RawQuery

Documentation

Current Version: 01.01.07
Released: 10/14/10

Table of Contents

Introduction

RawQuery is a transport level application that is able to fulfill multiple roles in dealing with CGI programs and HTTP requests.

Originally designed as a CGI debugger, RawQuery uses a generally static flat-file containing the contents of a query to submit to a CGI program. The idea was to get away from the cumbersome world of GUI while debugging CGI programs; you simply have a file, and the contents stay filled in. If you change your CGI program, necessitating a change in the form fields, you are forced to reload that form in a browser and fill it out all over again. With RawQuery, you just add or delete whatever fields you need to, and execute the program again. And unlike web browsers, you have access to the resultant HTML source (or file contents) without any rendering; there is no hunting for View Source. In this role, it is meant to be a brute testing method that gets you to the point where you would want to bother with a browser to look at the esoteric layout of your resultant page. The emphasis is on proving functionality.

After completing RawQuery, it became obvious that the tool had other applications. One of these is pure content retrieval of either static files (HTML or other formats), or of dynamic content derived from a CGI application, which may or may not require query data to be submitted.

Still yet another use is as a query broker. While this may fall partialy under the role of content retrieval, when it involves actually supporting the infrastructure of an enterprise, it becomes far more. One typical use is to use RawQuery to bridge the gap between a public web server and material on a private, isolated server, through a firewall. Another use is to pass along the data from CGI requests locally to a third party data supply house, and return their results for local use. In both cases, you are essentially brokering a request for information.

RawQuery performs no parsing of the results of a query or transmission. Whatever you get back in the result file, it is up to other software to parse or disseminate the data as desired. RawQuery is a transport mechanism, and could be considered an HTTP "driver" of sorts, which your applications may use as needed. The beauty of this is that the program does one thing, and does it well. It is a truly generic transport mechanism, but one robust enough to fill multiple roles within the same enterprise, even on the same server.

RawQuery is a command line tool. There is no GUI, nor will there be one written by Fairlight Consulting. However, RawQuery is not limited to command line use, and may easily be used as an embedded application or "driver" within other software, which would call it as an external.

Features

- Supports GET or POST methods.
- Supports newer 'multipart/form-data' encoding scheme.
- Supports HTTP File Upload, including multiple uploads.
- Supports accessing documents/CGIs protected with usernames and passwords via the HTTP Basic Authentication scheme.
- Supports cookies, including backing up cookie states.

- Ability to specify your target URL on the fly. Change which CGI you're sending the same query to in order to test different versions of the same CGI!
- Ability to use files containing XML queries and have them compiled and encoded into your complete CGI query. Includes multiple XML queries.
- Ability to use proxy servers.
- Ability to redefine the User-Agent.
- Ability to manually define the HTTP_REFERER setting.
- Ability to do dumps of your formatted queries to file to more easily facilitate debugging sessions or other analysis.
- Ability to access HTTP headers.
- Ability to show headers of HTTP requests from all redirects en-route to the final resolved URL.
- Ability to display the final URL after all redirections as part of the output result file.
- Ability to simply retrieve a document without posting any data. Simple GET operation is a snap.
- Ability to specify query and output filenames.
- Ability to override working directory.
- Ability to comment your query files.
- Built-in paged help.

Requirements

This program requires Perl version 5.6 or higher to run. It also requires several Perl modules to be installed on your system to operate correctly:

- libwww-perl (LWP)
- Getopt::Long
- Crypt::SSLeay

The compiled Windows version takes care of these prerequisites for you.

Licensing

It should be noted that there is no demo version of RawQuery, nor will there be a "Lite" version. Even stripped of many features, it has the ability to be a foundation block of an enterprise's HTTP communications infrastructure, and there is no way to eliminate that possibility and still provide a working demo version.

"Why can't you compile it for unix/linux?" Because the compiler used is Perl2Exe, and while they provide for cross-compiling, We would need the Crypt::SSLeay module for every platform we cross-compile for. We don't have access to every supported platform to make this a reality. In the interests of low overhead and uniformity, there will simply be no Lite or demo versions.

We will provide whatever pre-sales assistance is required to demonstrate whether RawQuery is the right product for your needs. If you have questions or comments, please direct them to sales@fairlite.com without hesitation, and we will answer as promptly as possible.

We realise that software is an investment, and hope that you preview the documentation available for this product and make sure that it will suit your needs before purchasing a license. We also hope you avail yourself of the opportunity to ask any pre-sales questions you might have. All license sales are final and non-refundable.

RawQuery is licensed at a cost of **$250.00 USD** *per server it is installed on*. Bulk discounts may be negotiated for purchases more than five copies at the same time.

Each license fee entitles you to use RawQuery on one server, in any role you require. You may modify the program to further suit your needs, and are under no obligation to release changes back to Fairlight Consulting. However, derivative works and/or modified versions **may not** be resold or otherwise distributed. Similarly, you may not copy RawQuery, modify it, and run the altered version on another machine. You must purchase another license to use it in any form, altered or otherwise, on an additional machine. You may use an altered version and the original version on the same machine under a single license, however. ***"Machine" shall be defined as one instance of an operating system, for the purpose of this license. Machines which run multiple concurrent operating systems (virtual machines) count as multiple machines, and require additional licenses for each instance.*** The licensee agrees to keep the source code confidential and protected.

