File Query

Version 2.0

User Manual

AgileData Software

Table of Contents

Getting Started

For file query to be installed and run successfully, you must be using one of the following supported operating systems:

Windows 2000 SP 4; Windows XP; Windows Vista

Installing File Query

To install File Query:

1. Download the installer package from www.agiledatasoftware.com/downloads.html
2. Unzip the two files contained within the installer package.
3. Run the file called Setup.exe.
4. Follow the instructions on screen to finish the installation.

Note: File Query requires the Microsoft .NET 3.5 Framework, and if it is not located on your system, you will be prompted to install it first.

Uninstalling File Query

To uninstall File Query, you can simply go to the Add/Remove Programs dialog within control panel.

Overview

What is File Query, and Why Should I Use it?

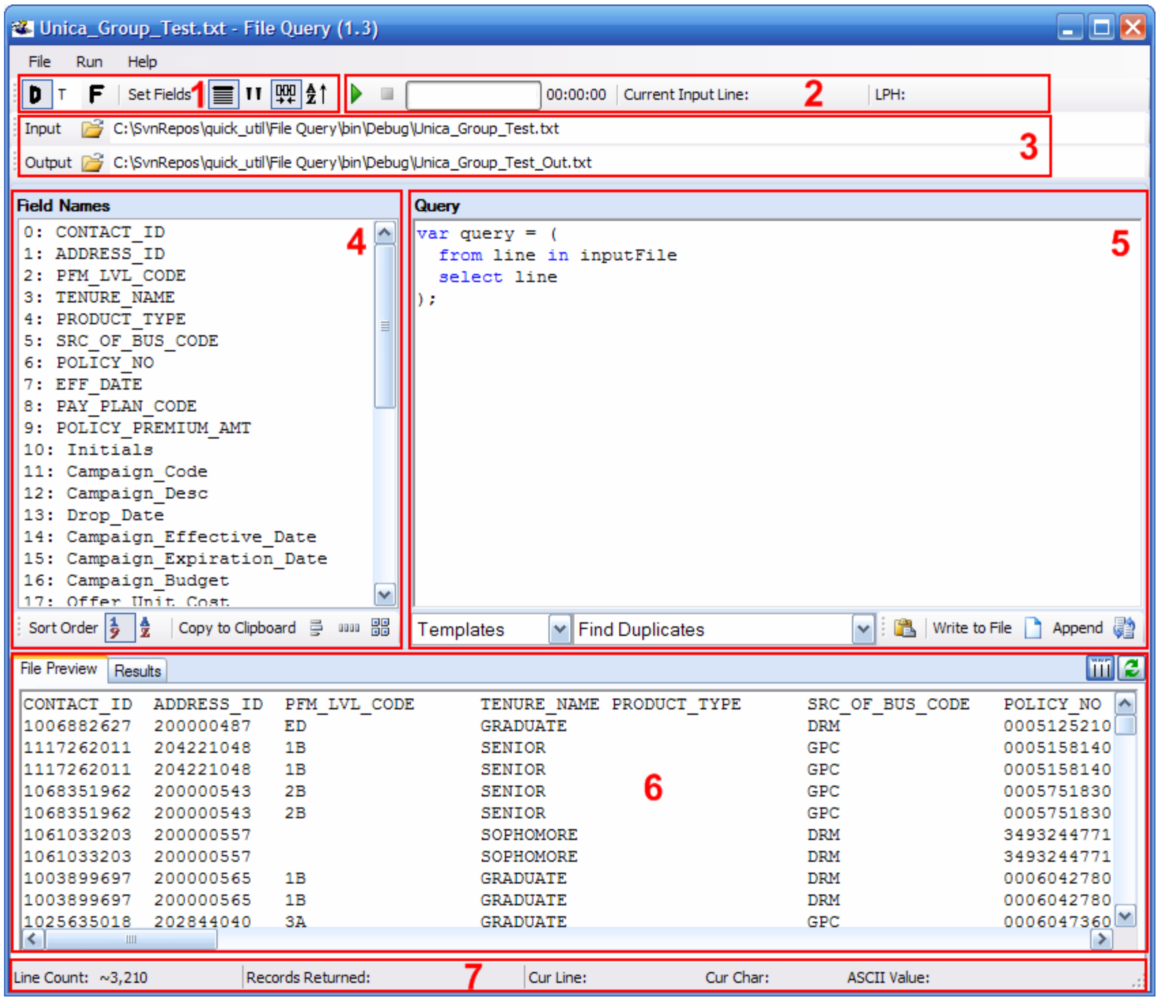
Flat files are a ubiquitous way to transfer information from one data system to another, but are difficult for people to work with directly. They are usually loaded by a program into some form of database as soon as possible so that they can be accessed, but if a program doesn't already exist, or there is a problem with the file's format or data, then every thing falls apart. File Query is meant to give the same flexibility of access and modification to flat files as exists for databases. This is especially useful for analyzing new files that you receive for the first time, figuring out what is wrong with ones that break a current job and possibly modifying the file to fix any errors within. While there are many tools that make it easier to write a program to handle them, or to run a few pre-defined programs against them, File Query is the only one that truly allows ad-hoc access to them with even greater flexibility than SQL. (SQL is the common querying languages used by almost all databases.)

