<zS.q	b#E8<)S""@<w#Nq	uE##w",E#.q	@'."L<qSo.^S
3	C?4C	~<>oE##E<^"F;.##	FN E<"aNq.^	6N).qS".Ez.	..zS"FE\$."FSNqw
4	C?4r	iq.kNqw"l.;Q	L^^."u<:qN&S	x<JE\$"a.<^	a<Dq.^;"N&"Lq<)E<
5	C?4Y	FE\$^.w"INESE.q	I<SqE;E<"D.<#	@N^w"6E;.<q\$zN^	@No""N^.z
6	C?41	6.>"<qqEzN^	*9#E."L^\$q.Oz	i.Nqk."H9QNq	~w'<Eq"ac\$w
7	C?4R	a."~<qJE^	*9#E."H'qEzSE.	6N).qS".Ez.	@'."FN9^\$'N&"~9zE;
8	C?44	I<9#F;N&E.##	b#E8<)S""@<w#Nq	jq.\$")E^&.o<^	L"~<^"jNq"l##'F.<zN^z
9	C?4X	6N\$'FS.Ek.q	M<S'.qE^".:.)9q^	~EQ."DE;N#z	e""@'."<S"-&'@'."DEK'S
10	C?4I	H#E&"6N).qSzN^	M<S'.qE^".:.)9q^	FEq" H<qN#".\$.	-#E.J.q
11	C?4?	*N^&.<w^.	~<kkE."FoES'	*N^&F;#.zE^k.q	~E\$^Ek'S"HN0)Nw
12	C?X	i.Nqk."Hn" F;NSs	#.^.<".<:QzN^	jq<^Q#E^&F;.<&&.q	I<SSN^
13	C?XC	i.^".<:Qo<^	*^."jN^&`	,E##E<o"jqE.\$QE^	@'."jq.^;"HN^&SEN^
14	C?Xr	~<q#N^"uq<^\$N	aE8<"E^.&##E	uN)"jNzz.	@'."IN\$&<S'.q
15	C?XY	*;<Q;a.oN^	#.^.<".<:QzN^	i.Nqk."6Nw";E##	@'."FSE^k
16	C?X1	LqS" H<q^.w	b##.^.u9qzSw^	jq<^Ez"jNq\$"HN N#<	@'."IN\$&<S'.q"l<qS"ee
17	C?XR	*;<Q"DE;N#zN^	aN9Ez."j#.S'.q	~E#Nz"jNqo<^	~."j#.0"J.q"@"."H9;QNNz"D.zS
18	C?X4	I.S.q"IE^	jkw."x9^<0<w	*N^&in"lJ#E\$z.^	6N;Qw
19	C?XX	6E;.<q\$"xq.w&9zz	xE<^."M.<SN^	,NN\$w"l##.^	L^&E."<##
20	C?XI	*N^&KNEK'S	*^."jN^&`	~E;.<#HEoE^N	@'."x;q"9^S.q
21	C?X?	x9zSE^&N&&o<^	F<##w"IE.##	6N).qS"u.^SN^	Mq<o.q"jz" Mq<o.q
22	C?I	6N).qS" x."DEqN	FEzzw"Fl;Q	6N).qS"6.\$&Nq\$	-q\$E^<qw"l.N #.

We should also warn you that before you encrypt a grid you should turn off the display of the header because only the editable cells that have a white background will be enciphered.

Although our encryption algorithm is not particularly strong, it is more than sufficient for its intended purpose. We only replace printable ASCII characters with other printable ASCII characters. However, as the above example shows, both numbers and names will become disguised and are made unreadable by this process.

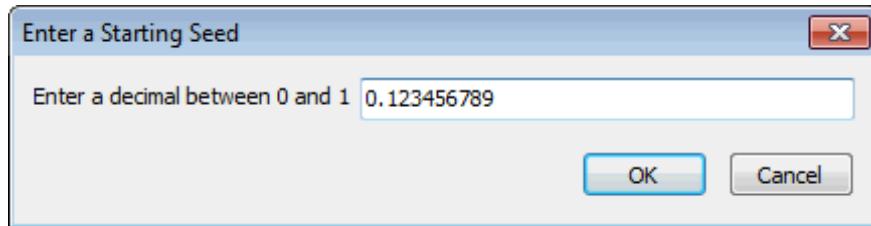
Just remember that once you have enciphered a grid, the only way to get back the original grid is to decode the enciphered grid using the same key seed value when you performed the encoding. Think of the decimal seed value as acting like a **PIN** number, but unlike many **PIN** numbers that may only use 3 or 4 digits, our seed values generally use 10 or more digits after the decimal point.

You always create a key based on the seed value you enter each time you encode a grid. In fact, you could perform double or triple encoding of a grid by selecting the encryption function two or three times in a row where for each invocation you enter a starting seed value. However, invoking multiple encodings is not necessary.

Decrypting the Grid

After you encipher the characters in a grid you will later need to eventually decipher that grid. To perform the decryption, you should first have the enciphered grid opened as the current grid. Remember to turn off the header display if it was turned off previously when the original grid was encoded.

Then select **Utilities | Grid Encryption/Decryption | Decrypt the Grid...** and you will be prompted to enter a starting seed value, just like when encrypting.



You must enter the same decimal value you entered when you encrypted the grid in the first place.

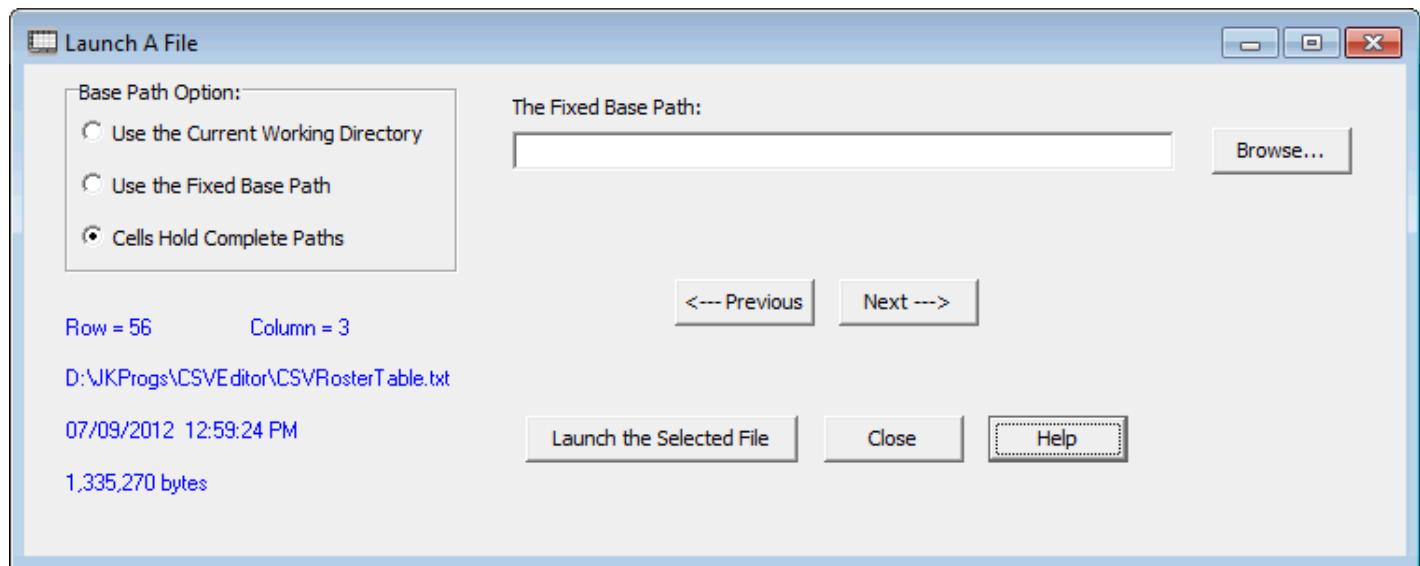
Think of the decimal seed value as acting like a **PIN** number, but unlike many **PIN** numbers that may only use 3 or 4 digits, our seed values generally use 10 or more digits after the decimal point.

When you click the **Ok** button the program will decode the current grid.

We won't show an example, because you can see the example on the previous two pages. Decryption is the inverse operation to encrypting.

Launching Files

There is a menu item under the **Utilities** menu that has the title **Launch a File From the Current Column...**. This brings up a dialog similar to that shown below that can be used to open or launch any file on your computer. The result would generally be the same as if you double clicked on the filename when in a file explorer window. Whatever the file type, the program will attempt to open that file. The intention here is that some column in your grid contains a list of filenames. Before opening the dialog, you should normally click on any cell in that column that has the name of a file that is to be launched or opened.



The blue text on the lower-left side of this dialog should show the current information for the currently selected file. The **Row** and **Column** information tells where the current selection is in terms of the currently selected cell in the main grid.

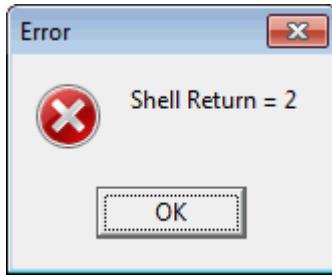
When that cell contains the name or the complete path to a file, then you should also see the complete path to that file and you will normally also see the date and time stamp for that file along with the file size in terms of the number of bytes in that file. Just look at the blue text to the left of the **Launch the Selected File** button.

When you click the button with the caption **Launch the Selected File** then the program will try to open or launch that file. For example, if the file is a text file or a **PDF** file then you should see that file opened in a text editor or in the Adobe Reader or Adobe Acrobat program. If the file is an image file then that file should be opened in an image editor program.

The controls in the above dialog work similar to those in the dialog that are used to View Images, with the main difference being that when you press the **Previous/Next** buttons, the program will change the selected file, and it will update the file information, however, the program does not try to do anything with the file until you press the button with the caption **Launch the Selected File**. Thus launching only works manually on one file at a time.

If the operating system cannot determine what to do with that file you may see an error message like that shown on the following page.

A **Shell Return** value less than 32 is an indication of an error. We don't try to interpret the error, but if you see any such message it just means your computer could not determine a default program that could open the file.



The dialog on the previous page is intended to be used with files that don't contain picture images, because another menu item under the **Utilities** menu displays picture images. If you do use this function to launch an image file, then rather than just display the image, this function should open the image in your default image editor program. Also, it would not make sense to try to open a **CSV** file with this function, so all **CSV** files will be ignored when you try to **Launch the Selected File**.

You can use a special function in the **Block Fill** dialog that will automatically fill a column of cells with filenames. Just select **Blocks | Block Fill...** and then click the button with the caption **Setup Regular Filenames Information....**

Setting the **Base Path Option** in this dialog gives you one of three choices. The assumption the program makes is that the grid column contains either just simple filenames, or it contains the complete filepath to the file. In the case of simple filenames you would normally select one of the first two **Base Path Options**. The program will automatically attach the beginning path to the filename in these two cases. **The Current Working Directory** is normally the last directory used that contained a **CSV** file that was loaded into the grid or saved from the grid.

If you choose to use the **Browse...** button to set a particular path, then you will see a regular file open dialog and you need only click on any filename as if you were going to open that file. Except of course the program will not open a file, but it will extract the complete path to that file and it will put that path in **The Fixed Base Path** edit box and it will automatically change the **Base Path Option** to the second choice.

If the cell does not contain the name of an existing file then the cell information may appear like one of the two following sets of information:

Row = 6 Column = 2
Empty Cell

or

Row = 6 Column = 2
Not A File ? = I am not a file

When either **Empty Cell** or the **Not A File ? =** line appears, then you should select another cell from the grid by clicking either the **Previous** or the **Next** buttons until you reach a cell that references a real file that has a possibility of being opened or launched. Either that, or you may need to change the **Base Path Option**.

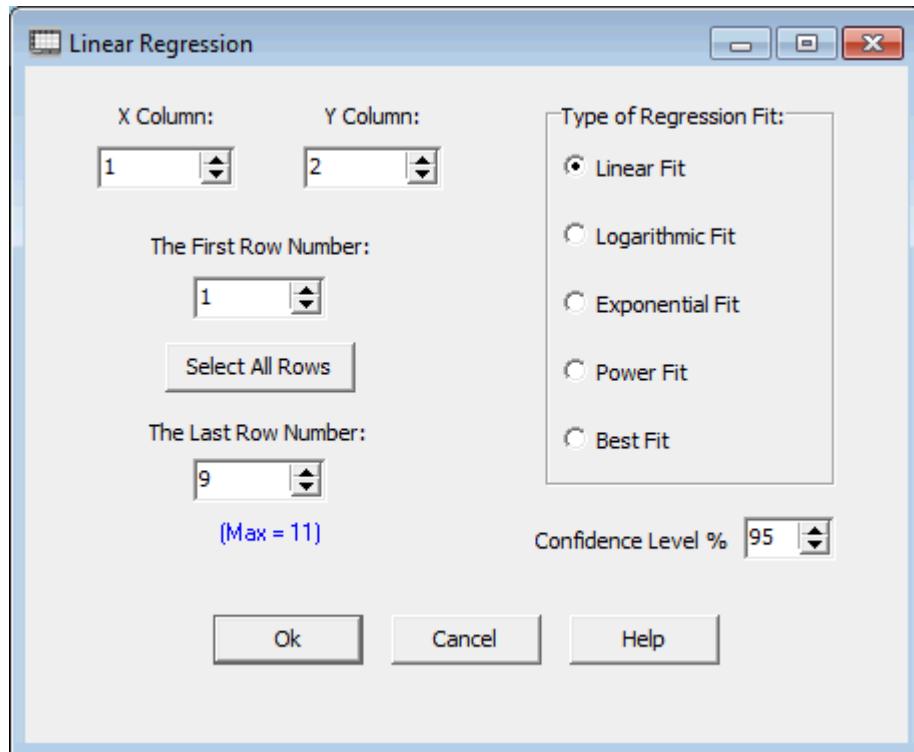
Linear Regression Report

The **CSV Editor** program has a function that allows you to analyze data to discover a mathematical relationship between two variables. The program derives an equation and produces a statistical report. The derived equation describes a mathematical curve that best fits the given data.

Before applying this function you need to have a grid that has two columns of data that we call the X-Data column and the Y-Data column. You need at least two rows of data, but normally you will have more than several rows. In fact, there is no limit on the maximum number of rows. All the data is assumed to consist of real numbers (floating point values) that can optionally be given in scientific notation. As a simple example, if you had the grid:

	Column 1	Column 2
<	X Data:	Y Data:
1	586	39906
2	601	39972
3	602	39983
4	607	40001
5	614	40015
6	619	40045
7	620	40053
8	632	40314
9	637	40348

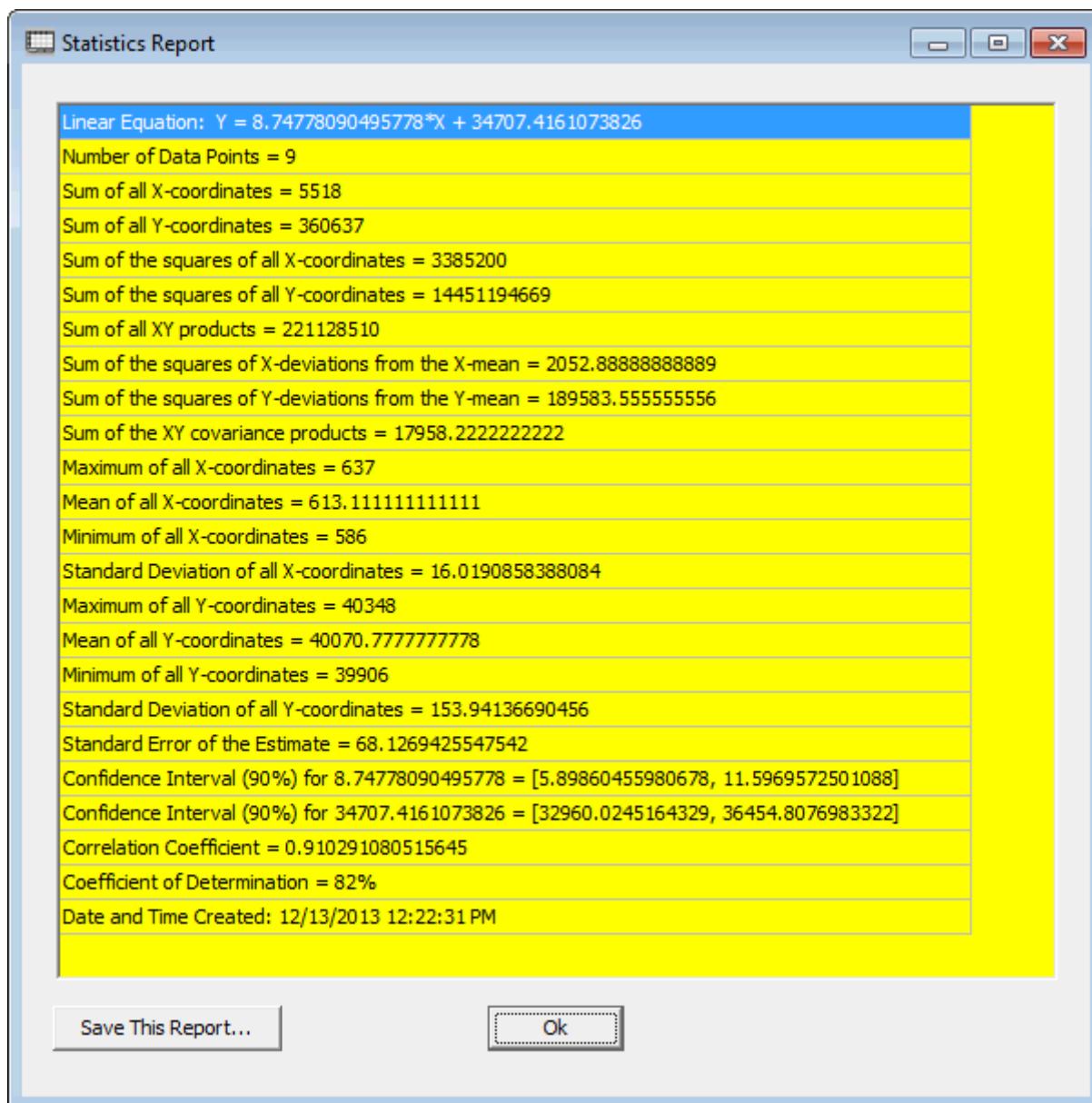
you could select the menu item **Utilities | Linear Regression Report...** and you will bring up the following dialog.



Using this dialog you should first identify the **X Column** and identify the **Y Column** and then select the appropriate range of rows. The **X** and **Y** columns can be any two columns in any order; they do not have to be adjacent columns. You have 5 choices for the **Type of Regression Fit**. The first four choices will force the type of expected equation to be one of **Linear** or **Logarithmic** or **Exponential** or **Power** equations. The 5th choice will let the program automatically determine the best type of equation.

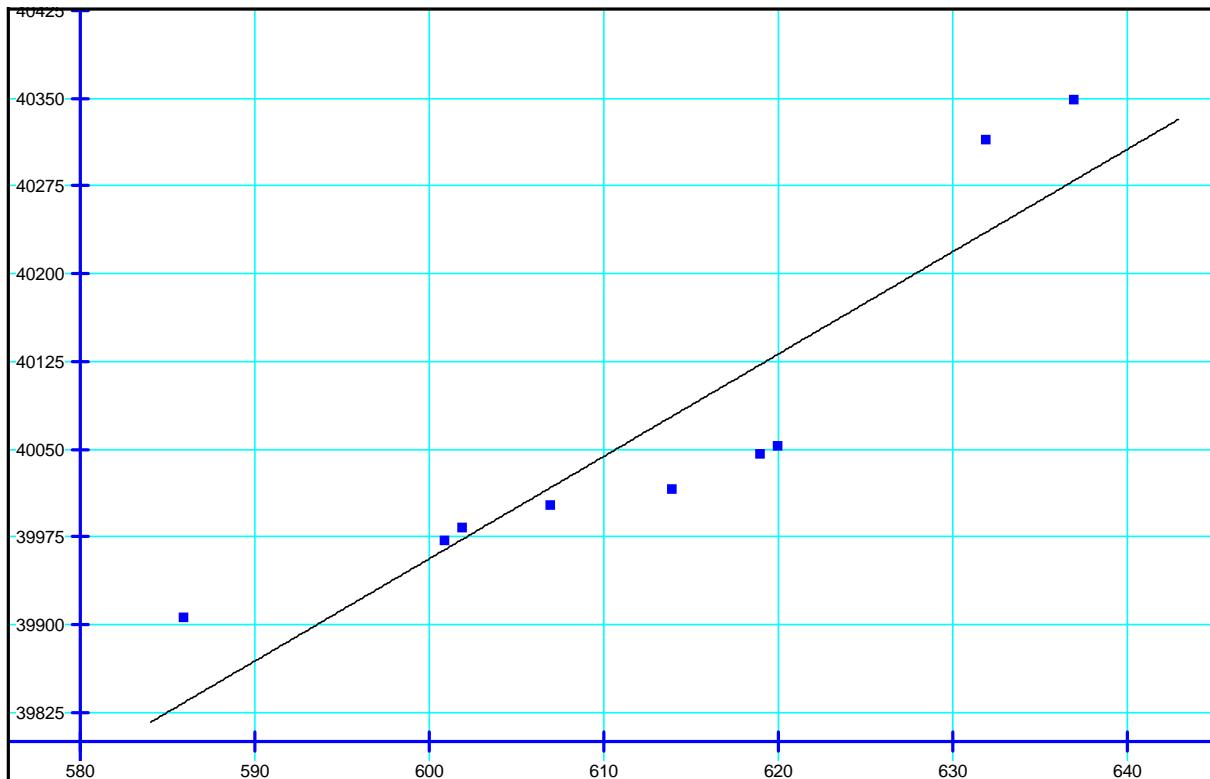
The final parameter is called the **Confidence Level %** and usually this integer value will be 90 or 95. This value is used to determine confidence intervals for the coefficients in the equation that this program generates. When the value is set at 90, it means that 90% of the time the true coefficients in the derived equation will lie within the intervals that are reported. See the two confidence intervals in the example report below. The valid range of integers is 1-99, but values between 90 and 97 are the norm.

An example type of report output is the following:



By clicking the button with the caption **Save This Report...**, you can save the results in a text file. Otherwise, when you click the **Ok** button the above dialog will close.

We won't explain the mathematics behind a typical report like this in this help file. In fact, our purpose here is not to explain anything about linear regression or the type and meaning of the statistics that are reported. We can however show a graph from another program. The graph below shows a plot of the data points as well as the line of best fit for the data. The equation of the line is what is shown in the first line of the report on the previous page. That equation is: $y = 8.74778090495778*x + 34707.4161073826$.



The above graph is called a scatter diagram of actual data taken from a particular computer system. The nine points in the graph are plotted from the data table given previously. The horizontal axis represents days of use of the computer while the vertical axis represents the number of files on that particular computer. For this example, the slope of the line is about 8.74 and this value could be interpreted as saying the number of files is growing on average at a rate that is a little under 9 files per day. We could also describe this same situation by saying the number of files on the computer is growing at a rate of about 3,200 files per year.

Just so you can have some idea of the kinds of curves the program can generate, below we show the four possible curve fit equations. The constants **A** and **B** are what the program computes based on your experimental or sample data. The **A** and **B** constants are the midpoints of the two related confidence intervals that appear in any statistical report.

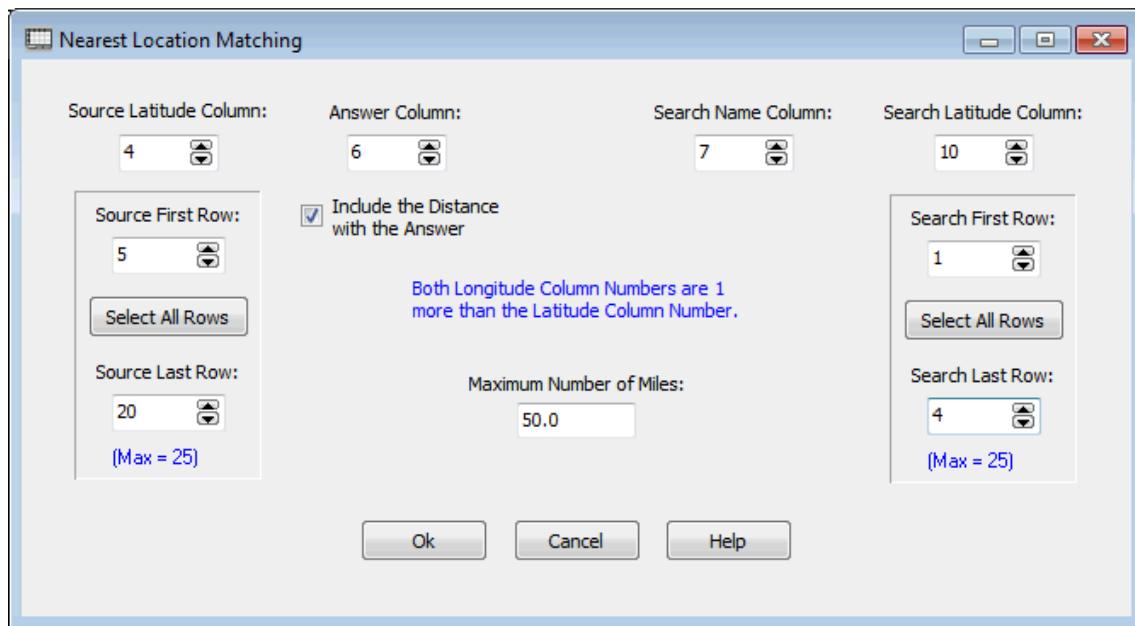
Linear Fit Equation: $Y = A*X+B$
Logarithmic Fit Equation: $Y = A*\ln(X) + B$
Exponential Fit Equation: $Y = A*(e^{(B*X)})$
Power Fit Equation: $Y = A*(X^B)$

Nearest Location Matching

A special menu item under the **Utilities** menu is named **Nearest Location Matching**. The purpose of this function is to search for and match one location that is nearest another location, based on the **GPS** coordinates of both locations. To illustrate a typical example, consider the following grid in which rows 1-25 and columns 1-6 contain a list of names and addresses for what we call **Source** persons living in southern California. Further assume the block contained in columns 7-11, and rows 1-4, is a list of four Technicians who service customers. The idea is to try and match each **Source** person with the name of the Technician that lives nearest that **Source** person. The list of Technicians is considered to be the **Search** list.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Source Name:	Address:	City:	Source Latitude:	Source Longitude:	Answer Column:	Search Name:	Street Address:	City:	Latitude:	Longitude:
1	Boone, Paula	953 Cole Ave	Los Angeles	34.0986333	-118.330128		Bob Crawford	13425 Washington Blvd.	Culver City	33.9917070	-118.4460640
2	Gray, Camila	2718 W Avenue 34	Los Angeles	34.1138841	-118.2358407		James Nelson	922 Gayley Ave.	Los Angeles	34.063090	-118.4479960
3	Meadows, Sheila	100 World Way	Los Angeles	33.9449481	-118.4005654		Jane Hartree	9149 South Sepulveda Blvd.	Westchester	33.9537110	-118.3963480
4	Burns, Clint	10011 Avalon Blvd	Los Angeles	33.94519500000001	-118.265532		Sheila Jackson	9245 Venice Blvd.	Los Angeles	34.02629790	-118.39419110
5	Leblanc, Edith	1007 N Western Ave	Los Angeles	34.0886995	-118.309546						
6	Britt, Mike	22030 Roscoe Blvd	Canoga Park	34.219274	-118.607119						
7	Holt, Ralph	6750 DE Soto Ave	Canoga Park	34.193382	-118.587984						
8	Snyder, Perry	130 E Sepulveda Blvd	Carson	33.807711	-118.273884						
9	Hardy, Russell	17455 Central Ave	Carson	33.8723058	-118.2494539						
10	Campos, Gordon	20315 Avalon Blvd	Carson	33.847118	-118.265221						
11	Reese, Kenneth	21836 Avalon Blvd	Carson	33.8291436	-118.2633744						
12	Baird, Julia	12701 Towne Center Dr	Cerritos	33.8720094	-118.0616051						
13	Haynes, Edith	133 Los Cerritos Mall	Cerritos	33.8656019	-118.0922543						
14	Woods, Ryan	20932 Devonshire St	Chatsworth	34.256823	-118.589573						
15	Morton, Camila	9185 DE Soto Ave	Chatsworth	34.2379233	-118.5890416						
16	Shannon, Lydia	1600 S Azusa Ave # 169b	City Of Industry	33.9935867	-117.9274765						
17	Barrett, Dean	17150 Gale Ave	City Of Industry	33.9986801	-117.932445						
18	Vincent, Clint	17951 Colima Rd	City Of Industry	33.9901366	-117.9125362						
19	Snyder, Hector	860 S Indian Hill Blvd	Claremont	34.0803216	-117.7185324						
20	Hill, Faith	101 W Compton Blvd	Compton	33.8963069	-118.2250263						
21	McLeod, Janice	1105 N Long Beach Blvd	Compton	33.903869	-118.2089797						
22	Carpenter, Renee	1733 S Alameda St	Compton	33.8782798	-118.217699						
23	Deleon, Harry	901 S Long Beach Blvd	Compton	33.8886391	-118.2070582						
24	Barnes, Hector	121 N Grand Ave	Covina	34.086592	-117.872909						
25	Brooks, Dorothy	1275 N Azusa Ave	Covina	34.1027076	-117.9095329						

To perform this matching, select **Utilities | Nearest Location Matching...** and when the dialog appears, assume you fill the entries so you have:



When you press the **Ok** button, the **Answer Column**, column 6, should get filled in so it looks as follows:

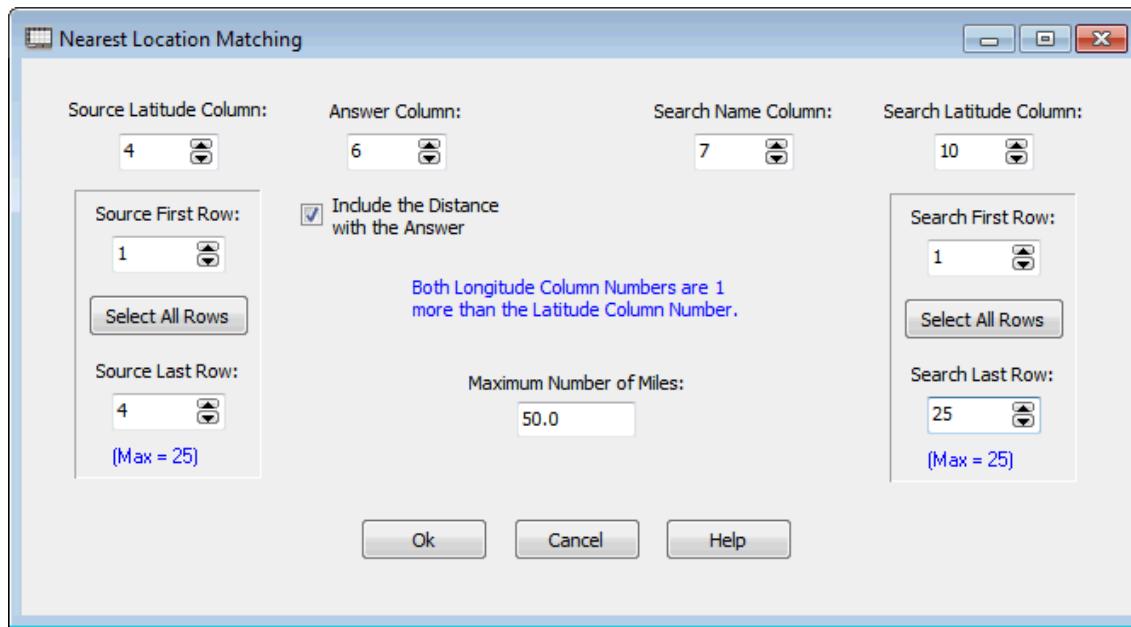
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Source Name:	Address:	City:	Source Latitude:	Source Longitude:	Answer Column:	Search Name:	Street Address:	City:	Latitude:	Longitude:
1	Boone, Paula	953 Cole Ave	Los Angeles	34.0886333	-118.330128		Bob Crawford	13425 Washington Blvd.	Culver City	33.9917070	-118.4460640
2	Gray, Camila	2718 W Avenue 34	Los Angeles	34.1138841	-118.2358407		James Nelson	922 Gayley Ave.	Los Angeles	34.063090	-118.4479960
3	Meadows, Sheila	100 World Way	Los Angeles	33.9449481	-118.4005654		Jane Hartree	9149 South Sepulveda Blvd.	Westchester	33.9537110	-118.3963480
4	Burris, Clint	10011 Avalon Blvd	Los Angeles	33.94519500000001	-118.265532		Sheila Jackson	9245 Venice Blvd.	Los Angeles	34.02623790	-118.39419110
5	Leblanc, Edith	1007 N Western Ave	Los Angeles	34.0886395	-118.309546	Sheila Jackson + 6.49					
6	Britt, Mike	22030 Roscoe Blvd	Canoga Park	34.219274	-118.607119	James Nelson + 14.13					
7	Holt, Ralph	6750 DE Soto Ave	Canoga Park	34.193382	-118.587984	James Nelson + 12.06					
8	Snider, Perry	130 E Sepulveda Blvd	Carson	33.807711	-118.273884	Jane Hartree + 12.31					
9	Hardy, Russell	17455 Central Ave	Carson	33.8723858	-118.2494539	Jane Hartree + 10.14					
10	Campos, Gordon	20315 Avalon Blvd	Carson	33.847118	-118.265221	Jane Hartree + 10.54					
11	Reese, Kenneth	21836 Avalon Blvd	Carson	33.8231436	-118.2633744	Jane Hartree + 11.51					
12	Baird, Julia	12701 Towne Center Dr	Cerritos	33.8720094	-118.0616051	Jane Hartree + 20.03					
13	Haynes, Edith	133 Los Cerritos Mall	Cerritos	33.8656019	-118.0922543	Jane Hartree + 18.49					
14	Woods, Ryan	20932 Devonshire St	Chatsworth	34.256823	-118.589573	James Nelson + 15.66					
15	Morton, Camila	9185 DE Soto Ave	Chatsworth	34.2379233	-118.5890416	James Nelson + 14.54					
16	Shannon, Lydia	1600 S Azusa Ave # 169b	City Of Industry	33.9935867	-117.9274765	Sheila Jackson + 26.86					
17	Barrett, Dean	17150 Gale Ave	City Of Industry	33.99866801	-117.9324445	Sheila Jackson + 26.54					
18	Vincent, Clint	17951 Colma Rd	City Of Industry	33.9901366	-117.9125362	Sheila Jackson + 27.73					
19	Snyder, Hector	860 S Indian Hill Blvd	Claremont	34.0803216	-117.7185324	Sheila Jackson + 38.90					
20	Hill, Faith	101 W Compton Blvd	Compton	33.8963069	-118.2250263	Jane Hartree + 10.60					
21	McLeod, Janice	1105 N Long Beach Blvd	Compton	33.9038693	-118.2089797						
22	Carpenter, Renee	1733 S Alameda St	Compton	33.8782798	-118.217699						
23	Deleon, Harry	901 S Long Beach Blvd	Compton	33.8886391	-118.2070582						
24	Barnes, Hector	121 N Grand Ave	Covina	34.086592	-117.872909						
25	Brooks, Dorothy	1275 N Azusa Ave	Covina	34.1027076	-117.9095329						

Because we limited the **Source** rows to be 5 through 20, only rows 5 through 20 have an answer in the answer column. The new contents of the answer column are such that when we look at the block in rows 5-20 and columns 1-6 we can discern which Technician is associated with the **Source** persons. The numerical values in the **Answer Column** represent the distance in miles with two decimal places of accuracy.

