xSkrape Fundamentals

Contents

This document is intended to give both business stakeholders and technical resources enough basic background about xSkrape to evaluate it and start using it. More detailed reference documentation and detailed examples are available on-line, but these assume you already understand basic concepts which will be covered here.

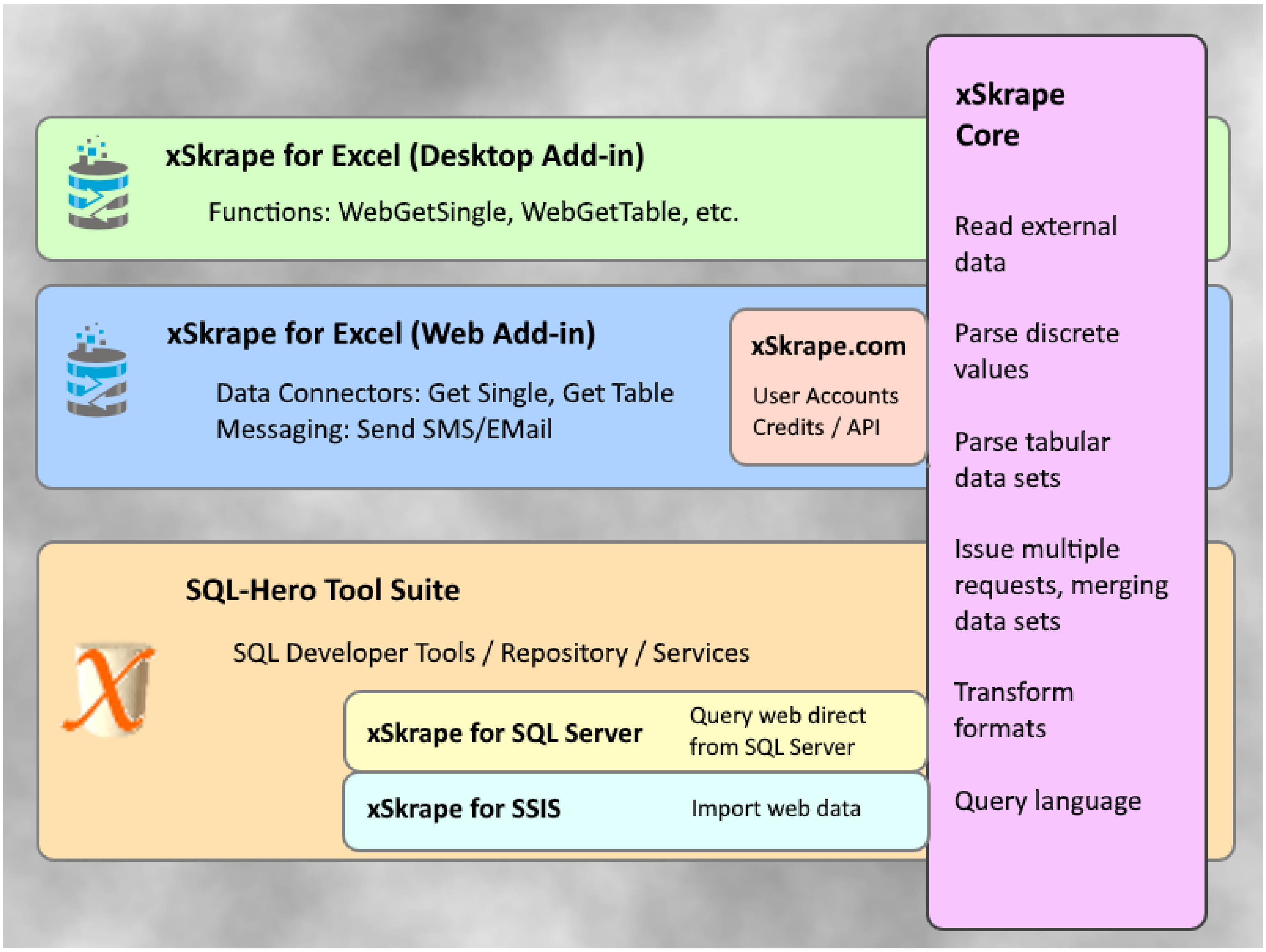
What *is* xSkrape?

Data sources exist everywhere: from well-defined Application Programming Interfaces (API's) - to not-so-well-organized HTML web pages. xSkrape offers two main data access paradigms: extraction of discrete values, and extraction of tabular data. In the xSkrape web-based add-in, a third feature area has been introduced: messaging right from within Excel. The latest version of xSkrape also introduces the xSkrape web site and API, available on credit-based usage.

In extracting values, xSkrape offers options that other solutions simply don't. For example, allowing use of XPath to extract values from both XML *and* JSON sources offers both consistency and a high level of flexibility given the maturity of XPath. Another rather unique capability is in being able to merge multiple requests, as illustrated in many of our examples and samples. More complex concepts such as *join sets* are being introduced over time and the product evolution will allow users to take advantage of improvements immediately (for web-based) or as an upgrade choice (for the desktop add-in).

xSkrape is really a *family* of products with similar capability. xSkrape functionality is exposed to Excel users through the desktop add-in *and* a web-based add-in. The desktop add-in adds a handful of new Excel functions. For SQL developers who use SQL Server, similar capabilities are available in xSkrape for SQL Server. This tool suite offers a way to query remote data sources in real-time, using familiar SQL constructs such as views and user-defined functions. For SQL developers who prefer to capture the remote data in an ETL (Extract, Transform and Load) type process, xSkrape for SSIS (SQL Server Integration Services) offers a way to take advantage of the "smarts" of xSkrape in both *tasks* and *data sources*.