Who Should Use it?

File Query is meant to be as simple as possible, and useable by almost any one with a simple understanding of how flat files work. While its capabilities are similar to SQL, it's built in templates and help system make it so that even those who have not dealt with SQL before can be off and running in no time. Be sure to read through the quick start guide if you want to jump right in. This manual is more in-depth, and certain parts, such as the functions listings, are meant to be used mainly as reference, except for those who love learning about all of the possible features of their new toy.

The Interface

Here is an example screen shot of File Query's interface. Different sections are numbered for easy reference from the following sections.



```
C:\SvnRepos\quick_util\File Query\bin\Debug\Unica_Group_Test.txt
C:\SvnRepos\quick_util\File Query\bin\Debug\Unica_Group_Test_Out.txt
```

```
0: CONTACT_ID
1: ADDRESS_ID
2: PFM_LVL_CODE
3: TENURE_NAME
4: PRODUCT_TYPE
5: SRC_OF_BUS_CODE
6: POLICY_NO
7: EFF_DATE
8: PAY_PLAN_CODE
9: POLICY_PREMIUM_AMT
10: Initials
11: Campaign_Code
12: Campaign_Desc
13: Drop_Date
14: Campaign_Effective_Date
15: Campaign_Expiration_Date
16: Campaign_Budget
17: Offer_Unit_Cost
```

```
var query = (
  from line in inputFile
  select line
);
```

```
CONTACT_ID  ADDRESS_ID  PFM_LVL_CODE       TENURE_NAME PRODUCT_TYPE      SRC_OF_BUS_CODE    POLICY_NO
1006882627  200000487   ED                 GRADUATE                      DRM                0005125210
1117262011  204221048   1B                 SENIOR                        GPC                0005158140
1117262011  204221048   1B                 SENIOR                        GPC                0005158140
1068351962  200000543   2B                 SENIOR                        GPC                0005751830
1068351962  200000543   2B                 SENIOR                        GPC                0005751830
1061033203  200000557                      SOPHOMORE                     DRM                3493244771
1061033203  200000557                      SOPHOMORE                     DRM                3493244771
1003899697  200000565   1B                 GRADUATE                      DRM                0006042780
1003899697  200000565   1B                 GRADUATE                      DRM                0006042780
1025635018  202844040   3A                 GRADUATE                      GPC                0006047360
```

File Paths and Formats

Sections 1, 3 and 4 are used to both determine and display information about the input file's format and location, as well as the location of the output file, if one is to be created.

File Format

Section 1 determines how the input file is interpreted. File Query will do it's best to set the format appropriately for each file, but the settings can be modified if needed. The first two buttons from the left switch the file type from delimited to fixed, and if it's delimited, what delimiter is used. The 'Set Fields' button does exactly what it says; it allows you to set the fields that are used, and has slightly different options depending on if the file is fixed or delimited. The last four buttons, from left to right, determine in the file has a header, if it is quoted (any delimiter within quotes is treated as normal text), if the fields should have their white space trimmed, and if the text should be uppercased on input.