Our first example was such that the **Search** list had a much smaller length than the **Source** list. For our second example, we are going to change the initial grid so it looks as follows:

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Source Name:	Address:	City:	Source Latitude:	Source Longitude:	Answer Column:	Search Name:	Street Address:	City:	Latitude:	Longitude:
1	Bob Crawford	13425 Washington Blvd.	Culver City	33.9917070	-118.4460640		Boone, Paula	953 Cole Ave	Los Angeles	34.0886333	-118.330128
2	James Nelson	922 Gayley Ave.	Los Angeles	34.063090	-118.4479960		Gray, Camila	2718 W Avenue 34	Los Angeles	34.1138841	-118.2358407
3	Jane Hartree	9149 South Sepulveda Blvd.	Westchester	33.9537110	-118.3963480		Meadows, Sheila	100 World Way	Los Angeles	33.9449481	-118.4005654
4	Sheila Jackson	9245 Venice Blvd.	Los Angeles	34.02623790	-118.39419110		Burris, Clint	10011 Avalon Blvd	Los Angeles	33.94519500000001	-118.265532
5							Leblanc, Edith	1007 N Western Ave	Los Angeles	34.0886395	-118.309546
6							Britt, Mike	22030 Roscoe Blvd	Canoga Park	34.219274	-118.607119
7							Holt, Ralph	6750 DE Soto Ave	Canoga Park	34.193382	-118.587984
8							Snider, Perry	130 E Sepulveda Blvd	Carson	33.807711	-118.273884
9							Hardy, Russell	17455 Central Ave	Carson	33.8723858	-118.2494539
10							Campos, Gordon	20315 Avalon Blvd	Carson	33.847118	-118.265221
11							Reese, Kenneth	21836 Avalon Blvd	Carson	33.8231436	-118.2633744
12							Baird, Julia	12701 Towne Center Dr	Cerritos	33.8720094	-118.0616051
13							Haynes, Edith	133 Los Cerritos Mall	Cerritos	33.8656019	-118.0922543
14							Woods, Ryan	20932 Devonshire St	Chatsworth	34.256823	-118.589573
15							Morton, Camila	9185 DE Soto Ave	Chatsworth	34.2379233	-118.5890416
16							Shannon, Lydia	1600 S Azusa Ave # 169b	City Of Industry	33.9935867	-117.9274765
17							Barrett, Dean	17150 Gale Ave	City Of Industry	33.99866801	-117.932445
18							Vincent, Clint	17951 Colma Rd	City Of Industry	33.9901366	-117.9125362
19							Snyder, Hector	860 S Indian Hill Blvd	Claremont	34.0803216	-117.7185324
20							Hill, Faith	101 W Compton Blvd	Compton	33.8963069	-118.2250263
21							McLeod, Janice	1105 N Long Beach Blvd	Compton	33.9038693	-118.2089797
22							Carpenter, Renee	1733 S Alameda St	Compton	33.8782798	-118.217699
23							Deleon, Harry	901 S Long Beach Blvd	Compton	33.8886391	-118.2070582
24							Barnes, Hector	121 N Grand Ave	Covina	34.086592	-117.872909
25							Brooks, Dorothy	1275 N Azusa Ave	Covina	34.1027076	-117.9095329

Note that essentially we have traded the **Source** and **Search** persons from the first example. We have done this deliberately to show an example where the **Search** list is longer than the **Source** list. We now setup the dialog so it looks as shown on the following page.



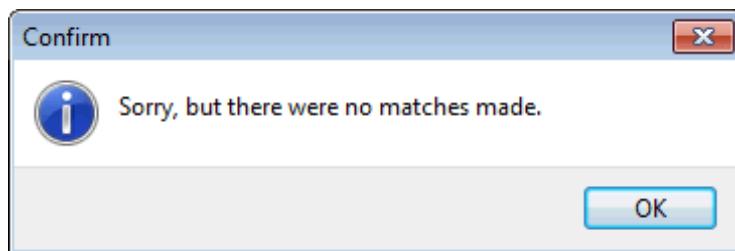
In this example we have selected all 25 rows for the range of the **Search Rows**. When you click **Ok** you should see the **Answer Column** gets filled in with just four answers as shown below.

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11
>	Source Name:	Address:	City:	Source Latitude:	Source Longitude:	Answer Column:	Search Name:	Street Address:	City:	Latitude:	Longitude:
1	Bob Crawford	13425 Washington Blvd.	Culver City	33.9917070	-118.4460640	Meadows, Sheila + 4.16	Boone, Paula	953 Cole Ave	Los Angeles	34.0886333	-118.330128
2	James Nelson	922 Gayley Ave.	Los Angeles	34.063090	-118.4479960	Boone, Paula + 6.98	Gray, Camilla	2718 W Avenue 34	Los Angeles	34.1138841	-118.2358407
3	Jane Hartree	9149 South Sepulveda Blvd.	Westchester	33.9537110	-118.3963480	Meadows, Sheila + 0.65	Meadows, Sheila	100 World Way	Los Angeles	33.9449481	-118.4005654
4	Sheila Jackson	9245 Venice Blvd.	Los Angeles	34.02629790	-118.39419110	Meadows, Sheila + 5.64	Burris, Clint	10011 Avalon Blvd	Los Angeles	33.94519500000001	-118.265532
5							Leblanc, Edith	1007 N Western Ave	Los Angeles	34.0886395	-118.309546
6							Britt, Mike	22030 Roscoe Blvd	Canoga Park	34.219274	-118.607119
7							Holt, Ralph	6750 DE Soto Ave	Canoga Park	34.193382	-118.587984
8							Snider, Perry	130 E Sepulveda Blvd	Carson	33.807711	-118.273884
9							Hardy, Russell	17455 Central Ave	Carson	33.8723858	-118.2494539
10							Campos, Gordon	20315 Avalon Blvd	Carson	33.847118	-118.265221
11							Reese, Kenneth	21836 Avalon Blvd	Carson	33.8291436	-118.2633744
12							Baird, Julia	12701 Towne Center Dr	Cerritos	33.8720094	-118.0616051
13							Haynes, Edith	133 Los Cerritos Mall	Cerritos	33.8656019	-118.0922543
14							Woods, Ryan	20932 Devonshire St	Chatsworth	34.256823	-118.589573
15							Morton, Camila	9185 DE Solo Ave	Chatsworth	34.2379233	-118.5890416
16							Shannon, Lydia	1600 S Azusa Ave # 169b	City Of Industry	33.9935867	-117.9274765
17							Barrett, Dean	17150 Gale Ave	City Of Industry	33.9986801	-117.932445
18							Vincent, Clint	17951 Colma Rd	City Of Industry	33.9901366	-117.9125362
19							Snyder, Hector	860 S Indian Hill Blvd	Claremont	34.0803216	-117.7185324
20							Hill, Faith	101 W Compton Blvd	Compton	33.8963069	-118.2250263
21							McLeod, Janice	1105 N Long Beach Blvd	Compton	33.903869	-118.2089797
22							Carpenter, Renee	1733 S Alameda St	Compton	33.8782798	-118.217699
23							Deleon, Harry	901 S Long Beach Blvd	Compton	33.8886391	-118.2070582
24							Barnes, Hector	121 N Grand Ave	Covina	34.086592	-117.872909
25							Brooks, Dorothy	1275 N Azusa Ave	Covina	34.1027076	-117.9095329

Of course this set of answers is different from what we had before. Neither Paula Boone nor Sheila Meadows were in the **Search Name** column in the first example search. They were in the **Source Name** column.

We should make a comment about the fact that nowhere in the dialog is there a way for you to enter a **Longitude** column for either the **Source** or the **Search** entities. The reason is that we assume the **Latitude** and **Longitude** columns will always be adjacent to each other with the **Latitude** appearing on the left and the **Longitude** appearing on the right. This explains why you only enter **Latitude** columns for both the **Source** and **Search** entities. As the instruction in the dialog says, each **Longitude** column number is always exactly 1 more than the corresponding **Latitude** column number.

You can set the **Maximum Number of Miles** that the search function uses. The default value is 50.0 miles. When the search is tried, it may happen that no matches are found within that number of miles. If that happens you will see the message:



If you see this message then you can try increasing the **Maximum Number of Miles** and run the function again. You should not set this number to anything less than 1.0.

Your **Search** list entities should have unique names, because if the names are not unique, then you can't really tell who the exact match is with.

Under the hood, when the program tries to find a match, it actually begins with one **Source** item at a time. Then it runs through all the items in the **Search** range of rows that you have selected, and it tries to find any matches within a 1-mile radius of the **Source** item. If no matches are found the program will repeat the search but with a 2-mile radius. In fact, the program continues searching as long as no matches are found by increasing the radius by 1 mile at a time for each new scan until you exceed the **Maximum Number of Miles**.

It is also possible that no matter the current search radius, there could be several matches all within that radius. When that is the case the program will always select the **Search** item that is closest. This is why this function has been given the name **Nearest Location Matching**.

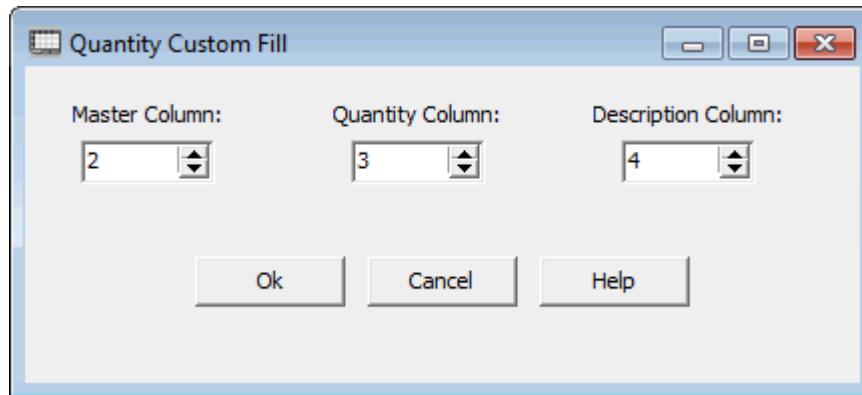
This function is unusual in that it really uses two separate blocks of information all contained within a single grid. We always consider the **Answer Column** to be part of the **Source** information. Of course your actual data will differ from our examples. Items that are either a **Source** or that are **Search** items don't always have to be people. The **GPS** coordinates can refer to business locations or National Parks or any other two entities that differ by a certain geographical distance.

It is an option to add the distances we compute to the answers in the **Answer Column**. We implicitly assume that all entities are somewhere on the surface of the earth. Internally we compute the great circle arc distance between any two points on the earth. We know that the great circle distances we compute are not the same as actual travel routes determined by real roads or human made trails. However, in almost all cases our distance calculations and comparison functions should be more than adequate. The distances shown are calculated in miles and are accurate to the nearest hundredth of a mile.

The reason for writing a **+** sign before the distance is in case you wish to split the **Answer Column** into 2 columns. You can use the **+** sign as the split character and after splitting you can apply the trim function to both columns.

Quantity Custom Fill

A special menu item under the **Utilities** menu is named **Quantity Custom Fill**. When you select the menu item **Utilities | Quantity Custom Fill...** you will bring up the following dialog:



To explain the purpose of this dialog, we assume you have first opened a grid such as that shown below.

<	Column 1	Column 2	Column 3	Column 4
>	Date:	Master Product Line:	Quantity:	Item Description:
1	8/23/2013	TempleTouch		10" Smart Screen
2	8/23/2013	TempleTouch		10" Smart Screen
3	8/23/2013	TempleTouch		10" Smart Screen
4	8/23/2013	TempleTouch		10" Smart Screen
5	8/30/2013	TempleTouch		MDT 860
6	8/30/2013	TempleTouch		RFID Reader
7	8/30/2013	TempleTouch		Smart CPU
8	8/30/2013	TempleTouch		Smart CPU
9	8/30/2013	TempleTouch		Smart CPU
10	05/01/2013	Ghost Software		598U
11	7/22/2013	Phoenix Flight		8800
12	7/22/2013	Phoenix Flight		8800
13	7/22/2013	Phoenix Flight		8800
14	7/22/2013	Phoenix Flight		8800
15	7/22/2013	Phoenix Flight		8800
16	9/18/2013	WORLD Music		8800
17	9/18/2013	World Music		8800
18	9/18/2013	World Music		8800
19	7/3/2013	World Music		8800
20	7/3/2013	World Music		8800
21	7/3/2013	World Music		8800
22	7/3/2013	World Music		8800
23	10/11/2013	Application Nero		8" Res Screen
24	10/11/2013	Application Nero		8" Res Screen
25	7/3/2013	Application Nero		8" Res Screen
26	8/14/2013	Application Nero		8" Res Screen
27	5/30/2013	Application Nero		8" Res Screen
28	06/27/2013	Application Nero		8" Res Screen
29	8/26/2013	Application Nero		8" Res Screen
30	7/3/2013	Application Nero		8" Res Screen
31	06/27/2013	Application Nero		Snap 1
32	10/1/2013	Application Nero		Snap 1
33	8/14/2013	Application Nero		Snap 1

If we were to fill out the dialog as shown on the previous page, the program would change the grid on that page so it would look like the following grid. The Quantity column is what has been partially filled with counts of the number of different items in Column 4. Each new quantity number is filled in the first row that changes to a new item description. These quantity counts also reset each time the cell in Column 2 changes. Now you should begin to understand that this function operates somewhat like a master-detail level in one table. For this example the Master list is in Column 2 while the details are in Column 4.

<	Column 1	Column 2	Column 3	Column 4
>	Date:	Master Product Line:	Quantity:	Item Description:
1	8/23/2013	TempleTouch	4	10" Smart Screen
2	8/23/2013	TempleTouch		10" Smart Screen
3	8/23/2013	TempleTouch		10" Smart Screen
4	8/23/2013	TempleTouch		10" Smart Screen
5	8/30/2013	TempleTouch	1	MDT 860
6	8/30/2013	TempleTouch	1	RFID Reader
7	8/30/2013	TempleTouch	3	Smart CPU
8	8/30/2013	TempleTouch		Smart CPU
9	8/30/2013	TempleTouch		Smart CPU
10	05/01/2013	Ghost Software	1	598U
11	7/22/2013	Phoenix Flight	5	8800
12	7/22/2013	Phoenix Flight		8800
13	7/22/2013	Phoenix Flight		8800
14	7/22/2013	Phoenix Flight		8800
15	7/22/2013	Phoenix Flight		8800
16	9/18/2013	World Music	7	8800
17	9/18/2013	World Music		8800
18	9/18/2013	World Music		8800
19	7/3/2013	World Music		8800
20	7/3/2013	World Music		8800
21	7/3/2013	World Music		8800
22	7/3/2013	World Music		8800
23	10/11/2013	Application Nero	8	8" Res Screen
24	10/11/2013	Application Nero		8" Res Screen
25	7/3/2013	Application Nero		8" Res Screen
26	8/14/2013	Application Nero		8" Res Screen
27	5/30/2013	Application Nero		8" Res Screen
28	06/27/2013	Application Nero		8" Res Screen
29	8/26/2013	Application Nero		8" Res Screen
30	7/3/2013	Application Nero		8" Res Screen
31	06/27/2013	Application Nero	3	Snap 1
32	10/1/2013	Application Nero		Snap 1
33	8/14/2013	Application Nero		Snap 1

Column 3 is simply the column that will receive the counts.

In general, you will first need to perform a multiple column sort, using what we are calling the Master and the Description columns, in that order to perform the sort.

This function depends on doing the multiple column sort before you apply this function.

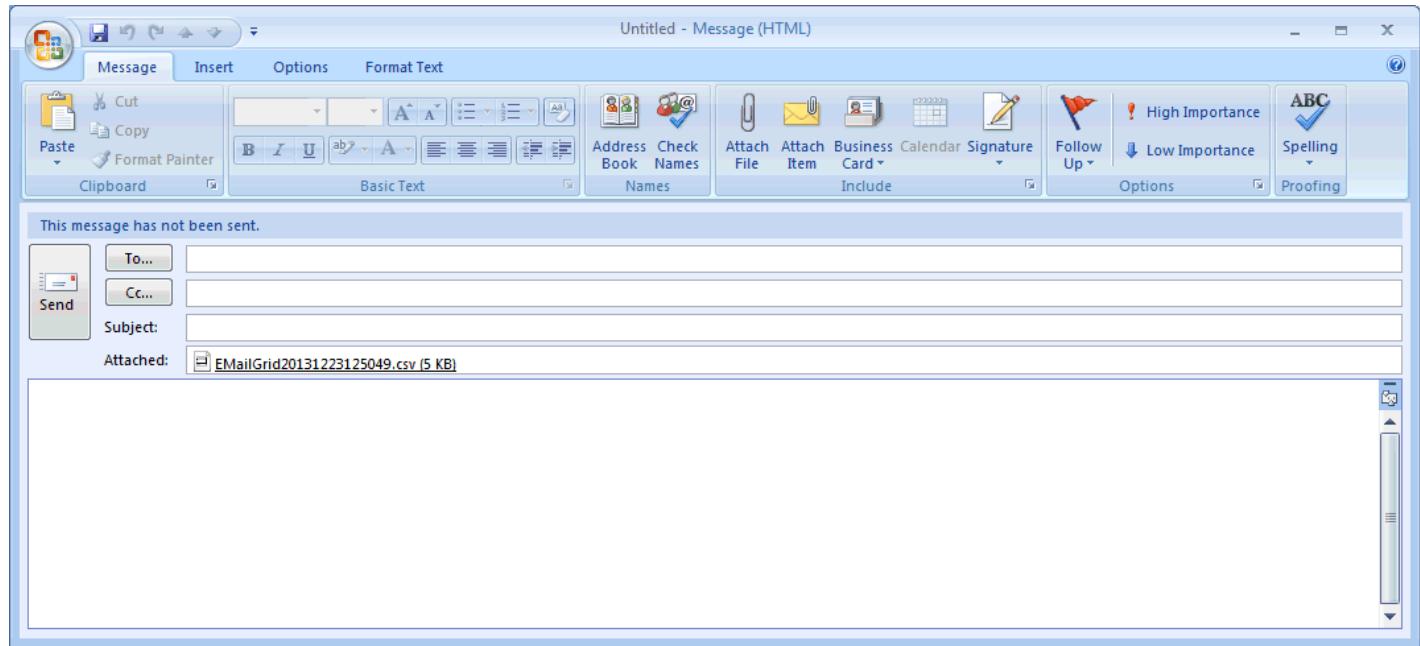
We recognize the very special nature of this function. Not everyone will have their data organized like that shown in the above table, and even if you do have data like this, you might not have a need to create the counts.

We should also say that Column 1 is not used by this function, but the dates in that column might represent dates when transactions were made. In fact, our example table assumes there is one transaction per line in the grid, and this special counting function can be used to summarize how many transactions there were for each unique description that combines the Master and Description columns.

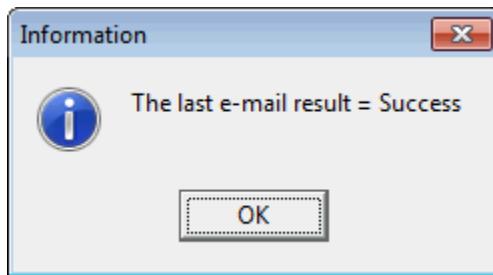
Sending the Grid in an e-mail Message

You can send the current grid as an attached file in an e-mail message at any time by choosing **Utilities | Send Grid in an e-mail Message....**

You should then see your e-mail application open. The current grid should already show as an attachment with the filename **EMailGridYYYYMMDDHHMMSS.csv** where **YYYYMMDD** and **HHMMSS** will be a series of digits in the range from 0-9 that represent the date (year month day) and the time (hours minutes seconds) when you started executing this function. An example Outlook e-mail application might appear like the following.



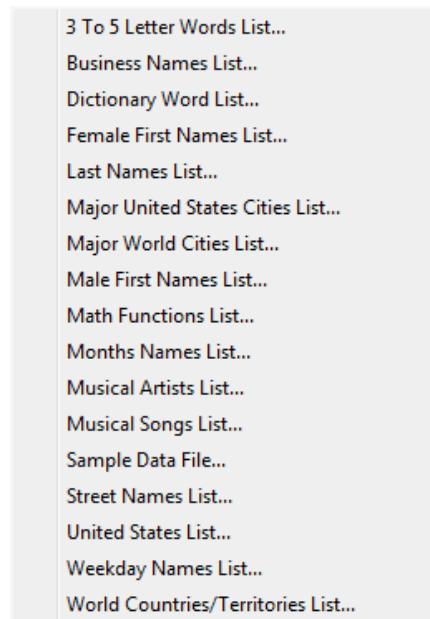
You should fill out the **To...** e-mail address information was well as the **Subject:** information. You can type in any text you want to complete the message. When you are finished, click the **Send** button. You should then see a short confirmation message like the following:



All of the functionality is dependent on how you have setup your e-mail application. We can suggest that you try using Microsoft Outlook. If you haven't already configured Outlook, then you can go into the Windows control panel and select the Mail function to setup Outlook. We are unable to supply any more configuration information, but you if have trouble setting this up and making it work you should know we are sympathetic.

Showing Sample Data Lists

Under the **Utilities** menu is a submenu **Show a Sample Data List ▶** that allows you to view various lists that are distributed with the **CSV Editor** program. The choice of lists are shown below.



The titles of these menu items are pretty self-explanatory. We will just mention that these viewing functions are only good for simply viewing the lists. An example of displaying a sample list is the following:

List of Major Cities in the World					
	City:	Country:	Country Region:	Latitude:	Longitude:
1	Abidjan	Cote d'Ivoire		5.341216	-4.027863
2	Abu Dhabi	United Arab Emirates		24.4734	54.367218
3	Accra	Ghana		5.559265	-0.196381
4	Addis Ababa	Ethiopia		9.040393	38.746033
5	Ahmadabad	India		23.040561	72.566071
6	Aleppo	Syria		36.216579	37.159538
7	Alexandria	Egypt		31.200468	29.917831
8	Algiers	Algeria		36.752914	3.042054
9	Almaty	Kazakhstan		43.257206	76.912537
10	Amman	Jordan		31.956641	35.945635
11	Amsterdam	The Netherlands		52.371826	4.894409
12	Ankara	Turkey		39.927642	32.853241
13	Antananarivo	Madagascar		-18.910834	47.530975
14	Antwerp	Belgium		51.222368	4.402771
15	Ashgabat	Turkmenistan		37.934992	58.366928
16	Asmara	Eritrea		15.333194,	38.926277
17	Astana	Kazakhstan		51.173316	71.433105
18	Asunción	Paraguay		-25.281023	-57.634964
19	Athens	Greece		37.984528	23.728924
20	Atlanta	United States	Georgia	33.750606	-84.388733
21	Auckland	New Zealand		-36.84556	174.763641
22	Baghdad	Iraq		33.326512	44.422531
23	Baku	Azerbaijan		40.383951	49.842911

Ok

If you want to edit any of these lists, to add new row elements, we caution you not to change the filenames of the pre-built lists and not to change their format in terms of the number of columns of data. All such lists can be loaded as regular **CSV** editor files and then they can be edited and/or printed. These files cannot be edited or printed in the viewing dialog. The viewing dialog is only for viewing.

The size of this dialog box dynamically changes as you change to view different lists. Many of the lists contain only one or two columns. Also note the caption of the entire dialog gives information about the list you are currently viewing.

Here is another list view that contains unique street names. This particular list has over 1,000 street names and the view below is somewhere in the middle of that list.

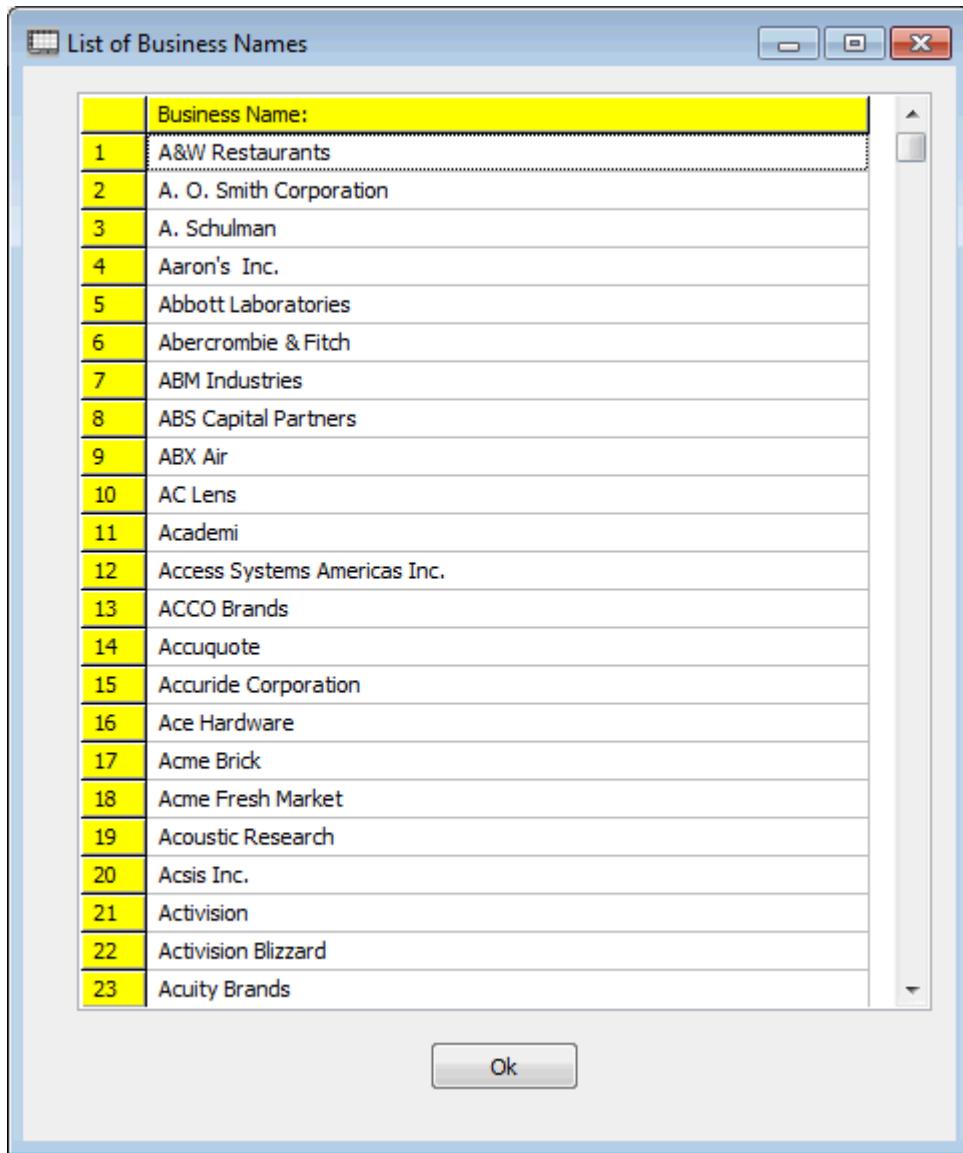
	Street Names:
613	Meadow Pond
614	Meadow Ridge
615	Meadow Run
616	Meadow Wood
617	Megan
618	Mellon
619	Mercury
620	Meridian
621	Merrivale
622	Miami
623	Michael
624	Middlefield
625	Midlothian
626	Milan
627	Mill
628	Mill Run
629	Mill Valley
630	Miller
631	Millpond Run
632	Mills
633	Minorca
634	Miramar
635	Miria

The list below shows world country names and territories where the names have been ranked in terms of geographical size.

List of World Countries/Territories		
	Size Rank:	Country Name or Territory Name (Owner):
1	1	Russia
2	2	Antarctica
3	3	Canada
4	4	China
5	5	United States
6	6	Brazil
7	7	Australia
8	8	India
9	9	Argentina
10	10	Kazakhstan
11	11	Algeria
12	12	Democratic Republic of the Congo
13	13	Greenland (Denmark)
14	14	Saudi Arabia
15	15	Mexico
16	16	Indonesia
17	17	Sudan
18	18	Libya
19	19	Iran
20	20	Mongolia
21	21	Peru
22	22	Chad
23	23	Niger

Ok

Yet another list is the list of business names shown below. This list contains over 1500 names of businesses that can be found in the United States. We certainly don't claim this list has the names of all businesses, but you should be able to find the names of most major businesses. A few of these names are simulated and the list is not intended to list real businesses only. If you need to generate random data with business names then you should load the file [BusinessNamesList.csv](#) into the dialog used to **Setup a Random Data Pick List** after you select the function **Blocks | Block Fill....**



Another customized list is intended to be used as part of Help. This is a list of the built-in mathematical functions that can be used when you enter any math expression. The list has two columns, the first of which contains a description of the function and the second shows the exact syntax for invoking the function. The corresponding file is named [MathFunctionsList.csv](#). There are 61 built-in functions in this list.

There are two lists named **Musical Artists List** and **Musical Songs List**. These are mostly Rock and Roll artists and songs from the 50's 60's and 70's, but there are a few other artists and genres and other types of music present. Neither list is intended to be uniform or exhaustive. Both lists can be considered somewhat eclectic. There are over 3100 songs and over 1000 artists. Just don't be surprised when you see something you think is out of place. We like having lists of real people and real things that might be recognized or not. Below we show the beginning of the **Musical Songs List**, where the list is sorted by song titles.

List of Musical Songs

	Song Title/Abbreviation:	Performer or Group or Author:
1	123	Len Barry
2	16 Candles	Crests
3	1999	Prince
4	19th Nervous Breakdown	Rolling Stones
5	2000 Light Years From Home	Rolling Stones
6	25 Or 6 To 4	Chicago
7	26 Miles	Four Preps
8	35 Of A Mile In 10 Seconds	Jefferson Airplane
9	409	Beach Boys
10	422am	Mystic Moods
11	50 Ways To Leave Your Lover	Simon and Garfunkel
12	500 Miles	Peter Paul Mary
13	500 Miles	Kingston Trio
14	5d	Byrds
15	7 O'clock News	Simon and Garfunkel
16	9 To 5	Dolly Parton
17	96 Tears	Question Mark
18	98.6	Keith
19	A B C	Jackson 5
20	A Banda	Herb Alpert
21	A Big Hunk Of Love	Elvis Presley
22	A Blossom Fell	Nat King Cole
23	A Boy Named Sue	Johnny Cash

Ok

We should make some comments about the contents of the above list. First, the strings in both columns are customized in that both the names of songs and the names of performers or groups can sometimes be different than what you expect. Mostly both song titles and artists names are written in a convenient brief form. The names should not be expected to be either consistent or correct in all cases.