The following diagram illustrates the relationship between the various "flavors" (editions) of xSkrape. All editions share a core set of functionality that includes our proprietary query language. Purchasing the desktop or SQL Server editions implies your computing resources will be running the xSkrape engine, whereas using the web-based Excel add-in as offered in the Microsoft Store uses *our* computing resources, hosted on xSkrape.com. This is a completely different model where we charge for use, by transaction, using a credits-based system. You can use xSkrape.com to purchase credits to "top up" your account, share your account over multiple clients if you choose, manage your profile, and more.


xSkrape for Excel (Desktop Add-in)
Functions: WebGetSingle, WebGetTable, etc.
xSkrape for Excel (Web Add-in)
Data Connectors: Get Single, Get Table
Messaging: Send SMS/EMail
xSkrape.com
User Accounts
Credits / API
SQL-Hero Tool Suite
SQL Developer Tools / Repository / Services
xSkrape for SQL Server
Query web direct from SQL Server
xSkrape for SSIS
Import web data
xSkrape Core
Read external data
Parse discrete values
Parse tabular data sets
Issue multiple requests, merging data sets
Transform formats
Query language

A natural question might be "What are the legal implications of pulling data from web sites – isn't that *screen scraping*?" xSkrape can be used to effectively "screen scrape" – however, it can do much more, such as retrieving data from published API's. We address some of these legal issues on our Licensing and Legal page. xSkrape is "robots.txt aware" and offers warnings (or errors) if denials might be implied here which means you can feel more confident about your source data, although as noted on "Licensing and Legal" (and in the xSkrape license agreement), you are responsible for understanding the terms of use for all of your possible data sources.

Benefits Summary

We've already alluded to some of xSkrape's benefits, but this list summarizes them – although it isn't necessarily exhaustive!

1. **Simple functions & data connectors.** These can be embedded into larger solutions, building live feeds from different sources with minimal effort. The fact we're only offering a handful of functions is a testament to their power and flexibility as demonstrated in our examples.

2. **Page Explorer.** Closely related to "simple functions" is "simplified path from start to finish," where *Page Explorer* is a large contributor to that path. Rather than having to construct table and value queries yourself, *Page Explorer* can make suggestions for these queries based on example requests, taking any potential "mystery" out of the process. *Page Explorer* is available with the Excel Desktop Add-in, and in the SQL-Hero package (Developer Edition).

3. **Built-in intelligence.** "Just Point" could easily be xSkrape's tag-line: simply pointing at a data source with minimal additional configuration yields *results*. Extra configuration adds even more value to the intelligence of the parsing engine: aggregation, filtering and sorting capabilities that can deliver exactly what you want.

4. **Quicker development, lower costs.** The above points all lead to less effort required by xSkrape users compared to many other approaches which try to achieve similar results. For example, consider an example we've seen: a business analyst was maintaining an Excel spreadsheet in Sharepoint. With xSkrape we could easily include the spreadsheet data in some larger SQL queries; as the analyst updated details, it was available in real-time. An alternative would have been to build an import process to "land" the data in a table (run on a schedule or on-demand) and use it from the populated table – but we eliminated that step entirely (and also eliminated any work to configure an import task, etc.).

5. **Solves real-world problems.** Virtually all aspects of xSkrape evolved by trying to solve real problems for real customers. By buying xSkrape instead of building your own solutions, you gain the benefit of *our* experiences in building a general data access tool (and broader framework) that solves common problems. Those experiences include addressing tricky issues with data parsing, correctness and performance. And if you encounter a specific scenario that isn't working for you – let us know! We're interested in addressing *every* possible use case, and your feedback does make a difference.

6. **Automation.** xSkrape can automate tasks that today require you to perform extra steps (and you may just take it for granted that this is the only way to get an end result).

7. **Resilience to change.** We have examples that show how xSkrape can be implemented in a way that's less sensitive to changes in your source data *structure* compared to other types of solution where similar changes might lead to rework.

8. **Consistent API.** If you start using the Excel Add-in and later decide to build an application using SQL Server that uses the same data – you can do so very easily since the functions exposed by xSkrape are very similar between product "flavors."

9. **Performance.** In addition to the parsing engine being written with multi-threading in mind, there are optimizations available such as *caching* that can be tweaked through configuration settings. We've got concrete examples of performance *exceeding* Excel's native web data source by an order of magnitude. The desktop add-in takes advantage of your local compute resources for performance in the current version of xSkrape. The web-based version uses *our* compute resources which we plan to *scale up* as demand dictates.

10. **Roadmap and support.** Expect new features and iterative improvements in up-coming releases. When you purchase licenses for "local" xSkrape, you're getting the current version plus one additional major version upgrade in the price. If you never need to upgrade further: no need to pay on-going subscription costs for your local installations. The web-based version always has the latest version, at a very reasonable cost.

11. **Price.** In any buy-vs-build decision, ask if you could build similar functionality for less. We feel strongly that it's unlikely given the maturity of code in xSkrape. Another consideration is we do offer a cloud-based alternative that's "pay as you go/need" (i.e. low commitment) – or you can commit to run the components on your resources for a one-time cost (plus optional future upgrades).

Feature Summary

We've indirectly covered a number of xSkrape features already. This list represents a more detailed list specifically for xSkrape for Excel and SQL Server, for reference:

- Read data from http and https sources (HTTP GET), and for desktop versions, file sources (local or UNC)
- Access live data, with optional in-memory caching to improve performance
- Interpret tabular formats from Excel, CSV, TSV sources; also from HTML sources, where possible
- Retrieve discrete values using regular expressions, XPath or other specialized expressions supported by xSkrape
- Apply filter expressions and sorts to isolate or aggregate data, helping to get exactly what you need in its final format

- Interpret JSON as if it were XML, meaning XPath can be used against JSON sources, and Page Explorer assists with this by rendering JSON as XML
- Identify tabular structures in HTML using built-in parsing engine, with an option to provide hints to clarify source data or preprocess it
- Merge results from HTTP and HTTPS over multiple requests where query parameters vary sequentially, or for a fixed list
- Excel: import tabular data in either a fixed size cell range, or in a dynamic range, sized based on the source data
- SQL Server: create wrapper objects that present tabular source data using the native column names and data types as found in the source
- Parsing engine looks for any published robots.txt file for hosts being referenced: if one exists, it is honored for explicit denials
- Configurable throttling settings mean that you won't necessarily overwhelm sources with requests (although this is something to still be mindful of)
- Page Explorer: identify potential table and value matches in source data along with suggestions for how to "pick" them
- Web-based add-in includes a messaging connector that can send email or SMS messages
- Web REST API is publicly available to cover all functionality that's available in the web-based add-in
- SQL-Hero licensed product (Developer or Advanced Edition) includes a free license of "xSkrape for Excel Add-in" (needs to be installed on a machine where SQL-Hero is installed)
- SQL-Hero Enterprise Edition allows unlimited deployment of xSkrape (both Excel Desktop Add-in and for SQL Server) within your organization

Our "URL Expression" syntax, also dubbed "XS Query Language" (XS.QL) supports the following:

- Issue multiple requests with URL's that contain text that can be changed over the multiple requests. This supports merging paged results, and issuing requests using a parameter list.
- Issue multiple requests that access different files (local or cloud-based), merging results. (Available when running xSkrape code locally.)
- Optionally set one or more header values on requests.
- Transform JSON and/or XML into a tabular format. Use XPath expressions to build simple to complex transformations. Identify columns explicitly or through metadata in the source. Use *join sets* to flatten hierarchical data.

