Csv File Search 3.0 Help

CsvFileSearch 3.0 is a simple search utility that provides an easy way to search the contents of multiple csv files.
It is easy to use for the novice and also flexible enough for the advanced user.

How it works.

You choose a directory of csv files to be scanned. You can then search all csv files found in that directory and sub directories.
The search results are displayed as a separate block for each file.
The column titles found on the first row of the searched file are used as the headings for the search result block.

The path of the parent file is shown in column 'A' ('Source'). A double click on this will open the parent file in your prefered application. The row number is shown in column 'B' (Row)

The first row of all csv files must contain column titles.

Getting Started

1. **Choose a search path**. Click on the yellow folder icon, or select '**File>Choose Search Folder**' from the top menu bar.

2. **Enter the search criteria** in the **'Search Criteria'** box. This can be a single word/phrase or multiple words/phrases separated by double slash (**//**).

3. **Press <**ENTER**>** or click on the **'Search'** button.

Note: You can cancel the search at any time. Records from the incompleted search will be displayed.
If a search is cancelled, the current file being processed will complete before search is stopped.

IMPORTANT

Criteria separators

The criteria separators in this version are different to previous versions.
A double forward slash (//) must be used to separate each criteria.
Specific columns may be searched by including a column id or title and "**==**" before the criteria.
Example: Surname==hardy // firstname==john

Note that in version >=3, **the comma is no longer used to separate criteria**. It is now used to separate multiple columns.
Example: firstname,forename==john

The example above will look for "**john**" in columns "**Firstname and Forename**".
Only the columns specified will be searched.

Search options.

Search Path

This is the base folder to search for csv files.
Click on the yellow folder icon and choose a folder to scan. Sub folders will also be scanned for csv files.

Search Type

*** FullText Search** Searches the csv files directly. This search type can be slow.

*** Index Search** A fast search of a prebuilt database.
The database index will be built automatically when you first search or when the search folder/files change.

Lookup Type

This is the record lookup type. 'Soundex', 'Metaphone' and 'Regex' always use a 'FullText' search.

*** Contains**: Search criteria can be anywhere in a field.

*** Starts With**: Search field must start with the search criteria.

* **Exact Match**: Search field contents must be exactly the same as the search criteria.

*** Soundex:** Search using the soundex algorithm

*** Metaphone:** Search using the metaphone algorithm

*** Regex**: Search using regular expressions.

*** SQlite:** Search the database using sql statements.
For more information on this option, see '**Using the SQlite search option**'.

Match Type

Set this the way you want to match search criteria to the field data.
Note: See '**Criteria separators'** for more info.

Match Any: If any of the search criteria is found. (OR)
example: **Surname,FathersName==Hardy//Forename==John** would search as follows:-
Surname == 'Hardy' **OR** FathersName == 'Hardy' **OR** Forename == 'John'

Match All: All the search criteria must be found. (AND)
example: **Surname,FathersName==Hardy//Forename==John** would search as follows:-
Surname == 'Hardy' **OR** FathersName == 'Hardy' **AND** Forename == 'John'

Search Criteria

The text to search for. You may specify columns.
Multiple words and phrases may be used. Separate criteria with double forward slash (//).

'Search' Button

Click this to start a search. (or press <ENTER>)

'Clear Grid' Button

This will clear the grid.

Column Id Reference

This is simply a list of column titles found in the files you are searching. A reference to the master worksheet column id is appended to the title. Example: **Surname (F).** This means that a column title 'Surname' was found in column 'F'.

Note: *CsvFileSearch requires all searchable files to have column titles.*
If you have files that do not have column titles in the first row, then you will see garbage here.

Column 'A' source file link.

To open the original parent file of a search result, 'Double Click' column 'A' (Source)'. The number of the row in the parent file is shown in the left column.

Column 'B' row number.
This is the rownumber of the record found in the source file.

Menu options

File menu

Saving all results into a single file

Select '**File>Save results as single file**' from the top menu bar.
All results are saved into a single csv file.

Saving results into a separate file for each searched file.

Select '**File>Save results as multiple files**' from the top menu bar.
Choose the folder to save the files into. The results are saved as separate csv files.

Tools menu

Refresh column reference list

The purpose of column reference list is simply to supply you with a reference to the column title locations.
When searching mixed files, a column title may be in several different columns.

Rebuild Index

Rebuild the database for the current search folder. This is usually done automatically by the application when changes are detected.

Context menu (right click)

Use this to copy some or all of the results to the clipboard.

Copy Current Row
This will copy the current row to the clipboard as tabulate data.
The column titles will also be copied.

Copy Selected Rows
This will copy the all selected rows to the clipboard as tabulate data.

To selected rows.
1. Place the cursor on the first row to start select.
2. Hold down 'SHIFT' and click on the last row you want to select.
3. Right click with the selection and choose 'Copy Selected Rows'

Copy All
This will copy ALL results to the clipboard as tabulate data.

Criteria separators

Each criteria must be separated by double forward slashes (**//**). Depending on your selection of the 'Match', the **//** will represent AND or OR.