File Paths

Section 3 shows the both the path of the input file, as well as where the output file would be written to. You can change them by typing directly into the text boxes and hitting enter, or by clicking on the folder icons to the left, which would open a file dialog.

Field List

Section 4 lists all of the fields defined for the current file. When File Query opens a delimited file, it will automatically populate this list. If there is a header, it will use it for the field names. Otherwise, it will use default names. If you are opening a fixed file, then you will need to manually set the fields yourself by hitting the 'Set Fields' button above the field list. You can also use this button to override the field names for a delimited file as well if you want to.

Below the field list are some options for changing the sort order of the list from the order they appear in the file (default) or alphabetically. There are also three buttons for copying the field list to the clipboard. You can either copy it with each field name on a separate line, or have them separated by a comma. They last button allows you to specify more options for how you want to copy the list, such as using a different delimiter, uppercasing the text or replacing spaces with a different character.

Query Controls

Section 2 is the main control area for running queries. It allows you to start and cancel queries as well as view their progress. (The simplest way to run a query though, is to use the F5 keyboard shortcut though.) If you have multiple queries, then you must highlight the one you want to run before hitting the run button. The two status labels to the right of the controls tell you what line the query is currently on and how fast it is processing them (in lines per hour).

Query Entry

Section 5 is where the real action takes place. Here you will enter queries, pull up the in-line help and determine how a query is written out. The two text boxes in the bottom left are the in-line help, and they contain information and examples for almost any thing that you can do with File Query. The left one lists all of the help categories, and the right one has the actual help items for the current categories. When looking through them, notice that as you mouse over each item, a help box comes up telling you what the item does, and any additional information there is about it. The most useful category for when you are just beginning is the 'Templates' category, as this contains fully written queries that allow you to do many of the most common tasks without having to write the code yourself. You simply modify the properties at the top of the template, and can usually ignore the rest of the code. Once you start to get the hang of it though, you can begin modifying the templates to gain even more control, and that's where the other categories come in, as they show you what else is possible.

The two buttons to the right determine how the query is written out. If the 'Write to File' button is selected **before** a query is run, then the query will be written out to the output path shown at the top of the screen. If it is not selected, then the query will be displayed in the results screen, and not written out. If you already ran the query, but didn't check the 'Write to File' button, then you can use the 'Save Results to File' menu option from the File menu. Last, there is the 'Append' button which only takes affect when 'Write to File' is checked. When 'Append' is not checked, if a file already exists in the output location, a dialog will ask you if you want to overwrite the file or cancel the query. When 'Append' is checked, the query will just be written to the end of the file if one already exists, or create a new one if it does not.

File Preview and Query Results

Section 6 is pretty much the beginning and end of most queries. When you first open a file, the first 100 lines are displayed in the preview window, allowing you a quick glance of what is in the file. One of the nicest features of the preview window is that it allows you to open up a file of any size, since only the very beginning is read. You can be opening a 20GB file from across a network, and it will still open up immediately. Once it's open, you will notice two buttons up in the top right of the preview area. The button on the left with the vertical lines is called the 'Normalization' button. This is used for delimited files, and will simply spread out each field to the width of the largest field value, so that each field is displayed as an easy to read column. If you have to look at delimited fields often, this can be a life saver. Go open up a file now and try it. I'll wait. Back already? OK, the other button on the right is the refresh button. This will simply re-read the file into the preview window. This is handy if either the file is changed, or you have normalized it, and want to go back to seeing what the file actually looks like.

The second window is the results screen. This will automatically get selected after a query is run, and is (surprisingly) where the results are displayed. The results will either contain the actual output of the query, or if the 'Write to File' button was selected, then it will simply list the location of where the output was written to. Also, the results screen can only display a maximum of 10MB of text, so if the query returns more than that, it will be written out to a file as well. The only other thing to notice about this screen is the two buttons up in the top right. The refresh button is disabled, since it is not reading directly from a file. This also causes a warning message to pop up when you try to normalize the results. It reminds you that if you want to go back to an un-normalized view of the results, you will have to re-run the query. For a large query, this may not be some thing that you want to do.