XS.QL has become an important part of xSkrape for several reasons:

- It offers a path for future enhancements to be added without breaking *any* existing code you may write. This is because we don't need to change the *signatures* of function or data connectors in place today (i.e. add or change parameters passed in). The slots used for the URL or data queries simply allow for more and more options.
- The query language can change the nature of what's being retrieved *and* how the retrieved data is parsed. XS.QL is really applicable to multiple parameter types, and its keywords and syntax change depending on the parameter.
- Use cases drive the evolution of XS.QL, so please keep challenging us with your requirements!

Background & xSkrape Shared Concepts

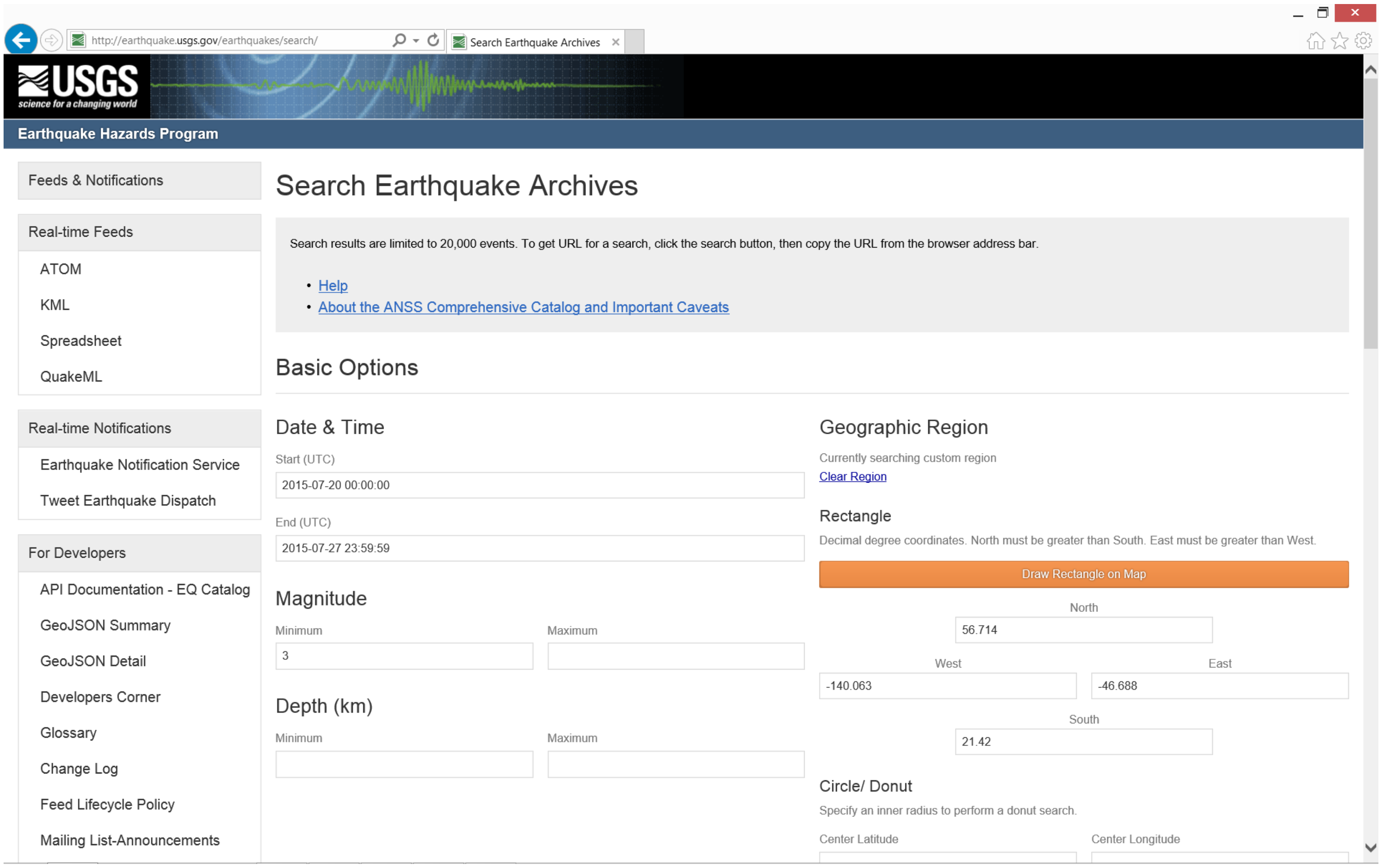
Given the wide range of possible data sources that xSkrape supports, we need to develop an understanding of:

- HTTP requests and query strings
- Web-based API's
- HTML concepts to help isolate data of interest

HTTP requests are something you make every time you use a browser. A URL like http://finance.yahoo.com is something that the finance.yahoo.com server recognizes, processes, and returns HTML content for. When we see a more complex URL such as http://finance.yahoo.com/q/ae?s=GS+Analyst+Estimates it's still serviced by the finance.yahoo.com server, but the content is controlled by the page being requested: *q/ae*, plus parameters that can be passed to the page: *s=GS+Analyst+Estimates*. In this case, the *s=GS+Analyst+Estimates* is also known as "query parameters." Using the web site, you may not even notice the changes in your URL as you typed in a stock symbol at the top of the page, but that's what happened when we entered "GS" as a stock symbol and picked a menu option for "Analyst Estimates."

Understanding how the web site of interest behaves is central to making xSkrape work for you. In the above example, we discovered through observation that a URL formatted as http://finance.yahoo.com/q/ae?s=GS+Analyst+Estimates would give us what we want (analyst estimates, for example) for a specific stock symbol: GS. It's not a leap to assume if we instead used http://finance.yahoo.com/q/ae?s=MSFT+Analyst+Estimates, we'd get different results – this time analyst estimates for MSFT instead of GS.