Upon receipt of payment for a license, access to the program will be generated for the licensee, and such information shall be delivered to the email address associated with the PayPal payment.

Upgrades for the product are *currently* free when moving to new minor and major versions. Fairlight Consulting reserves the right to change this policy in the future, with no prior warning.

There is ***no warranty*** for this software. This software is offered **"AS-IS"** and without warranties as to performance or merchantability or any other warranties, whether expressed or implied.

Good computing practice dictates that any program should be thoroughly tested with non-critical data before deploying it for production. The user assumes the entire risk of using the program. In no event shall Fairlight Consulting be held liable for loss of data, failure of performance, or any other damages, be they real or perceived.

If you agree to these terms, click here to order RawQuery!

Already registered and have your account information?
Download RawQuery!

Support

Extended (non-bug-related) support for RawQuery is available at our standard hourly rates. Because we offer pre-sales assistance in determining if RawQuery is right for your needs, and because documentation is readily available, anything not covered by either of these is deemed an at-cost support issue.

You may request *any* kind of technical assistance with RawQuery by sending email to rawquery@fairlite.com, including bug reports and feature requests.

Feature requests may be commissioned for special functionality, if desired. Any non-commissioned requests are subject to being implemented soley at the discretion of Fairlight Consulting. This may include not being implemented at all, depending on how useful we think the feature would be in general. Commissioned requests can be price-negotiated based on whether the features requested are allowed to be re-integrated into the main product, or whether they shall remain proprietary and exclusive to the commissioning party.

We also offer consulting on how to achieve specific results using the product through this support mechanism, and would be happy to assist you in this regard.

Query File Format Specification

The RawQuery query file specification is fairly straightforward, but should be adhered to strictly to ensure proper performance and results.

All query fields are in the form **fieldname=fieldcontents**, each with one field entry per line. No leading whitespace should precede the field name, or be present on the left-hand side of the equals sign. Any whitespace preceding non-whitespace on the right-hand side of the equals sign is treated as literal space to be encoded as data.

Comments are lines that start with a pound sign (#), and are considered comments throughout the entire line. The pound sign should be the first character on the line, with no leading whitespace. Lines with no characters are simply ignored. Please note that comments are optional, and were not even honoured until v01.01.03. Also please note that comments may not be used at the end of a query field line, as a pound sign could legitimately be part of the data, and there is no way to discriminate the intent of its presence past the first character of the line.

When defining fields that denote HTTP File Uploads, the format of the data is specific. Immediately following the equals sign, the pathname to the file to upload should be presented. Optionally, you may specify a MIME Content-Type by following the path with a single space, and a string of the format **-MIME_TYPE=image/gif**, where the appropriate MIME type is substituted for your needs. Should you omit a MIME definition and provide only the path, the default MIME type of *application/octet-stream* will be used for that particular file during upload.

When defining fields for inclusion of XML files, you follow the same procedure as with HTTP File Upload definitions. The difference in this field type is that you do not have any optional MIME type; you specify only the path to the file containing the XML to be integrated into the data stream.

For both HTTP File Upload and XML file fields, you should always remember to use the respective argument for that field in the command line call. Failure to do so will result in the query file field contents being used as the data, rather than inclusion of the appropriate file.

The following example demonstrates a correct query file:

```
# Name
first=Mike
last=Larson
# Picture
portrait=/home/mikel/portrait.jpg -MIME_TYPE=image/jpeg
uploadfile=/home/mikel/someprogram.exe
# XML Data Set
xmlfield1=/home/mikel/xml/somexmldata.xml
# Other information
email=mikel@somehost.com
city=Phoenix
state=AZ
```

Again, please note that the comment lines were not at all necessary, and could have been omitted. These were included purely for demonstration purposes.

Note that to simulate multi-line <TEXTAREA> field input, you will have to have a single line and emulate linebreaks by inserting raw carriage returns. The major graphical browsers use a sole carriage return (\r or ^M for non-coders) as indication of a line break within these fields. In an editor under a unix of any flavour, you can insert this by hitting ^V and then ^M. (The ^M and ^V mean control-M and control-V as keystrokes, if you are unfamiliar with the notation.)

RawQuery Basics

We will now acquaint you with the basics of how RawQuery works, and how to use it at its basest levels.

Perhaps the first thing you should do is try the **--defaults** option by itself to get a feel for some of the settings the program has defined by default. Don't worry about every single one of them just yet. The options page is there to help you sort them all out later.

First, pay attention to the working directory. By default, RawQuery uses the current working directory of whatever process it is called from. This behaviour can be changed with the **--working-dir** option. The important thing to note is that the default filenames for queryfile, outfile, and cookie-jar are all just filenames, not full paths. RawQuery expects to find its files in whatever it is set to use as the working directory. If you change the location of the query file to a full path, and the working directory is still the current working directory ('.'), then it would try to find the full path *under* the working directory. This is likely not the behaviour you want. Always set the working directory to be the path to your files, and set the filenames to simple filenames without paths.