Status Bar

Section 7 is the status bar, and is simply a place where information about the current status of the file, cursor location, and query results are displayed. Starting from the left, you can see the line count for the file. It is important to note, however, that on a delimited file, this is only an estimate and will have a tilde (~) in front of it to signify that. If it is a fixed file, it will be an exact count. The next item is the number of records returned from a query (not including any header) and is only set when a query finishes. The remaining items simply show where your cursor is located, the ASCII value of the current character (in both decimal and hex), and the number of characters selected.

Opening Files

While it may not seem like it deserves it's own section, there are many different ways to open files in File Query. The three most obvious from the interface are to use the File->Open menu item, click on the folder icon next to the input path at the top of the screen, or type it directly into the file path and hit Enter on the keyboard. If three different options weren't enough though, there are two more as well, and they usually more convenient than the others. The first is using the Ctrl+O keyboard shortcut, and the last (and most convenient) is to use the 'Send To' menu from Windows Explorer or your desktop.

Simply right-click on any file, click ‘Send To’ and then click on File Query. This way, if you already see the file some where, you don’t have to open File Query and then locate it again. Even better, this is the only way that allows you to open multiple files at once, which you can then query as a single file!

The Language

What is a Query?

Most of the work done in File Query is done through (surprisingly) queries. These are short blocks of text that tell the program what to do with the file and data. It's very similar to an SQL query, but structured a little differently. In fact, queries are written in C#, which is a standard programming language from Microsoft. Don't let that scare you though. It is only a very small portion of it that you are using, and there are lots of features that make your queries much simpler than a normal program.

The Basics

Queries are very simple once you know the basics. Let's start by looking at the most basic query that simply selects every thing from the file, un-changed. This is already entered in the query entry area when the program opens. Almost all queries will be created by modifying this.

```
var query = (
  from line in inputFile
  select line
);
```

Going from the top, the first line is simply saying that our query will be equal to whatever we put between the open and closed parenthesis. In almost all cases, you never need to change this. The next to lines can be read as "For each line in the input file, select the entire line." The last line ends the query.

Now let's try some thing more useful.

```
var query = (
  from line in inputFile
  where line[0] == "VALUE"
  select line
);
```

This query is the exact same as the first, except it only selects lines where the first field is equal to the text VALUE. This demonstrates the two main ways of selecting data from the file. `line` by itself will select the entire line. `line` with a number after it (enclosed in square brackets) represents an individual field in the line. The indexes start at 0, so the 4th field in a line has an index of 3. If you look in the field list to the left of the query entry area, all of the fields will be listed with their indexes, so you don't have to worry about figuring them out. If using a number seems odd for accessing a field, you can use the fields name instead, such as `line["CONTACT_ID"]`. It does the exact same thing, but indexes are usually preferred, since they are much quicker to type.

Templates

One of the nice features of File Query is that it is filled with plenty of templates for some of the most common tasks that people use. Templates contain a fully written query, and simply give you options that you can set. While you usually don't have to change the query itself in a template, you are able to if you want, and they can often be a good starting point for other queries. Here is a query for finding all duplicate values within a certain field.

```
var fieldIndex = 0;


//Implementation-------------------------------
var query = (
  from line in inputFile
  select line[fieldIndex]
).Freq().Where(f => f.Count > 1);
```

As with most templates, you will see one or more settings at the top, and the actual code will be below the word Implementation. Any text that starts with // is considered a comment, and will be colored green. These have no affect on the query, but help to make it more readable. In this template, all you have to do is change the 0 to whatever field index you want to check (or leave it alone if you are checking the first field) and hit the run button.

Functions

The three types of functions listed below are what really set File Query apart from any other tool for analyzing text files. By allowing you to use any combination of them wherever and however you want, you are no longer in limited in what you can do. Want to select only lines where the first is not empty and the second field starts with "PRE_"? No problem. You want those same lines, but with out the 7th column? Go nuts!

You can get a list of all available functions for any type by selecting the appropriate category from the help boxes below the query entry area. Remember to read the help information that pops up as you move your mouse over each one. It will explain what each one does and how to use it.