Let's look at another example. Visiting the USGS web site, we can search for a list of earthquakes that includes an assortment of metrics. The output can be delivered directly from the web site in different formats, including comma-separated values. The web application that can be used to do this download is shown here:


http://earthquake.usgs.gov/earthquakes/search/
Search Earthquake Archives
USGS
science for a changing world
Earthquake Hazards Program
Feeds & Notifications
Real-time Feeds
ATOM
KML
Spreadsheet
QuakeML
Real-time Notifications
Earthquake Notification Service
Tweet Earthquake Dispatch
For Developers
API Documentation - EQ Catalog
GeoJSON Summary
GeoJSON Detail
Developers Corner
Glossary
Change Log
Feed Lifecycle Policy
Mailing List-Announcements
Help
About the ANSS Comprehensive Catalog and Important Caveats
Clear Region
Draw Rectangle on Map

Search Earthquake Archives

Search results are limited to 20,000 events. To get URL for a search, click the search button, then copy the URL from the browser address bar.

Basic Options

Date & Time

Start (UTC): 2015-07-20 00:00:00

End (UTC): 2015-07-27 23:59:59

Magnitude

Minimum: 3

Maximum:

Depth (km)

Minimum:

Maximum:

Geographic Region

Currently searching custom region

Rectangle

Decimal degree coordinates. North must be greater than South. East must be greater than West.

North: 56.714

West: -140.063

East: -46.688

South: 21.42

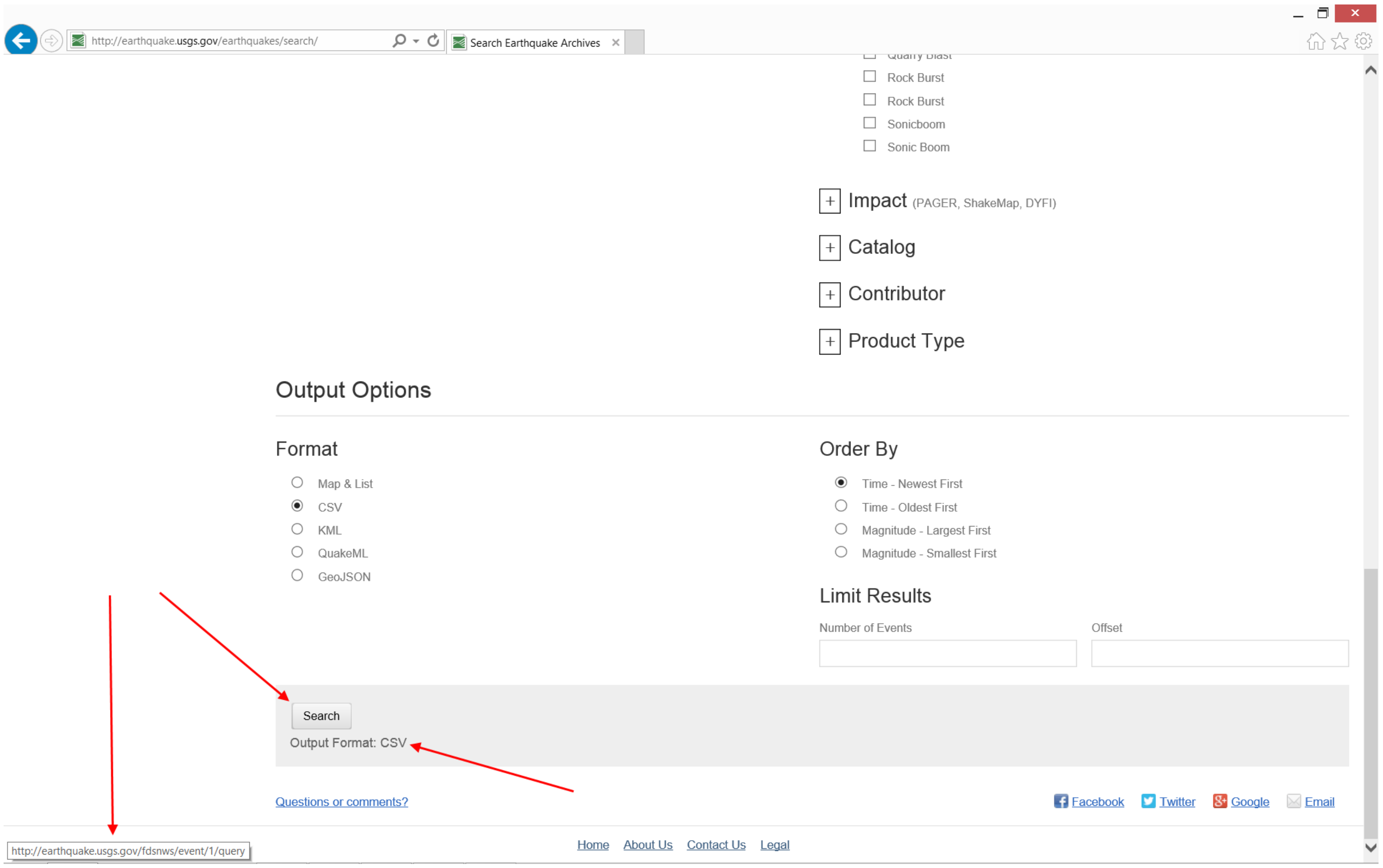
Circle/ Donut

Specify an inner radius to perform a donut search.

Center Latitude

Center Longitude

… Scrolling to the bottom of the page:



http://earthquake.usgs.gov/earthquakes/search/

Search Earthquake Archives

Quarry Blast

☐ Rock Burst

☐ Rock Burst

☐ Sonicboom

☐ Sonic Boom

+ Impact (PAGER, ShakeMap, DYFI)