The next important setting is the query file. This is the file that your query contents are pulled from to make the request. (We'll just assume for now that you're actually making a query; you don't always have to, and *can* just grab a web page with RawQuery.) The query file contains the field names and values to be submitted during a request. The full specification for this file is available, and is quite easy to use. Note that you *can* override the default filename, however. This becomes important as you start using RawQuery more heavily and for specific tasks.

Similarly, you can override the output and cookie jar filenames.

With the ability to override the working directory and all three filenames, you can use RawQuery concurrently and without conflict on specific files for many different requests (even of multiple types, in multiple roles) at once. Keep this in mind when considering your overall solutions.

Cookies are disabled by default. If you need to use the cookie functionality, you'll have to enable it with the **--cookies** option. You may also wish to make use of the backup facility, which will be explained further in the section on using RawQuery to debug CGI.

Perhaps the most important option is the **--url** option. Without this, RawQuery is useless. This option tells the program where to submit its request. Multiple instances are not handled at once. Only the last instance encountered on the command line is honoured, should you try and submit more than one.

Using what we know now, we can make a simple query. Let's assume that we are simply debugging a CGI program, so we change directories to our project directory (let's say */home/bob/testcgi*), and our working directory default is fine. We edit the *raw.query* file.

Let's assume the program in question is a simple email support form submission program. First, we write our query file:

```
first=Mike
last=Larson
email=mikelarson@somehost.com
subject=Tech Issues
body=This is a test body.  Do not fold, spindle, or
mutilate.
```

We now have a query file. We're actually all set to make the request at this point. Assuming the default filenames and working directory, the syntax is quite simple:

rawquery -u www.testhost.com/cgi-bin/techsupp.cgi

Rawquery will read your query file, compile the full query, and make the transaction. At the completion of the run, the file *raw.response* contains the raw output of the request, be it HTML, a graphic file, or whatever. We can then look at that file and see if we got the expected results at a low level. Keep in mind that RawQuery does no rendering of resulting HTML. You're looking at the raw results, just as they're sent to any browser.

That's the simplest form of using RawQuery to make a simple request that pushes data to a CGI, and gives you the results back to look at.

It should be noted that if there is a network or protocol level error, like the page not being found (404), or an unreachable host, the error will be present in the response file in the form: *Error 404: Not Found*. The word Error is static, and will always be present. The number is variable depending on the actual error, followed by a colon, a space, and the error message as translated by libwww. Errors will always be a single line. Keep errors in mind for advanced usage (data population, query brokering), so that you make allowances for conditions that might cause an error. If you see an Error 500, you're going to have to look at the error logs on the web server to see what went wrong. There is no way to detect this from remote. The debugging part of this tool is not meant to find that kind of error, but to help trace functional errors with easy, repetative queries that require you to do a minimal amount of input.

At its most basic level, this is how RawQuery works. It is capable of doing a lot more, but these are the basics that will get you started. See the tutorial and options pages from the table of contents for this documentation for more details on what else is possible.

Uploads and XML

Using RawQuery for HTTP File Uploads and sending XML datasets is particularly easy, and offers a lot of power and flexibility. Please keep in mind that these features of course depend on the remote end CGI being able to handle the data properly. If the CGI isn't designed to handle a file upload, using RawQuery on the local end won't change that. Remember that RawQuery is the client-end transport.

Uploading a file with RawQuery

Uploading files is achieved in several simple steps:

- Define an entry in your query file for the file to be uploaded. The format of this entry is:
  fieldname=/path/to/file -MIME_TYPE=mime/type
  The MIME type specification is optional. If not specified, *application/octet-stream* will be used as a relatively safe assumption. If specified, this option should be separated from the pathname by a *single space*.

- Ensure that when you call RawQuery, you specify **--file-field=fieldname**, where "fieldname" is the actual name of the field you defined in the query file to be an upload definition. This tells RawQuery to upload the contents of the file pointed to, rather than simply send the information in that entry as encoded data for the path and MIME type.

- Ensure that you specify the **--multipart** option. This is *required* for HTTP File Upload to work, and is mandated by the appropriate RFC.

- Ensure that you leave the method as **POST**, either by using the default, or by manually using the **--method** option. The **--multipart** and **--file-field** options will not allow themselves to be used without the **POST** method, and RawQuery will give a verbose error and exit, if you try and use the wrong combination by mistake.

Sending XML files with RawQuery

Sending XML files is very similar to uploading files. The main differences are that **--multipart** is not required, and there is no extra optional field for a MIME type.

The following steps should be taken to send an XML file's contents encoded in **POST** data transactions:

- Define an entry in your query file for the file to be uploaded. The format of this entry is:
  fieldname=/path/to/file

- Ensure that when you call RawQuery, you specify **--xml-field fieldname**, where "fieldname" is the actual name of the field you defined in the query file to be an XML file. This tells RawQuery to encode the contents of the file pointed to, rather than simply send the information in that entry as encoded data for the path.

- Ensure that you leave the method as **POST**, either by using the default, or by manually using the **--method** option. The **--xml-field** option will not allow itself to be used without the **POST** method, and RawQuery will give a verbose error and exit, if you try and use the wrong combination by mistake.