As an example, the band officially named **Creedence Clearwater Revival** is more simply named by us as **Creedence**. For the band that was originally named **Jefferson Airplane**, we kept that name for all its songs, even though the group later became known as **Jefferson Starship**, a name we never really liked.

You may see some songs by the group named the **Jackson 5** and see other songs with the name **Michael Jackson**. Whether we use a group name or an individual artist's name may seem somewhat arbitrary. We don't always care when a leader left a band and became a star on their own. As another example, most songs by **Simon and Garfunkel** show both artists names, but the song **Kodachrome** is an exception where the artist is shown as **Paul Simon**.

For most of the songs by the **Beatles** we show that group name, but songs done by **Paul McCartney** when he was with **Wings** will show the group name **Wings**. We claim neither consistency nor correctness in all cases. Sometimes we just chose the name or the abbreviation that first came to our mind. Also note that we have removed apostrophes and quotes and parentheses and all punctuation marks from all names. This was done primarily to avoid operating system filename conflicts with actual MP3 files that we own.

The above list is sorted by song titles but you could also sort the list by artist when you wish to see what songs are included by a particular artist. We don't include all songs by all artists; we usually only include the most popular songs. To sort the list by the artist's names you should just manually load the regular **CSV** file:

[**MusicalSongsList.csv**](#).

Then select **Columns | Sort Using Multiple Columns...** and choose Column 2 as the first sort column and then choose Column 1 as the second sort column.

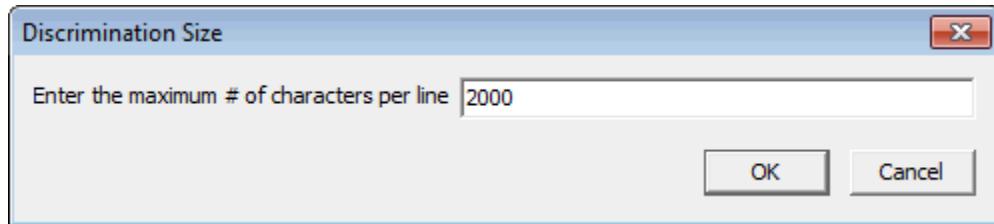
You can add your own music or delete some of ours. You will probably find the list is very limited for songs after about 1972. If you don't find your favorite artist or favorite song, don't be disappointed. The songs and artists that were selected were just the ones we could easily find and the ones we had in our personal music collection. Of course, there may be artists or songs you never heard of because the list just contains things we had in our own personal database.

There are two lists that deal with words. The first is a list of words that consist of between 3 and 5 letters. This list contains over 13,000 words and is in the file named [**Three2FiveLetterWordsList.csv**](#). A larger list contains what we call a list of dictionary words. This larger list has over 42,000 words in the file named [**DictionaryWordList.csv**](#). You could use either list whenever you need to generate some random words. Just load these files into any Pick List. We use the smaller list when we generate passwords, although we also mix the case of those words at random and we also randomly sprinkle special characters in the passwords that we generate. See the function **Utilities | Generate a List of Random Passwords....**

Splitting a File By Line Size

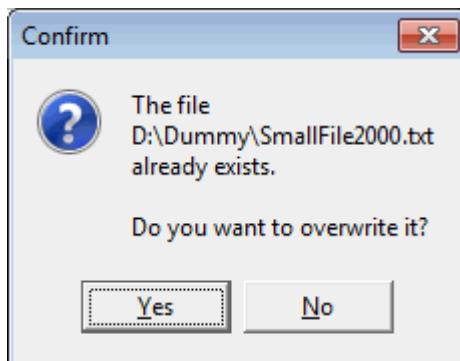
There may be times when you need to split a text file (a **CSV** file) according to the size of the lines of text that exist in that file. This may be especially true when the file may be a mal-formed **CSV** file. When you select the menu item **Utilities | Split a File By Line Size...**, you will be immediately prompted with a standard file open dialog in which you need to select the file that is to be split into two other newer files. The original file that is selected will remain unchanged by this special function.

Next, the program will prompt you to enter the maximum number of characters you expect to see in a typical valid line from the selected file. For example, you may be prompted with a dialog like the following:



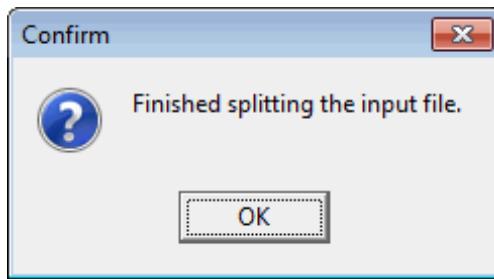
After entering the maximum number of characters you expect to see in a typical line in the selected file, and selecting **Ok**, the program will immediately open that file and it will read that file line by line. All lines that are read will be placed into either of two output text files that have automatically-formed names. Those names will be like **OutputSmall2000.txt** or **OutputLarge2000.txt**. The number in the filename, like **2000** in this example, is the same number you enter in the above dialog box. For this example, any line that is read that has more than **2000** characters in the line will be output to the file with the name **OutputLarge2000.txt**. Any line that is read that has **2000** or fewer characters in that line will be output to the file named **OutputSmall2000.txt**. Of course in a different example the number will be different from **2000**.

Since the two filenames are automatically formed, there is a chance that you may need to overwrite any existing file that has the same name as one of the two output files. When that is the case you will see one or two additional prompts like the following:



If you select the **No** button it means the entire operation will be aborted and no new files will get created. If you select the **Yes** button it means you intend to overwrite the named file. Of course you won't see a prompt like that shown above if there is no output filename conflict.

If you don't abort the operation, then you should see a final confirmation message that appears as:



After the two output files are created you can perform further operations on those two files as needed.

More often than not, the mal-formed lines will all be in the file named like **OutputLarge2000.txt**, for this example.

See also the next topic that may help you identify a mal-formed **CSV** file.

Suspicious CSV File Column Counts Report

Under the **Utilities** menu is a menu item that says **Suspicious CSV File Column Counts Report....**

This function should be used when you have a file that may be a mal-formed **CSV** file. The program will first prompt you to select the suspicious **CSV** file, and then it will prompt you for the report file that will be saved. That second report file is itself another **CSV** file, and that report file will be opened or automatically loaded into this **CSV Editor** program when this menu function finishes. A typical Column Report file might look like the following:

<	Column 1	Column 2
>	D:\CSVEditor\AcademyAwards.csv	Column Counts:
1	Line 1	6
2	Line 2	6
3	Line 3	6
4	Line 4	6
5	Line 5	6
6	Line 6	6
7	Line 7	6
8	Line 8	6
9	Line 9	6
10	Line 10	6
11	Line 11	6
12	Line 12	6
13	Line 13	6
14	Line 14	6
15	Line 15	6
16	Line 16	6
17	Line 17	6
18	Line 18	6
19	Line 19	6
20	Line 20	6
21	Line 21	6
22	Line 22	6

The Column Report file always has a header line in which Column 1 contains the filepath of the suspicious **CSV** file that was first selected, and Column 2 always shows the header title **Column Counts:**. The remainder of the report will show for each Line in the suspicious file how many columns of data that line actually has. In the above example, the suspicious file was the **AcademyAwards.csv** file in which each of its 22 lines contained 6 columns of data.

By quickly looking down the second column you can get some sense of just how mal-formed the suspicious **CSV** file really is. If all the column counts are the same then at least you know the format is consistent.

If you have a very large suspicious **CSV** file, then it may be worthwhile to sort the second column of the report. Just remember when you specify the Sort operation that the second column contains numbers, not regular text. If you see a lot of different numbers in the second column then you will have some idea of just how mal-formed the **CSV** file really is. Running this column report function is more useful than just trying to open a mal-formed **CSV** file. This is especially true if trying to open the file generates error messages over which you have no control.

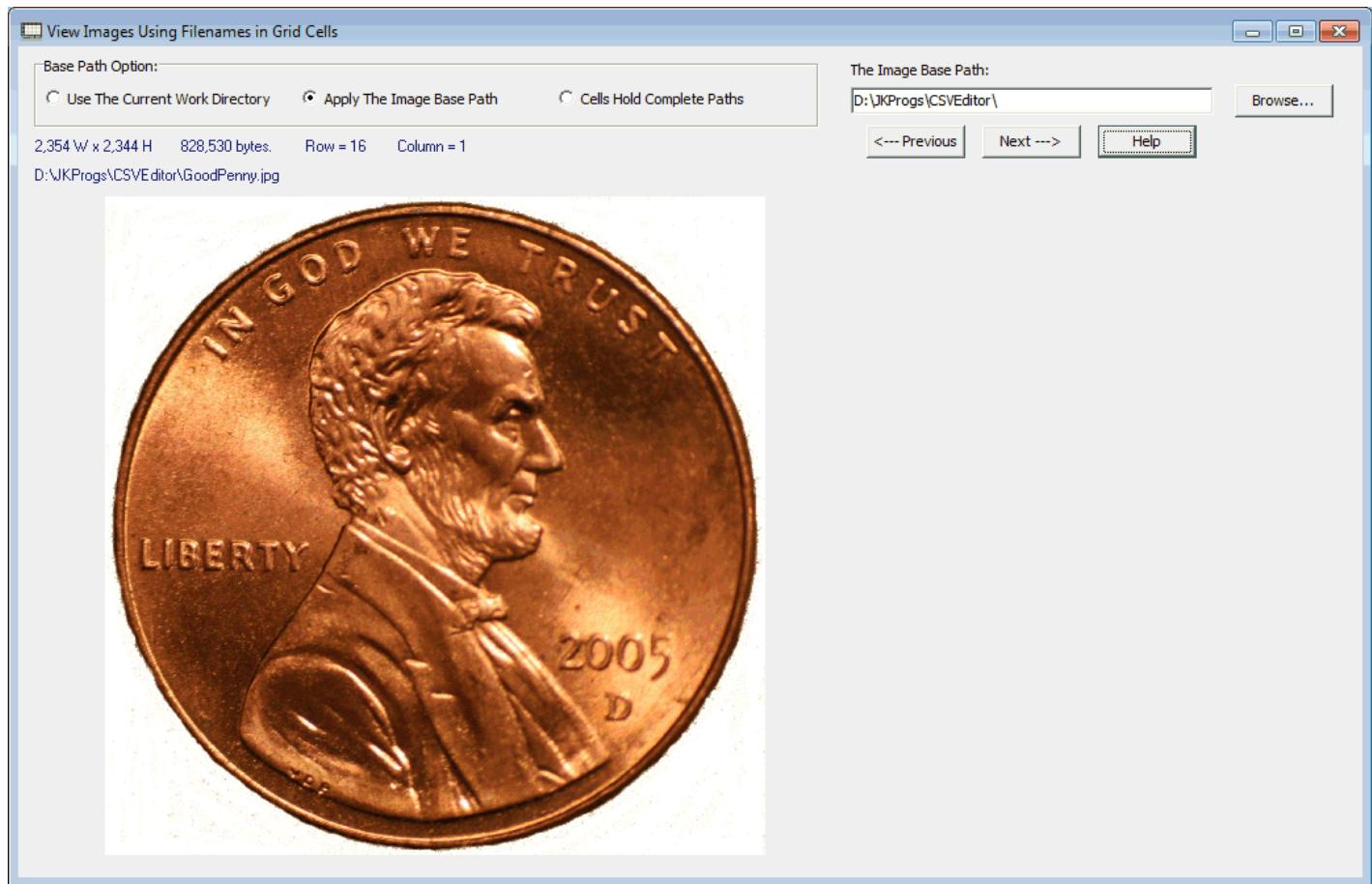
In severe cases it will be necessary to first edit the mal-formed **CSV** file using a text editor like **Notepad** (or **TextPad**) to make manual corrections to mal-formed lines. Then you can try making another column report, or you could try to open the **CSV** file directly after you have manually corrected all the errors.

See also the list of fifteen [**Rules For Creating CSV Files**](#) at the end of this help file to help you understand the many things that can be wrong with any mal-formed **CSV** file.

Viewing Images

A menu item under the **Utilities** menu has the title **View Images Using Grid Cell Filenames...** This brings up a dialog similar to that shown below that can be used to display picture images.

The first thing to learn before you open this dialog is the meaning of the **Base Path Option** and its relationship with the grid cell contents. The intention here is that some column in your grid contains a list of image filenames. Before opening the dialog, you should normally click on any cell in that column that has the name of a file that contains a picture. This program displays only images contained in files that are either ***.BMP** or ***.GIF** or ***.JPG** or ***.PNG** files. These are the only four image types the program uses. Then as you click the **Previous** and **Next** buttons as shown in the dialog below, you should see your picture images displayed, one at a time. The program will move the grid selection up and down within the chosen column.



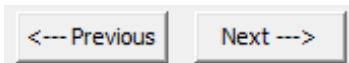
The blue text shown above any image gives the width and height of the image in pixels, and gives the file size in bytes, and it shows the grid **Row** and **Column** position and it shows the complete path to the displayed file.

If you choose to use the **Browse...** button to set a particular path, then you will see a regular file open dialog and you need only click on any filename as if you were going to open that file. Except of course the program will not open a file, but it will extract the complete path to that file and it will put that path in **The Image Base Path** edit box and it will change the **Base Path Option** to the middle choice.

We make one of two assumptions about the file names and their associated file paths. We assume all your files are in either one single directory, or we assume they can reside anywhere on your computer. For the single directory case we assume that directory is either the current working directory of the **CSV Editor** program itself, or it can be any other subdirectory whose path you manually type in the edit box in the above dialog that is **The Image Base Path**. **The Current Work Directory** is normally the last directory used that contained a CSV file that was loaded into the grid or saved from the grid.

You must select either the first or second radio button in the **Base Path Option** radio group to select either of the single directory options. If you select the third radio button with the caption **Cells Hold Complete Paths** then we assume each cell in the selected column contains the complete path to the file that will be displayed. In this case the image files don't have to all be in one subdirectory.

Before you open the menu item **Utilities | View Images Using Grid Cell Filenames...** you should first click on the first cell that contains an image filename or complete file path. Then when the above dialog opens all you need do is click the two buttons that appear as:



The program will automatically select the **Previous/Next** cell that is up or down in the selected column and the program should then display the image named in the selected cell. When the selections reach the bottom or the top of a column, they will automatically wrap around and continue with the first or last cell in the given column.

If a given cell is blank, or does not name a proper image file, then no image will be displayed and you may see a message like:



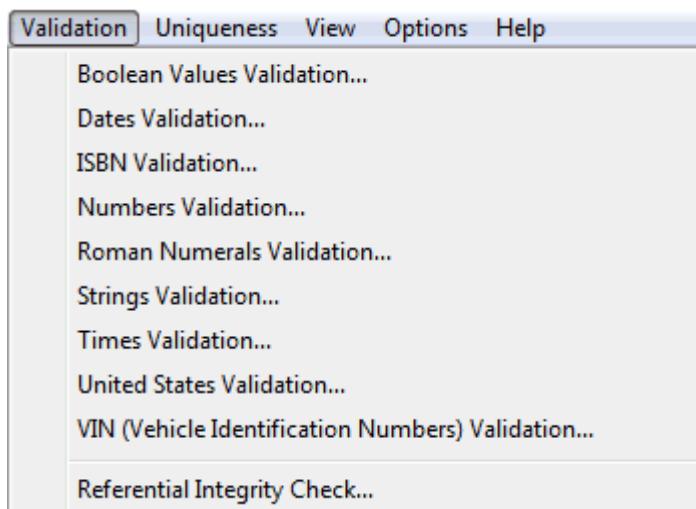
An error message like the second one means the file did not contain an image the program could work with, or, perhaps you may need to change either **The Image Base Path** to make a correct path, or you may need to change the **Base Path Option** to another choice.

The only purpose of this dialog is to help you visualize file images when a column of cells are used to name existing image files. You can only display one file at a time. The size of the displayed image is determined by the program and cannot be customized. However, the image will fill the space that is available in the dialog. In other words, each image will stretch or shrink to automatically fill the space available. The aspect ratio of the original image should be maintained as the image is displayed.

The above dialog box is modeless which means you could move it to one side and then edit grid data while it is still showing. For example, you could click on another grid column to start working with that column. You could also edit or change the viewing parameters such as change the **Base Path Option** or edit **The Image Base Path**. To close the dialog box you should click the upper-right corner close button .

You can use a special function in the **Block Fill** dialog that will automatically fill a column of cells with image filenames. Just select **Blocks | Block Fill...** and then click the button with the caption **Setup Image Display Filenames....**

The Validation Menu



The **Validation** menu contains a few menu items that are concerned with testing your data and verifying that it meets certain specifications. Normally you should format your data before you verify it. Validation does not change your data in any way, but it will point out potential weaknesses and inconsistencies in your data. For example, all strings in a column may be validated against a set of master strings to insure the data strings all have consistent spelling and form. Strings like **JANUARY**, and **January**, and **Jan** and **JAN** are inconsistent. Even though a human might think these all represent the same month, a computer would normally see these as four different strings.

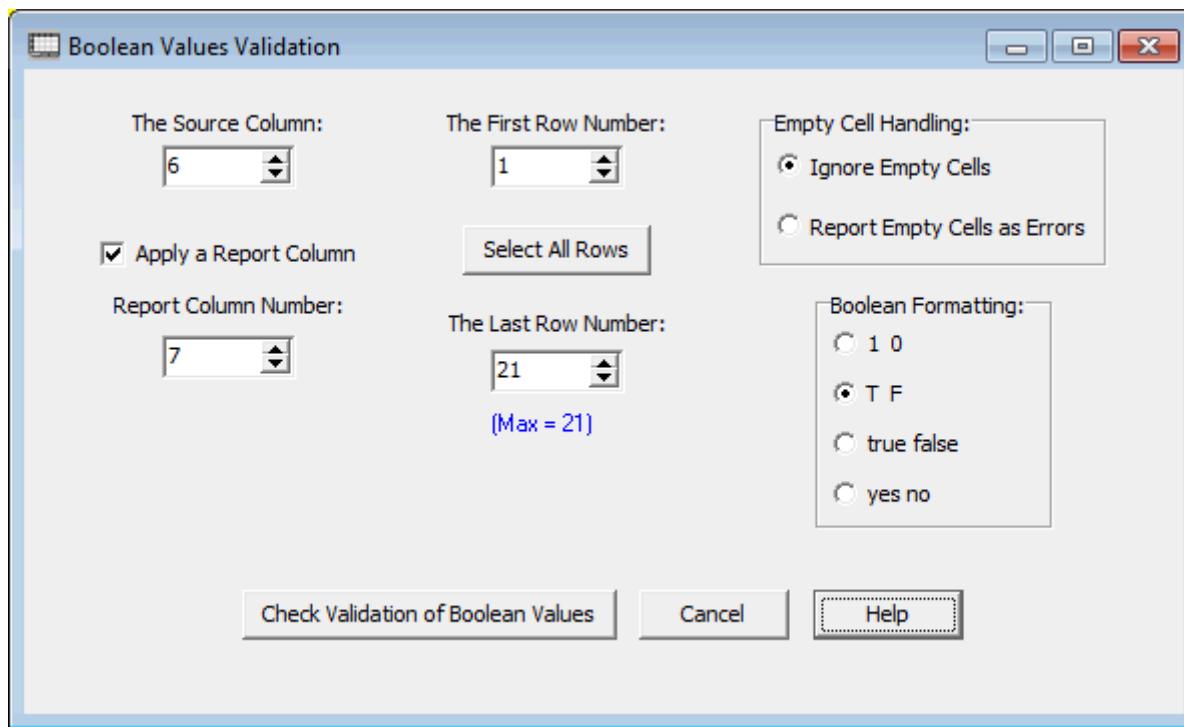
Validation can also be done with special kinds of numbers that are formatted to contain check digits or checksum values. The validation process will verify that the check digits or the checksums are correct. ISBN (International Standard Book Number) and Vehicle Identification Numbers (VIN) are two popular examples of numbers that contain check digits.

Other kinds of data that don't have check digits or checksums can also be validated. Roman Numerals can be validated as well as dates. Neither Roman Numerals nor dates have check digits. Both kinds of data can be validated by converting back and forth between certain standard forms or representations. Dates can also be checked to insure they are all within a selected range of dates. Numbers can also be validated and checked that they all lie within a selected range.

There is a function that performs a simple **Referential Integrity Check** that can be used to determine if all elements in one column of a particular table can be found in another column in another table. This same function can even be applied to two different columns in the same table.

Validating Boolean Values

If you select **Validation | Boolean Values Validation...** you will bring up the dialog box shown below. Validating Boolean data differs from editing or even formatting Boolean data. All validating Boolean data does is warn you when your Boolean values are either invalid or inconsistent.



The above dialog has the controls needed to specify a source column and to select a range of rows within that column. It is an option to **Apply a Report Column** or not. Generally we recommend you do this unless you already know most of your Boolean values are valid. Before you open the above dialog you might want to insert a new blank column just to the right of **The Source Column** that can serve as the **Report Column**. You can also choose an option for how you want to handle **Empty Cells**. The final option is to choose one of the four **Boolean Formatting** choices.

Before you perform a validation test, we suggest you use the **Block Formatting** function under the **Blocks** menu to format your Boolean values so they appear more consistent. Mainly this is to choose the proper case for Boolean values. Note that the only Boolean values that use upper-case are the **T** and **F** values. All other Boolean values should be lower case.

When you perform a validation test, you may find all your Boolean values are proper or you may generate an error message. When you have bad data, you will see an error message and the program will highlight the first cell that it found that is suspect. You should then edit the Boolean data to correct it before you try to validate it again.

When you apply a **Report Column**, the first word in that column will be either **Ok**, or **Error**. You can then correct those rows that have an error.

The grid below on the left contains some random Boolean values in Column 1. The grid on the right shows the result of applying Boolean validation to all the rows in Column 1, but using Column 2 as the **Report Column**. For this validation example we chose the **T F** formatting option.

This is a good example to illustrate why so many of the Boolean values in Column 1 did not pass the validation test, even though all the Boolean values in Column 1 are good Boolean values. The problem is only one of consistent formatting. Only those values that were written as either **T** or **F** passed the validation test because of our choice of **Boolean Formatting**.

<	Column 1	Column 2
1 >	T	
2	yes	
3	T	
4	no	
5	T	
6	1	
7	no	
8	no	
9	no	
10	1	
11	no	
12	yes	
13	1	
14	T	
15	yes	
16	T	
17	F	
18	F	
19	no	
20	F	
21	F	

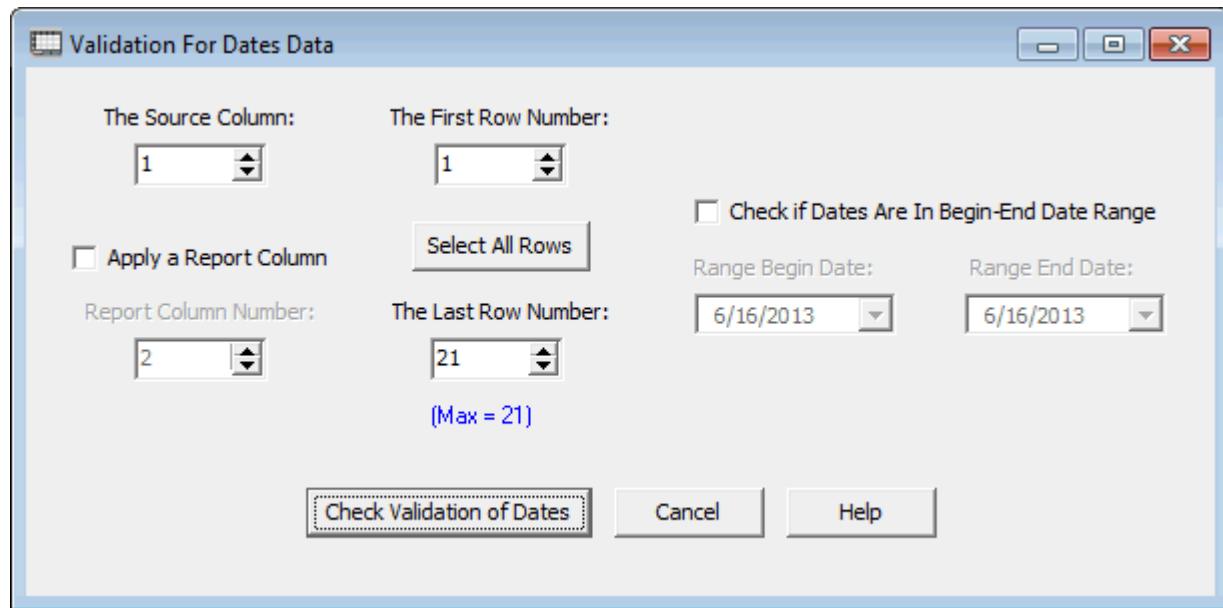
<	Column 1	Column 2
1 >	T	Ok
2	yes	Error Boolean value is improper.
3	T	Ok
4	no	Error Boolean value is improper.
5	T	Ok
6	1	Error Boolean value is improper.
7	no	Error Boolean value is improper.
8	no	Error Boolean value is improper.
9	no	Error Boolean value is improper.
10	1	Error Boolean value is improper.
11	no	Error Boolean value is improper.
12	yes	Error Boolean value is improper.
13	1	Error Boolean value is improper.
14	T	Ok
15	yes	Error Boolean value is improper.
16	T	Ok
17	F	Ok
18	F	Ok
19	no	Error Boolean value is improper.
20	F	Ok
21	F	Ok

If we applied formatting to Column 1 first, all of the values would pass the validation test because they would all be forced to have the same formatting. So the lesson to be learned here is the importance of formatting. All validation tests require strict formatting. If we had chosen a different type of input **Boolean Formatting**, like **yes no**, then we would have generated very different validation results from those shown above.

In general, if you format first you will generate fewer validation errors because you will be starting with more consistent data. To pass Boolean value validation tests, all values must be of the same **Boolean Formatting** type and this means only and exactly two different values are expected.

Validating Dates

If you select **Validation | Dates Validation...** you will bring up the dialog box shown below. Validating dates data differs from editing or even formatting dates data. All validating dates data does is warn you when your dates data is guaranteed to be invalid! Sometimes it will even suggest how you might reformat a date that is found to be in a questionable date format. The dates validation function also includes the ability to screen out dates that are not within a selectable range of dates.



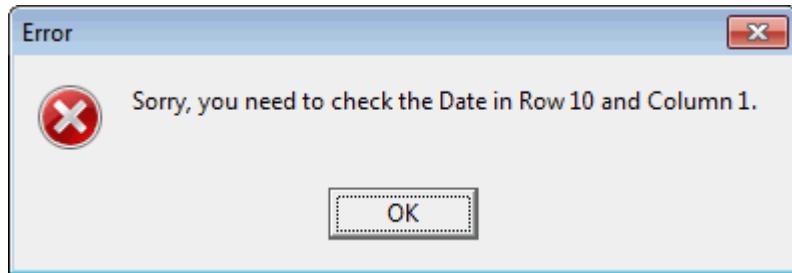
This dialog has the controls needed to specify a date source column and to select a range of rows within that column. It is an option to **Apply a Report Column** or not. Generally we recommend you do this unless you already know most of your dates are valid. Before you open the above dialog you might want to insert a new blank column just to the right of **The Source Column** that can serve as the report column. Finally there is an option to **Check if Dates Are In Begin-End Date Range**. When this last item is checked then you should also set the **Range Begin Date** and set the **Range End Date**. Data in the form of dates can then be checked to insure each date lies between these two date range extremes. The range includes both extreme values.

Each cell entry must be in one of six possible input formats. Examples of those formats are shown in the following grid in which the second yellow row really shows the expected format corresponding to each column that displays some sample dates. The purpose of this table is just to show what valid dates should look like.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
>	MM/DD/YYYY	YYYYMMDD	DD-MonthName-YYYY	YYYY-MonthName-DD	MonthName DD, YYYY	Weekday MonthName DD, YYYY
1	03/30/2013	20130330	30-March-2013	2013-March-30	March 30, 2013	Saturday March 30, 2013
2	03/31/2013	20130331	31-March-2013	2013-March-31	March 31, 2013	Sunday March 31, 2013
3	04/01/2013	20130401	01-April-2013	2013-April-01	April 01, 2013	Monday April 01, 2013
4	04/02/2013	20130402	02-April-2013	2013-April-02	April 02, 2013	Tuesday April 02, 2013
5	04/03/2013	20130403	03-April-2013	2013-April-03	April 03, 2013	Wednesday April 03, 2013
6	04/04/2013	20130404	04-April-2013	2013-April-04	April 04, 2013	Thursday April 04, 2013
7	04/05/2013	20130405	05-April-2013	2013-April-05	April 05, 2013	Friday April 05, 2013
8	04/06/2013	20130406	06-April-2013	2013-April-06	April 06, 2013	Saturday April 06, 2013

For the dates shown in any column (especially column 2), the program will normally expect to see (and for Column 2 to require) exactly two digits for any month number **MM**, or two digits for any day number **DD**. As an example, a month number for the month of February is expected to appear as 02 and not simply 2. Similarly the seventh day of the month is expected to appear as exactly 07 and not simply 7. If you have dates with single digits for some months or days then you might want to first convert those values into 2-digit numbers. The easiest way to do this automatically is to use the **Block Formatting** function under the **Blocks** menu to format the dates so they appear more consistent. You can do this before you try to validate those same dates, although this is not a strict requirement.

When you select a set of consecutive rows in a given column to check it for valid dates data, you may generate an error message that tells you a particular cell has an invalid date. When you see an error message, the program will highlight the first cell that it found that is suspect. You should then edit the date data to correct it before you try to validate it again. A typical error message might appear like the following:



The date 02/31/1968 will be found to be invalid because there is no February 31st for any year. Another example invalid date is 02/29/1970. Since 1970 is not a leap year, there is no February 29 for that year. The next table below shows how some dates will be reported using the **Dates Validation** function.

	Column 1	Column 2
1 >	8/3/99	Error ? Change it to 08/03/1999
2	5/03	Error ? Change it to 05/01/2003
3	January 5, 2012	Ok
4	48	Error ? Change it to 01/01/1948
5	1902	Error ? Change it to 01/01/1902
6	05-Jun-12	Error ? Change it to 06/05/2012
7	9.15.2010	Error ? Change it to 09/15/2010
8	1/1/2010	Ok
9	01/1/2010	Ok
10	1/01/2010	Ok
11	5 January 2013	Error Unrecognizable Date Format
12	5-January-2013	Ok
13	05-January-2013	Ok
14	2013-January-5	Ok
15	2010.9.15	Error Unrecognizable Date Format
16	Saturday March 30, 2013	Ok
17	Friday March 30, 2013	Error Unrecognizable Date Format
18	Saturday April 6, 2013	Ok

A questionable date would be the string 02/1/2012. In this case, the **DD** part of the date is suspect because it consists of only a single digit. This is mostly a date formatting question as opposed to a real invalid date error.

The six rows in the previous table that begin with **Error ?** show even more questionable date errors, which is why the program flags them with a question mark and makes a suggestion on how to change those dates.

Another thing to note is how we handle the dates in rows 8, 9, and 10. In these rows the expected date format is of the form **MM/DD/YYYY**. However, none of the dates in these three rows fit this exact format because either the month number or the day number or both are given as single digits. For five of the six date formats we use, we will tolerate single-digit month numbers or single-digit day numbers and report dates with these single digits as **Ok**.

However, whenever the **CSV Editor** program writes a date that is expected to be in any of our standard six date formats, it will always write 2-digit month and day numbers. Thus our program always writes leading 0's where 2-digits are possible, but we tolerate single digits in your data and don't generate questionable date errors for most dates with single-digit month or day numbers. This of course is NOT true when we deal with the **YYYYMMDD** format because this particular date format requires the full 2-digits for both months and days.

In the above table, note the differences between the dates in rows 11, 12, and 13. Row 11 is not recognized as valid because the date does not use dashes to separate the month name from the numbers.

After making a report with a **Report Column**, you could sort that column to arrange all the Ok rows together and to arrange all the **Error** rows together. If your dates generate a lot of these questionable errors, then you should definitely try the **Block Formatting** function before you try to validate such dates. In six of the first seven rows in the above table the program is able to make a suggestion for how you might change the date to make it valid. Only three of the 18 dates were found to be unrecognizable by the program.

Another thing to note is that when a date is in the format like Saturday March 30, 2013, then we also check that the weekday name Saturday is correct for the March 30, 2013 date. If you tried entering a date like Sunday March 30, 2013 or Friday March 30, 2013 then you will see the program return an error message for that date which will be found to be invalid. If you just drop the weekday name entirely then such a date will pass scrutiny. Note the difference between rows 16 and 17 in the above table.

A date such as 13/45/2012 is easily found to be invalid because there is no 13th month number and there is no 45th day in any month. When a date is determined to be invalid, no further explanation as to why will be given, but you can be sure there is something wrong with any such date.

We use a special but simple algorithm to validate dates. That algorithm computes a **Julian Day Number (JDN)** for the given date and then it converts that **JDN** back into the expected date string format. If the original date string does not match the **JDN** string then your date is invalid.