+ Catalog

+ Contributor

+ Product Type

Output Options

Format

○ Map & List

● CSV

○ KML

○ QuakeML

○ GeoJSON

Order By

● Time - Newest First

○ Time - Oldest First

○ Magnitude - Largest First

○ Magnitude - Smallest First

Limit Results

Number of Events

Offset

Search

Output Format: CSV

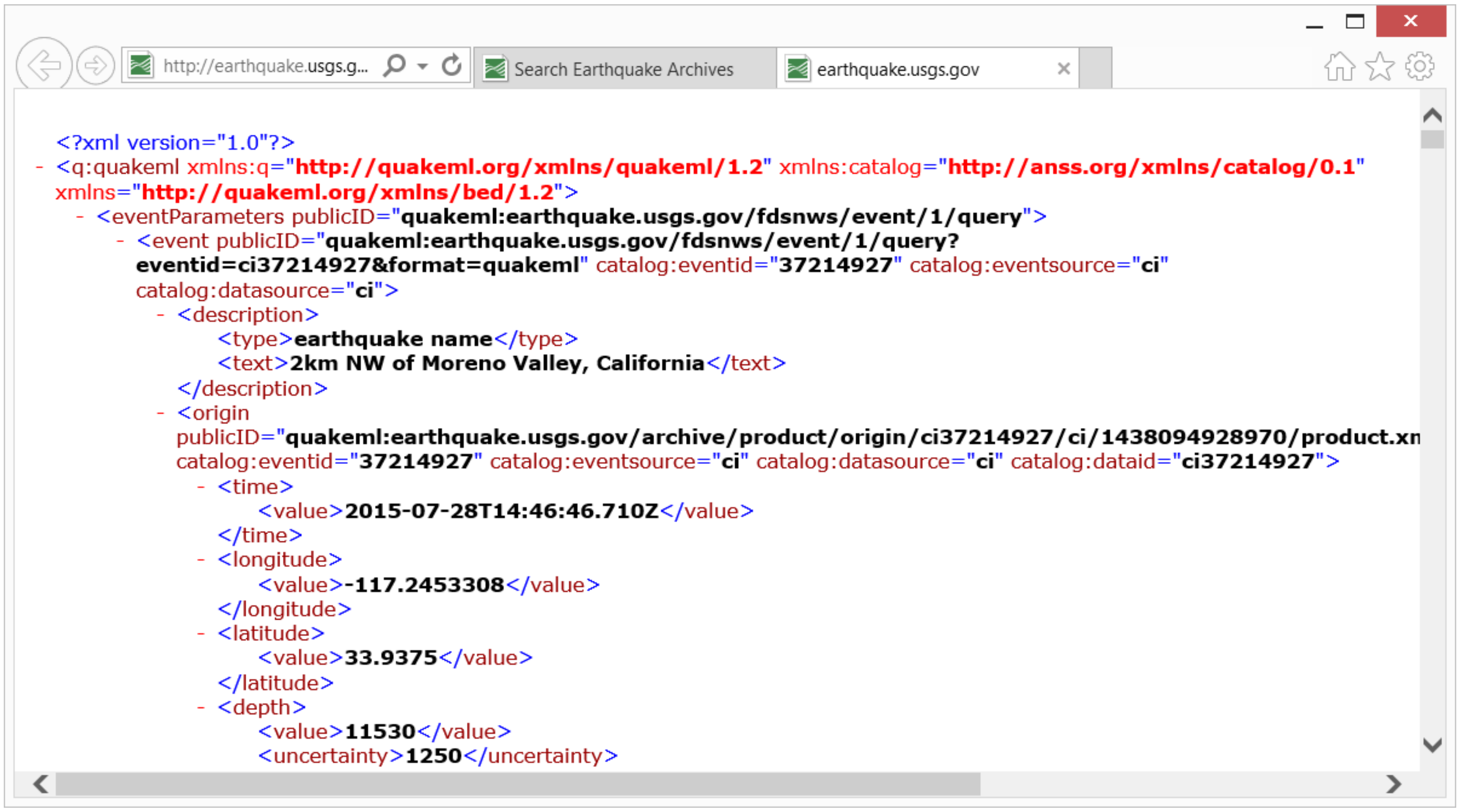
Questions or comments?

Facebook Twitter Google Email

Home About Us Contact Us Legal

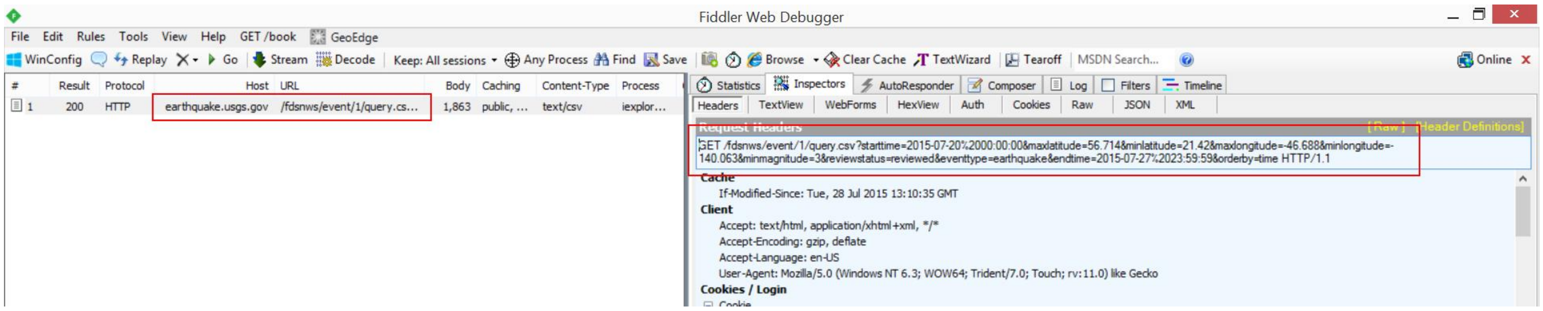
http://earthquake.usgs.gov/fdsnws/event/1/query

Notice if we hover over the "Search" button, a URL is shown that is directly connected to the search operation: http://earthquake.usgs.gov/fdsnws/event/1/query. Even though when we click on the "Search" button we do get results from the application, we can't simply copy this URL into a browser window (or xSkrape Page Explorer) and get the same results. In fact, we get XML results that are not filtered at all:



```
  <?xml version="1.0"?>
- <q:quakeml xmlns:q="http://quakeml.org/xmlns/quakeml/1.2" xmlns:catalog="http://anss.org/xmlns/catalog/0.1"
  xmlns="http://quakeml.org/xmlns/bed/1.2">
  - <eventParameters publicID="quakeml:earthquake.usgs.gov/fdsnws/event/1/query">
    - <event publicID="quakeml:earthquake.usgs.gov/fdsnws/event/1/query?
      eventid=ci37214927&format=quakeml" catalog:eventid="37214927" catalog:eventsource="ci"
      catalog:datasource="ci">
      - <description>
          <type>earthquake name</type>
          <text>2km NW of Moreno Valley, California</text>
        </description>
      - <origin
        publicID="quakeml:earthquake.usgs.gov/archive/product/origin/ci37214927/ci/1438094928970/product.xn
        catalog:eventid="37214927" catalog:eventsource="ci" catalog:datasource="ci" catalog:dataid="ci37214927">
        - <time>
            <value>2015-07-28T14:46:46.710Z</value>
          </time>
        - <longitude>
            <value>-117.2453308</value>
          </longitude>
        - <latitude>
            <value>33.9375</value>
          </latitude>
        - <depth>
            <value>11530</value>
            <uncertainty>1250</uncertainty>
```