CGI Debugging/Advanced Usage

This section of the documentation strives to teach you both how to debug CGI programs using RawQuery, but also how and when to use some of the more advanced features of the program. You may not be using the program for debugging CGI, but rather for content retrieval or query brokering. However, this page also illustrates the finer points of using cookies, setting things like the referer and agent, using Basic Authentication, helping you debug whatever logic flow you *are* using RawQuery within, and more.

Debug Mode

The debug mode can be helpful in showing you what RawQuery is actually doing during execution. The default level (**-d** is the same as **--debug=0**) will show you various program settings, and the final resolved URL if there was a redirection. The next level (**--debug=1**) will show you the server's HTTP response headers to STDOUT. The level after that (**--debug=2**) will also show you the state of the cookie jar both before and after the transaction, also to STDOUT. The debug mode may be helpful if you can't seem to figure out what you have set incorrectly when encountering unexpected results. *If you are reborting a potential bug, please include the output of* ***-d2*** *in your bug report.*

Debugging CGI programs with RawQuery

Let's take an example of debugging a program with RawQuery from real life. Besides being its developer, I am also heavily dependant on it for helping me develop my CGI software. It minimises the amount of data I must enter just to test my software during its development cycle, and adds flexibility and ease to my actual testing. Let's illustrate how I use it to achieve these goals by going through the debugging process I used while developing and debugging the core of a web-based message forum system.

The particular CGI that I used RawQuery the most for debugging was the message posting mechanism. This module underwent probably fifty different revisions, because it is responsible for doing so much within the larger application. The module name (the individual script) is called *cb_createpost*. A sample of the query file I used shows what I was eventually sending to the program with RawQuery:

```
parentid=2
forumid=6
genform=1
anon=0
notifyreply=0
subject=     Third reply in second thread
msgtext=This is a test message.
ticket=post10
```

As you can see, I was sending eight fields in the final version. Some versions had more, some had less.

One *annoying* thing about debugging CGI with a graphical browser (or with any browser) is that if you change the CGI field requirements by either adding or removing one, and your CGI is programmed securely so that it rejects requests with extra or missing fields, you need to reload the page in the browser, fill out all your data again, and *then* finally resubmit the data.

Moreover, I had not one, but thirteen different submissions to make through this engine, as I was eventually testing message threading, which is actually mostly handled through the storage mechanism rather than the viewing mechanism. So the values you see above would change between iterations.

Let's look at the first case, where you just change field requirements. In this event, instead of filling out a form all over again (painful if you have a form with more than about three fields), you just add or delete a line in the query file, and run the program again. Running the program again is easy in most modern shells, which have both arrow history and usually the bang-history (!command_unique_prefix) methods. So I could make changes, and simply call **!raw**, and the test was run again. No pain, no mucking about with a clumsy GUI, no reloading a form. Edit only what you need and move on and *get results quickly, with a minimum of effort*.

In the second case, where I had to run thirteen posts in a row, I did not edit the file between each run. I set up a quick program to generate thirteen files (or I could have just made thirteen files in an editor), and wrote a simple one-liner of Perl that called RawQuery on all thirteen files in turn, using the **--query** and **--outfile** options. I would just run this "testposts" command, and RawQuery was called on each file in turn. All my submissions with differing sets of data (notably the subject, message text, parent ID, and ticket) were all submitted within seconds, rather than the several minutes it would take with a browser. When I needed to check what results I got back, I had my separate result files and could consult them. If I needed to run the test again, everything was in place; all I had to do was flush the database table of the entries and run the program again. Simplicity itself.

Of course, for mild security to prevent cross-site scripting, one can attempt to use the HTTP_REFERER server variable. This can be set with the **--referer** option. It is incredibly easy to spoof this variable, so using it in CGI is a mild nuisance to the determined, at best. But one does everything one can to ensure high security standards. When testing whether or not my referer handling was working correctly, I made use of this option. It was also handy to set it to test what would come back on redirects based on the referer variable.

For those that like to test what happens for different platforms, the HTTP_USER_AGENT variable can be invaluable. You can use RawQuery's **--agent** option to test your program's responses to multiple platforms, all from the same program. This can also be a handy option if you happen to be doing something like polling a site repeatedly and the default agent name gets blocked out. You can then set it to appear to be Netscape, Internet Explorer, lynx, or whatever your heart desires. This isn't considered good manners, so use your judgement. The capability does exist, however.

At one point, I was testing data and wasn't at all convinced that my CGI was handling the data correctly. The **--dump** option lets you look at the raw query stored locally, just as it was sent to the server. This is excellent for making sure you are sending what you think you're sending.

Of course, my CGI makes use of cookies. This isn't a problem with RawQuery. Simply specify the **--cookies** option, and your cookies are automatically handled correctly. The problem was, I kept having to test the same submission condition over and over again, and it was based on what was in the cookie going in, and then it set the cookie to a different data set coming back. And as I said, I needed to make the same test repeatedly. This is where the cookie jar **--backup** option comes in handy. I could just run the program once using the backup option, and see what I got. If I needed to run the test again with the same cookie settings as before, all that was necessary was to copy the backup file to the current file, and then rerun the test. This would be awkward at best in a browser environment. With RawQuery, it's trivial.