Incidentally, the format **YYYYMMDD** is probably the most universal date format there is because it uses the least number of characters and it is also good for sorting as string data. However, this doesn't mean it is a preferred date format because most people find it difficult to read. The format **MM/DD/YYYY** is most popular in the United States, but this is probably not true for most other parts of the world. Error messages like the ones in the above table are only intended to convey that the date needs some kind of changing before it can be validated.

If you check the checkbox with the caption **Apply a Report Column** then you will be able to change the column number for that report. Otherwise the control will be disabled and grayed out.

Similarly, the checkbox with the caption **Check if Dates Are In Begin-End Date Range** will allow you to enter both the **Range Begin Date** and the **Range End Date**. Otherwise these two controls will be disabled and grayed out.

In the next table below, we list some random dates in the first column. We want to screen out those dates that occur before 1951 or after the year 2050. So we set our date range with the **Range Begin Date** of 1/1/1951 and we set the **Range End Date** as 12/31/2050. We also checked the checkbox with the caption **Check if Dates Are In Begin-End Date Range**.

The form contains a checked checkbox labeled "Check if Dates Are In Begin-End Date Range". Below it are two dropdown menus labeled "Range Begin Date:" and "Range End Date:". The "Range Begin Date:" dropdown shows "1/ 1/1950" and the "Range End Date:" dropdown shows "12/31/2050".

You can see the report results in Column 2 in the grid below. Column 1 in the grid just contains a list of some random dates in the **MM/DD/YYYY** format.

<	Column 1	Column 2
1 >	06/28/2009	Ok
2	12/05/2077	Error Date is out of range
3	03/27/1932	Error Date is out of range
4	11/18/2071	Error Date is out of range
5	04/22/2012	Ok
6	10/22/2038	Ok
7	05/17/2041	Ok
8	01/10/2009	Ok
9	05/04/1960	Ok
10	05/01/1920	Error Date is out of range
11	01/29/1903	Error Date is out of range
12	01/01/1951	Ok
13	12/31/2050	Ok

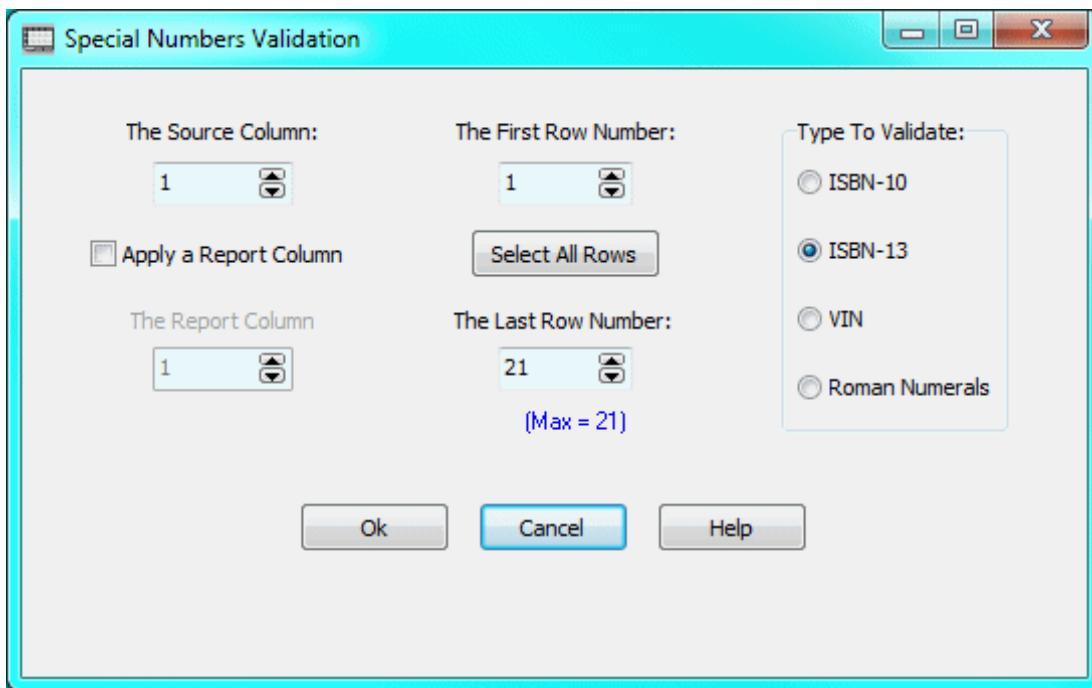
The program has found several dates to be out of range. Those are the dates that are either before 01/01/1951 or dates that are after 12/31/2050.

We can also see that in the last two rows we have dates that are exactly on the range borders. Both dates were determined to be **Ok** because both date border values are in the included range. So the range includes both the **Range Begin Date** and the **Range End Date** and it includes all dates in between.

It is possible to make the **Range Begin Date** the same day as the **Range End Date**. In this case validating dates will insure all dates are exactly that single date. This kind of a check could also be done using string validation with a single string in a validation list. The program won't let you set a **Range End Date** that is strictly before the **Range Begin Date**.

ISBN Validation

When you select the menu item **Validation | ISBN Validation...** you will see the following dialog that is used to validate International Standard Book Numbers, or ISBNs. As with all validation functions, this function will not change your data. This function only tries to verify that your ISBNs are correct and, and when one isn't correct, this function will show a message that indicates the location of the incorrect data.



You must select a **Source Column** which is the column that contains the ISBN numbers that are to be validated. You can also select an appropriate range of rows within that source column.

It is optional to check the box with the caption **Apply a Report Column**. When this checkbox is checked, it means the program will actually write a message in the **Report Column** that will either say **Ok** or will tell why the given ISBN number was not valid. In this case a report message will be written in every selected row, but in the chosen **Report Column**. We usually only **Apply a Report Column** when we anticipate there may be many errors in the **Source Column**. When that is the case, before we open the above dialog, we insert a blank **Report Column** next to the **Source Column**. Later, when we are finished validating all the ISBN numbers, we will delete that **Report Column**.

If you only expect very few errors to occur then you can leave the **Apply a Report Column** checkbox unchecked and the program won't use the **Report Column**. Instead, the program will stop with an error message at the first row that contains an invalid ISBN.

The **Type To Validate** radio buttons allow you to select the type of number to validate. For ISBN validations you would only select one of the first two buttons in that radio group. You should already know the type of ISBN numbers that are in your **Source Column**.

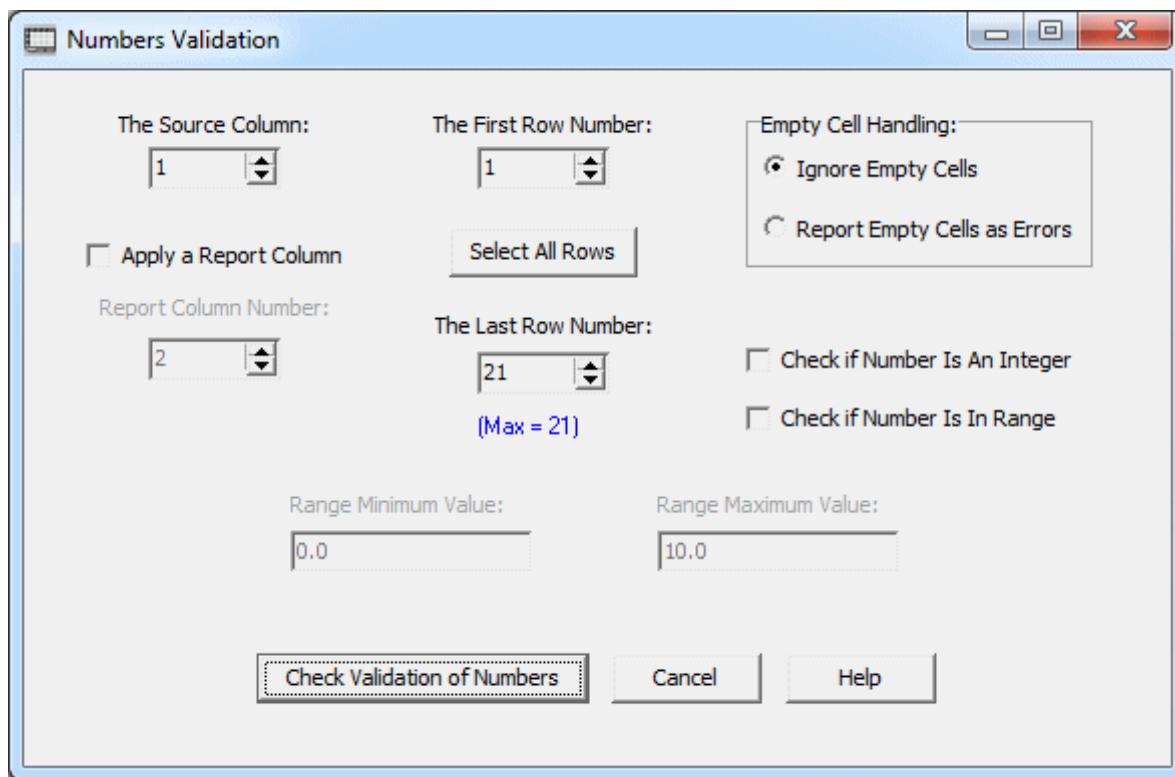
When you click the **Ok** button you should see the validation result.

If you checked the checkbox to **Apply a Report Column** then look at all relevant rows in the **Report Column**. Sometimes it is nice to sort the **Report Column** to group similar errors by consecutive rows. If you are lucky, most of your rows will say **Ok**. But if any row contains an invalid ISBN, the reason will be written in the corresponding **Report Column**. When there are any errors, the first word in the **Report Column** will be **Error**. You could also search for the word **Error** in the **Report Column**.

If you don't **Apply a Report Column** then you will see one of two messages. One message will congratulate you on having all valid ISBN numbers. The other message will warn you about the first row the program has found that contains an invalid ISBN number. Other parts of any error message should indicate exactly why the ISBN was determined to be invalid. When any error occurs, you should correct the data and then run this same validation test again until everything checks out. When there is an error the cell containing the first error will be highlighted and made visible.

Validating Numbers

Another **Validation** menu item has the caption **Numbers Validation....** This menu item is used to guarantee that each string in your grid can be converted to a floating point value or an integer or a number that lies within a given range. This function is useful if you accidentally type an invalid character as part of a regular number. For example, the letter O might be confused with the digit zero 0, or the letter l could be mistaken for the digit one 1. This validation check will avoid these errors as well as other errors. When you select this menu item you will see the following dialog box.



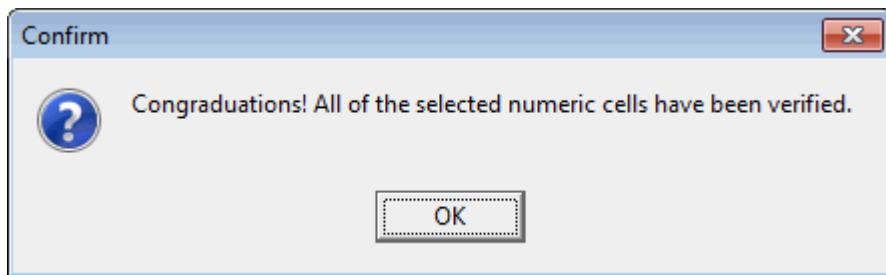
You should first set **The Source Column** that determines which column will be used for the validation test. Next you can choose to **Apply a Report Column** or not. If you check this checkbox then you should also set the number of the column that will hold the report results. Next, you can set the range of rows that will be visited by the validation function. Then you can set an option for how you want to handle empty cells. You can either ignore empty cells or have this function report empty cells as having invalid numbers.

Last, check either or both or neither of the checkboxes with the captions **Check if Number Is An Integer** or **Check if Number Is In Range**. If you choose to perform a range check then you must further enter the **Range Minimum Value** and the **Range Maximum Value**. The **Range Maximum Value** should always be at least as large as your **Range Minimum Value**. The two range values will be grayed out and disabled whenever you don't check the checkbox to **Check if Number Is In Range**.

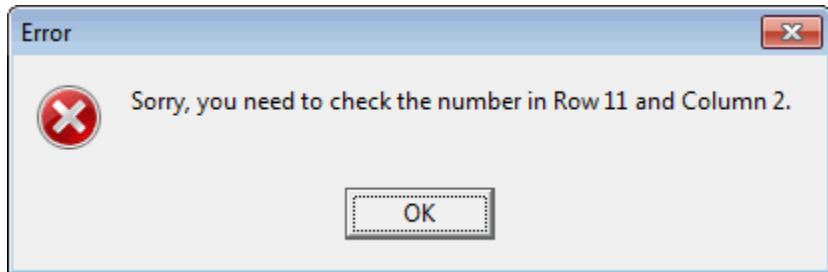
When you press the button with the caption **Check Validation of Numbers** the program will scan the chosen rows within **The Source Column**. The program will check each cell to insure the cell contains a valid floating point number.

If you are applying a report column then all you need do is inspect the results in that column when the function finishes. In some cases you may wish to sort the report column to group the valid and invalid rows together. If there are any invalid numbers, the grid selection will change to show the first invalid number cell that was found. When there are any errors, the first word in the report column will be **Error**, in the corresponding row. You could also search for the word **Error** in the report column.

If you are not applying a report column then you should see one of the following two messages. If all cells are good you should see the message:



On the other hand, the program will stop at the first cell that it finds that contains an invalid number and you will then see a different message similar to:



If your numeric data is in the form of currency values that have \$ signs and commas, you can use this function on that data if you first use the **Block Formatting** function to format it as ordinary numeric data. Then after the validation test you can perform a second **Block Formatting** to turn the data back into the currency format.

Depending on the options you choose, you can see several other kinds of error messages in the report column.

To **Check if Number Is In Range** means the program will verify that your cells numbers lie between the range minimum and the range maximum, inclusive. Another way describing this is that you will only generate an error if any cell value is strictly less than the range minimum or is strictly larger than the range maximum. Values exactly equal to a range minimum or equal to a range maximum will be accepted as Ok. To check for single exact values, you can set the **Range Minimum Value** equal to the **Range Maximum Value**. As an alternative, if you needed to check a single value, you could also perform a validation using strings where the validation list only contains one string that is the number you anticipate.

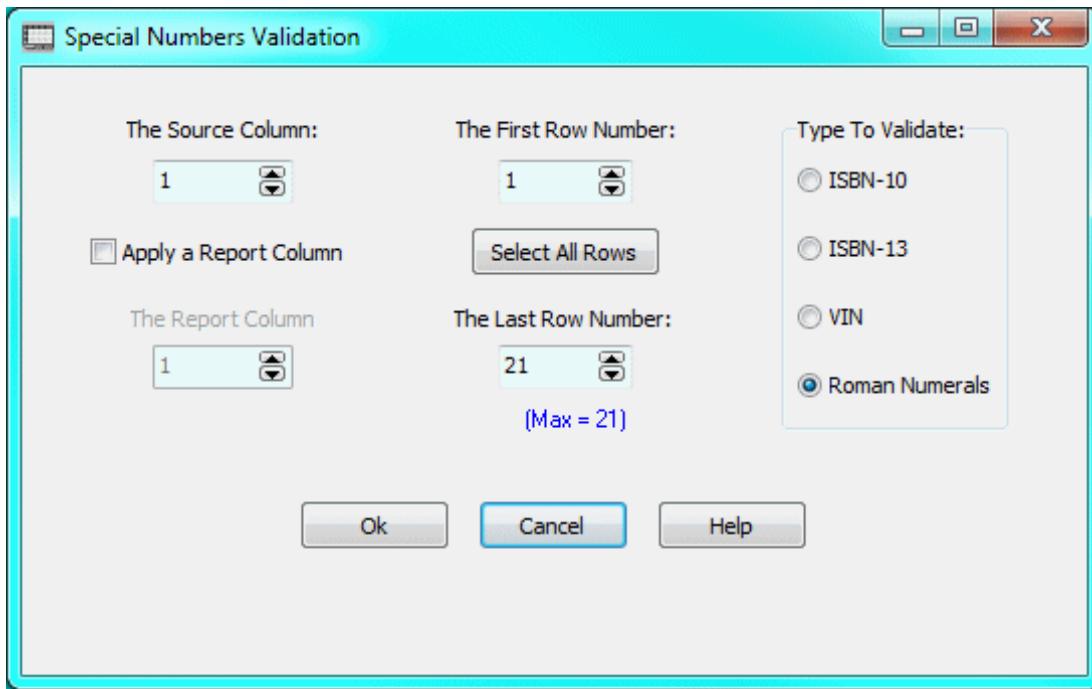
The default range is the closed interval [0.0, 10.0] and in almost all cases you will need to change one or both of these values. Either or both range values can be negative numbers as long as the range maximum is never smaller than the range minimum.

Also note that the two controls for entering the range minimum and maximum values will be grayed out and disabled until you check the checkbox with the caption: **Check if Number Is In Range**. Then those two controls will be enabled and then you can enter a proper range of values.

If you need check that your numbers are integers, and you also want to perform a range check at the same time, then you can check both of the available checkboxes. However, when any error is reported, then only the first violation will be reported. If there are no errors then you can be assured both the integer and range checks passed for all the cells that were checked. It is only when there are errors that you should correct those initial errors and then run another validation check until no errors are reported.

Validating Roman Numerals

Another **Validation** menu item has the caption **Roman Numerals Validation....** This menu item is used to verify that each string in a given column contains a valid or proper Roman Numeral. This function is useful if anyone accidentally types an invalid character as part of a Roman Numeral. When you select this menu item you will see the following dialog box.



The **Type To Validate** radio button that should already be checked is the one that has the caption **Roman Numerals**.

If the **Source Column** number is not correct then you should first set the **Source Column** number that determines which column will be checked for valid Roman Numerals. If the range of rows is not correct then you should also set the proper range of rows that will be visited by this validation function.

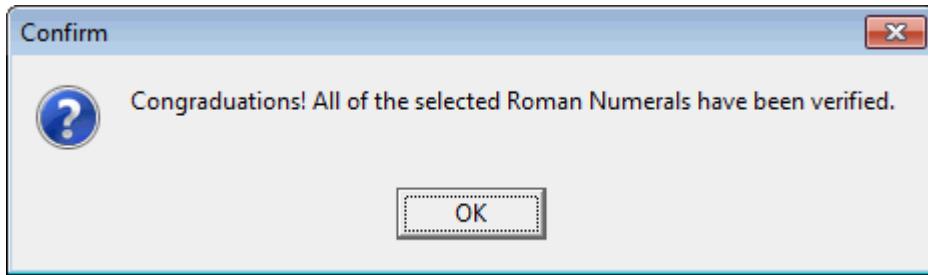
It is optional to check the checkbox with the caption **Apply a Report Column**. When this checkbox is checked, it means the program will actually write a message in the **Report Column** that will either say **Ok** or will tell why the given Roman Numeral was invalid. In this case a report message will be written in every selected row, but in the chosen **Report Column**. We usually only **Apply a Report Column** when we anticipate there may be many errors in the **Source Column**. When that is the case, before we open the above dialog, we insert a blank **Report Column** next to the **Source Column**. Later, when we are finished validating all the Roman Numerals, we will delete that **Report Column**.

If you only expect very few errors to occur then you can leave the **Apply a Report Column** checkbox unchecked and the program won't use the **Report Column**. Instead, the program will stop with an error message at the first row that contains an invalid Roman Numeral.

When you click the **Ok** button in the above dialog box the program will scan the relevant rows within the **Source Column**.

If you don't **Apply a Report Column** then you will see one of two messages. One message will congratulate you on having all valid Roman Numerals. The other message will warn you about the first row the program has found that contains an invalid Roman Numeral. Other parts of any error message should indicate exactly why the Roman Numeral was determined to be invalid. When any error occurs, you should correct the data and then run this same validation test again until everything checks out.

If all cells are good you will see a message like the following:



On the other hand, the program will stop at the first cell that it finds that contains an invalid Roman Numeral and you will then see a different message. There can be many reasons why a Roman Numeral isn't valid and the corresponding error message should give appropriate details of what was wrong.

Only Roman Numerals that are in the range of 1-4999 when converted to whole numbers will be considered valid. It doesn't make sense to have a Roman Numeral larger than 4999 or less than 1.

The program performs a similar operation to what it does to validate dates. The program will convert the Roman Numeral to a whole number and then it will convert that whole number back to a Roman Numeral and it will compare that with the original Roman Numeral. If the two don't match then the original Roman Numeral is invalid. In that case you may see the message:

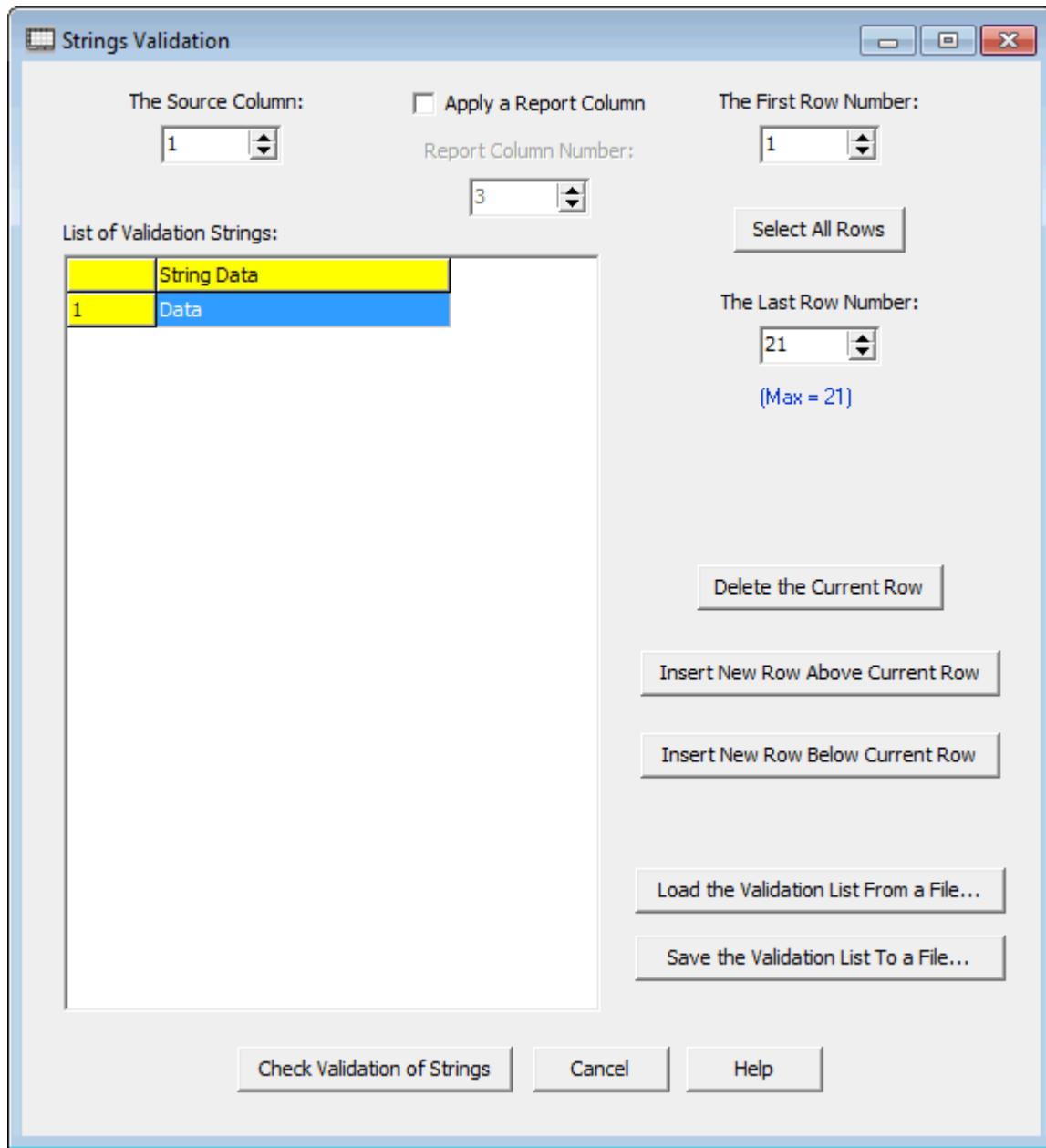


If you checked the checkbox to **Apply a Report Column** then look at all relevant rows in the **Report Column**. Sometimes it is nice to sort the **Report Column** to group similar errors by consecutive rows. If you are lucky, most of your rows will say **Ok**. But if any row contains an invalid Roman Numeral, the reason will be written in the corresponding **Report Column**. When there are any errors, the first word in the report column will be **Error**. You could also search for the word **Error** in the report column.

When there is any error, the program will highlight and show the first cell in question.

Validating Strings

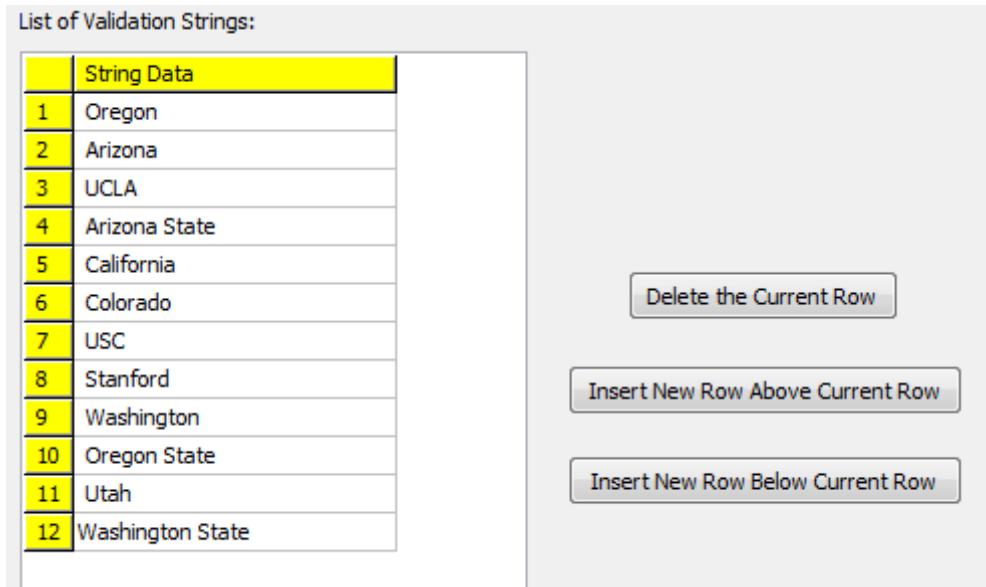
The next **Validation** menu item is concerned with validating string data. At first blush you may think this is odd because virtually all the cells in a grid contain only string data. When you choose the menu item with the caption **Strings Validation...** you will see a dialog box similar to the following:



The controls in the top of this dialog allow you to select a particular source column to work in and they allow you to specify exactly the rows within that source column that you will inspect. The main option is controlled by the checkbox to **Apply a Report Column**. If you choose to apply a report column then you must also set the correct **Report Column Number**. In fact, when you are going to use a report column then, before you open the above dialog, you should normally insert a new blank column just after the data column to be validated.

The other main control in this dialog is a grid that is to contain a list of what we call the **Validation Strings**. What is a list of Validation Strings?

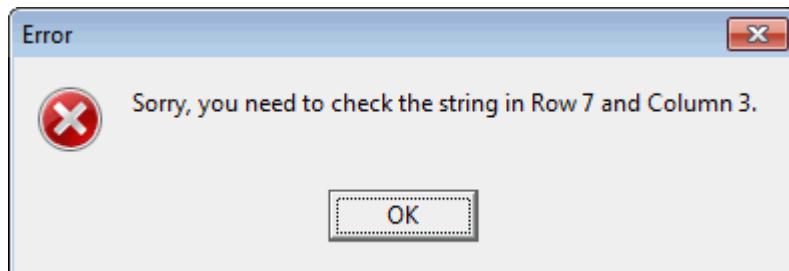
It is a list of items (usually a small list) that your grid data is to be chosen from. As an example, suppose you have a grid column in which every entry is to represent one of the schools in the Pacific 12 Athletic Conference. What you should do in this case is fill in the **List of Validation Strings** so that it appears as:



In this case you would manually type in the list of the school names as shown above. You could use the buttons to the right of the grid to help you manage editing the list in terms of adding or deleting rows. The items in your list can be in any order. For validation purposes, they do not have to be sorted in any particular way.

After this list of Validation Strings is correct, you would press the button with the caption **Check the Validation of Strings**. The program will then march down your main CSV grid and for each cell it will read the string data and then it will try to find that same entry in the list of Validation Strings. The reason this is so important is that it not only verifies that each school name is spelled correctly, it guarantees your data is consistent. In database systems, checks of data similar to this are given a name that is called **referential integrity**. This is an important concept in modern relational database systems that helps maintain accuracy as well as consistency in your data.

If you accidentally had a data cell with the school name Notre Dame, a school that is not in the Pacific 12 Conference, this function would catch that error and you might see a message like the following:



Another, but different kind of validation error that will be flagged would be if you had an entry like **Southern Cal** that was intended to be **USC** but you didn't abbreviate it as such.

If you had a large database table, say one that had a list of all the football and basketball and baseball players from each school, you might have 2000 rows of data. If you wanted to verify that all 2000 rows had the correct spelling of every corresponding school name, this function would beat the pants off of trying to verify the data manually by reading or inspecting each cell in your grid. This function makes for very fast comparisons of large data sets.

This feature of the **CSV Editor** program is most useful whenever someone else sends you a data list that has not been inspected and cleaned and maintained as it should. This function makes it relatively easy to find and correct all the errors in such mal-formed data. It is much more efficient than trying to correct errors by using search and replace functions in a text editor. It is way better than trying to do it manually.

We should mention that after you enter a list of validation strings, you will probably want to save that list for future use. Press the button with the caption **Save the Validation List To a File....** For the above example your saved grid could be in one of two formats. We show both formats below in a regular text editor.

"Oregon"	Oregon
"Arizona"	Arizona
"UCLA"	UCLA
"Arizona State"	Arizona State
"California"	California
"Colorado"	Colorado
"USC"	USC
"Stanford"	Stanford
"Washington"	Washington
"Oregon State"	Oregon State
"Utah"	Utah
"Washington State"	Washington State

The format on the left is for a single column **CSV** file. The format on the right is the same as on the left except none of the data is delimited by double quote characters. If you load a list of Validation Strings from a text file, that text file can be an ordinary list without the double quotes, or it can be a real **CSV** file with only one column that has the double quotes.

Such a list also may or may not have a first header row. A header row is something we like to always include, but it is really an option as to whether your list has a header row or not. If you have such a row and don't want it, you can click the button, **Delete the Current Row**, to get rid of it. All of the sample validation data lists and all the sample data pick lists that are distributed with the **CSV Editor** program have a header row.

If any of your strings needed to contain the double quote character or a comma character, then you should save the data using the above dialog function that will always insert the double quotes and create a true **CSV** file. Otherwise it doesn't matter. When a file is loaded in the above dialog, the loading function will strip off the double quotes that are not needed and it will contain only those double quotes that are part of real data.

You can use any text editor program to create a list of single column data and this program can load it and use it as a regular text file. The number of strings you have in a list of validation strings can be quite large. In fact, such a list could have hundreds or even thousands of entries. When that is the case, you will want to save and load that list using a file.

When you choose to **Apply a Report Column**, then the program will not show any error messages like the one shown above, but for each row in the selected data column, the program will report whether that data was found in the list of validation strings. The answer is written in the same row but in the report column. A typical report might look like the following where we took the liberty to first insert a blank report column and even made a header for Column 2. When the report was made, each row in Column 2 got filled with a report answer. When there are any errors, the first word in the report column will be **Error**. You could also search for the word **Error** in the report column. You could also just sort the **Report Column** to put all the **Error** rows together as well as put all the **Ok** rows together.

<	Column 1	Column 2
>	School Name:	Report Column:
1	UCLA	Ok
2	Oregon	Ok
3	Notre Dame	Error Not Found
4	USC	Ok
5	Southern Cal.	Error Not Found
6	Arizona	Ok
7	Utah	Ok
8	Colorado	Ok
9	Ohio State	Error Not Found
10	Alabama	Error Not Found
11	Stanford	Ok
12	Washington	Ok
13	Oregon State	Ok
14	Washington State	Ok
15	Arizona State	Ok
16	Ball State	Error Not Found

In this example we can see that most school names were found to be **Ok**. But there were five schools listed in the first column that were **Not Found** when we tested the School Names against the list of schools in the Pac-12 Conference.