Why? The web application collects a lot of search criteria that the URL alone as seen above does *not* convey. It's reasonable to expect here that there are query parameters involved that we don't *see*: and normally we don't *need to* since the web application is doing the work of collecting search criteria, issuing the search itself, and returning the results as CSV (if that's our choice of delivery type). Are we out of luck? Not necessarily. Sites like this may publish their *API* (Application Programming Interface) to make it clear how you can issue your own queries. Another alternative is to use a tool like Fiddler to capture the back-and-forth traffic from the web site, including the actual query parameters used. Here's an example of what we see in Fiddler 4 when we've clicked the "Search" button on the site:



#	Result	Protocol	Host	URL	Body	Caching	Content-Type	Process
1	200	HTTP	earthquake.usgs.gov	/fdsnws/event/1/query.cs...	1,863	public, ...	text/csv	iexplor...

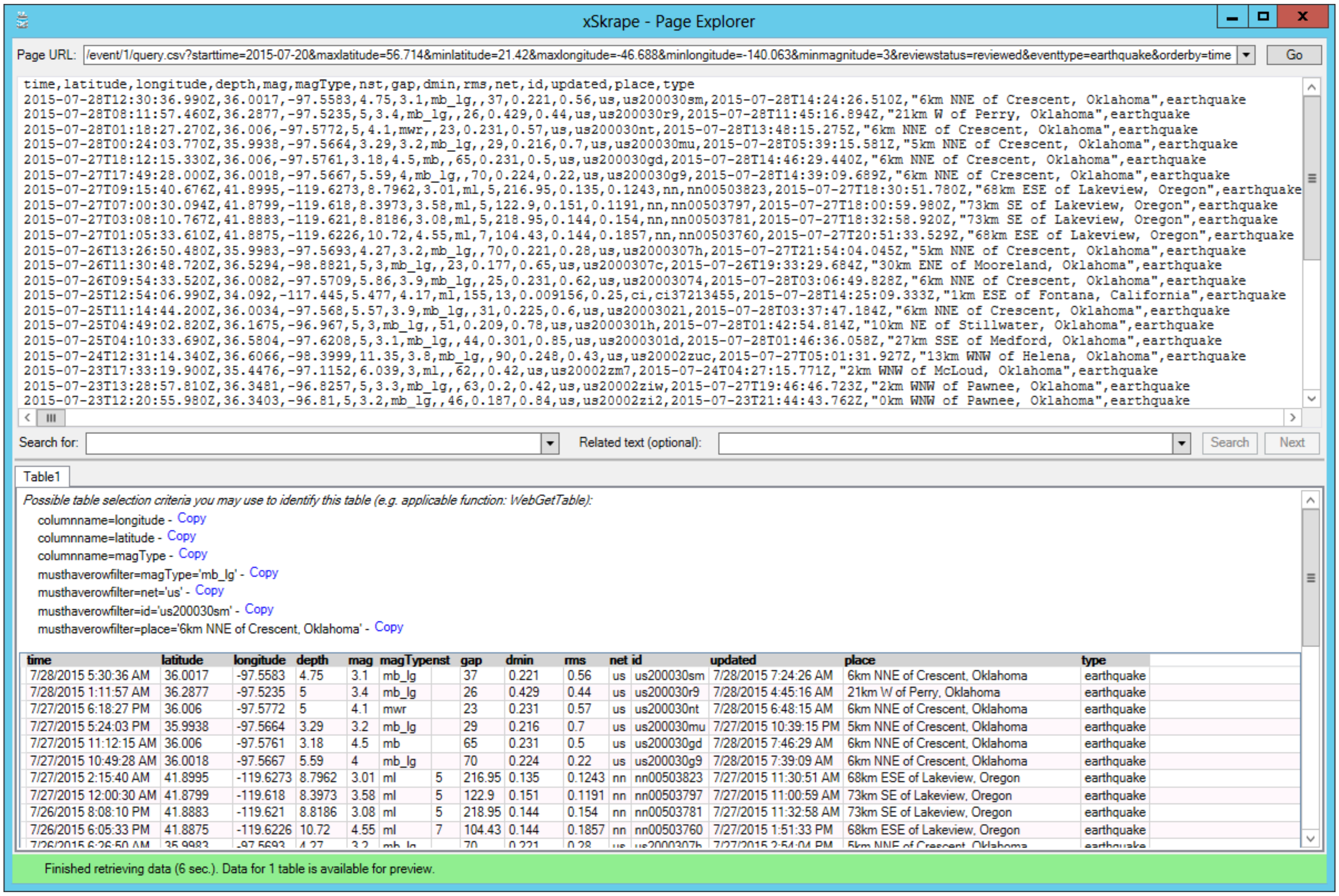
```
GET /fdsnws/event/1/query.csv?starttime=2015-07-20%2000:00:00&maxlatitude=56.714&minlatitude=21.42&maxlongitude=-46.688&minlongitude=-
140.063&minmagnitude=3&reviewstatus=reviewed&eventtype=earthquake&endtime=2015-07-27%2023:59:59&orderby=time HTTP/1.1
Cache
    If-Modified-Since: Tue, 28 Jul 2015 13:10:35 GMT
Client
    Accept: text/html, application/xhtml+xml, */*
    Accept-Encoding: gzip, deflate
    Accept-Language: en-US
    User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko
Cookies / Login
    Cookie
```

Notice we *do* see query parameters, starting with the "*?starttime=…*" text. The captured full URL serves as a great starting point to understand the underlying API, even if it's not documented. As you can see, query parameters are composed of a *name=value* format, and multiple parameters are delimited with the "&" character. What if you needed to include the "=" or "&" characters in the *value* of a parameter? This is where URL encoding can be used.