One can also use different sets of cookie information on individual runs. For some tests I might want to use the jar that listed me as the user "Fairlight". For others, I might want to use the jar that listed me as the user "Librax", which was necessary for testing administrative override of the posting mechanism in this case. One is an administrative account, one is not. The **--jar** option let me pick and choose who I was at will. You can dictate exactly which cookie file is used with a single option.

Of course, during development, I protected my CGI software with HTTP Basic Authentication so that only I could get at it. This is also a non-issue with RawQuery. Simply using the **--auth** and **--pass** options sets the username and password for this authentication scheme, and you never have to input a thing. These options are best used on systems you trust as secure, like a private development system where you aren't being observed, or can trust the users that are online, since **this information *will* show up on the command line**. Care should be taken when using these options, but they exist for a good reason.

All of these options were used while testing that one piece of CGI. There are other options that I've found useful for testing other CGI, or just plain web pages.

I got a URL for a game site once. The site had been in existance for quite some time, but for some reason it was redirecting me to a porn site in Brasil, of all places. I threw RawQuery at the page with the **--no-data**, **--method=GET**, **--show-final-url**, and **--trace** options, and saw exactly the routing of five redirects that I was passed along to get to my false destination. Apparently the site had been taken over by a traffic aggregator. But I could see the entire path, including all HTTP headers received at each server along the way. The **--trace** option is very helpful when dealing with redirects.

No matter what you're fetching, you can set the standard libwww variables for proxy use, and they will be honoured by RawQuery. These variable names are always in the form *protocol_proxy*, where "protocol" is "http" or "https". In theory, other protocols are available for proxy use. However, RawQuery is limited to doing one thing and doing it well; it only performs HTTP or HTTP SSL requests. But if you want to use a proxy, feel free to set something like **http_proxy="http://myproxy.com";export http_proxy** in a Bourne-like shell. Users of [t]csh should use *setenv* as appropriate. Windows users can set these variables as well, using *SET*.

Conclusion

As you can see, RawQuery offers great versatility and ease of use when dealing with debugging both CGI and general HTTP transaction connectivity. It cuts the pain out of debugging CGI, lets you do things expediantly, and get on with developing your applications, rather than tinkering about with clunky browser interfaces. In addition, you have the bonus of looking at the raw output from the server, rather than needing to hunt for a browser's "View Source" option.

In this role, RawQuery is *not* meant for viewing rendered output. The focus of RawQuery in the role of CGi debugging is on functionality testing, not cosmetics. Once you have your functionality down, you can use a browser to go through it and iron out the cosmetics without having to fill out the forms again, as you've already breezed past all that tedium by using RawQuery for the grunt-work instead.

Content Retrieval

RawQuery may be used for content retrieval from HTTP (and SSL encrypted HTTP) data sources. There are other tools that simply perform **GET** operations (like *wget*), but RawQuery goes one better. You can **POST** data to CGI data sources in order to obtain your data. What you do with the resultant data is entirely up to you, but a few points are worth noting.

The first thing to consider is whether you are fetching a raw web data source or posting to a CGI. If the CGI in question requires (or allows) **POST** operation, the default method setting or an explicit **--method POST** will work fine. However, this will *not* work for obtaining raw web pages or content without posting any data. RawQuery will still look for a query file from which to push data to the remote location unless you take a critical step. The way to fetch raw web content without submitting any data is use of the **--no-data** option. When this option is specified, RawQuery will not look for a query file, but will perform a simple **GET** on the data source in question. It is worth noting that **--no-data** may only be used with the **GET** method. If you use this option, you *must* set the method accordingly.

As when debugging CGI, you may enable cookies, utilise HTTP Basic Authentication, use a proxy server, or utilise any of the other features you require to successfully achieve your content retrieval.

You should *always* check for error conditions in any automated process which does content retrieval. The error format is specified in RawQuery Basics. The facility exists to check for protocol or network layer faults, and any professional use of the product should account for potential errors and handle them gracefully. In addition, if the success code happens to matter to you, you can use the **--show-code** option to have this inserted at the beginning of the file. The documentation for how it is appended and in what format is documented on the options page.

RawQuery does not care if the page you receive is HTML, XML, a GIF, JPEG, or an MP3. It will simply store the resultant data to the filename specified (or the default if you didn't specify an explicit filename). What you do with the data is up to you, and you can use any tools you like to massage, slice, dice, fold, spindle, or mutilate the data to suit your needs.

The beauty of RawQuery is that it's "simply" the transport agent. It's also completely generic and task indiscriminate. Therefore, it is ideal for embedding in many situations. You can use it to fetch the local weather or news at one place in your enterprise, and in another you can use it to fetch the latest stock updates and use that information to update your trend-tracking software, if you like. The end disposition of the data is entirely up to you, and you get to decide what software to use and how to manipulate the data for your specific task. RawQuery is the generic transport layer for your operations. Whether you embed it in an actual HTML Server Side Include, a cron job, or another application, you can fit it to your particular needs.

Query Brokering

Query brokering is a concept that encompasses several uses. At its heart, query brokering is content retrieval. However, it is more in that it is essentially a "relay" transaction. A request comes in to a primary web server. An application then parses out the data specifying the request, and uses RawQuery to query yet another data source behind the scenes. The result is a transparent presentation of information from one site, all of which may have come from multiple sources at different sites.