Field Functions

When you select a field, such as `line[0]`, you are selecting the text exactly as it exists in the file. Field functions allow you to then change its value, or gather information about it. To use one, simply place it after the field, separated by a period. The easiest way to do this is to place the cursor after the field and then hit the paste button next to the help boxes (or Ctrl+Shift+V). This will type in the period, the function, and default values for any parameters that are required. Below are some common examples of field function uses.

```
where line[0].Length < 5
where line[0].EndsWith("_ID")
where line[0].ToUpper() == "MISSING"
where line[0].Contains() == "NAME"
select line[0].PadLeft(10)
```

Line Functions

When you select a line, such as `line`, you are selecting the entire line as a single value, even if it's delimited with multiple values within. Using line functions however, you can modify it based on its individual fields, or gather information about it, such as its line number and length. The most common uses of line functions are to remove or modify individual fields, and to change the format, such as from fixed to delimited, or from comma delimited to pipe delimited. Here are some examples of line functions in use.

```
select line.AllFieldsExc(6, 12, 39)
select line.SetField(5, "01/01/1800")
select line.ReplaceFieldValue(2, "OLD_VALUE", "NEW_VALUE")
select line.AllFields
```

All of the examples above should be pretty self explanatory, except for the last one. As stated above, selecting just `line` will select the whole line as a single value. If you have a fixed file being read in and you want to write it out as a delimited file, just changing the output format to comma delimited and selecting `line` will not work as it will select the whole fixed line as a single value that does not need any delimiters. By selecting `line.AllFields`, you are now selecting the fields individually, and File Query will then put a delimiter between them on output.

Query Functions

In all of the full queries you have seen, you may have noticed that they all end with a closed parenthesis and a semi colon. It was simply mentioned that this ends the query, but really the closed parenthesis ends the query and the semi colon ends the entire statement. What's the difference? Between the end of the query and the statement you can place functions that will operate against the entire results of the query. If you select all of the values from a field, you can then use a query function to get just the unique values. You can even chain multiple functions to then sort those unique values. Want to know what the maximum length of a field is? Select the field's length, and then use the `Max()` function against the query to get the maximum length (Although, since this is so common, File Query comes with a template for doing just that.) Here are some examples of using query functions.

Get the maximum field length:

```
var query = (
  from line in inputFile
  select line[4].Length
).Max();
```

Get a sorted list of all unique field values:

```
var query = (
  from line in inputFile
  select line[2]
).Unique().Sort();
```

Get a random 10% sample of the file:

```
var query = (
  from line in inputFile
  select line
).Sample(10);
```

If you happened to have a look at the help categories in File Query, you may have noticed that there are two categories of query functions, with the second one being labeled as 'Memory Intensive'. Both categories work the same as far as syntax goes, but the difference lies in the fact that the memory intensive ones **may** need to store a lot of information in memory to run their processing. The `Sort()` function needs to store all of the data it's sorting in memory before sorting, so it is almost always memory intensive on a large file. The `Unique()` function just needs to store a list of each unique, so for an ID field on a large file it will probably take a lot of memory, on a gender field, where there are usually only three distinct values, it won't take much memory no matter how large the file is.

Other Code

Besides the templates and the functions, there is still some other code that can be used in your queries, and the following sections round up the rest of it.

Comparison Operators

Operators deal with equality (are two things equal or not equal) and logic operations (and, or, not, ternary). While logic operations may sound confusing to some, it's really pretty simple. They are used within a `where` clause and allow you to combine multiple test, such as saying that a certain field must start with the letter 'A' and be less than 5 characters. (`where line[1].StartsWith("A") && line1.Length < 5)` The not operator can also be used to change the value of a true or false statement. This is most useful when dealing with a function that returns true or false, such as the IsEmpty() function. If you want all lines where the first field was not empty, you could use the following query:

```
var query = (
  from line in inputFile
  where line[0].IsEmpty()
  select line
);
```

The only operator that can be a little difficult to understand at first is the ternary operator. This takes a statement that can be true or false, such as `line[0] == "INDIVIDUAL"`, and returns one of two values depending on the outcome. The following example would check if a field is empty, and if it is then return `"NULL"`. Otherwise, if it had a value, it would just return the field itself.

```
(line[0].IsEmpty) ? "NULL" : line[0]
```

Miscellaneous

For the code that really doesn't fit into any of the sections listed above, we have the Miscellaneous section. This code will most often deal with reading various information about the file (but not from the file), as well as setting information about the query results, such as the header or footer. These will normally only be needed for advanced queries, or ones that need an extra level of automation.