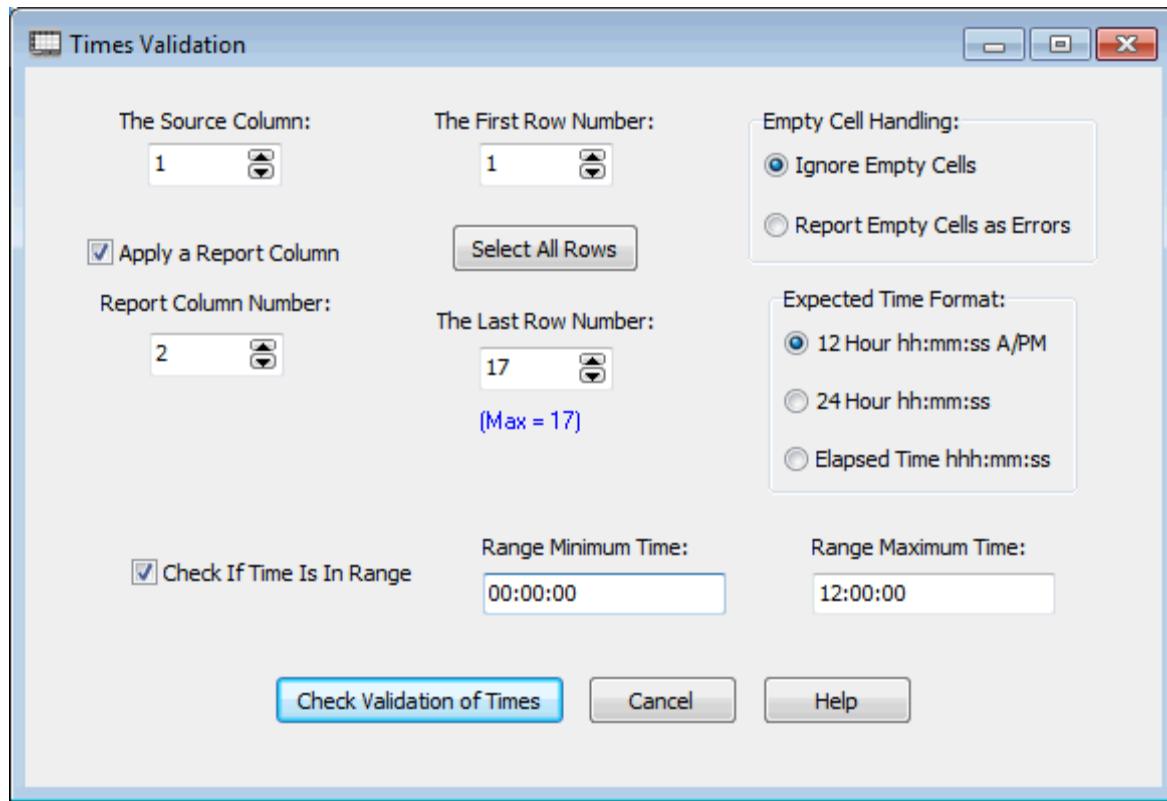
In any case, whenever a string cannot be validated the program will highlight the first cell that contains the invalid data string.

Usually validating strings takes only a few seconds. However, for very large lists you may find it helpful to view a countdown number in the lower left of the **Status Bar**. The number will count down to zero and then the process will be finished. You will also see a message flash to indicate the validation report is complete.

There is no option for case sensitivity when using a list of validation strings. In other words, the use of case must be correct in both your data column and in the validation list. As an example, the strings **Washington** and **washington** don't match.

Validating Times

To validate data that represents times, select the menu item **Validation | Times Validation....**. You should then see the following dialog.



This has all the usual controls for setting up a standard validation check. The two main options are for determining how you want to handle empty cells and for setting the **Expected Time Format**.

There are three controls related to setting and performing a range check on time values. This feature is used when you need to insure all times fall within a given range of values. As an example, if you wanted to insure all times are within working hours between 9 AM and 5 PM you can do that. The time range defines a closed interval that includes both endpoints.

Either the **Range Minimum Time** or the **Range Maximum Time** can be entered in any valid time format, although the **Elapsed Time** format may be the simplest. In fact, if the times in your grid are all in the **Elapsed Time** format then the two time values you enter could both have more than 24 hours. If you think about it, it is probably best to enter the two time range values using the same format as you choose for the **Expected Time Format** in the grid, even though this is not required. Leave the checkbox **Check If Time Is In Range** unchecked when you don't care about validating the range of time values. When the checkbox with the title **Check If Time Is In Range** is not checked then both the **Range Minimum Time** and the **Range Maximum Time** controls will become inactive and grayed out.

The two grids below show the before and after results when we applied **Times Validation** to the grid on the left. For this example we assume all the controls were set as shown in the above dialog.

As with most validation results, when we apply a report column, the first thing that starts each cell in the report column will either be **Ok** or **Error**.

<	Column 1	Column 2
>	Times:	Validation Result:
1	02:03:04 AM	
2	02:05:06 PM	
3	14:00:00	
4	0	
5	14:00:00 PM	
6	11:00:00 PM	
7	12345	
8	59:159:180	
9	124:40:38	
10	23:59:59	
11	24:60:60	
12	0:0:0	
13	12:00:00 AM	
14	12:00:00 PM	
15	23:59:60	
16	01:02:03x	
17	-5	

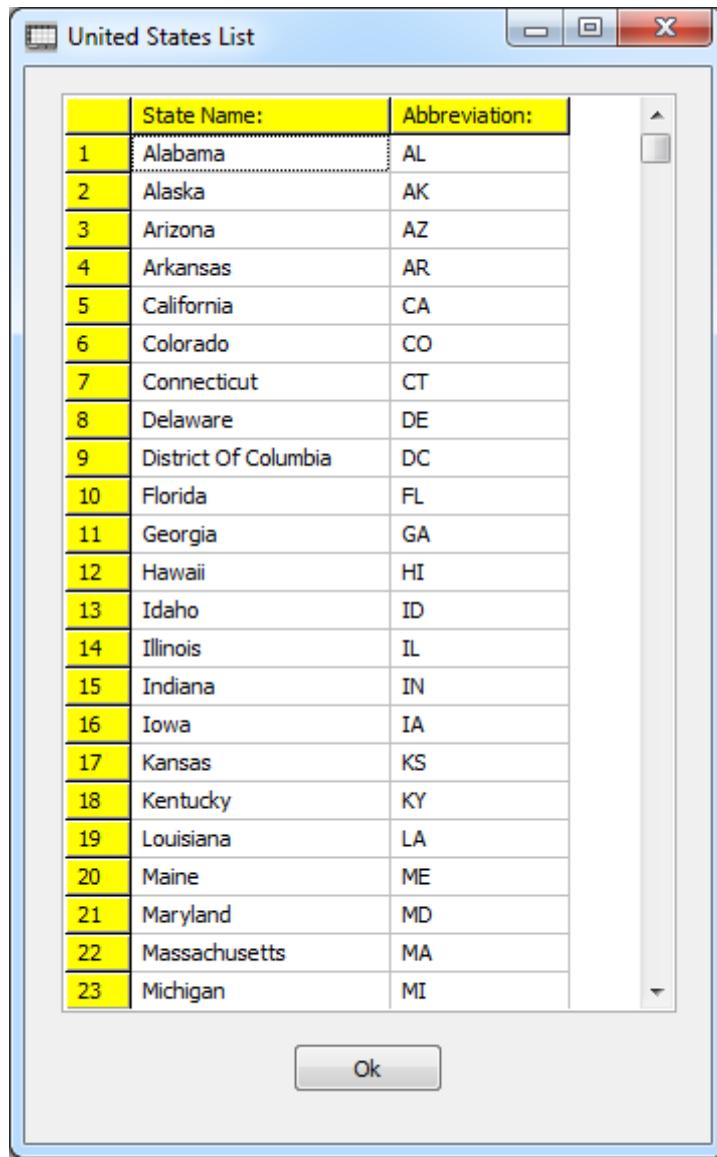
<	Column 1	Column 2
>	Times:	Validation Result:
1	02:03:04 AM	Ok
2	02:05:06 PM	Error Range Check Failed
3	14:00:00	Error 2-way Conversion Failed
4	0	Error 2-way Conversion Failed
5	14:00:00 PM	Error 2-way Conversion Failed
6	11:00:00 PM	Error Range Check Failed
7	12345	Error 2-way Conversion Failed
8	59:159:180	Error 2-way Conversion Failed
9	124:40:38	Error 2-way Conversion Failed
10	23:59:59	Error 2-way Conversion Failed
11	24:60:60	Error 2-way Conversion Failed
12	0:0:0	Error 2-way Conversion Failed
13	12:00:00 AM	Ok
14	12:00:00 PM	Ok
15	23:59:60	Error 2-way Conversion Failed
16	01:02:03x	Error Invalid Time
17	-5	Error Invalid Time

The reason we got so many **Errors** in the above example is mostly because of failure of the 2-way conversion. This means the time format found in the grid differed from matching what was chosen for the **Expected Time Format**. In fact, all but the last two rows contain valid time data, but that data is not formatted consistent with the **Expected Time Format**. It is the inconsistent formatting that is the primary problem that the above validation check has found. You might also note that only two of the input times failed the range check.

If we reformat the first 15 rows to the same format (12-hour or 24-hour or elapsed), and then do a validation check on those rows, without a range check, we will find all the times validate as **Ok**.

Validating United States Names

Another **Validation** menu item has the title **United States Validation....** This is just a special case of string validation in which we automatically pre-load a list that looks like the following.

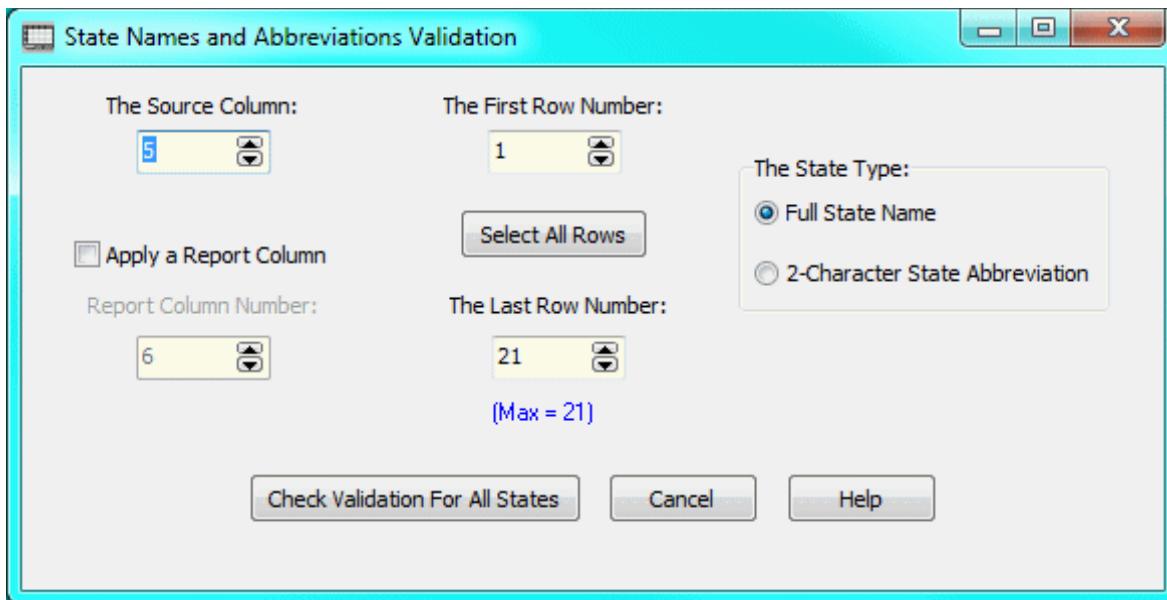


The screenshot shows a standard Windows-style dialog box with a title bar 'United States List'. Inside is a grid table with two columns: 'State Name:' and 'Abbreviation:'. The first column contains numbers 1 through 23, and the second column contains the corresponding state names and abbreviations. An 'Ok' button is at the bottom.

	State Name:	Abbreviation:
1	Alabama	AL
2	Alaska	AK
3	Arizona	AZ
4	Arkansas	AR
5	California	CA
6	Colorado	CO
7	Connecticut	CT
8	Delaware	DE
9	District Of Columbia	DC
10	Florida	FL
11	Georgia	GA
12	Hawaii	HI
13	Idaho	ID
14	Illinois	IL
15	Indiana	IN
16	Iowa	IA
17	Kansas	KS
18	Kentucky	KY
19	Louisiana	LA
20	Maine	ME
21	Maryland	MD
22	Massachusetts	MA
23	Michigan	MI

The above list has two columns. The left column spells out the name of each state. The right column contains the 2-letter abbreviation for the state. You can just think of this picture as containing two lists of Validation Strings.

When you have a grid that is to contain either the spelled out state names or the 2-letter abbreviations, you can select the menu item with the caption **United States Validation...** and you will see the following dialog:



Like all data validation dialogs, this one has the usual controls in the top for selecting a particular source column and selecting a range of rows within that source column. You can also choose to **Apply a Report Column** or not. The final option is to choose the **State Type** and all this means is you will either check the fully spelled out state name or you will check the 2-character state abbreviation.

Thus this dialog is a special case of performing string validations. The only thing special is that it uses the pre-built list of state information. If you want to inspect that list you can choose the **Utilities** menu item with the caption **Sample a Data List ► | United States List...** and you will see the list already shown above. This list has 51 entries. There is one entry for each of the 50 states and there is one additional entry for the District of Columbia.

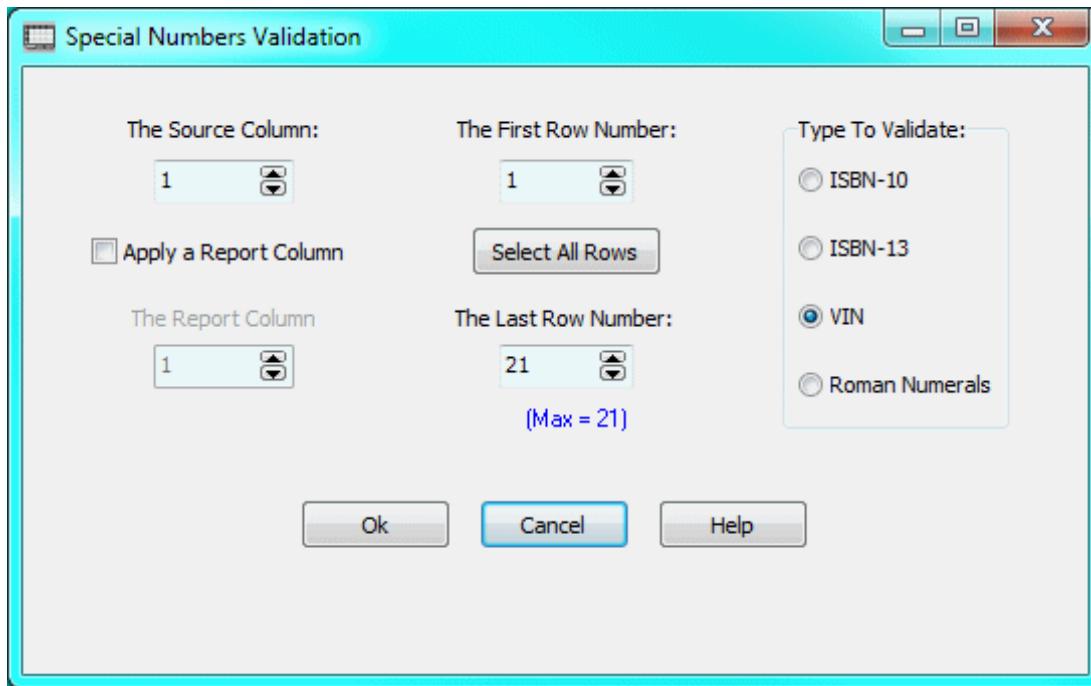
If you chose to **Apply a Column Report** then when finished executing you should inspect the report column in which all the cells will either show **Ok** or **Error Not Found**. When there are any errors, the first word in the report column will be **Error**. You could also search for the word **Error** in the report column.

If you don't apply a column report then you will see one of two messages. You may see a message that tells you all the source column cells were valid, or you will see a message telling you the first row that contained an invalid state name or abbreviation.

In case any cell can't be validated, then the program will highlight and show the first cell that contains the invalid data.

VIN (Vehicle Identification Numbers) Validation

When you select the menu item **Validation | VIN (Vehicle Identification Numbers) Validation...** you will see the following dialog.



You must select a **Source Column** which is the column that contains the **VIN** vehicle identification numbers that are to be validated. You can also select an appropriate range of rows within that **Source Column**.

It is optional to check the box with the caption **Apply a Report Column**. When this checkbox is checked, it means the program will actually write a message in the **Report Column** that will either say **Ok** or will tell why the given **VIN** was not valid. In this case a report message will be written in every selected row, but in the chosen **Report Column**. We usually only **Apply a Report Column** when we anticipate there may be many errors in the **Source Column**. When that is the case, before we open the above dialog, we insert a blank **Report Column** next to the **Source Column**. Later, when we are finished validating all the **VIN** numbers, we will delete that **Report Column**.

If you only expect very few errors to occur then you can leave the **Apply a Report Column** checkbox unchecked and the program won't use the report column. Instead, the program will stop with an error message at the first row that contains an invalid **VIN**.

The **Type To Validate** should already be selected as **VIN**. In this case, **VIN** stands for Vehicle Identification Number.

When you click the **Ok** button the program will validate the **VIN** data.

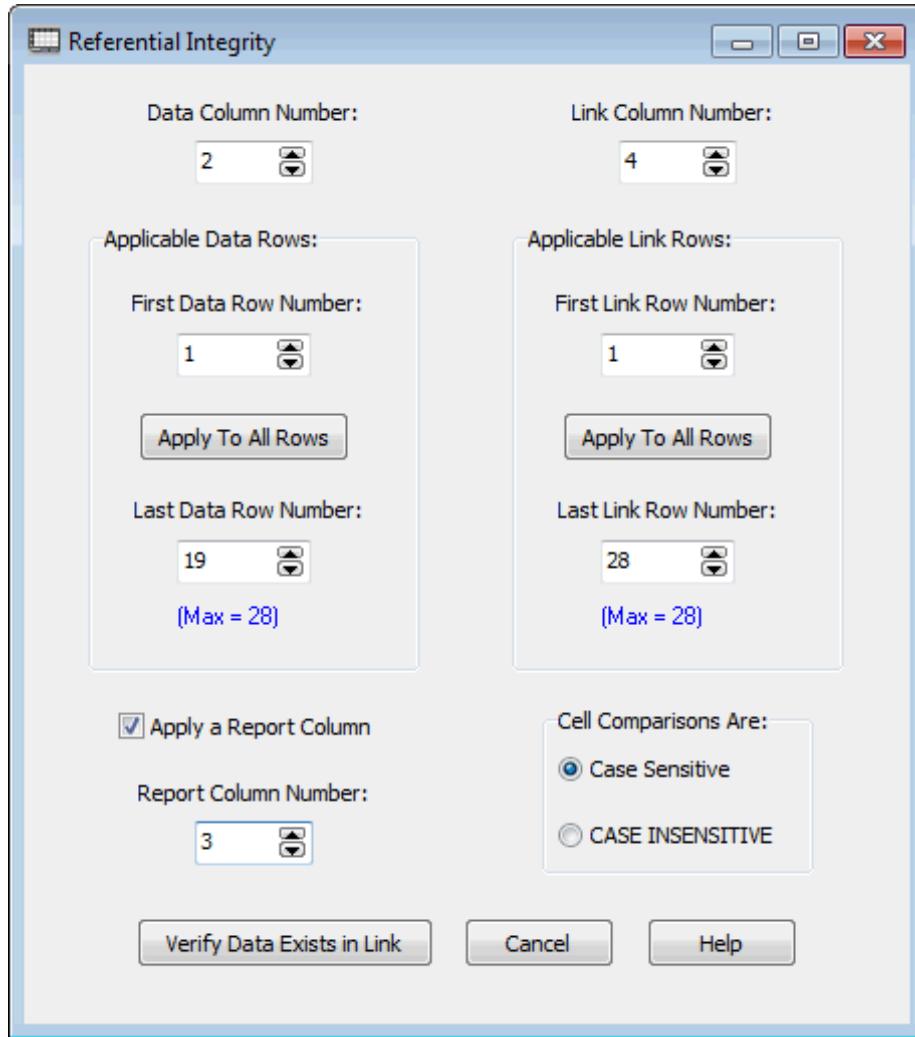
If you did not check the checkbox to **Apply a Column Report**, then you should see one of two messages. One message will congratulate you on having all valid **VIN** numbers. The other message will warn you about the first row the program has found that contains an invalid **VIN** number. Other parts of any error message should indicate exactly why the **VIN** was found to be invalid.

If you checked the checkbox to **Apply a Report Column** then look at all relevant rows in the **Report Column**. Sometimes it is nice to sort the **Report Column** to group similar errors by consecutive rows. If you are lucky, most of your rows will say **Ok**. But if any row contains an invalid **VIN**, the reason will be written in the corresponding **Report Column**. When there are any errors, the first word in the report column will be **Error**. You could also search for the word **Error** in the report column.

In case any cell can't be validated, then the program will highlight and show the first cell that contains the invalid data.

Referential Integrity

The **Validation** menu contains a special function that is called **Referential Integrity**. When you select the menu **Validation | Referential Integrity Check...** you will see a dialog like the following.



We will explain how to setup and use the controls in the above dialog after we present a problem that involves asking a question related to two tables.

Imagine we have two tables like those shown on the next page. **Column 2** in both tables contain a list of ISBN numbers. We would like to determine which, if any, of the ISBN numbers in the list on the left exist anywhere in the list on the right.

<	Column 1	Column 2
1 >	Cummings	978-1-56881-721-7
2	Spence	978-1-449-30468-3
3	Strong	978-0-486-64725-8
4	Grant	978-0-262-51668-6
5	Vinson	978-0-8176-4704-9
6	Hardin	978-0-486-47417-6
7	Guzman	978-1-4129-1314-6
8	Kennedy	978-0-8176-4372-0
9	Haney	978-0-465-01775-1
10	Andrews	978-0-486-47883-8
11	Hogan	978-0-691-15265-3
12	Gates	978-0-691-14039-1
13	Bennett	978-0-471-11709-4
14	Alvarez	978-0-596-80552-4
15	Shields	978-1-59184-492-1
16	Rosa	978-1-56158-891-6
17	Gray	978-0-486-65241-2
18	Sears	978-0-19-974044-3
19	Anderson	978-0-321-88691-0

<	Column 1	Column 2	Column 3	Column 4
1 >	1/29/2012	978-1-449-30468-3	O'Reilly & Associates	Johnson, Clay
2	2/15/2012	978-0-691-14039-1	Princeton University Press	Bernstein, Dennis S.
3	2/15/2012	978-0-19-974044-4	Oxford University Press	Levitin, Anay; Levitin, Maria
4	2/15/2012	978-0-691-14714-7	Princeton University Press	MacCormick, John
5	2/23/2012	978-0-486-47883-8	Dover Publications	Michaelson, Greg
6	2/27/2012	978-0-465-01775-1	Basic Books	Stewart, Ian
7	3/2/2012	978-0-486-47417-5	Dover Publications	Pinter, Charles C.
8	4/12/2012	978-0-471-11709-4	John Wiley & Sons	Schneier, Bruce
9	4/14/2012	978-0-8218-4418-2	American Mathematical Society	Mullen, Gary L.
10	7/6/2012	978-0-691-15270-7	Princeton University Press	Cook, William J.
11	7/17/2012	978-0-691-14342-2	Princeton University Press	Havil, Julian
12	8/21/2012	978-0-321-62930-2	Addison-Wesley	Vickers, Andrew
13	9/13/2012	978-0-674-05755-5	Belknap Press	Lockhart, Paul
14	9/25/2012	978-1-118-46446-5	Wiley Computer Publishing	Upton, Eben; Halfacree, Gareth
15	10/4/2012	978-0-307-72095-5	Crown Publishers	Anderson, Chris
16	10/4/2012	978-1-59420-411-1	The Penguin Press	Silver, Nate
17	10/4/2012	978-1-59184-492-1	Portfolio/Penguin	Steiner, Christopher
18	10/5/2012	978-0-984-72511-3	Digital Frontier Press	Byrnjolfsson, Erik; McAfee, Andrew
19	10/11/2012	978-1-118-20413-9	Wiley Computer Publishing	Thurrott, Paul; Rivera, Rafael
20	10/15/2012	978-0-321-88691-0	Peach Pit Press	Revell, Jeff
21	10/23/2012	978-1-285-42457-6	Cengage Learning	Busch, David
22	11/19/2012	978-0-07-180783-8	McGraw-Hill	Monk, Simon
23	11/23/2012	978-0-470-40765-3	John Wiley & Sons	Kjellstrom, Bjorn; Elgin, Carina
24	12/8/2012	978-1-56881-721-7	CRC Press	Wapner, Leonard
25	12/27/2012	978-0-691-14892-2	Princeton University Press	Van Brummelen, Glen
26	12/28/2012	978-0-7484-0304-2	CRC Press	Snyder, John; Bugayevskiy, Lev
27	1/15/2013	978-0-691-15271-4	Princeton University Press	Gray, Jeremy
28	1/15/2013	978-0-321-81958-1	New Riders Publishing	Kelby, Scott

To answer this question we would extract **Column 2** from the above table on the right and we would append that single column to the table on the left, after we insert an extra blank **Column 3** to the table on the left.

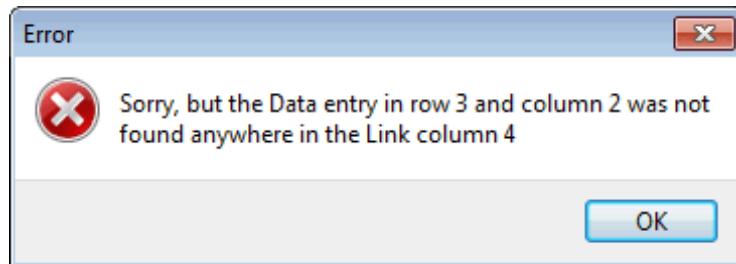
When we open the combined table we would have a list that would appear as shown on the next page.

This means we have essentially combined two tables into one because there are essentially two different sizes of rows in the sets of data.

We are going to use the blank column, **Column 3** as a **Report Column** when we execute the function that performs a **Referential Integrity Check**. The **Report Column** is always cleared (using only the corresponding data rows) before the checks are made. In general database terms, a key column in one table might be supposed to have only elements that come from another key column in another table.

<	Column 1	Column 2	Column 3	Column 4
1 >	Cummings	978-1-56881-721-7		978-1-449-30468-3
2	Spence	978-1-449-30468-3		978-0-691-14039-1
3	Strong	978-0-486-64725-8		978-0-19-974044-4
4	Grant	978-0-262-51668-6		978-0-691-14714-7
5	Vinson	978-0-8176-4704-9		978-0-486-47883-8
6	Hardin	978-0-486-47417-6		978-0-465-01775-1
7	Guzman	978-1-4129-1314-6		978-0-486-47417-5
8	Kennedy	978-0-8176-4372-0		978-0-471-11709-4
9	Haney	978-0-465-01775-1		978-0-8218-4418-2
10	Andrews	978-0-486-47883-8		978-0-691-15270-7
11	Hogan	978-0-691-15265-3		978-0-691-14342-2
12	Gates	978-0-691-14039-1		978-0-321-62930-2
13	Bennett	978-0-471-11709-4		978-0-674-05755-5
14	Alvarez	978-0-596-80552-4		978-1-118-46446-5
15	Shields	978-1-59184-492-1		978-0-307-72095-5
16	Rosa	978-1-56158-891-6		978-1-59420-411-1
17	Gray	978-0-486-65241-2		978-1-59184-492-1
18	Sears	978-0-19-974044-3		978-0-984-72511-3
19	Anderson	978-0-321-88691-0		978-1-118-20413-9
20				978-0-321-88691-0
21				978-1-285-42457-6
22				978-0-07-180783-8
23				978-0-470-40765-3
24				978-1-56881-721-7
25				978-0-691-14892-2
26				978-0-7484-0304-2
27				978-0-691-15271-4
28				978-0-321-81958-1

If we bring up the dialog shown two pages previously and set all the controls as shown, then when the referential integrity report is made we should see an initial error message that appears as:



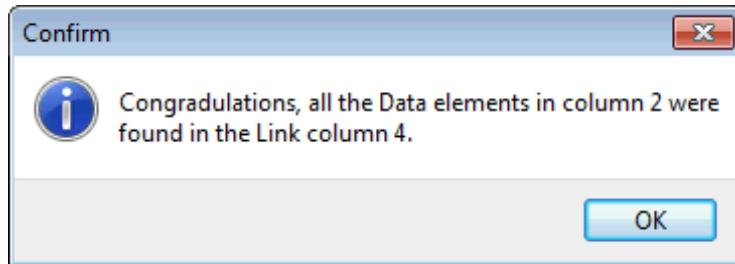
The full report is shown in **Column 3** in the grid on the next page.

<	Column 1	Column 2	Column 3	Column 4
1 >	Cummings	978-1-56881-721-7	Ok	978-1-449-30468-3
2	Spence	978-1-449-30468-3	Ok	978-0-691-14039-1
3	Strong	978-0-486-64725-8	Error Not Found	978-0-19-974044-4
4	Grant	978-0-262-51668-6	Error Not Found	978-0-691-14714-7
5	Vinson	978-0-8176-4704-9	Error Not Found	978-0-486-47883-8
6	Hardin	978-0-486-47417-6	Error Not Found	978-0-465-01775-1
7	Guzman	978-1-4129-1314-6	Error Not Found	978-0-486-47417-5
8	Kennedy	978-0-8176-4372-0	Error Not Found	978-0-471-11709-4
9	Haney	978-0-465-01775-1	Ok	978-0-8218-4418-2
10	Andrews	978-0-486-47883-8	Ok	978-0-691-15270-7
11	Hogan	978-0-691-15265-3	Error Not Found	978-0-691-14342-2
12	Gates	978-0-691-14039-1	Ok	978-0-321-62930-2
13	Bennett	978-0-471-11709-4	Ok	978-0-674-05755-5
14	Alvarez	978-0-596-80552-4	Error Not Found	978-1-118-46446-5
15	Shields	978-1-59184-492-1	Ok	978-0-307-72095-5
16	Rosa	978-1-56158-891-6	Error Not Found	978-1-59420-411-1
17	Gray	978-0-486-65241-2	Error Not Found	978-1-59184-492-1
18	Sears	978-0-19-974044-3	Error Not Found	978-0-984-72511-3
19	Anderson	978-0-321-88691-0	Ok	978-1-118-20413-9
20				978-0-321-88691-0
21				978-1-285-42457-6
22				978-0-07-180783-8
23				978-0-470-40765-3
24				978-1-56881-721-7
25				978-0-691-14892-2
26				978-0-7484-0304-2
27				978-0-691-15271-4
28				978-0-321-81958-1

When we read the entries in **Column 3** we can clearly see most elements in **Column 2** do not also appear in **Column 4**. What we have performed here is easier than trying to compute a set intersection, or a set difference. In fact, at this point we could delete **Column 4** because that column is no longer needed. **Column 4** was essentially imported as a single extra and last column.

We should point out that when you setup the dialog, you should first identify two columns that we refer to as the **Data Column** and the **Link Column**. When the **Referential Integrity Check** is performed, the program will run down all the elements in the **Data Column** and it will search for each of those in the all the rows you specify, but in the **Link Column**. The reason for having two different row ranges is because the number of elements in the **Data Column** is usually very different from the number of elements in the **Link Column**.

Whether you use a **Report Column** or not, the program will always report the first data element, if any, that was not found in the **Link Column**. In a different example where all the elements are **Ok** then you could see a different confirmation message like the following.



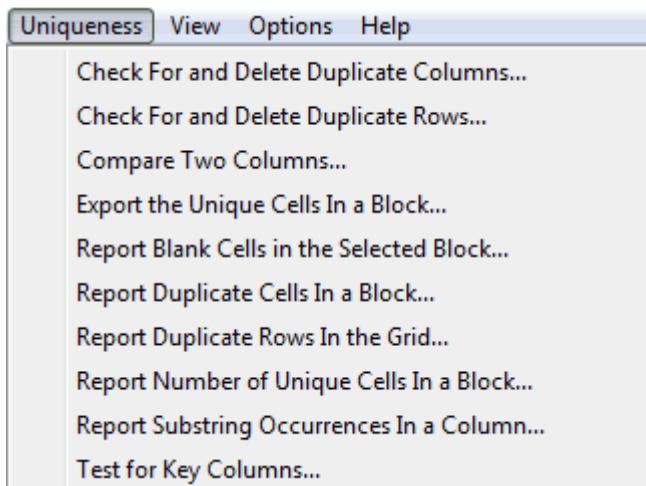
Just remember that when more than one element cannot be found, then only one message will display information, but only for the first error that is found. Using a **Report Column** makes it easy to see all the elements that are both found and not found. When the **Report Column** contains thousands of rows you could always search that column for the word **Error** that always appears whenever a corresponding element cannot be verified as existing in the **Link Column**. If you search for **Error** and request that only the number of occurrences be reported, then you can quickly determine exactly how many **Errors** there were.

It really doesn't matter which of the **Data** and **Link** columns contains more elements. The program always checks the **Data Column** against the **Link Column**. This function is especially useful when both columns contain thousands of rows. In all such cases, you might note the **Link Column** acts like a list of **Validation Strings** as if we were validating strings. Except we didn't load a separate list of validation strings. Instead, we appended an extra column. Another difference is that we allow setting a **Case Sensitive** option in our dialog. Such a **Case Sensitive** option is not available when validating strings.