The other use for query brokering is for enhanced security. If you have a non-network-aware database or application on one system, and want that system to be isolated from the publicly accessible web server, yet need to get data from that application or database *to* the web server and in the end to the user's browser, you can use RawQuery and some minimalistic "bridge" software to accomplish this task.

Site to Site, Business to Business Brokering

Let's assume we run a service providing information. Not all the information we have access to is directly on our own servers, but we have access to it via a CGI interface.

Rather than sending users directly to the "data wholesaler" we may be obtaining our information from, we can make it come transparently from our site.

A user makes a query to a CGI application on our server. This application takes the data that serves as the criterion for the request, parses and reformats it into a query file, and then calls on RawQuery to contact the real data source for the information that matches the given specifications. RawQuery hands back the file containing the results, and with or without further massaging (whichever is necessary or desirable), the CGI can then pass the data on to the browser.

Using this methodology, it is possible to utilise many online data sources, be they CGI or otherwise, so long as they're available via HTTP, and present them as coming from one unified source. It's a method of making your site a "one stop shopping" experience for your users.

In addition, this methodology can be used to accomplish things like database population using XML data transfers from one site to the other when necessary, for example.

Security Brokering

Assume we have an application or database that is not network-aware. Normally you would have to have this on the same server as the public web server itself. This isn't necessarily desirable for security reasons.

Let's implement two servers instead. One is a public server, simply hosting the web site and some CGI applications (at least the one we need to use RawQuery and make this work). We then have a firewall in place to protect our critical systems. Finally, we have a private machine behind the firewall, which houses our application or database. The firewall is configured to allow access to the private machine *only* from the public machine, and *only* on ports 80 and 443. Thus, the private machine is isolated to only being reachable if the public machine is breached, and then only on those two ports. This minimises the amount of trouble someone can cause on the private server.

Let's assume we have a small piece of "bridge" software. This is a simple CGI that detects what URI was originally requested, gathers the data that was originally requested, and sends a request to the same URI on the private server, which houses a CGI to work directly with the target application. The private server responds with the end result, which RawQuery passes back through its response file to the public machine's CGI, which then goes back out to the browser.

In this way, we can minimise exposure to the private machine to two ports, accessible only from one machine. To the outside world, it may look like the data comes right from the web server, but we have just isolated that application, increased its security, and provided the same functionality as if the application sat directly on the more exposed public web server.

Conclusion

RawQuery is the transport layer for query brokering. It requires some minimal software to act as the major application, but it is more than capable of handling the communications requirements of the process. Again, RawQuery is task indiscriminate. Any parsing or logic flow belongs in the application being developed. RawQuery is your transport level "driver" to help you get the job done effectively.

Options

- --defaults

  This option shows you the default settings that RawQuery will use for many of its options if they are not provided/overridden with arguments on the command line.

- --debug, -d integer

  This option takes an integer expression (or can stand alone, in which case it assumes zero *but not 'off'*) to set the debug level of the program. The default setting of **0** employs most of the debugging. Currently, a setting of **1** shows you incoming HTTP headers to STDOUT during execution, and a setting of **2** shows you that information, as well as information about cookie states before and after a transmission.

- --agent, -A string

  With this option, you may change the default User Agent name that the program reports to web servers. For example, you might want to show something like "My-Web-Crawler/3.0". The proper format is **NAME/VERSION**.

- --working-dir, -w path

  Use this option to specify a directory in which to work. This affects where the query files are read from, and where the result files are written to. XML and HTTP File Upload locations are not affected by this option, as they are assumed to be sensibly pathed in the query file, if they are used. The default is the current working directory of the parent process launching RawQuery.

- --query, -q filename

  This option denotes the filename to read the query data from. The default is **raw.query**, and will be read from the working directory.

- --outfile, -o filename

  This option denotes the filename to write the resultant data to. The default is **raw.response**, and will be written in the working directory.

- --url, -u URL

  The most important option, this tells the program which URL to POST to or GET from. You may specify **http://** or **https://**. If you do not specify either (ie., *www.myhost.com/cgi-bin/test.cgi*, then **http://** is automatically assumed.

- --referer, -r URL

  This option allows you to "spoof" the HTTP_REFERER environment variable that the remote server sees. This is particularly important to set if you are testing or using a script that checks this variable for "appropriate" contents. The value should be a valid URL.

- --method, -m GET|POST

  This option lets you override the method used for your transaction. The default is **POST**, and if you plan on using this method, you do not need to specify a method (although it will not hurt anything, and accepts **POST** for the sake of uniformity). If you need to, specify **GET** with this option. You would want to do this for CGI's that require the GET method, or in conjunction with the **--no-data** option, which would let you simply retrieve a web page without posting any data at all.

- --multipart, -M

  The original CGI encoding method was *x-www-application-urlencoded*, which URL-encodes the data and sends it to the CGI application. This is the default encoding that RawQuery uses to submit data.