Tips and Tricks

Here are a few quick tips and tricks to use while working with File Query.

Save a Common File Format:

If you find yourself opening up fixed length files with the same format multiple times, you can easily save that format so that you don't have to enter it each time you look at a file. When you have the Set Fields dialog open, you will see a section in the bottom left labeled 'Saved Layouts'. After filling out the layout, Just hit the save button, and enter in a name for the layout when a dialog pops up (it defaults to the file's name). The next time you open up a file with the same layout, go back to the Set Fields Dialog and you will see the layout listed in the combo box. When the correct layout is selected, hit the load button (has a picture of a folder) and it will open and load the layout.

Find Invalid Characters:

A huge problem for many data feeds is receiving invalid characters within a data field. Perhaps you have a field that is supposed to contain only numeric characters, but is screwing up a process because some of the values have other characters. While the `Contains()` function makes it easy to find fields that have a certain character or value within them, it doesn't allow you to look for types of characters. This is where the `Like()` function comes in with it's support for regular expressions. While the subject of regular expressions is too large to get into here (There is a quick reference in the in-line help for the `Like()` function and countless resources online.), here is a quick example of how to find all lines where the first field contains a non-numeric character.

```
var query = (
  from line in inputFile
  wwhere line[0].Like(@"[^0-9]")
  select line
);
```

Here is another variation of the where statement that allows extra characters in the field, in case of negative, decimal or currency values.

```
where line[0].Like(@"[^0-9.+\-$]")
```

Creating a Test Query

When working on a large file, it can be frustrating to run a long query, only to find out that the result wasn't what you wanted, and you need to re-write and re-run it. Using the `.Take()` function, you can drastically speed up testing and collection of partial data. By using `.Take()` on the input file, you can tell the query to only run against the first few records, meaning that if your file has 10,000,000 records, you can run against 1,000 until you know the query works, and only then run against the full file. Here is an example.

```
var query = (
  from line in inputFile.Take(1000)
  select line.SetField(9, "18000101")
);
```

The other option is to use the `.Take()` function against the results of the query. This tells it to run the query until a certain number of records are returned. Say you have that same 10,000,000 record file, and you know that there is a small number of records with an empty CONTACT_ID field (the first field). If you just want to look at a few examples of the bad lines, and don't need to see all of them, you can use the following query, and it will return as soon as 10 of the offending records have been found, and won't try and go through the entire file (unless there are less than 10 throughout the entire file).

```
var query = (
  from line in inputFile
  where line[0].IsEmpty()
  select line
).Take(10);
```

View a Fixed File as Delimited

Fixed files are not the easiest files to read manually, especially when the number of fields gets large. To make things easier to visually analyze a fixed file, you can view it as a delimited file with a header. You can then use the normalize button in the results screen to put every thing into nice and even columns. Doing this requires nothing more than replacing `select line` with `select line.AllFields` in the default query. This will cause File Query to select each field individually, and since the default output format is a pipe delimited file, that is how they will be outputted. To have the header get written out as well, simply go to the 'Set Output Format' dialog box (Ctrl+T) and select the 'Write Header' option, but leave the 'Custom' option un-checked, telling File Query to use the field names for the header.

Remember the `.Take()` function if you only want to view the beginning of a larger file.

Entering Fixed Formats

When entering in the format for a fixed width file, you are supposed to enter in the start position and length of each field. If you happen to have just the start and end positions, without lengths, that is fine as well. File Query will check the Length column, and if every value appears to be an end position, it will ask you if you want it to automatically convert them for you. While you can always get the lengths yourself when you have the start and end positions, it's still a handy time saver to let File Query to the work for you.

Also remember that you can paste an entire format in at once using Ctrl+V if you happen to have it already in a spreadsheet or Word document. You can select all three columns of data at once, so there is no need to do one at a time. And to save even more time with file format that you use often, you can hit the little disk button at the bottom of the Set Fields dialog to save that format for later re-use.