To complete our example of using this API, we might infer the parameters we need to retrieve all "3 magnitude and greater earthquakes that have been reviewed since 7/20/15, in a specific geography" as:

http://earthquake.usgs.gov/fdsnws/event/1/query.csv?starttime=2015-07-20&maxlatitude=56.714&minlatitude=21.42&maxlongitude=-46.688&minlongitude=-140.063&minmagnitude=3&reviewstatus=reviewed&eventtype=earthquake&orderby=time

If you click on the above link, chances are good Excel will open on your local computer (assuming it's installed), and you'll see the raw tabular data from the search. We can see the same thing using *Page Explorer*:



Page URL: /event/1/query.csv?starttime=2015-07-20&maxlatitude=56.714&minlatitude=21.42&maxlongitude=-46.688&minlongitude=-140.063&minmagnitude=3&reviewstatus=reviewed&eventtype=earthquake&orderby=time

```
time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated,place,type
2015-07-28T12:30:36.990Z,36.0017,-97.5583,4.75,3.1,mb_lg,,37,0.221,0.56,us,us200030sm,2015-07-28T14:24:26.510Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-28T08:11:57.460Z,36.2877,-97.5235,5,3.4,mb_lg,,26,0.429,0.44,us,us200030r9,2015-07-28T11:45:16.894Z,"21km W of Perry, Oklahoma",earthquake
2015-07-28T01:18:27.270Z,36.006,-97.5772,5,4.1,mwr,,23,0.231,0.57,us,us200030nt,2015-07-28T13:48:15.275Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-28T00:24:03.770Z,35.9938,-97.5664,3.29,3.2,mb_lg,,29,0.216,0.7,us,us200030mu,2015-07-28T05:39:15.581Z,"5km NNE of Crescent, Oklahoma",earthquake
2015-07-27T18:12:15.330Z,36.006,-97.5761,3.18,4.5,mb,,65,0.231,0.5,us,us200030gd,2015-07-28T14:46:29.440Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-27T17:49:28.000Z,36.0018,-97.5667,5.59,4,mb_lg,,70,0.224,0.22,us,us200030g9,2015-07-28T14:39:09.689Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-27T09:15:40.676Z,41.8995,-119.6273,8.7962,3.01,ml,5,216.95,0.135,0.1243,nn,nn00503823,2015-07-27T18:30:51.780Z,"68km ESE of Lakeview, Oregon",earthquake
2015-07-27T07:00:30.094Z,41.8799,-119.618,8.3973,3.58,ml,5,122.9,0.151,0.1191,nn,nn00503797,2015-07-27T18:00:59.980Z,"73km SE of Lakeview, Oregon",earthquake
2015-07-27T03:08:10.767Z,41.8883,-119.621,8.8186,3.08,ml,5,218.95,0.144,0.154,nn,nn00503781,2015-07-27T18:32:58.920Z,"73km SE of Lakeview, Oregon",earthquake
2015-07-27T01:05:33.610Z,41.8875,-119.6226,10.72,4.55,ml,7,104.43,0.144,0.1857,nn,nn00503760,2015-07-27T20:51:33.529Z,"68km ESE of Lakeview, Oregon",earthquake
2015-07-26T13:26:50.480Z,35.9983,-97.5693,4.27,3.2,mb_lg,,70,0.221,0.28,us,us2000307h,2015-07-27T21:54:04.045Z,"5km NNE of Crescent, Oklahoma",earthquake
2015-07-26T11:30:48.720Z,36.5294,-98.8821,5,3,mb_lg,,23,0.177,0.65,us,us2000307c,2015-07-26T19:33:29.684Z,"30km ENE of Mooreland, Oklahoma",earthquake
2015-07-26T09:54:33.520Z,36.0082,-97.5709,5.86,3.9,mb_lg,,25,0.231,0.62,us,us20003074,2015-07-28T03:06:49.828Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-25T12:54:06.990Z,34.092,-117.445,5.477,4.17,ml,155,13,0.009156,0.25,ci,ci37213455,2015-07-28T14:25:09.333Z,"1km ESE of Fontana, California",earthquake
2015-07-25T11:14:44.200Z,36.0034,-97.568,5.57,3.9,mb_lg,,31,0.225,0.6,us,us20003021,2015-07-28T03:37:47.184Z,"6km NNE of Crescent, Oklahoma",earthquake
2015-07-25T04:49:02.820Z,36.1675,-96.967,5,3,mb_lg,,51,0.209,0.78,us,us200030lh,2015-07-28T01:42:54.814Z,"10km NE of Stillwater, Oklahoma",earthquake
2015-07-25T04:10:33.690Z,36.5804,-97.6208,5,3.1,mb_lg,,44,0.301,0.85,us,us200030ld,2015-07-28T01:46:36.058Z,"27km SSE of Medford, Oklahoma",earthquake
2015-07-24T12:31:14.340Z,36.6066,-98.3999,11.35,3.8,mb_lg,,90,0.248,0.43,us,us20002zuc,2015-07-27T05:01:31.927Z,"13km WNW of Helena, Oklahoma",earthquake
2015-07-23T17:33:19.900Z,35.4476,-97.1152,6.039,3,ml,,62,,0.42,us,us20002zm7,2015-07-24T04:27:15.771Z,"2km WNW of McLoud, Oklahoma",earthquake
2015-07-23T13:28:57.810Z,36.3481,-96.8257,5,3.3,mb_lg,,63,0.2,0.42,us,us20002ziw,2015-07-27T19:46:46.723Z,"2km WNW of Pawnee, Oklahoma",earthquake
2015-07-23T12:20:55.980Z,36.3403,-96.81,5,3.2,mb_lg,,46,0.187,0.84,us,us20002zi2,2015-07-23T21:44:43.762Z,"0km WNW of Pawnee, Oklahoma",earthquake
```

Possible table selection criteria you may use to identify this table (e.g. applicable function: WebGetTable):