Example: If 'Match Type' is set to 'Match ALL' then a criteria of **William // John // Richard** would search anywhere for **William OR John OR Richard**

Note that in version >=3, the comma is no longer used to separate criteria. It is now used to separate multiple columns.

Specific columns may be searched by including a column id or title and "**==**" before the criteria.

Important: A double forward slash (//) must be used to separate each criteria.
Example2: Surname==hardy // firstname==john

Example2 will look only in column 'Surname for "**hardy**" and only in column 'Firstname' for "**john**".
The AND/OR will be decided by your choice of 'Match Type'.

Note: Only the columns specified will be searched. Searches are NOT case sensitive.

Multiple searches for the same column are also possible .
There are two ways to do this...

1. Specifiy each column separately with the same criteria.

Example: Surname==Hardy // FatherName==Hardy
In this example, if the "Match" option is set to "Match **ALL**", the result would be true if "Hardy" is found in "Surname" **AND** "Name".
If the "**Match**" option is set to "**Any**", the result would be true if "Hardy" is found in "Surname" **OR** "FatherName".

2. Specify multiple columns

Multiple columns may be specified and each column must be separated by a comma.

Example1: Surname,FatherName==Hardy

The columns in each criteria are '**OR**'. That is to say, in **example1** above, the cell contents of either **Surname** **OR** **FatherName** need to match '**Hardy**'.

In **example1**, if a '**Match**' type of **'Match All' (AND)** is selected, the '**//**' separator would represent AND , but the comma separated columns in each criteria would still be OR.

(Surname OR FatherName) == Hardy AND (Forename OR Fornames) == John

Notes:
Column "A" and "B"are never searchable. *These are the 'source' and the result 'row' number.*

Soundex, Metaphone and Regex searches are considerably slower than normal text searches, and can only be used with 'FullText' searches..

The sqlite search lookup option allows extra search options for advanced users.

Using the SQlite LookUp Type

The sqlite lookup type allow advanced user to search the database using sql statements.

The Index (database) Tables

In the database each csv file found in the search folder is automatically given a unique table name. Each tables contain records (rows) of data taken from a single csv file.
The table information can be viewed via the edit menu. **Edit > Show table info**

In CFS all tables are searched with a single statement, and you do not need to know the names of the tables. CFS will add the table name to your sql statement.
You can omit the first part of the statement (*SELECT * FROM INDEX WHERE*) and just enter the column_title and expression.
Example: **surname LIKE '%Hardy%';**

If you choose to enter the full statement, use 'INDEX' as an alias for the real table name. The correct table name will be added automatically.
Example: SELECT * FROM INDEX WHERE surname LIKE '%Hardy%';

Wildcard Characters

A wildcard character can be used to substitute for any other character(s) in a string.

In CFS wildcard characters may be used with the **LIKE** operator.

The wildcards are:

%	A substitute for zero or more characters
_	A substitute for a single character

Using the '%' (percent) and '_ ' (underscore) wildcards

The following SQL statement selects all records with a Surname starting with the pattern "Har".
Example: **surname LIKE 'Har%';**

The following SQL statement selects all records with a Surname containing the pattern "ard".
Example: **surname LIKE '%ard%';**

The following SQL statement selects all records with a Surname starting with any character, followed by "ardy".
Example: **surname LIKE '_ardy';**

Operators and Conditions.

LIKE

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
Syntax: *columnname* **LIKE** *pattern*;
Example: **surname LIKE '%hardy%';**

Note: When using the 'LIKE' operator, the criteria is not case sensitive.

GLOB

The GLOB operator is used to match only text values against a pattern using wildcards.
Unlike the LIKE operator, GLOB **is case sensitive** and it follows syntax of UNIX for specifying the following wildcards.

The asterisk sign (*)
The question mark (?)

The asterisk sign represents zero or multiple numbers or characters. The ? represents a single number or character.

The basic syntax of ***** and **?** is as follows:
SELECT * FROM *tablename* **WHERE** *columnname* **GLOB '***criteria***?'**

Example: SELECT * FROM index WHERE surname GLOB '?ardy';

You can combine any number of conditions using AND or OR operators.
Here are number of examples showing different GLOB clauses with **'*'** and **'?'** operators:

column_name GLOB '184*'	Finds any values that start with 184
column_name GLOB '*200*'	Finds any values that have 200 in any position
column_name GLOB '?00*'	Finds any values that have 00 in the second and third positions
column_name GLOB '2??'	Finds any values that start with 2 and are at least 3 characters in length
column_name GLOB '*2'	Finds any values that end with 2
column_name GLOB '?2*3'	Finds any values that have a 2 in the second position and end with a 3
column_name GLOB '2???3'	Finds any values in a five-digit number that start with 2 and end with 3

Example, which would display all the records from 'INDEX' where BaptismDate starts with 'Apr' and has 4 characters at the end starting with '184' (1840 to 1849).
BaptismDate GLOB 'Apr*184?';

Example which would display all the records from 'INDEX' where a Surname has a hyphen (-) inside the text: SELECT * **Surname GLOB '*-*';**
Note: When using the 'GLOB' operator, the criteria is case sensitive.