The function to perform **Referential Integrity** is almost the same as the function under the **Columns Menu** that is named **Two-Column Search and Match**. The main difference is in how the **Report Column** or the **Answer Column** gets filled in.

The Uniqueness Menu

The **Uniqueness** menu contains a few menu items for checking uniqueness of columns or rows or cells and also contains a special function for reporting the number of blank cells in a selected block of cells. You can also determine the number of unique cells in a block and you can export those cells in another file. You can also report all the substring occurrences within a single column. The **Uniqueness** menu appears as:



Although there are only a few menu items, these are all the ones that are needed to perform basic uniqueness tests.

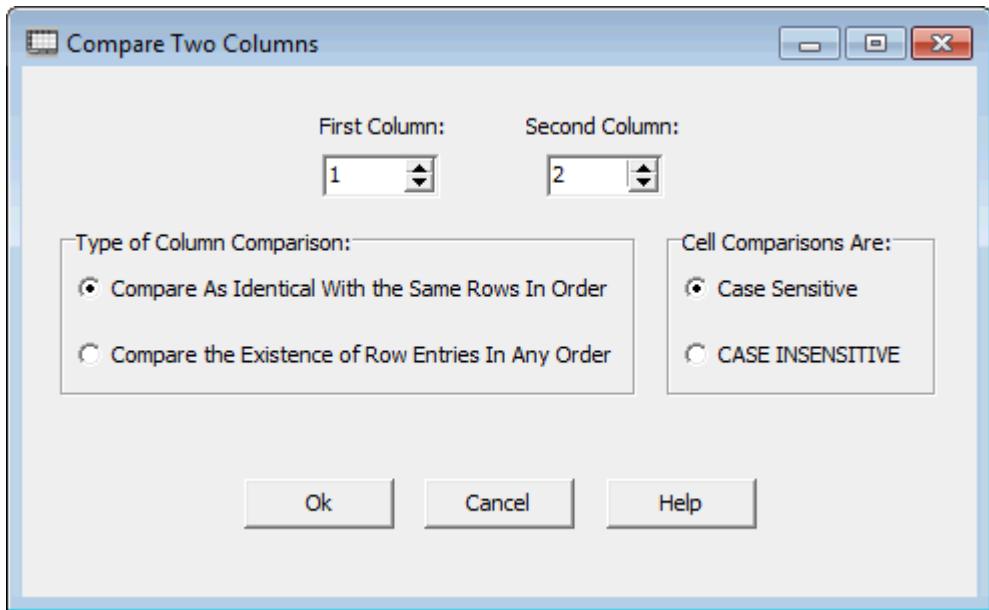
The first two menu items duplicate functions that are under the **Columns** menu and the **Rows** menu and these two functions are documented elsewhere in this help file under those other two menus. These two functions are the only ones under the **Uniqueness** menu that can be used to edit or change your data. These two functions are duplicated here just for your convenience when you are browsing the **Uniqueness** menu items.

Comparing Two Columns

One of the checks for uniqueness under the **Uniqueness** menu is used to compare two columns. However, there are two significant ways in which two columns may be compared. If we wanted to know if the two columns were identical on a row by row basis, then we would compare the columns to see if they were identical in every aspect. In all comparisons, we ignore the yellow header row, if there is a header row.

The other kind of comparison would be where we want to know if the two columns contain identical entries, but the order of those entries might be different in terms of the rows. For example, if one column was sorted and the other column wasn't sorted, it would be possible for the two columns to contain exactly the same entries, but just in two different row orders. In this case it would be nice to verify that the two columns contained exactly the same data. In another case it would be nice to know which cell in either column was missing from the other column.

To compare any two columns in a grid select **Uniqueness | Compare Two Columns...** and you should see the dialog box:



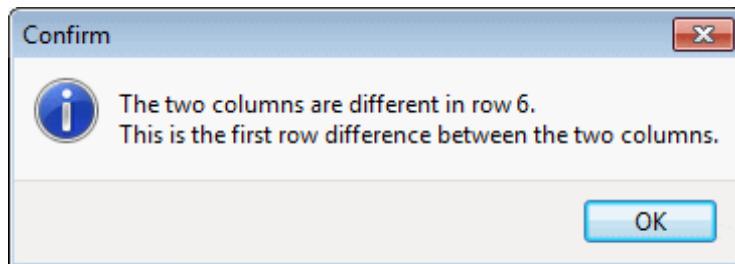
The **Type of Column Comparison** radio buttons are used to select the type of comparison that is to be made with respect to the columns. Although you might think it would be easy to compare two columns by visual inspection when they might be identical, that is not the case when the two columns are far apart in a grid nor is it easy to do when the grid has many rows that are not immediately visible.

A final option to set is how the cell comparisons should be made. The choice is **Case Sensitive** or **CASE INSENSITIVE**.

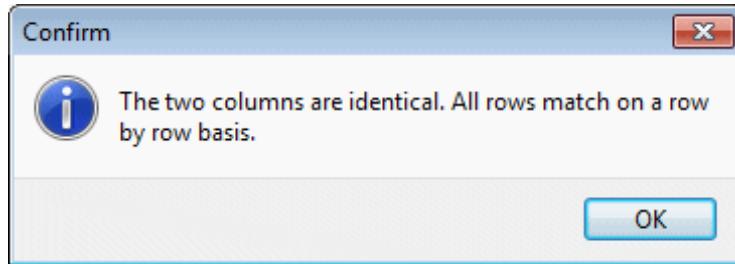
As an example, consider the grid that appears as shown on the next page.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9
1 >	03/01/2016	02/18/2013	11/07/2016	11/30/2016	01/05/2017	07/12/2018	08/21/2014	02/18/2013	03/01/2016
2	05/13/2016	05/20/2015	08/15/2017	02/19/2016	04/25/2018	05/29/2017	08/24/2018	05/20/2015	05/13/2016
3	02/24/2016	01/05/2017	07/08/2016	12/28/2013	05/20/2015	09/05/2015	09/22/2013	01/05/2017	02/24/2016
4	05/15/2013	06/06/2015	04/25/2018	12/24/2013	09/06/2017	04/29/2013	02/20/2018	06/06/2015	05/15/2013
5	10/01/2014	03/06/2013	12/08/2015	08/27/2013	03/06/2013	07/19/2014	10/18/2016	03/06/2013	10/01/2014
6	09/06/2015	01/29/2013	07/20/2017	01/18/2018	06/15/2017	05/31/2018	06/27/2018	01/29/2013	09/14/2016
7	09/01/2015	02/28/2013	09/02/2016	01/31/2018	05/23/2013	04/29/2014	08/15/2017	02/28/2013	09/01/2013
8	09/26/2013	06/15/2017	10/27/2016	04/23/2015	04/14/2018	06/28/2017	11/07/2016	06/15/2017	09/03/2017
9	01/04/2013	04/23/2013	01/01/2013	07/09/2016	01/29/2013	05/12/2014	10/05/2014	04/23/2013	08/11/2013
10	03/30/2014	03/04/2018	05/24/2017	05/09/2013	06/06/2015	03/09/2015	07/25/2014	03/04/2018	02/24/2017
11	09/17/2017	07/29/2015	02/16/2017	01/19/2018	02/28/2013	07/27/2013	08/28/2017	07/29/2015	09/04/2015
12	09/30/2013	04/25/2018	10/08/2013	05/07/2017	07/29/2015	06/16/2015	07/08/2016	04/25/2018	06/14/2014
13	03/30/2015	04/14/2018	11/06/2013	11/21/2015	04/23/2013	03/25/2016	08/01/2017	04/14/2018	05/22/2013
14	01/28/2016	09/06/2017	02/04/2013	03/12/2013	02/18/2013	07/12/2016	04/03/2014	09/06/2017	07/05/2013
15	06/03/2013	05/23/2013	02/16/2016	10/29/2016	03/04/2018	11/02/2013	04/25/2018	05/23/2013	09/16/2015

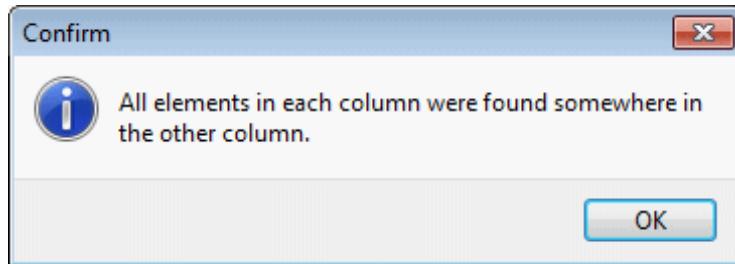
If we were to test Column 1 and Column 9 to see if these two columns were identical, and if we chose the **Type of Column Comparison** as the first radio button, **Compare As Identical With the Same Rows In Order**, then we would see the report come back as:



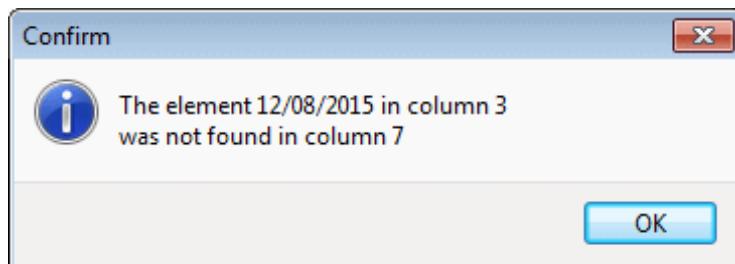
If we were to test Column 2 and Column 8 to see if these two columns were identical, and if we chose the first radio button, **Compare As Identical With the Same Rows In Order**, then we would see the report come back as:



Next, if we compared Column 5 with Column 8, and if we chose the second radio button, **Compare the Existence of Row Entries In any Order**, we would see a different report. This report would be hard to do if you tried doing it by visual inspection of the two columns.



As an example of another report we could compare Column 3 with Column 7, again choosing the second **Type of Column Comparison** radio button.



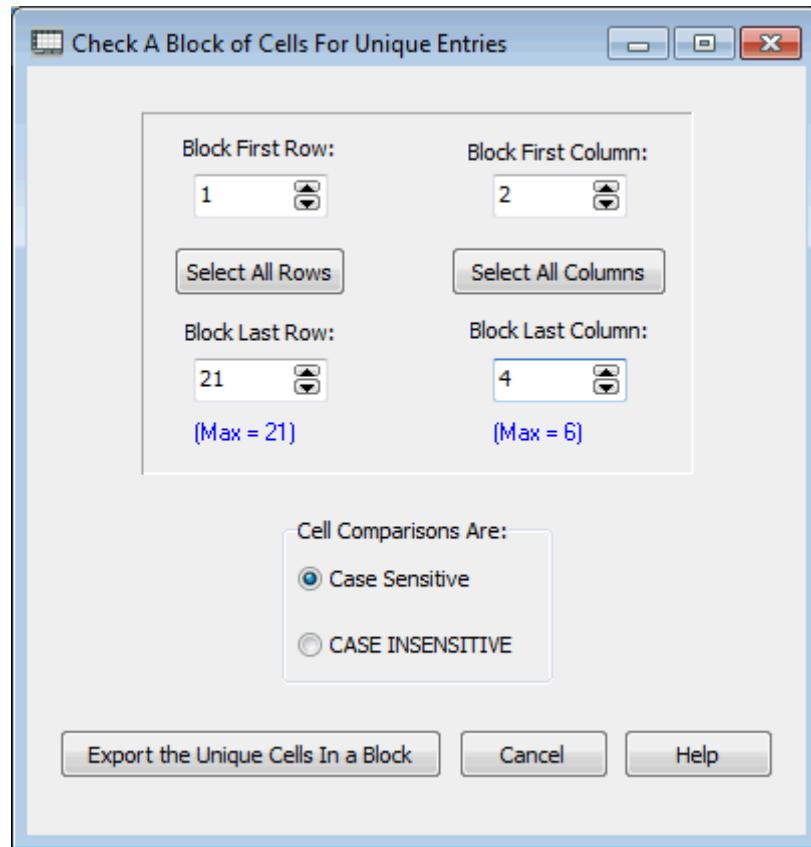
In this last report, only the first non-matching element is what is reported. There could be other elements that would also not be found.

If you select the second option to **Compare the Existence of Row Entries In Any Order** then this function can be time consuming, depending on the number of rows in the grid. To help you understand the progress that is being made, there is a counter that appears in the lower-left corner of the program window in the status line. This counter changes rapidly, but it is a countdown counter and when it reaches 0 the function will be complete and you will then see the final report message.

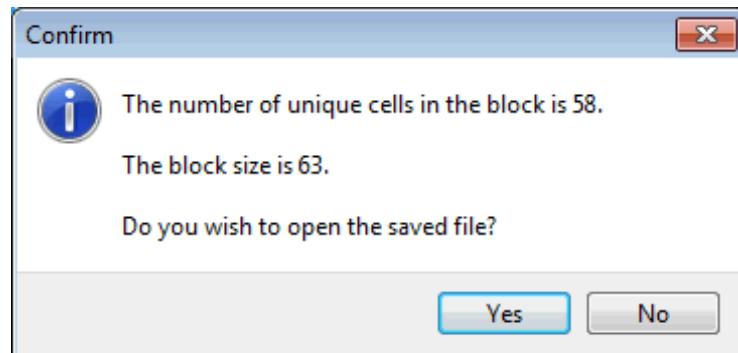
So if you watch the lower-left corner you can gauge how long the computation will take. Just be patient and wait for the final message.

Exporting the Unique Cells In a Block

For those times when you want to export all the unique cells in a block, you can select the function **Uniqueness | Export the Unique Cells In a Block...** and you will see the following dialog:



If we chose the **Academy Awards** grid and selected Columns 2 through 4 inclusive we could get a list of all the unique names of the actors and actresses and directors. When you click the button with the caption **Export the Unique Cells In a Block**, you will first see a standard file save dialog in which you should type the name of the new file you want to save. If you name an existing file you will be given a chance to change your mind or to overwrite the existing file. Next, you will see a brief report like the following:



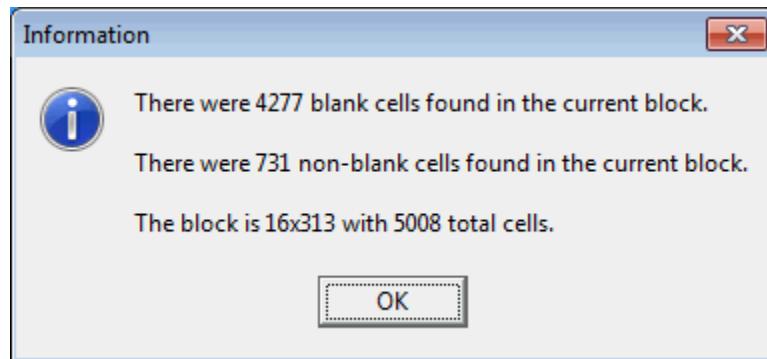
If you click the **No** button then you will remain with the currently opened grid. Otherwise, when you click the **Yes** button the program will open the file you just saved. The unique cells are always saved in a single-column CSV file. For this example the new grid might appear as:

<	Column 1
>	Burt Lancaster
1	Elizabeth Taylor
2	Billy Wilder
3	Maxmillian Schell
4	Sophia Loren
5	Robert Wise
6	Gregory Peck
7	Anne Bancroft
8	David Lean
9	Sidney Poitier
10	Patricia Neal
11	Tony Richardson
12	Rex Harrison
13	Julie Andrews
14	George Cukor
15	Lee Marvin
16	Julie Christie
17	Paul Schofield
18	Fred Zinnemann
19	Rod Steiger
20	Katherine Hepburn
21	Mike Nichols
22	Cliff Robertson
23	Sir Carol Reed
24	John Wayne
25	Maggie Smith
26	John Schlesinger
27	George C. Scott
28	Glenda Jackson
29	Franklin Schaffner

You may want to turn the header row off at this point and then you could insert a new header row and edit its contents. There are actually 58 names in this list, but here we only show about half of them.

Reporting Blank Cells In a Block

There is a menu item under the **Uniqueness** menu that counts how many blank cells there are in the currently selected block of cells. Thus before using this menu item you should first select a rectangular block of cells. Then when you click the menu item **Uniqueness | Report Blank Cells in the Selected Block...** you will see a message like the following:



This function is smart enough to also tell you how many non-blank cells there were in the block as well as the total number of cells in the block.

Reporting Duplicate Cells In a Block

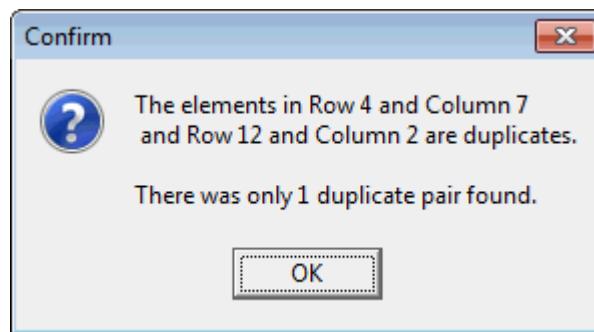
A choice under the **Uniqueness** menu is used to see if any cells in a block contain any duplicate entries. This differs from checking for duplicates within a single column because it can be applied to a very large block and it does not require any kind of sorting before it can be applied.

An example of where this function is really useful is the following. Imagine you have a grid full of random dates and you need to know if all these dates are unique. Such a grid might appear as shown next:

Are all the entries in the grid below unique?

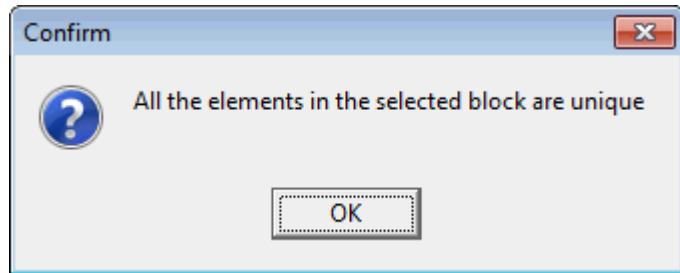
<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8
1 >	03/01/2016	02/18/2013	11/30/2016	09/30/2018	07/12/2018	08/21/2014	11/07/2016	02/22/2013
2	05/13/2016	05/20/2015	02/19/2016	05/26/2016	05/29/2017	08/24/2018	08/15/2017	01/19/2015
3	02/24/2016	01/05/2017	12/28/2013	07/17/2017	09/05/2015	09/22/2013	07/08/2016	02/05/2014
4	05/15/2013	06/06/2015	12/24/2013	05/03/2017	04/29/2013	02/20/2018	04/25/2018	10/11/2015
5	10/01/2014	03/06/2013	08/27/2013	01/13/2017	07/19/2014	10/18/2016	12/08/2015	04/17/2013
6	09/06/2015	01/29/2013	01/18/2018	07/30/2015	05/31/2018	06/27/2018	07/20/2017	09/14/2016
7	09/01/2015	02/28/2013	01/31/2018	02/08/2018	04/29/2014	11/21/2018	09/02/2016	09/01/2013
8	09/26/2013	06/15/2017	04/23/2015	04/28/2016	06/28/2017	01/20/2017	10/27/2016	09/03/2017
9	01/04/2013	04/23/2013	07/09/2016	05/11/2014	05/12/2014	10/05/2014	01/01/2013	08/11/2013
10	03/30/2014	03/04/2018	05/09/2013	06/18/2014	03/09/2015	07/25/2014	05/24/2017	02/24/2017
11	09/17/2017	07/29/2015	01/19/2018	04/07/2016	07/27/2013	08/28/2017	02/16/2017	09/04/2015
12	09/30/2013	04/25/2018	05/07/2017	02/13/2013	06/16/2015	11/04/2015	10/08/2013	06/14/2014
13	03/30/2015	04/14/2018	11/21/2015	09/20/2016	03/25/2016	08/01/2017	11/06/2013	05/22/2013
14	01/28/2016	09/06/2017	03/12/2013	01/22/2014	07/12/2016	04/03/2014	02/04/2013	07/05/2013
15	06/03/2013	05/23/2013	10/29/2016	06/06/2018	11/02/2013	11/22/2013	02/16/2016	09/16/2015

The answer is no, but if you haven't found the equal entries, you should spend a couple of more minutes trying to find them. Ok, give up? If you choose the menu item **Uniqueness | Report Duplicate Cells In a Block..** and choose the entire grid in the dialog that comes up, you will be able to generate the following kind of a report.

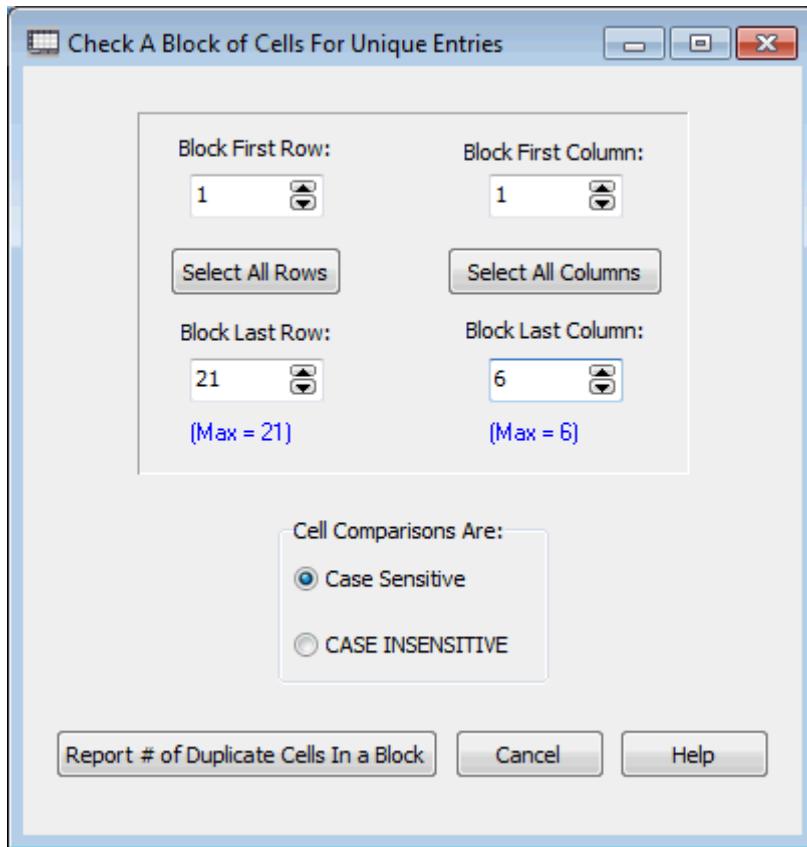


Now you should be able to appreciate the power of this function. It is not easy to find the same elements in the above grid, and this is a very small grid that you can visually see all at once.

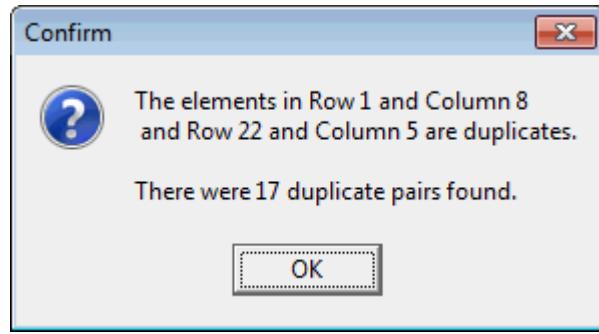
If this grid had 150 rows instead of 15 it would be nearly impossible to find two equal entries by simply trying to read the entire grid. When a grid has no duplicates you will see a different message like:



In any case, under the **Uniqueness** menu when you bring up the dialog to report the number of duplicate cells in a block you will see the dialog:



After you set the rows and columns of the grid to be checked, you should also choose the type of **Case Sensitivity** you would like to apply. The default means the string comparisons are not case sensitive. For a different grid than the above example, you might see a report like the following. In this case the first duplicate pair is reported and we also learn how many duplicates there were overall.



This function can be time consuming, depending on the number of cells in the selected block. To help you understand the progress that is being made, there is a counter that appears in the lower-left corner of the program window in the status line. This counter changes rapidly, but it is a countdown counter and when it reaches 0 the function will be complete and you will then see the final report message.

So if you watch the lower-left corner you can gauge how long the computation will take. Just be patient and wait for the final message.

Reporting the Number of Unique Cells

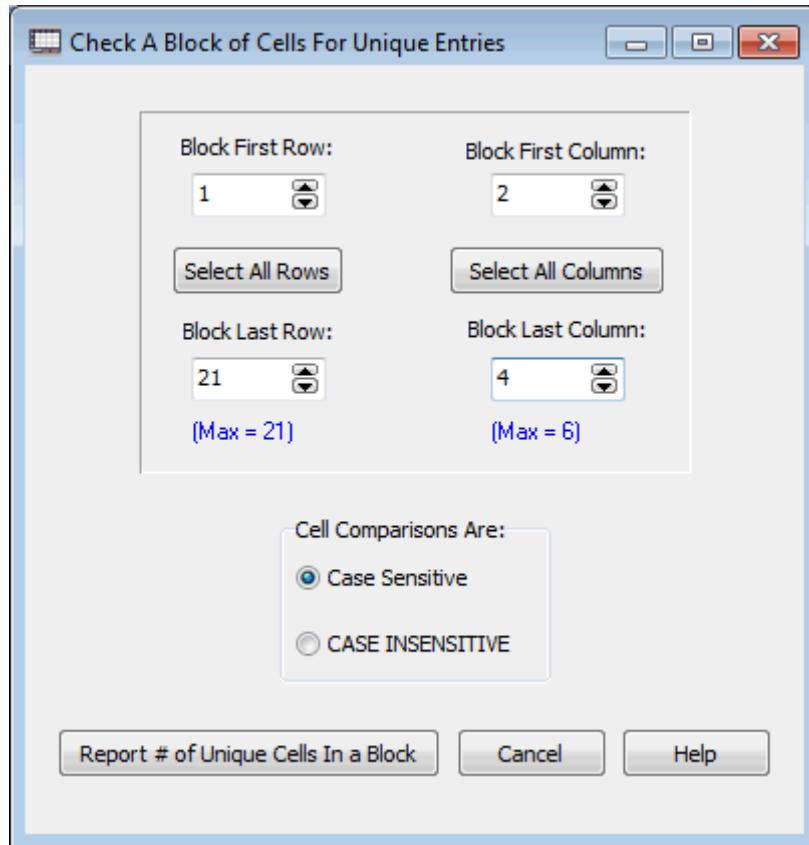
Sometimes you don't want to know which cells in a block are duplicates of each other, nor do you want to export the unique cells in a block. You may simply want to know how many cells in a given block are distinct or unique.

As an example, if you opened the **Academy Awards** grid and chose the block as shown below, we might want to know how many of the selected names are unique.

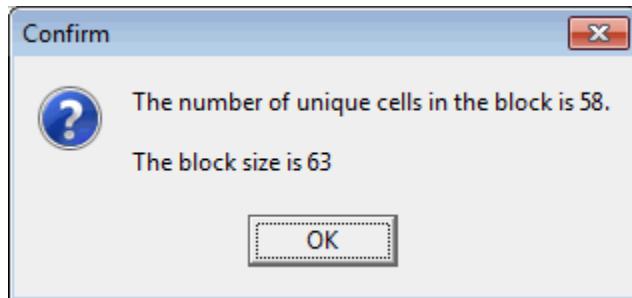
<	Column 1	Column 2	Column 3	Column 4	Column 5
>	Year:	Best Actor:	Best Actress:	Best Director:	Best Picture:
1	1960	Burt Lancaster	Elizabeth Taylor	Billy Wilder	The Apartment
2	1961	Maxmillian Schell	Sophia Loren	Robert Wise	West Side Story
3	1962	Gregory Peck	Anne Bancroft	David Lean	Lawrence of Arabia
4	1963	Sidney Poitier	Patricia Neal	Tony Richardson	Tom Jones
5	1964	Rex Harrison	Julie Andrews	George Cukor	My Fair Lady
6	1965	Lee Marvin	Julie Christie	Robert Wise	The Sound of Music
7	1966	Paul Schofield	Elizabeth Taylor	Fred Zinnemann	A Man For All Seasons
8	1967	Rod Steiger	Katherine Hepburn	Mike Nichols	In The Heat Of The Night
9	1968	Cliff Robertson	Katherine Hepburn	Sir Carol Reed	Oliver
10	1969	John Wayne	Maggie Smith	John Schlesinger	Midnight Cowboy
11	1970	George C. Scott	Glenda Jackson	Franklin Schaffner	Patton
12	1971	Gene Hackman	Jane Fonda	William Friedkin	The French Connection
13	1972	Marlon Brando	Liza Minelli	Bob Fosse	The Godfather
14	1973	Jack Lemon	Glenda Jackson	George Roy Hill	The Sting
15	1974	Art Carney	Ellen Burstyn	Francis Ford Coppola	The Godfather Part II
16	1975	Jack Nicholson	Louise Fletcher	Milos Forman	One Flew Over The Cuckoos Nest
17	1976	Peter Finch	Faye Dunaway	John G. Avildsen	Rocky
18	1977	Richard Dreyfuss	Diane Keaton	Woody Allen	Annie Hall
19	1978	Jon Voight	Jane Fonda	Michael Cimino	The Deer Hunter
20	1979	Dustin Hoffman	Sally Field	Robert Benton	Kramer vs Kramer
21	1980	Robert De Niro	Sissy Spacek	Robert Redford	Ordinary People

To determine that, you should select the menu item **Uniqueness | Report Number of Unique Cells In a Block...** and you will essentially see the same dialog used to actually export the unique cells in a block. However, the action of this dialog differs in that the output that is produced is very simple. We mention that the same dialog has a dual usage so you won't be confused. You will see different help topics, depending on which **Uniqueness** menu item was chosen and you will see different captions on the main function button. For reporting the number of unique cells in a block the main function button will have the caption **Report # of Unique Cells In a Block.**

In any case, if you accept the default values as shown on the next page,



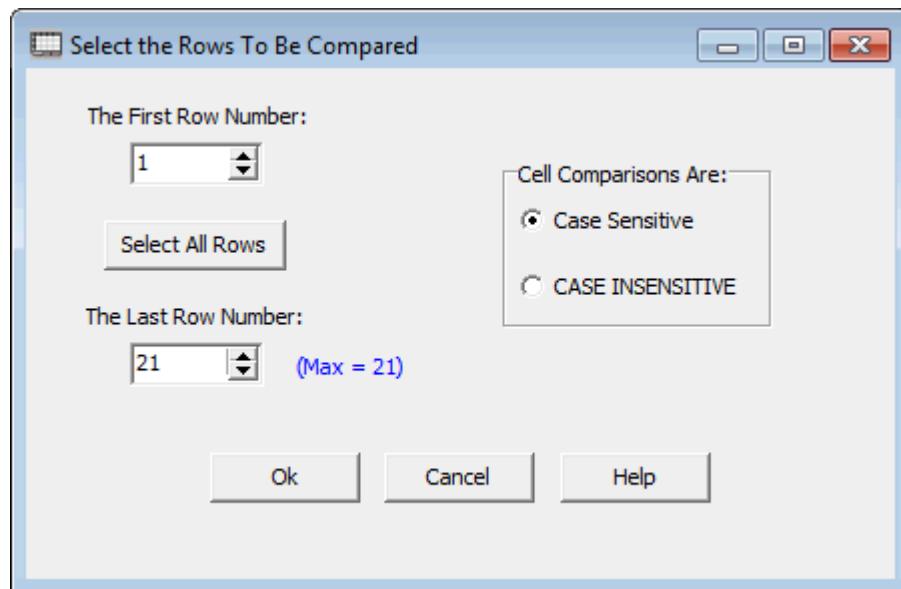
then after you press that function button, you should see the message:



The original grid had 63 selected names, but only 58 of these are unique names. Four of the women and one of the men appear twice in the block. Note that $63 - 58 = 5$. Five of the names are duplicates.

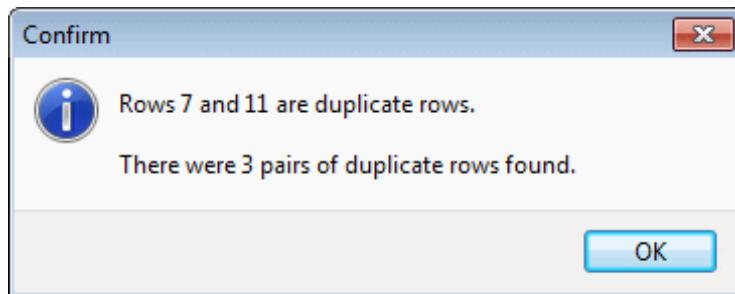
Reporting Duplicate Rows In the Grid

Another check for uniqueness under the **Uniqueness** menu is used to check for unique rows in an entire CSV grid. In this case, when we say the entire CSV grid we mean that the row comparison necessarily uses all the columns. You can still select a subrange of all the rows. Thus you can apply this check to any set of consecutive rows. You don't have to use all the rows. When you select **Uniqueness | Report Duplicate Rows In the Grid...** you will see the following dialog.