  A new encoding, *multipart/form-data* was introduced to handle specific tasks. Based in MIME, the encoding scheme is ideal for handling HTTP File Upload functionality, and very long data fields (such as data generated by use of TEXTAREA fields). If you use the HTTP File Upload functionality, you MUST specify both the **POST** method using the **--method** option, and this option.

- --dump-query, -D filename

  This option takes a filename as an argument and uses that location to dump the *encoded* query for local analysis. This is useful for looking at what your query looks like to the remote application, especially if you start getting unexpected results. This option is *not* usable as a logging facility, as the dump overwrites not appends. (The rationale for this is that if you're doing file uploads, you don't really want to store potentially several megabytes of data per transaction.)

- --file-field, -F fieldname

  Use this option to tell RawQuery which field name in your query file should be treated as an HTTP File Upload field. The format of the field is noted in the Query File Format Specification. Use multiple instances of this option, each with its own argument, to upload more than one file. An example would be **-F upload1 -F upload2 ...**. This option can only be used if **--multipart** is specified, which also requires the method to be **POST**.

- --xml-field, -X fieldname

  This option works similar to **--file-field**, but is used to tell RawQuery that the file contents of the XML file to incorporate into the query should be encoded as the contents of the field. While **POST** must be used with this method, you do *not* need to use **--multipart**, although it may be recommended if you are sending extremely large amounts of data, as multipart encoding is more robust, and you cannot overrun the environment space of the CGI's command line options, which is a concern when using **GET** with too much data.

- --cookies, -c

  This option enables the use of cookies. Cookies will be stored in a cookie jar in the working directory when set, and taken from the cookie jar when requested. Unless you specify this option, cookie use is entirely disabled.

- --backup, -b

  This option will make a backup of your cookie jar file in the working directory prior to performing the transaction. The backup file will be suffixed with a **.bak** extension. Keep in mind that the file that's backed up is whatever the current cookie jar file is for this run. If you set it with **--jar**, it will back up whatever file you tell it to use as a cookie jar.

  This option is very useful when you're testing cookie use with a script or web page, and want to replicate a test with previous cookie contents, rather than what you've received in intermediate tests. You can simply copy the backup file back to the jar file and re-run RawQuery in this event.

- --jar, -j filename

  You may set the name of the cookie jar file with this parameter. The default name is **raw.cookies**. The cookie jar will be stored to and read from the working directory.

- --show-headers, -s

  If this option is specified, all HTTP headers from the server are saved to the result file, followed by a blank line before the actual response data. This is useful for analysing what the server thinks it sees, and what it may be doing that is possibly unexpected.

- --trace, -t

  If specified, this directive will store HTTP headers from every redirection encountered on its way to the final resolved URL, which will differ from the URL specified if redirection did occur. In complex situations, this can be useful for tracing the path a request took and what else may have transpired at these hops.

- --show-code, -C

  If activated, this option causes the first line of the response file to contain the status code and message for the transaction. The format would read: *Status 200: OK* by way of example. The word "status" is constant, as is the colon. The numeric code and text message are variable. A blank line will follow this status line, if this option is active.

- --show-final-url, -S

  If activated, this option will insert four lines into the response file, denoting the final resolved URL. The second of these four lines is the actual URL, and the fourth is a blank line. The other two lines are strictly formatting.

- --auth, -a username

  To access pages or programs protected by HTTP Basic Authentication, you may specify the username for the protected resource using this option. You will also need to use **--pass** to give the password for the username you provide.

- --pass, -p password

  This option is used in conjunction with **--auth** to give the password for the username utilised in accessing a resource protected by HTTP Basic Authentication.

- --no-data, -n

  This option tells RawQuery *not* to send any query data, or look for a query file. This option is most useful when doing simple content retrieval from non-CGI sources (ie, regular HTML pages, or other resources that require no content submission), or when simply using **--show-headers** to check a server's headers for specific content by retrieving "any old existing page" from the server.

- --timeout, -T seconds

  This option sets the timeout for requests in seconds.

- --version, -v

  This option causes RawQuery to issue its version information and exit.

- --changelog

  This option results in a paged output of the change log for the product, citing what differs between versions of the program.

- --help, -h

  This option results in a paged listing of all available options for RawQuery.