NOT

NOT is used with other operators to find records NOT matching the criteria.
Example: **surname NOT LIKE 'hardy%';**

= or ==

To find records EXACTLY matching the criteria (*case sensitive*).
Example: **surname == 'Hardy';**

!=

To find records NOT exactly matching the criteria (*case sensitive*).
Example: **surname != 'Hardy';**

<>

To find records greater or smaller than the criteria (*case sensitive*).
Example: **surname != 'Hardy';**

>=

To find records greater or equal to the criteria (*case sensitive*).
Example: **year >= '1840';**

<=

To find records less or equal to the criteria (*case sensitive*).
Example: **year <= '1840';**

>

To find records greater than the criteria (*case sensitive*).
Example: **year > '1840';**

<

To find records less than the criteria (*case sensitive*).
Example: **year < '1840';**

IS NULL

To find records containing a null value.
Note: There should never be null records in this database.

IS NOT NULL

To find records not containing a null value.
Note: There should never be null records in this database.

AND & OR

The AND & OR operators are used to filter records based on more than one condition.

The AND operator displays a record if both the first condition AND the second condition are true.
The OR operator displays a record if either the first condition OR the second condition is true.

When combining AND OR conditions, it is important to use round brackets so that the database knows what order to evaluate each condition. (Similar to the way of ordering operations in maths)
Example: surname LIKE '%Hardy%' **AND (** forenames LIKE '%George%' **OR** forenames LIKE '%Mary%'**)**;

BETWEEN

The BETWEEN condition selects values within a range.
Example: aged BETWEEN 1 AND 14 ;

The BETWEEN condition can also be combined with the NOT operator.
Example: aged NOT BETWEEN 20 AND 40 ;

LIMIT

Limit the number of records returned (from EACH file).
Example: surname LIKE 'Hardy' LIMIT 20 ;

IN

The IN operator allows you to specify multiple values in a WHERE clause.
Syntax : column_name IN (value1,value2,...);

Example: forenames **IN** ('George', 'Mary','William');
Note: When using the 'IN' operator, the criteria is case sensitive.

ORDER BY **ASC|DESC**

The ORDER BY clause is used to sort the result-set by one or more columns (for EACH file).

ORDER BY sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.
Example: forename like '%william%' ORDER BY surname ASC, birthyear ASC ;

SPLIT

SPLIT is a special operator used to split a file into smaller files. SELECT must be used in this statement.
Syntax is: **SELECT * FROM** *tablename* **SPLIT** *numberofchunks* ;
This can be shortened to **SELECT** *tablename* **SPLIT** *numberofchunks* ;

Example: **SELECT table8 SPLIT 20 ;**
This would split the file indexed in table8 into 20 files and save them in a folder called 'CFS_SPLIT', which can be found in your 'Home' folder.

Note: The file is processed, not the database records. The original file is NOT changed.
The table name for each file can be viewed by clicking on the '**Tools**' menu and selecting '**Show table info**'.

COMPARE

COMPARE is a special operator to compare two tables. SELECT must be used in this statement.
Syntax is: **SELECT * FROM** *tablename* **COMPARE** *comparetablename* ;
This can be shortened to **SELECT** *tablename* **COMPARE** *comparetablename* ;
The results returned will be from the '*comparetablename*' *table* ;

Example: **SELECT table8 COMPARE table9 ;**
The above example would return results of records found in **table9** that were NOT found in **table8**.

To reverse the compare, just reverse the table names...
SELECT table9 COMPARE table8 ;

You can of course do this the sqlite way, column by column...
SELECT DISTINCT * FROM table8 WHERE (Surname NOT IN (SELECT DISTINCT Surname FROM table9));

You would need to repeat for every column...
*SELECT DISTINCT * FROM table8 WHERE (Surname NOT IN (SELECT DISTINCT Surname FROM table9)) OR (Forenames NOT IN (SELECT DISTINCT Forenames FROM table9)) OR (Age NOT IN (SELECT DISTINCT Age FROM table9)) OR .... etc..*

Which is why I created the COMPARE operator.

String Literals

String literals are always surrounded by single quotes (') or double quotes ("). For example:

'Les Hardy'	String literal with single quotes
"Les Hardy"	String literal with double quotes

Number Literals

Number literals can be either positive or negative numbers that are exact or floating point values. If you do not specify a sign, then a positive number is assumed.
Here are some examples of valid number literals:

72	Integer literal with no sign (positive sign is assumed)
+72	Integer literal with positive sign
-72	Integer literal with negative sign
72.607	Decimal literal

Date and Time Literals

Date and time literals can be expressed as strings. Here are some examples of valid date and time literals:

'2015-04-27'	Date literal formatted as 'YYYY-MM-DD'
'2015-04-27 11:44:23'	Datetime literal formatted as 'YYYY-MM-DD HH:MM:SS'

Boolean Literals

There are no boolean literals in SQLite, instead, boolean literals are stored as numeric values. Here are some examples of valid boolean literals:

1	Equivalent to TRUE (stored as a number)
0	Equivalent to FALSE (stored as a number)

License

CsvFileSearch is free for non-commercial use.

For more information contact les@webmayo.com