This kind of a check does not require any kind of sorting, because it would not make sense to select one key column. So it is best to just think that this function only compares any two rows. This function might be useful if you were trying to determine a set of key columns for a database table, although the **CSV Editor** program has another function that is devoted to that particular task. When the rows are not unique that is evidence of the need to perhaps add another key column or to change the data. When comparing one row with another row, the program has to compare cells on a column by column basis, so we also give you an option to handle the case sensitivity of strings.

If any two rows are equal you will see a report telling which first pair of equal rows were found. You will also see how many other total pairs of rows were equal. An example report is the following:

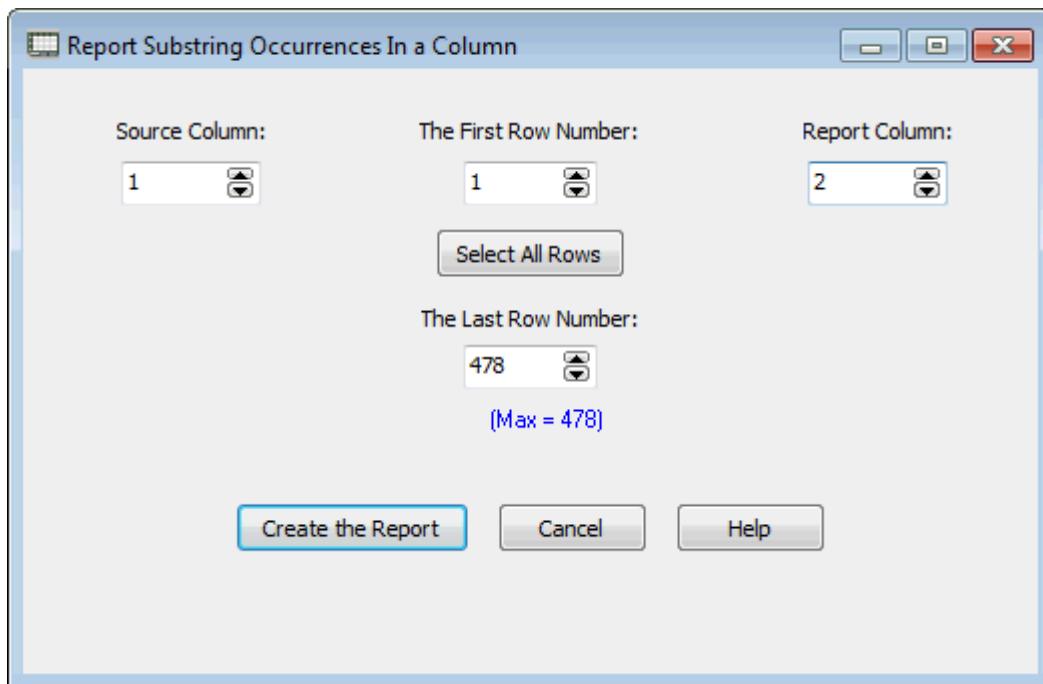


This function can be time consuming, depending on the number of rows and the size of the grid. To help you understand the progress that is being made, there is a counter that appears in the lower-left corner of the program window in the status line. This counter changes rapidly, but it is a countdown counter and when it reaches 0 the function will be complete and you will then see the final report message.

So if you watch the lower-left corner you can gauge how long the computation will take. Just be patient and wait for the final message.

Report Substring Occurrences In A Column

There is a special function under the **Uniqueness** menu that allows you to determine which cells are substrings in other cells, all within a single column. To perform this function select **Uniqueness | Report Substring Occurrences In a Column...** and you will bring up a dialog box like the following:



You need to specify a **Source Column** and a **Report Column** and after that you can select a range of applicable rows. The **Source Column** contains the strings (really substrings) that will be used to perform searches with, and also contains the strings that will be checked for substring occurrences.

When you click the button with the caption **Create the Report**, the program will clear all the cells in the **Report Column** that correspond to the selected range of rows. Then the program will select each cell in the **Source Column** and it will check if that cell occurs as a substring in any other cell within that same **Source Column** (excluding of course the row that the cell was selected from).

A typical response in a **Report Column** cell will be an empty cell if the substring is not found in any other rows. However if the substring is found one or more times in other rows then a typical **Report Column** cell will contain a response like:

```
3  Row List = [ 82; 323; 431 ]
```

The first number **3** means the substring was found three times. Then what follows is a **Row List** that contains a list of row numbers inside square brackets. Each row number is separated from the next by a semi-colon character. So the meaning of the above report cell would be that the string in the **Source Column**, but in the same row, occurs as a substring in rows **82** and **323** and **431** in the grid. Thus you not only know how many times the substring occurs, you also know exactly which other rows it occurs in.

Consider the following grid where row 81 and column 1 contains the string **comma**. The report for comma in this grid is what is shown on the previous page in blue text. You can inspect various rows in the grid, but mainly rows 81 and **82** and **323** and **431** where the last two rows only show what is Column 1.

<	Column 1	Column 2
65	character_	
66	cities lists	
67	clearing a column	
68	clearing a row	
69	clipboard	1 Row List = [465]
70	code point	
71	code space	
72	colon character	1 Row List = [365]
73	column character extents	
74	column display widths	
75	column move	1 Row List = [145]
76	column report	
77	column shortest	
78	column tallest	
79	Columns menu	
80	combining columns	
81	comma	3 Row List = [82; 323; 431]
82	command line parameters	
83	comparing columns	
84	comparing files	
85	comparison report	
86	conditional replacements	
87	constant payments	
88	Conversions menu	
89	converting data	
90	copy and paste	
91	copy matching data	
92	country names	
93	CR-LF Movement	

315	random passwords
316	random strings
317	random times
318	range check
319	raw bytes
320	read order
321	recent files list
322	recurse subdirectories
323	redo command
324	referential integrity
325	regression function
326	remove (word)
:	
425	toolbar
426	top (row move)
427	transpose of a grid
428	transpose read
429	trees
430	two-column search and match
431	undo command
432	Unicode
433	uniform distribution
434	union of sets
435	unique cells
436	unique cells exporting
437	unique columns

The first reported row number is **82** and when we look at the string in that row, **command line parameters**, we can see that **comma** is a contained substring.

The next reported row number is **323** and when we look at the string in that row, **redo command**, we can see that **comma** is a contained substring.

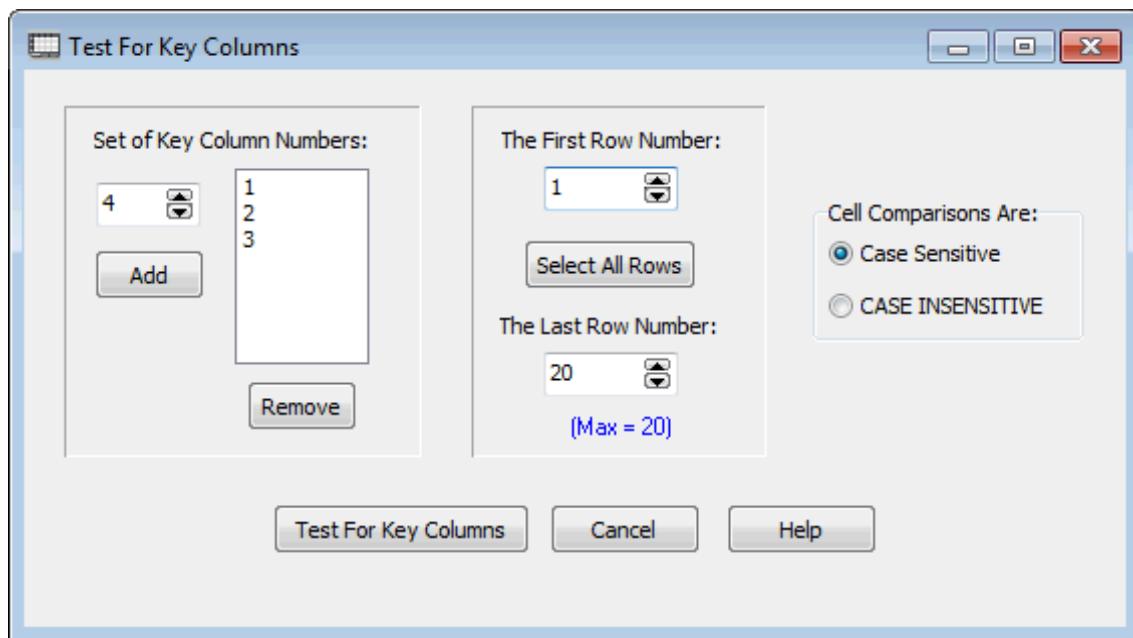
The last reported row number is **431** and when we look at the string in that row, **undo command**, we can see that **comma** is a contained substring.

While we have explained only one report result cell in detail, this should be sufficient for you to understand the complete nature of this function.

This function is useful when you want to determine which text entries in a book or document index are contained in other text entries within that same index.

Testing For Key Columns

If you know some database theory, then you know that the rows in a grid can sometimes be considered to be uniquely determined by a set of one or more columns in that same grid. The **CSV Editor** program has a special function that can be used to determine which set of columns in a grid might constitute a set of database key columns for that grid. To bring up the dialog that is used to perform the test, select **Uniqueness | Test For Key Columns....** You should then see the following dialog.



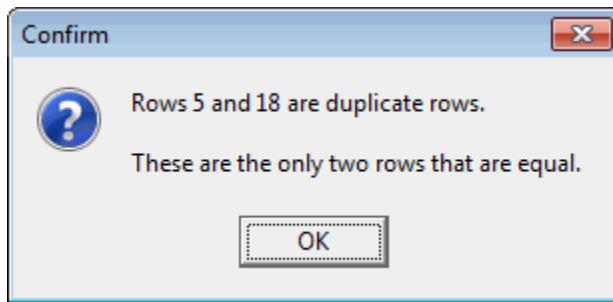
You use the controls on the left side to make a list of potential database key column numbers. Such a list usually only includes a few column numbers and rarely contains all the column numbers. In the above figure, the column numbers are simply the numbers 1, 2, and 3 that appear vertically in the list above the **Remove** button. You use the controls in the middle to select a range of rows to which the test for database key columns will be applied. Most of the time you will want to select all the rows so you can test an entire grid.

Testing for a set of database key columns is not something you can do by just looking at a grid. One reason is that when you look at a grid you may not see all the columns or all the rows at one time. In any case, you need a computer program to do this kind of checking because the human visual checking method cannot be applied reliably, even when you can see almost all the data at one time.

We will use the grid shown at the top of the next page for our test data. We will simply run a few tests that will check to see if certain combinations of columns might be used as key columns for a database table. All of our tests will assume cell comparisons are **Case Sensitive**. If and when you choose the **CASE INSENSITIVE** choice then all grid cells will temporarily be converted to upper case before any cell comparisons are made. Usually if your data is properly formatted then you will want to choose the **Case Sensitive** option.

<	Column 1	Column 2	Column 3	Column 4	Column 5
>	First Name:	Last Name:	Birth Date:	Street Address:	Social Security Number:
1	Camila	Burns	10/03/1984	802 Bittersweet	427-35-0021
2	Paula	Nielsen	11/05/1982	128 Shafer	621-63-4953
3	Ava	Curry	05/17/1985	850 Arno	568-36-5963
4	James	Horn	01/09/1997	732 Temescal Canyon	217-63-7854
5	Julie	Holmes	01/25/1989	983 Bluffside	706-77-9899
6	Kay	Hill	06/13/2006	658 Lamar	914-94-7289
7	Taylor	Jacobson	05/09/1994	142 Windermere	452-33-6002
8	Samantha	Deleon	05/25/1988	198 Summerlyn	413-81-2096
9	John	Shepherd	03/05/2006	344 Sunrise	586-89-2679
10	Kay	Stuart	02/20/1985	494 Las Lomas	598-13-2214
11	Taylor	Larson	01/17/2001	529 Windfield	606-97-9853
12	Paige	Jacobson	06/28/1998	209 Salerno	381-20-3716
13	Kim	Russo	12/05/2010	159 Longford	700-39-9379
14	Alexis	Hess	03/27/1985	91 15th	389-97-1894
15	Paula	Albert	11/18/2009	107 Sea Reef	568-77-7469
16	Ronald	Willis	04/22/1998	686 Brampton	561-08-7326
17	Gabriella	Bush	10/22/2003	925 De Marisa	612-96-8530
18	Julie	Holmes	01/25/1989	512 Crocus	331-19-3484
19	Stella	Shepherd	03/05/2006	344 Sunrise	341-98-3626
20	Edith	McMahon	05/04/1990	247 Charm Acres	330-82-3006

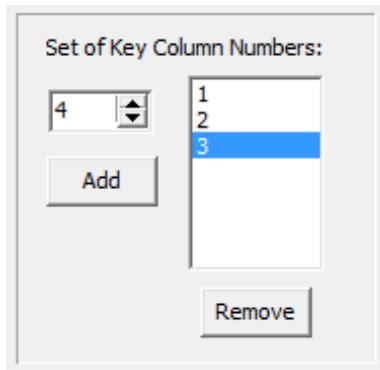
The first test we will run will use the values shown in the dialog on the previous page. After we open the dialog and setup the values and press the button **Test For Key Columns** the first report we see is:



All reports are subject to interpretation. In this case the report is claiming that rows 5 and 18 are equal, which anyone can clearly see is not the case if we look at either the Street Address or the Social Security Numbers. However, the meaning of equal rows as part of this key test function means that we only look at and compare the selected columns in the list. For the column numbers 1 and 2 and 3 in this example, these columns are such that rows 5 and 18 are the same and thus are considered duplicate rows. To qualify as a set of database key columns we want all the rows to be unique when considering only the listed columns.

For our next test we will change the column list to only look at columns 2 and 4, but we will still apply the test to all the rows from 1-20. Whenever you want to change the list of **Key Column Numbers** you may need to first remove some numbers from that list.

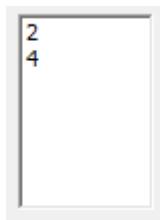
To remove any number from the list you should first click on the number to be removed and you will see that number highlighted in the color blue as shown below.



Once a number in the list is highlighted with the blue color, you can click the **Remove** button and only that highlighted number will be deleted from the list. Usually you will have only a few numbers in the list.

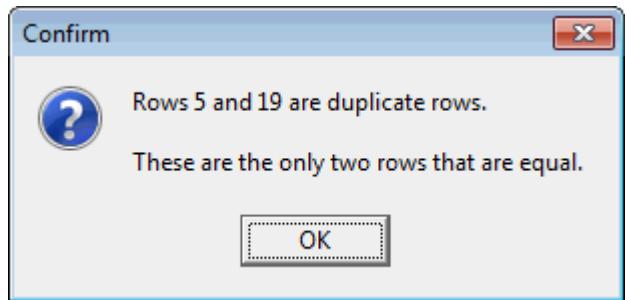
To add a number to the list, first use the up/down arrows in the left control to get to the column number. Then press the **Add** button that appears below the control. The number in the control will be added to the list and the control will automatically advance by 1 or wrap around to 1 after the last column is added. For the above figure, if we were to click the **Add** button, the number 4 would get added to the list and the number counter would automatically advance to show 5.

The list of column numbers is always kept sorted. No column number can be entered in the list more than once. This list of column numbers can be empty, but in that case no test will be applied and you will see a warning message when you try to apply the test.



For our next test the list of selected columns should look like:

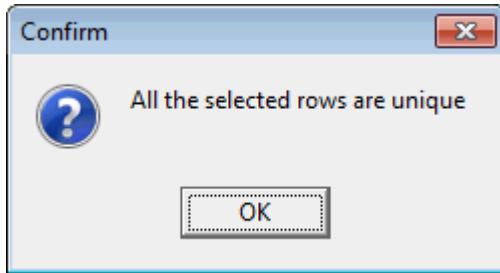
. When we execute the test with



only these two column numbers the report says:

When we look at the data on the previous page we can understand that rows 9 and 19 might represent either a married couple or a brother and sister that live at the same address. In this case we did not include the First Name column number in the list. So when the report says Rows 9 and 19 are duplicate rows, it only means they have the same data in columns 2 and 4, our two test columns.

If we choose four column numbers as the numbers 1, 2, 3, and 4 then the report says:



In this case we have included enough columns to get a unique answer. In general this might mean that the first four columns are sufficient to determine a unique database key. Of course there are no guarantees in real life, but in general most of us would expect that if we included both first and last names and included birth dates and addresses that this amount of information should be enough to uniquely identify an individual.

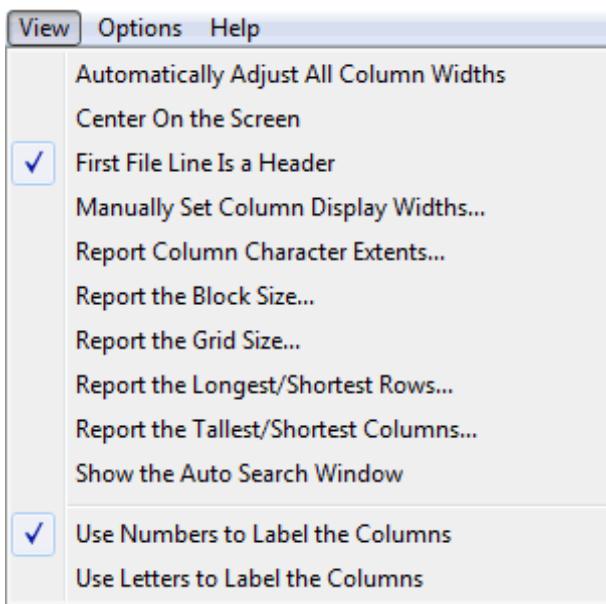
If you know about Social Security numbers you would also expect that using that single column should constitute a database key because Social Security numbers are expected to be unique. (Even identity thieves know this should be true.). If you try running a report with only column 5 in the selected column list, will you get the same report that says all the rows are unique. This means Social Security numbers alone are sufficient to uniquely determine any individual, or to determine a unique row in the grid.

Just because you get a report saying the rows are unique does not mean you should use the selected columns to define a database key. The reason is that your sample table may not happen to contain enough data to show up such a deficiency. Most database designers would not use the Last Name and Birthdate and Street Address to guarantee a unique individual. After all, if there were twins born in the same household they would share these characteristics, but these characteristics alone do not determine a unique individual.

In database terms, the values in a set of key columns are sufficient to uniquely determine what row those values belong to. If you change one entry in one of the key columns then you must also change to a different row in the table. Not all database tables are required to have a set of key columns, but in practice most tables do have a set of key columns that serve as an aid to sorting and indexing and to quickly finding information in a table.

The test for key columns is somewhat related to the test for unique rows. The difference is that when you test for unique rows you really end up using all the columns. Testing for key columns only uses the key column list, but in rare cases you could include all the columns in that list. The test for key columns avoids having to extract the column set to an intermediate table and then test for uniqueness of rows in that extracted column table.

The View Menu



The **View** menu contains a few items that help you control the view of the grid. The main items are concerned with setting the Column Widths, treating the first data row as containing header information, or not, and using either numbers or letters to label the columns. There is also an **Auto Search Window** that can be used for quick searching. The last two menu items are concerned with launching files and displaying pictures that are contained in files, where the filenames or perhaps the entire paths to the files are contained in the cells of a single column.

The first **View** menu item allows the program to automatically determine the displayed width of all your columns. You can also just grab and drag a column header dividing line to manually re-size the display width of any column. Clicking the toolbar button performs the same function as selecting the first menu item. Using the toolbar button is faster and more convenient than opening and selecting the menu item.

The second menu item **Center On the Screen** provides a quick way to exactly center the program window on your computer screen.

The third menu item, **First File Line Is a Header**, is a toggle switch that changes whether the first line is used to display header information or not. When this option is selected or turned on then the top of the grid will have two yellow lines. The first yellow line is used to display the column numbers as usual, but the second yellow line is used to display header information for the columns. Yellow parts of the grid are intended to be information only items. This means these areas are not used for normal editing. For example, using search and replace or any other type of actual editing, like selecting cells, never applies to the yellow parts of a grid. However, when you save a file then any second yellow row header information is saved with the file. If you need to edit the header information all you need do is turn off this option that treats the first file line as header information. Then you can edit that information and then turn this option back on.

An alternative to clicking the menu item **First File Line Is a Header** is to just click the far right toolbar button that appears as either  or  . The resulting button caption toggles to indicate whether the header option is turned on or off. It is faster to click the toolbar button than to open the **View** menu and then select a submenu item. If you have only 1 row of data then the above menu item and this button will both be changed to the option being turned off. You can't turn this option on if you have only 1 row of data.

There are several **View** menu items that are used to report various aspects of the current grid, such as the current block size or the grid size as well as the longest/shortest rows and the tallest/shortest columns.

One of the **View** menu items duplicates a function of a toolbar button for showing or hiding the **Auto Search Window**.

There are two **View** menu items that are used to select one of two options for labeling the columns.

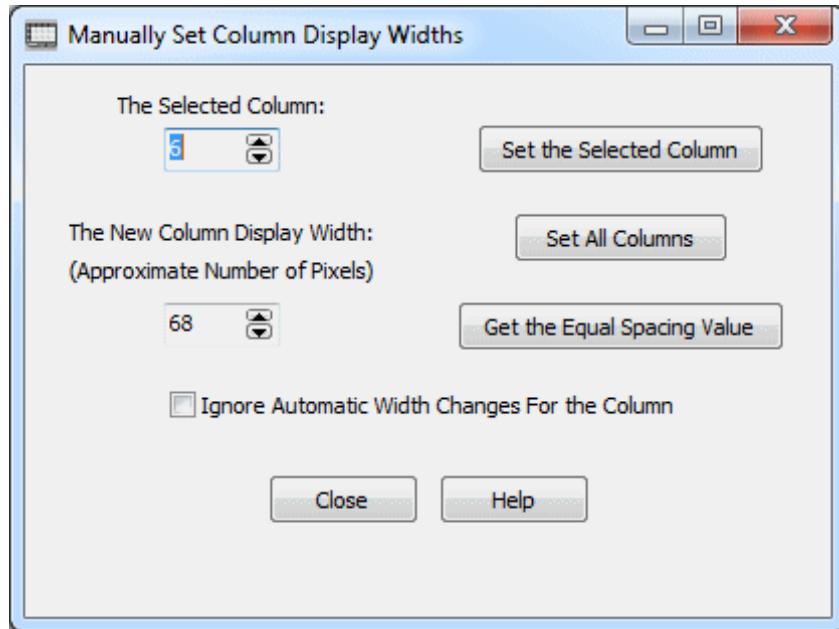
Normally the columns are labeled using numbers (the default). In fact, all dialog boxes that expect you to select a range of columns do so in terms of using column numbers. You can select the **View** menu item to label the columns using letters such as A-Z for the first 26 columns and using letter names like AB-AZ, BA-BZ, and CA-GR for all remaining columns, up through the program limit of 200 maximum total columns.

We provide the option to use letter names for columns simply because most spreadsheet programs do the same. If you feel more comfortable with letter names, then by all means you should select that **View** menu option. However, most functions of the program expect and use numbers to identify the columns to be operated on. You can quickly select between displaying numbers and/or letters as needed.

Manually Setting Column Display Widths

The **View** menu item, **Manually Set Column Display Widths...**, will normally only be used when you have loaded a **CSV** file in which one or more of the columns contains an unusually large amount of text. Such columns may have display widths that are equal to or larger than the entire grid control width. If that is the case, then you won't be able to change the column display width by grabbing a yellow column header division line and dragging it with the mouse. If you can't resize a column using your mouse, then should first click on any row in that column and then select this menu item to help you control the display width. You should then see a dialog box similar to that shown below.

You can change **The Selected Column** number to review the current information for any column.



Usually setting the approximate number of pixels to a value like 100 will be sufficiently small that you can later (after you close this dialog) grab a yellow column header division line and then drag it with the mouse to further resize the column display width. Incidentally, the largest number of pixels you can enter is 9999.

The checkbox with the caption **Ignore Automatic Width Changes For the Column** means the program will not attempt to resize the selected column, even when the program must automatically resize all columns. However, the program will only maintain this state while you continue to edit the current grid. If you load a new grid, then this checkbox state will be cleared or reset for all columns in that new grid.

If you insert or delete other columns, the program will still try to keep the same constant width for the selected column, even with its new column position. After you close the above dialog, if you manually drag the column header to resize the selected column with the mouse, the program will try to maintain the new column display width, provided you checked the checkbox with the caption **Ignore Automatic Width Changes For the Column**. So any manual column width changes you make will be accepted. Only the width changes the program tries to make automatically will be ignored.

At any time you can click the button with the caption **Set the Selected Column** and the program will set the information for just the selected column. The two values that are set are the column display width and the Boolean value to ignore automatic width changes for the current edit session. If you click the button with the caption **Get the Equal Spacing Value** the program will compute a suggested display width, but this width will never be smaller than 75. If you click the button to **Set All Columns**, the program will set the same width for all the columns in the grid and for each column it will also set the Boolean value to ignore automatic width changes for the current edit session, depending on whether the checkbox is checked or not.

Column Character Extents Report

To generate a report that tells both the minimum and the maximum number of characters used in each column, just select **View | Report Column Character Extents...**. You will see a report similar to the following in which the 3rd and 5th columns tell the minimum and maximum number of characters used by all cells in the column whose row number is in Column 1. The 4th column tells what row achieved the minimum while the 6th column tells what row achieved the maximum. The blue text just below the grid tells how many total characters are in all the cells of the entire grid and it tells how many characters are in the currently selected column.

Column #:	The Title:	Min. Width:	Min. Row #:	Max. Width:	Max. Row #:
1	Year	4	1	4	1
2	Best Actor	10	6	17	2
3	Best Actress	10	12	17	8
4	Best Director	9	13	20	15
5	Best Picture	5	17	30	16
6	Tagged	1	1	1	1

Total grid chars = 1,248 Total chars in column #4 = 285

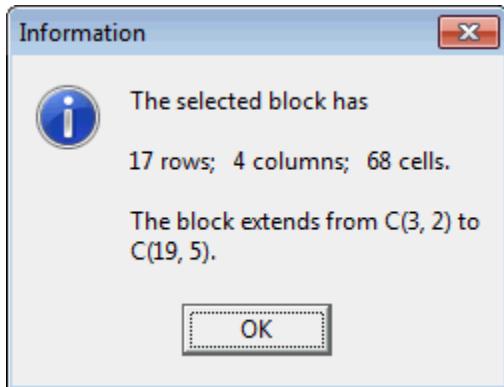
Ok Print the Report Help

This report is useful when you need to create the structure for a database application that will read in the data from the **CSV** file. The maximum character counts can be used to define the widths of the database columns.

An example report that is made from the Academy Awards **CSV** file would appear as shown above. In this report, the last column tells the first row number in the grid in which the given column actually achieved the maximum character count. Of course it is possible that a later row could also have the same maximum character count. This row number might be used to manually edit the data in the given cell to achieve a little smaller maximum count. Also note there are **1,248** total characters in this example grid and there are **285** characters in the current column which was **column #4**.

Reporting the Block Size

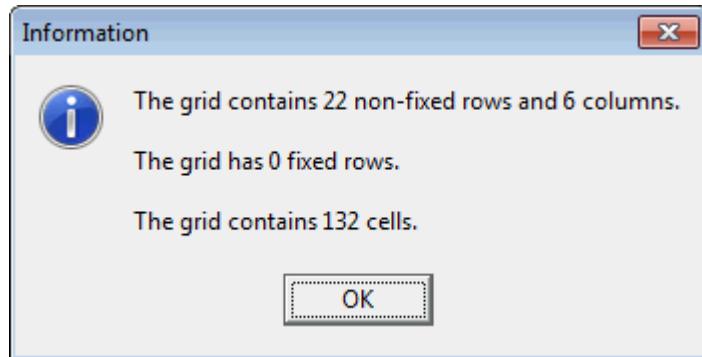
There may be times when you have selected a large block of cells and you want to know the basic information about the block. All you need do is select **View | Report the Block Size...** and you will see a simple report like the following.



The example grid coordinates like **C(3,2)** and **C(19,5)** refer to the upper left and lower right corners of the grid and use our notation for a cell reference. This notation is formatted as **C(row,column)**. So the row number comes first and the column number is second.

Reporting the Grid Size

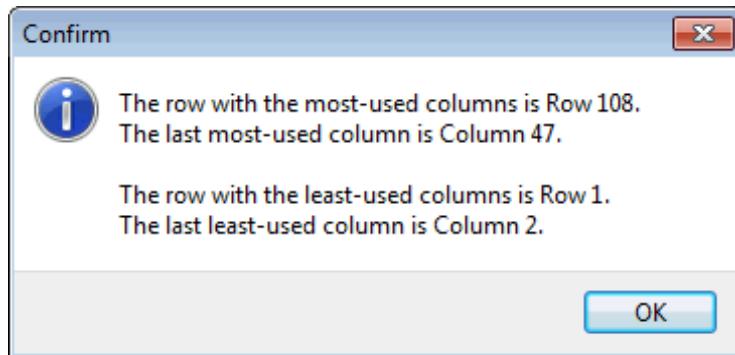
The menu item under the **View** menu with the title **Report the Grid Size...** means the program will report the total size of your existing grid. An example of such a report is:



This report won't tell you anything you could not have found by scrolling the grid to the last column and the last row, but this menu item will save you from having to do this whenever you are working with a very large grid that has more rows and columns than will fit in the main program window all at once.

Reporting the Longest/Shortest Rows

To determine the rows that occupy the least/most number of cells in the entire grid you can select the function **View | Report the Longest/Shortest Rows...** and you should see a report similar to the following.



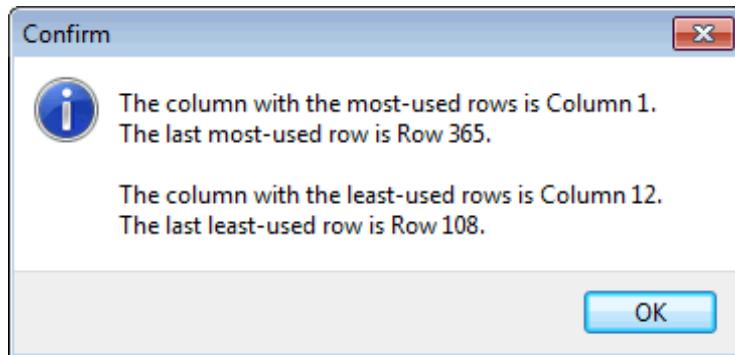
This function works by scanning all rows in the grid.

To determine the longest row, it determines the right-most column in that row that contains a non-empty cell. Among all such right-most columns it will report the corresponding row and column with the largest column number.

To determine the shortest row, it determines the left-most column in that row that contains a non-empty cell. Among all such left-most columns it will report the corresponding row and column with the smallest column number.

Reporting the Tallest/Shortest Columns

To determine the columns that occupy the least/most number of rows in the entire grid you can select the function **View | Report the Tallest/Shortest Columns...** and you should see a report similar to the following.



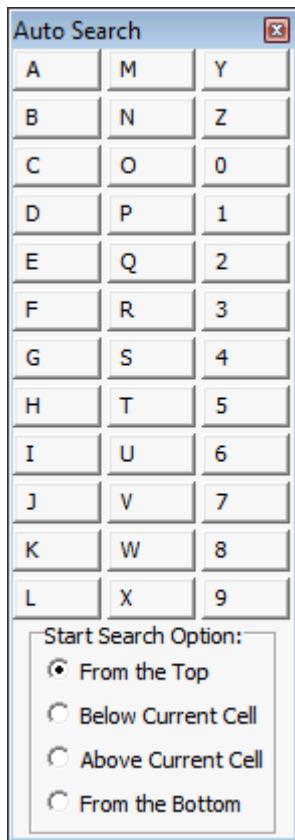
This function works by scanning all columns in the grid.

To determine the tallest column, it determines the bottom-most row in that column that contains a non-empty cell. Among all such bottom-most rows it will report the corresponding column and row with the largest row number.

To determine the shortest column, it determines the top-most row in that column that contains a non-empty cell. Among all such top-most rows it will report the corresponding column and row with the smallest row number.

Using the Auto Search Window

There is a program utility function that displays a special control window called the **Auto Search Window**. To make this window visible you can click the right-most button in toolbar  . You should then see the special control window in the upper-right of the main program window.



The purpose of this window is to provide a series of controls that let you quickly search up or down in a given column. The items in the column don't have to be sorted in any particular order, but when they are sorted this tool is very effective at quickly jumping near a desired position, especially when the column or grid has a large number of rows.

Whenever you click a letter button **A-Z** or you click a number button **0-9** the program will automatically search up or down the current column to find the first or next cell whose first character matches the letter or number that you clicked. Whether the search direction is up or down is explained in the next paragraph.

The **Start Search Option** always determines two parameters for the search process. The first two choices cause the search direction to be downward. The third and fourth choices cause the search direction to be upward. Besides determining the search direction, you also get to tell where to start the search. The first and last choices mean the entire column will be searched, if necessary. The second and third choices limit the starting position to be nearest the current cell. So the search direction may not be obvious until you think about where you are and think about where you can and should go.