Change Log

```
Fairlight RawQuery - Corporate Version v01.01.07
Copyright 2002-2008, Fairlight Consulting
=========================================
REVISION HISTORY

     v00.00.01      Original code base for alpha release.
     v00.00.02      Added code for Basic authorization, Cookies, and
                    the display of HTTP response headers.
     v00.00.03      Added the ability to just GET a page without posting
                    any data, so we can be used as a response, header,
                    and cookie debugger outside of CGI context.
     v00.01.00      Moved to beta after further testing.
     v00.01.01      Added cookie jar backups, in case you want to revert
```

```
             to an earlier state for testing purposes.
v00.01.02    Added paging to help, changelog, and defaults.
v00.01.03    Added ability to change user agent name manually.
v00.01.04    Changed some of the URL encoding of query strings so
             it would be more fault tolerant to careless testing.
v00.02.00    Added multipart/form-data encoding scheme for POST
             queries, complete with file upload capability.
v00.02.01    Fixed problem with boundaries in
             multipart/form-data.
             Added a close() for nicety in an exit condition.
v00.02.02    Added query dump for debugging of queries locally,
             so you can more easily see what you're sending.
v00.02.03    Added optional second argument on multipart file
             arguments in query file.  A valid Content-Type can
             follow the filename after a space.  If none is
             specified, the default is application/octet-stream.
             This is on a per query-entry basis in the query
             file.
v00.02.04    Added more comments to code.  Debugged and
             rearranged
             some conditions and logic.  Updated README file.
v00.02.05    Fixed --defaults exit location to falling -after-
             the pager filehandle code that was added on the
             wrong side of that line originally.
v00.02.06    Added user agent cookie_jar for redirect-handling of
             cookies inside the agent during redirecing.  Added
             printout to file of final resolved URL to account
             for redirects.
v00.02.07    Fixed --help verbiage for -F argument.  Added
             limitation and workaround notes regarding LWP to
             the README file.
v00.02.08    Added -S to make showing final resolved URL
             optional.  The feature is not used by default.
v00.02.09    Changed MIME type specification to account for
             filenames with spaces in them.  The MIME type
             should now start with a -MIME_TYPE= in the upload
             file field.  Fixed text-only file upload bug where
             I was optionally tacking on \r\n only if the buffer
             did not end in such.  It requires it to be there
             anyway or there is a file size mismatch when
             uploading text files from windows.  Altered README
             file with new spec for Content-Type definition on
             uploads.
v01.00.00    Added --trace functionality to show headers of all
             hops, including redirects.  Moved to gamma release
             code.
v01.00.01    Tweaked trace output formatting in report.
v01.00.02    Added encoding of ':' and '@' characters so there is
             no possible interference with GET URL types.



v01.01.00    Added in XML query building from files.  You can
             build an XML query file and specify the file name
             as the argument to the field name, similar to HTTP
             File Upload.  This function is valid only in POST
             mode.  This function first appeared in a stripped
```

```
               down version of the original basis of what
               eventually became the first rawquery version.
               The functionality has now been folded in.
v01.01.01      Added environment-based proxy use.
v01.01.02      Updated documentation for changes from v01.01.00 and
               v01.01.01 after doing stress testing.
v01.01.03      Added ability to include whitespace and comment (#)
               linesin query files.  Added -C option to retrieve
               the status code on non-error transfers.  Changed
               default user agent name.
v01.01.04      Added urlencoding of /, ?, ;, and $ symbols.
v01.01.05      Added -T timeout option.
v01.01.06      Added encoding of esoteric reserved characters.
v01.01.07      Removed –link-timeout option that didn't exist.
```

Output of --help

```
Fairlight RawQuery - Corporate Version v01.01.07
Copyright 2002-2010, Fairlight Consulting
==========================================
HELP

Usage:  rawquery [options]

Options:
     --defaults                    Show default values for settings.
     --debug=x, -d x               Debug level setting.
     --agent=NAME/VERSION -A N/V   Set user agent name manually.
     --working-dir=path, -w dir    Path to working directory where
                                   working files and image spools are
                                   stored.
     --query=file, -q file         Query file to use for input.
     --outfile=file, -o file       Output filename to write.
     --url=URL, -u URL             URL for CGI to query at remote end.
     --referer=URL, -r URL         HTTP_REFERER to send.
     --method=GET|POST -m GET|POST Method for query action.
     --multipart, -M               Use 'multipart/form-data' instead of
                                   the old
                                   'application/x-www-form-urlencoded'
                                   encoding scheme.  Useful for file
                                   uploads and other CGI's that may use
                                   it.
     --dump-query=file -D file     Dump the query to a file for
                                   analysis of constuction on the local
                                   system.


    --file-field=field, -F field   Field name in query file to
                                   recognize as a file to upload,
                                   rather than raw data to send as a
                                   field.  Valid for use only with
                                   the --multipart option.  Multiple
                                   files may be specified with repeated
```

```
                              uses of -F field -F field [etc...].
--xml-field=field, -X field   Field name in query file to
                              recognize as an XML file to encode
                              into the query under the field name
                              given.  Valid for use only with the
                              POST method.  Multiple files may
                              be specified with repeated uses of
                              -X field -X field [etc...].
--cookies, -c                 Use cookies.
--backup, -b                  Backup cookie jar file.
--jar=file, -j file           Cookie jar file in which to store
                              cookies.
--show-headers, -s            Show the HTTP response headers in
                              response file before the response
                              itself.
--trace, -t                   Show headers of all HTTP requests
                              that were
                              processed along the way from all
                              redirects.
--show-code, -C               Show status code and message as the
                              first line of the result file
                              (followed by a blank line) for
                              successful transfers.
--show-final-url, -S          Show the final URL that the query
                              engine could resolve to in the
                              report.
--auth=name, -a name          User name for Basic Authentication.
--pass=password, -p password  Password for Basic Authentication.
--no-data, -n                 Do not use a query file or send query
                              information.  Do a straight fetch.
                              Use this only with the -m GET
                              setting.
--timeout seconds, -T seconds Timeout for requests in seconds.
--version, -v                 Show only the program version and
                              exit.
--changelog                   Display revision information.
--help, -h                    Show this help screen and exit.
```