If the program finds a matching cell then you will see the highlighted grid position automatically change to show that cell. The program will scroll the entire grid in order to make the selected cell visible. Otherwise, if no match is found then the grid won't scroll and the current position of the grid selection will remain unchanged. Usually the highlighted cell will remain near the middle of all the displayed rows, but the highlighted cell can sometimes appear at the top or bottom of the listed rows. This happens when the found cell is the first or last cell in the column. The program always tries to show the most rows it can, in addition to always bringing the highlighted cell into view.

If you leave the **Start Search Option** as **Below Current Cell** or leave it as **Above Current Cell**, then you can repeatedly click the same letter or number button to quickly move up or down to the next matching cell in the column.

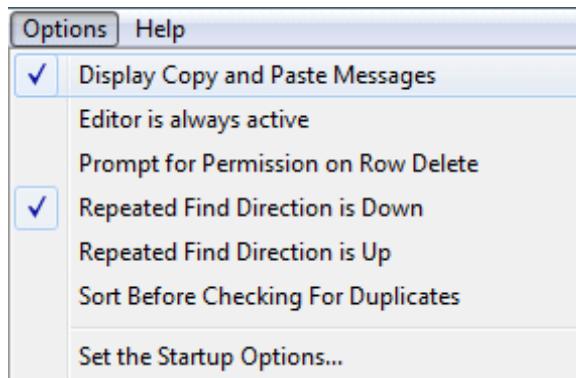
To search in a different column, just click on any cell in another column and then click any **Auto Search** button to find the first matching item in that column. The **Auto Search** function always starts working in the column that has the current cell selection.

You can close the **Auto Search Window** by clicking the same button  in the toolbar that you may have used to open it.

This utility function is an alternative to using the program's main search function that is controlled by a toolbar button. The main search function is for finding longer exact strings anywhere in a grid. The **Auto Search** functionality is intended for quick bi-directional searches and quick movements within a single column.

The Options Menu

There are several options you can set when you open the **Options** menu:

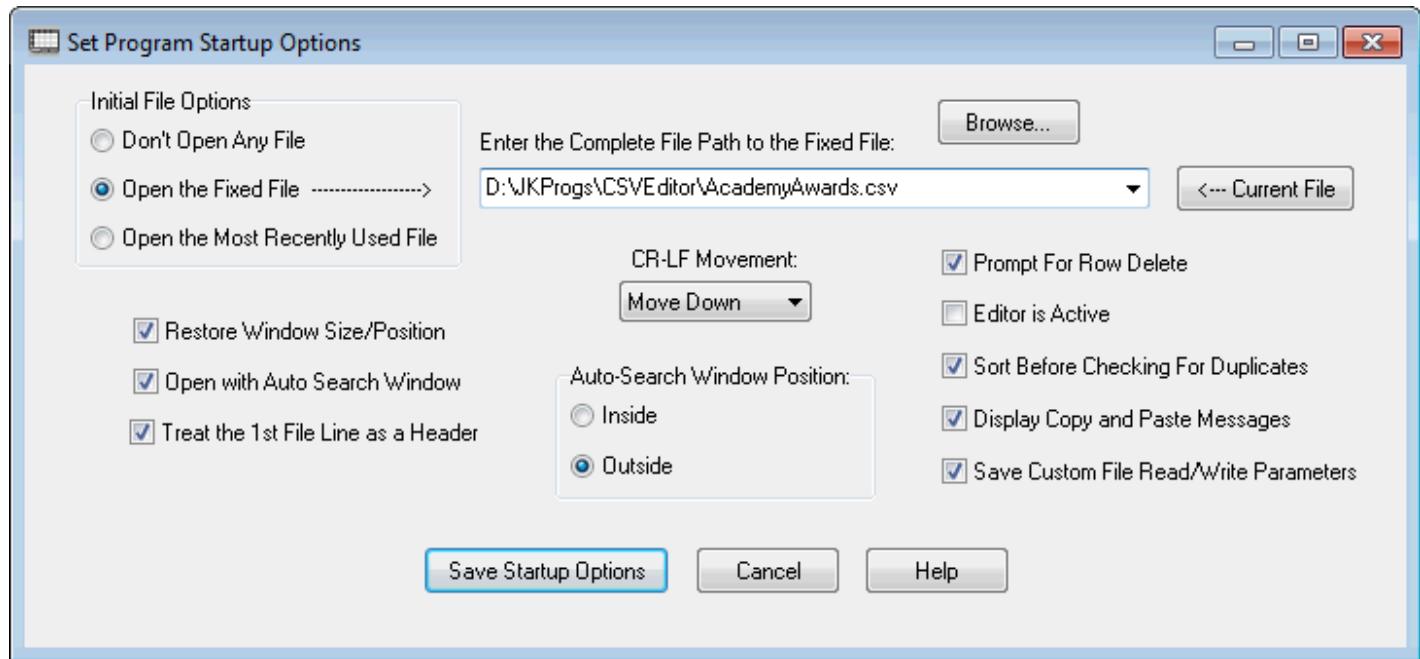


The first six items under the **Options** menu are what are called toggle switches. You can click these menu items to change the state of an option from being turned on to being turned off, or vice versa. A checkmark to the left of one of these menu items means that the corresponding option is turned on. Otherwise that option is turned off.

The **Display Copy and Paste Messages** menu item is intended to be effective only when you execute a copy and/or a paste command by releasing the keyboard keys **CTRL+C** and **CTRL+V**. This means if you use the **Block** menu items, or if you use the toolbar buttons, to do copy and paste, no corresponding message will be displayed, regardless of whether the above **Options** menu item is checked or not.

Setting the Startup Options

There are a number of program options that can be automatically initialized when the program first starts. Just pull down the menu item **Options | Set the Startup Options...** to set all the startup options as desired. You will see the following dialog.



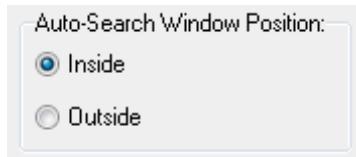
The **Initial File Options** radio group contains three choices. The program can always load the same fixed **CSV** file that you specify. An alternative is to just have the program re-load or **Open the Most Recently Used File**. The final option is to not have the program try to load any file when it starts. This third choice means the program will begin with an empty grid.

When specifying the **CSV** file to be loaded on startup, you can click the button with the caption **<--- Current File** to cause it to enter the path of any file you currently have open. You can also just use the **Browse...** button to select any other **CSV** file anywhere on your computer. The **Complete File Path to the Fixed File** is ignored if you chose to **Open the Most Recently Used File**, or if you choose to not open any file. If either kind of file does not exist at the time the program tries to load the initial file, then the program will simply ignore trying to load the file. In that case the program will open with an empty grid, after which you can manually open any **CSV** file you need to work on.

Whether you open any file or not on startup, you can have the program set both the initial size and the initial position of the program window as the size and position the program had when it was last closed down.

Another option is named **Open With Auto Search Window** and this means when the program first starts the **Auto Search Window** will be showing. This special window contains a series of alpha-numeric buttons used for quick searching. This window can be turned on or off at any time by clicking the **A** button in the toolbar.

Another setting related to the **Auto Search Window** is its position relative to the main program window.



In either case the **Auto Search Window** will attach itself at the right edge of the program window, either just **Inside** or just **Outside** the main program window.

There is a **CR-LF Movement** drop-down list box that when expanded shows three choices:



These three choices have to do with how the program moves the cursor or the selected grid cell when you press the **Enter** or **Return** key on your keyboard. This may occur when you are finished editing the cell contents and you want to move to edit the contents of the next cell. You can choose the primary movement to be either **Move Down** or **Move Right** or you can choose to have **No Movement**. Such movements are like reading a news paper where you may move down row by row until you reach the last row and then you move to the first row in the next column. Or you may move column-wise to the right until you reach the last column and then you will move down one row, but in the first column. Such movements occur automatically until you reach the final cell in the grid. After that no further movements are possible.

Another practical reason for having these choices has to do with using a hand-held barcode scanner to enter data into cells. Most scanners act as a keyboard that optionally send either a carriage return character (**CR**) or a line feed character (**LF**) in addition to the scanned data. Thus we refer to this option as the **CR-LF Movement** option. Having this option means you can continually use scanner clicks to enter data without ever having to manually move the focus to the next grid cell. The default setting is **Move Down**.

You can also automatically initialize five other main program options with checkboxes that cause the program to open in states with which you will feel most comfortable. The first of these is in the main **View** menu and is also controlled by a toolbar button. The remaining four options are available in the main **Options** menu.

The checkbox to **Treat the 1st File Line as a Header** duplicates the second menu item under the **View** menu. Controlling this option in this dialog gives you a chance to set the **View** menu item on program startup. However, it is fastest to later change this option by just clicking either the **H+** or **H-** button near the far right of the toolbar. The caption on this button reflects the current state of this option as being turned on or turned off. It is faster to click this button than to open the **View** menu item to set the option as desired. We like leaving this option turned on, and if we ever open a file that doesn't have a header, we just click the toolbar button to turn off the display of the header row. If you have only 1 row of data this option is automatically turned off and can never be turned back on until after you add at least one more row of data.

If you know you are going to be deleting a lot of rows in your **CSV** file, you may NOT want to be prompted every time the program tries to delete a row. On the other hand, if you are more paranoid about your data and you want to be more protected from accidental row deletions you may want to turn on the check box so the program will **Prompt For Row Delete**.

The next option is related to having the edit mode of the **CSV** grid active. If this is not checked, then to edit inside a cell you would normally just double-click on that cell. The desired setting for this option probably depends on whether you are more of a keyboard user or a mouse user. If you are primarily a keyboard user you might want to turn on the **Editor Is Active** checkbox. If you are a mouse user you might try turning off the **Editor Is Active** checkbox.

There is also an option that is related to checking for duplicates. Checking for duplicates in a given column first requires that the given column be sorted. You can choose to not automatically sort before checking for duplicates, or you can have the program sort before it checks for duplicates. There are reasons both for and against this option. If sorting requires more than 2 or 3 seconds you may NOT want to automatically perform a sort first. In this case you would want to leave the option **Sort Before Checking For Duplicates** in the above dialog unchecked. This option is almost not necessary because internally the program uses a sorting algorithm that normally sorts even large grids in less than a second. We like leaving this option turned on and you should only find it necessary to turn this option off if the time required to perform sorting seems to slow you down.

The next to last checkbox controls an option related to seeing copy and paste messages. When you perform copy and paste operations using the keyboard, those options are not executed until you release the corresponding keys. For this reason, you can sometimes think you performed an operation like copy when the program doesn't register your keystroke. For this reason the program will normally try to confirm these operations by flashing a message. If you feel the messages just slow you down unnecessarily, then you can leave this checkbox unchecked. If you prefer to see the confirming messages then check this checkbox. This option can also be turned on or off using the first item in the **Options** menu.

The last option allows you to **Save Custom File Read/Write Parameters** that saves all the parameters that are in the dialog for performing custom file reads and file writes. Once you use the dialog and have set the parameters, it is nice for the program to save these so you don't have to set them every time you open that dialog. To see the dialog, select **File | Custom File Read**. The parameters get reset only when you click the dialog button with the caption **Custom File Read** or **Custom File Write**.

All of the settings in the Startup Options dialog are saved in a text file that is named **StartupOptions.ini**. This file can be deleted or overwritten at any time without any harm. The **CSV Editor** program will automatically overwrite the file **StartupOptions.ini** every time you close the program.

Although it is not part of the **Startup Options** dialog, we should mention that the program can be started with one command line parameter that is the fully qualified file path to a file to be opened on startup. When this command line parameter is supplied, then all of the above startup options that would otherwise try to automatically load a file will be by-passed.

The **StartupOptions.ini** file will be read and its contents loaded, but the fixed file path and the most recently used file paths will not be used to try to load a file, because the command line parameter will over-ride these and dictate which file is to be opened.

When you save the **Startup Options**, the program will automatically reset all the checkbox options as well as the **CR-LF Movement** option in the main program. This means the program's five menu items that control those same options will be set to match the options in the dialog that you just saved. If you cancel instead of saving, then those options will remain as when you opened the **Startup Options** dialog.

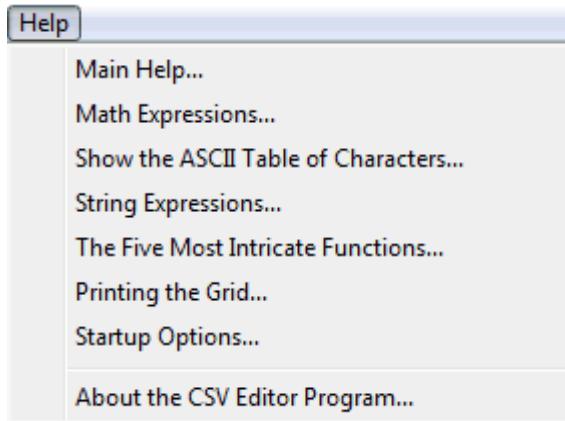
An example of the contents of an example **StartupOptions.ini** file is shown below.

All radio button index numbers specify the position of the corresponding radio button within its radio button group where you count starting from 0. For example, the **Auto Search Position Index** number is 0 for **Inside** and 1 for **Outside**.

```
Initial File Index = 2
The Fixed File Path = C:\CSVEditor\AcademyAwards.csv
The Most Recently Used File = C:\CSVEditor\AcademyAwards.csvRow
Delete Prompt = FALSE
Active Editor = FALSE
Sort Before Dup Check = TRUE
First Line Is A Header = TRUE
Display Copy And Paste Messages = TRUE
CR-LF Movement = MOVE DOWN
Restore Window = TRUE
Open With Auto Search Window = TRUE
Auto Search Position Index = 0
Window Left = 365
Window Top = 272
Window Width = 858
Window Height = 653
Field Delimiter Index = 0
Field Separator Index = 0
Special Character = @
Remove BOM on Read = TRUE
Remove Nulls on Read = TRUE
Insert BOM on Write = TRUE
Insert Nulls on Write = TRUE
```

The Help Menu

The **Help** menu contains several items for getting help. The last **Help** menu item gives you information about the program.



The **Help** menu item with the title **Math Expressions...** is very technical and is only needed for those who wish to exploit the more advanced features of applying mathematical functions to cells. You should first read the **Main Help** file that has basic examples of using math expressions before you try to read this more advanced material. You could also select **Utilities | Show a Sample Data List | Math Functions List...** to see a list of the 61 built-in mathematical functions.

The **Help** menu item with the title, **Show the ASCII Table of Characters...**, displays a special table for programmers that shows ASCII characters and their binary or hexadecimal or decimal number equivalents. This table can also be used to convert numbers between their binary or decimal or hexadecimal formats and can also be used to convert between some (not all) Unicode characters and their whole number equivalents. Otherwise, this table probably has little interest for non-programmers.

The same is true of the **Help** menu item with the title **String Expressions....**. This material is also somewhat technical so it may not be for everyone. However, learning to use the basic string expression functions may allow you to make editing changes to a **CSV** file that would otherwise be very difficult or very time consuming to make.

The **Help** menu item with the title, **The Five Most Intricate Functions...**, is a special help file that discusses the five most closely-related functions in the program. These functions usually involve using multiple columns all at once. The columns can come from different grids. These functions are fully discussed in the main help file.

The last **Help** menu item gives information about the **CSV Editor** program. The most important information is the version number and the most recent build date. This menu item can always be used to determine if your version is up to date.

Rules For Creating CSV Files

If you have the opportunity to create **CSV** files, you should find the following 15 rules helpful. Unfortunately, not all programs follow these rules. **Microsoft Excel** is a notable exception. **Excel** is supposed to be able to export a spreadsheet to a **CSV** file, but how it handles certain numbers and dates is sometimes inconsistent. The following 15 rules make it easy to import and export data from a database, if only all database programs obeyed these rules. Consider these rules as a simple list of free but mostly valuable advice.

A **CSV** file should be an ASCII text file that contains only the printable ASCII characters (including tabs) as well as either Line Feed or Carriage Return and Line Feed characters (we denote these by CRLF). Line Feed and CRLF characters are never allowed inside of any fields. The only non-data special characters inside a **CSV** file are the comma and double quotes. (No special escape characters or escape sequences should be allowed).

1. Each record is on one line. (In other words, records are never split across multiple lines.)
2. Fields are always separated with commas. (Note that this does NOT imply that fields are delimited by commas. It just means that if there is a field following a given field, then a comma character must appear in the input stream before the start of that next field. Thus most fields are terminated by commas, except the last field on a line does not and in fact should not be followed or terminated by a comma.)
3. Leading and trailing white-space characters adjacent to comma field separators are ignored. Note that white-space characters can be spaces or tabs. Adjacent means at the start or end of the field.
4. Fields with imbedded comma data must be delimited with double-quote characters.
5. Fields that contain double-quote characters must be surrounded by double-quotes, and the embedded double-quotes must each be represented by a pair of consecutive double-quotes.
6. If a field starts with a double-quote because it is intended to be delimited by double-quotes, it is an error for that field to not contain an ending double quote. (Especially for the last field on a line).
7. No field should contain embedded line breaks or CRLF combinations.
8. Fields with either leading or trailing spaces MUST be delimited with double-quote characters.
9. Fields may always be delimited with double-quotes, and when they are, the double-quotes will automatically be discarded when the file is read and processed.
10. The first record in a **CSV** file MAY be a header record that contains column field names.
11. No escape characters are ever used inside any field and no escape characters are ever used to delimit fields or to denote double-quotes or commas.
12. No line feed characters are ever allowed in any line except at the end of a line.
13. All lines should contain the same number of fields of data.

14. An empty field may be represented by two adjacent commas and any line that begins with a comma is assumed to start with an empty field. Similarly, any line that ends with a comma is assumed to finish with an empty field.
15. A line in a CSV file that is empty is allowed and is assumed to contain no fields.

If you use another program to parse CSV data, you can test that program by trying to parse the following lines of test data:

```
","",Robert,1122,
John,"Robert",1122,Anystreet,"",99999
"John ""Da Man""",Repici,120 Jefferson St.,Riverside, NJ,08075
John,Doe,120 jefferson st.,Riverside, NJ, 08075
Jack,McGinnis,220 hobo Av.,Phila, PA,09119
Stephen,Tyler,"7452 Terrace ""At the Plaza"" road",SomeTown,SD, 91234
,Blankman,,SomeTown, SD, 00298
"Joan ""the bone"", Anne",Jet,"9th, at Terrace plc",Desert City,CO,00123
,,1677,Pacific Palisades, CA, 90411
1677,Pacific Palisades, CA, 90411,,
""
1677,Pacific Palisades, CA, 90411,X,"John ""Da Man"""
John,"Robert",1122,Anystreet,"",99999
```

The following figure shows what this data should look like when it is properly parsed into columns.

<	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
1		Robert	1122			
2	John	Robert	1122	Anystreet		99999
3	John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	08075
4	John	Doe	120 jefferson st.	Riverside	NJ	08075
5	Jack	McGinnis	220 hobo Av.	Phila	PA	09119
6	Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
7		Blankman		SomeTown	SD	00298
8	Joan "the bone", Anne	Jet	9th, at Terrace plc	Desert City	CO	00123
9			1677	Pacific Palisades	CA	90411
10	1677	Pacific Palisades	CA	90411		
11						
12	1677	Pacific Palisades	CA	90411	X	John "Da Man"
13	John	Robert	1122	Anystreet		99999

Index

A

above current cell 412
Academy Awards 4, 16, 20, 39, 43, 58, 153, 155-157, 162, 164, 191, 256, 304-305
accuracy factor 222
address text file 302, 310-311
adjusting column widths 6, 405-406
Adobe Reader 1
ALabel
 data file 303-306
 grid table 307-309
 program 1, 303-309
alignment information 42
amortization schedule 312-314
answer column 169-177, 189, 301, 334-336
appendix
 data as columns 29
 data as rows 29
artists list 57, 263, 341, 343-346
ASCII characters 2, 21-28, 214-218, 419-420
asterisk character * 45
automatic
 creation 255-257, 272-281
 indenting 285
 numbers 61-63
 search window 6, 150, 403, 412-413, 415-418
 width changes 6, 403, 405
average 77, 165, 212, 331

B

barcode scanner 416
base path option 68-69, 80, 328-329
basic limitations 12, 420
below current cell 412
Big Endian 24
binary 225, 234-236
bitmap files 68, 351
blank
 cells 148, 181, 210, 388
 column removal 126
 row removal 181
block
 copy 46
 definition 51
 file 56, 58, 61, 64, 68, 70, 72, 75, 77, 79-80, 83
 formatting 11, 84-92
 interlacing 207-209
 move 100
 paste 46
 read/write order 47, 50
 replication 105, 209
 reshaping 47-55
 selection 7, 113
 size report 408
 width 47, 50
block-column reshaping 16, 47-55
Blocks menu 11, 46
BMP files 68, 351
BOM (byte order mark) 24-28

book/document index file 267-291
bookmarks 1
Boolean data 9, 90, 157-160, 354
bottom (row move) 206
business names 57, 71, 76, 263, 341, 344
byte order mark 24-28

C

C(I, J) notation 97, 116, 408
case sensitive 127, 142, 149, 151, 159, 174-175, 182, 189, 247, 370, 381, 383, 390, 394, 399
cell comparisons 127, 142, 149, 151, 159, 174-175, 182, 189, 247, 370, 381, 383, 390, 394, 399
 reference 97, 116, 408
Celsius 98, 240
center-justify 84-87, 92
centering the window 403
character repertoire 22
 22-23
cities lists 57, 263, 341
clearing
 a column 129
 a row 183
clipboard 7-8, 46, 167, 210
code
 point 22
 space 22
colon character 18
columns
 character extents 407
 display widths 6, 403, 405-406
 maximum 12, 15, 34, 162, 211, 404
 menu 125
 move 146-147
 report 349
 shortest 411
 tallest 411
combining columns 130
comma 18
command line parameters 13
comparing
 columns 128, 383
 files 292
comparison report 292
conditional replacements 258
constant payments 318
Conversions menu 10, 213
converting data 11, 213
copy
 and paste 7-8, 414
 matching data 293-301
country names 341, 343
CR-LF Movement 416, 418
CSV 1
CTRL key 4, 7-8, 144, 153, 167, 212, 414
currency 89, 155, 157, 229, 312-319
custom
 file read 18-28
 file write 18-28

D

data
 column 377-381
 types 9, 84-92, 153-160
date
 data 9, 87-88, 153-160, 219, 356
 formats 87-88, 219, 357
 sequence 64
 to JDN 219-221
de-interlace 203-205, 267-270, 283-288
decimal 77, 88-89, 93, 97, 167, 212, 221-226,
 229-230, 240, 242, 264, 312-319
decryption 327
DELETE key 8, 129, 183
deleting
 a range of columns 135
 a range of rows 184
 blank columns 126
 blank rows 181
 duplicate columns 127-128
 duplicate rows 182
 rows 184
delimiter 18
delta days 64
depleting an account 316
destination column 11, 47, 50, 214-243,
 296-301
dictionary word list 346
difference of sets 174-177, 301
divisor 204
double quotes 18
duplicate
 columns 127-128
 rows 182, 209, 394-395
duplicating
 a column 137
 a row 186, 209

E

e-mailing the grid 340
editing multiple columns 178-179
editor is active 416
elapsed time 9, 90, 244
electronic signature 320
empty string 8, 34, 89, 120-124, 143, 150, 152,
 155, 158, 164-165, 264
encrypting 325
endian-ness 23
equal spacing 406
error message 11, 43, 49, 77, 89-92, 98, 103,
 110, 122, 137, 145, 155, 158, 162, 165, 195,
 201, 211, 217, 219, 223, 226, 228, 230, 232,
 241, 243, 275, 281, 283, 289, 328, 350, 352,
 354, 357-358, 360-361, 363, 365-366, 370,
 375-376, 379
Excel 7, 420
exchanging
 two columns 138
 two rows 187
existing the program 45
exporting
 columns 139
 matching key rows 143, 189-191
 rows 143, 188-193
 unique cells 386

extents report 407
extreme left/right column move 146-147

F

Fahrenheit 98, 240
female first names 75, 341
filed
 names 303, 420
 separator 18
file
 count 68, 80
 information 68, 80-82, 328-329
file menu 14
filename filter 80-82
filenames 68, 80-82, 328-329
filter
 on a column 141-143
 rows with matching cells 194-197
financial calculations 312-319
finding duplicates 144, 389
fingerprint 320
first names 75, 341
fixed file path 418
font 22
formatting data 11, 84-92, 355
fractions 222-224
from
 the bottom 412
 the top 412
function key F8 178-179
functions list (math) 344, 419

G

GeoCoding 264-266
GIF files 68, 351
globally unique identifier 58-60
glyph 22
Google Maps 264-266
GPS coordinates 54-55, 264-265, 333-336
grid size report 409
groups of rows 17, 192
GUID 58-60

H

hash values 320-322
header row 4, 188, 209, 403, 416
help 1
Help menu 419
hexadecimal
 21-28, 58, 225, 320
 editor 21, 28
horizontal position 39
HP-12C calculator 314
Huffman codes 10, 233-239

I

image
 base path 351
 files 10, 68, 80, 351-352
importing
 columns 29
 rows 29

improper page number 275
in place sorting 156-158
indenting 285
Index 422
index
 formatting options 285
 page replacements 271-281
initial file index 417
inserting
 columns 145
 rows 198
insertion sort 153
inside 416, 418
integers 77-78, 225-226
integrity check 272, 281, 283-284
interlacing rows 199-205, 207
Internet 265
intersection of sets 174-177, 301
ISBN 10, 227-228, 360-361, 377

J

JDN (Julian Day Number) 10, 67, 219-221
join operation 37, 247, 250, 254, 301
JPG files 10, 68, 351
justify cells 84-87, 92

K

key
 columns 31, 34, 254, 399-402
 rows 143, 175, 189-191
keyboard shortcuts 7-8, 414, 417
keyname 303-306

L

last names 75, 341
latitude 264, 333-335
launch a file 10, 328-329
Least Significant Byte (LSB) 24
left-justify 84-87, 92
length function 117, 120-121
line
 endings 25
 length 285
linear regression 10, 99, 330-332
link column 377-381
Little Endian 24
loan amortization 312-314
location matching 333-336
longest row 410
longitude 264, 333-335
lump sum earnings 314

M

Macintosh 26
mal-formed CSV file 347-350
male first names 341
matching
 angle brackets 18
 curly braces 18
 data 169-173, 189-191, 293-301
 key rows 143, 175, 189-191
 locations 333-336
 parentheses 18
 square brackets 18

math
 expressions 10, 93-99, 419
 functions list 344, 419
maximum
 chars per cell 9, 12
 columns 12, 15, 34, 162, 211, 404
 rows 12, 15, 186, 192, 323
mean 77-78, 165-168, 212, 331
merging data 29-37, 324
meta-data 92
Microsoft
 Excel 7, 420
 Windows 25
 Word 7
mimetype names 75
files 333-336
money 229-230, 312-319
month
 pattern 64-67
 range 72-74
most recently used file 417
Most Significant Byte (MSB) 24
move
 backwards 111
 Down 416
 forwards 111
 Right 416
movement option 416
moving
 a block 100
 columns 146-147
 pages 271-281
 rows 187, 206
 the cursor 416-417
multiple
 column sorting 159-161
 columns 6, 159, 178-179
musical
 artists list 57, 263, 341, 343-346
 songs list 57, 263, 341, 343-346

N

names lists 57, 75-76, 263, 341, 344-346
nearest location matching 333-336
new grid 15
NST 320
No Movement 416
normal distribution 77-78
Notepad 3, 12, 350
numeric data 9-11, 77-79, 157-160, 330-332, 362-364

O

octal 225
open a file 18, 328-329
Options menu 414
outside 416, 418

P

passwords list creation 323-324, 346
paste button 7-8, 414
pattern 64-67
PDF file 1, 42, 81, 92, 328
periodic payments 318
pick list 57, 70-71, 346

pick up column 296-301
picture files 10, 68, 80, 351-352
PIN number 326-327
PNG files 68, 351
position of auto search window 416
post-conversion blanking 258-262
precision 89, 223, 240
printable characters 21
printing
 38-43
 alignment information 42
 header information 43
process subdirectories 68, 80
program limitations 12, 15, 209, 420
prompt for row delete 416
punctuation character 216, 323

Q

quantities 59
quantity custom fill 337
quick selection 7
QuickSort algorithm 153
quitting the program 45

R

random
 booleans 79
 dates 72
 names 75-76, 344-346
 numbers 77, 110
 passwords 323
 strings 79, 323
 times 79
range check 356-359, 362-364, 371-372
raw bytes 26-27
read order 50
recent files list 14, 44, 415
recurse subdirectories 68, 80
redo command 5, 46
referential integrity 2, 173, 353, 368, 377-381
regression function 330-332
remove (word) 276
replacement lists 246, 248, 255, 257, 259, 273, 275
replacing book indexes 271-281
replicating a block 105, 207-209
report column 11, 354-381, 396, 407
reporting
 blank cells 148, 210, 388
 block size 113, 408
 duplicate rows 394-395
 grid size 409
 longest row 410
 shortest column 411
 shortest row 410
 tallest column 411
reshaping a block 47-55
restore window 417
reverse string function 117
right-justify 84-87, 92
Roman Numerals 10-11, 231, 365
rotating a block 110
row list 396

rows
 deleting 184
 exporting groups 17, 192
 longest 410
 maximum 12, 15, 186, 192, 323
 menu 180
 moving 187, 206
 reporting duplicate rows 394-395
 scrambling 110, 211
 shortest 410
rules for CSV files 3, 420

S

sample data lists 57, 263, 341
scanner 416
scatter diagram 332
scrambling
 a block 108
 rows 110, 211
search
 auto 6, 150, 403, 412-413, 415-418
 column 333
 miles 333-336
searching
 and replacing 151-152, 169-173
 for text 149
secure hash algorithm (SHA-256) 320-322
seed value 325
selecting 7, 113
selection without replacement 77
semi-colon character 18
separator character 18, 122-124
sequence rule (dates) 64-67
serial numbers 63
set
 difference 174-177, 301
 intersection 174-177, 301
 union 174-177, 301
SHA-256 320-322
shift key 7
shortcut keystrokes 7-8, 414, 417
shortest
 column 410
 row 411
signature 320
single quote 18
Social Security numbers 1, 59, 124, 250-254, 400-402
songs list 57, 263, 341, 343-346
sorting
 153-161, 289-291
 book index page numbers 289-291
 in place 156-158
 using multiple columns 159-161
source column 47, 214-243, 333, 396
special
 character 18, 84-85
 transpose read 16-17, 55
splitting
 a column 162
 a file 347-348
SQL 2
standard data types 9
starting seed 73, 77, 325, 327
startup options 13, 415-418

state
abbreviations 83, 195-197, 373
names 83, 195-197, 341, 373

statistics 165, 212, 330-332
street names 57, 263, 342

string
data 9, 57, 157-160, 367
expressions 114-124, 419
length function 117, 120-121
reverse function 117
substring function 116-124, 396
token function 122-124

subdirectories 68, 80

subheading 288

substring 116-124, 141-143, 163, 194, 238, 396

occurrences 288, 396

summing entries 165, 212

supported data types 9

suspicious CSV file 347-350

T

tab character 18

tallest column 411

temperatures 98, 240

template design 305

territory names 343

testing key columns 399

text

ASCII 21-28, 214-218, 419-420

prefix 63

suffix 63

Unicode 21-28

TextPad 2, 12, 350

time

data 9, 90-92, 157-160, 168, 212, 242, 371

formatting 90-92, 168, 212, 242, 371

token

function 122-124

separator character 122-124

tool bar 5

top (row move) 206

transpose

of a grid 16-17, 47, 55

read 16-17, 55

trees 234-239

two-column search and match 169-173, 381

U

undo command 8, 46

Unicode 21-28, 214, 216, 419

uniform distribution 77

union of sets 174-177

unique

cells exporting 386-387

cells reporting 389-390

columns 127, 382-383

rows 182, 382, 394

substrings 396

uniqueness

checks 389, 392, 394

menu 382

Unix 26

US

business names 344
cities 341
State abbreviations 83, 195-197, 373
State names 83, 195-197, 341, 373
using the auto search window 6, 150, 403, 412-413, 416
UTF-16 18, 25-26
UTF-32 25
UTF-8 25, 216
Utilities menu 263

V

validating data 11, 353-376

validation

menu 10, 353

strings 70, 368-373, 381

vehicle identification numbers 10, 375

View menu 8, 403

viewing images 351-352

VIN (Vehicle ID Number) 10, 375

W

warning 12

window

auto search 6, 150, 403, 412-413, 415-418

centering 403

height 418

left 418

top 418

width 418

Windows clipboard 7-8, 46, 167, 212

Windows(Microsoft) 26

word lists 346

Word Microsoft 7

word wrap 285

working hours example 79

world

cities list 57, 263, 341

countries list 57, 263, 341

write order 50

X

X

data 330

variable 93-98, 115-117, 124

Y

Y data 330

year range 72-73

Z

zero 2, 21, 25-28, 40, 63, 89, 98, 100, 155, 158, 244, 312, 325, 327, 362