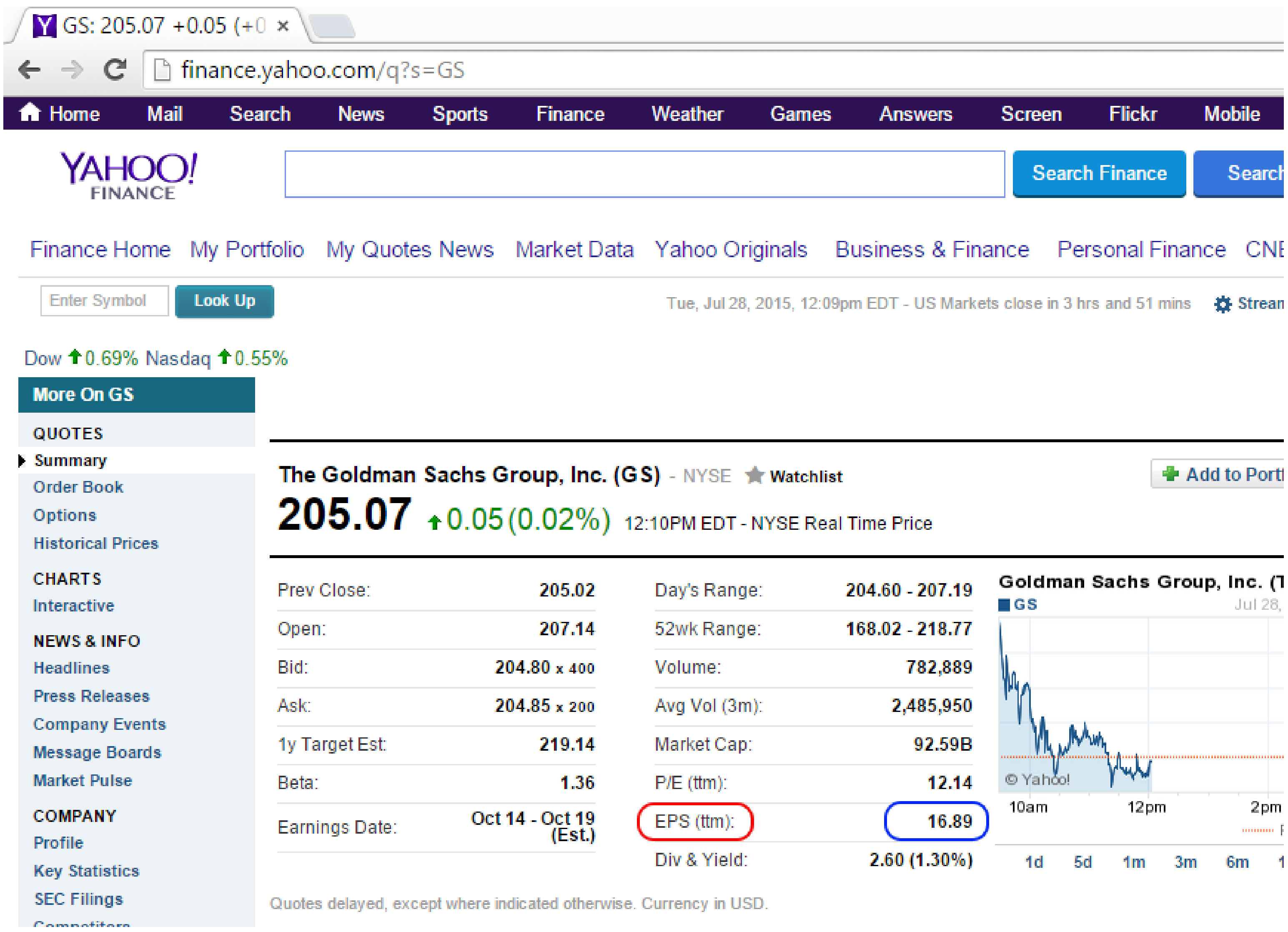
- columnname=longitude - Copy
- columnname=latitude - Copy
- columnname=magType - Copy
- musthaverowfilter=magType='mb_lg' - Copy
- musthaverowfilter=net='us' - Copy
- musthaverowfilter=id='us200030sm' - Copy
- musthaverowfilter=place='6km NNE of Crescent, Oklahoma' - Copy

time	latitude	longitude	depth	mag	magType	nst	gap	dmin	rms	net	id	updated	place	type
7/28/2015 5:30:36 AM	36.0017	-97.5583	4.75	3.1	mb_lg		37	0.221	0.56	us	us200030sm	7/28/2015 7:24:26 AM	6km NNE of Crescent, Oklahoma	earthquake
7/28/2015 1:11:57 AM	36.2877	-97.5235	5	3.4	mb_lg		26	0.429	0.44	us	us200030r9	7/28/2015 4:45:16 AM	21km W of Perry, Oklahoma	earthquake
7/27/2015 6:18:27 PM	36.006	-97.5772	5	4.1	mwr		23	0.231	0.57	us	us200030nt	7/28/2015 6:48:15 AM	6km NNE of Crescent, Oklahoma	earthquake
7/27/2015 5:24:03 PM	35.9938	-97.5664	3.29	3.2	mb_lg		29	0.216	0.7	us	us200030mu	7/27/2015 10:39:15 PM	5km NNE of Crescent, Oklahoma	earthquake
7/27/2015 11:12:15 AM	36.006	-97.5761	3.18	4.5	mb		65	0.231	0.5	us	us200030gd	7/28/2015 7:46:29 AM	6km NNE of Crescent, Oklahoma	earthquake
7/27/2015 10:49:28 AM	36.0018	-97.5667	5.59	4	mb_lg		70	0.224	0.22	us	us200030g9	7/28/2015 7:39:09 AM	6km NNE of Crescent, Oklahoma	earthquake
7/27/2015 2:15:40 AM	41.8995	-119.6273	8.7962	3.01	ml	5	216.95	0.135	0.1243	nn	nn00503823	7/27/2015 11:30:51 AM	68km ESE of Lakeview, Oregon	earthquake
7/27/2015 12:00:30 AM	41.8799	-119.618	8.3973	3.58	ml	5	122.9	0.151	0.1191	nn	nn00503797	7/27/2015 11:00:59 AM	73km SE of Lakeview, Oregon	earthquake
7/26/2015 8:08:10 PM	41.8883	-119.621	8.8186	3.08	ml	5	218.95	0.144	0.154	nn	nn00503781	7/27/2015 11:32:58 AM	73km SE of Lakeview, Oregon	earthquake
7/26/2015 6:05:33 PM	41.8875	-119.6226	10.72	4.55	ml	7	104.43	0.144	0.1857	nn	nn00503760	7/27/2015 1:51:33 PM	68km ESE of Lakeview, Oregon	earthquake
7/26/2015 6:26:50 AM	35.9983	-97.5693	4.27	3.2	mb_lg		70	0.221	0.28	us	us2000307h	7/27/2015 2:54:04 PM	5km NNE of Crescent, Oklahoma	earthquake

Finished retrieving data (6 sec.). Data for 1 table is available for preview.

A natural question might be "Should we build the same kind of functionality we see in Fiddler into xSkrape tools like *Page Explorer*?" We feel the answer is "no" *currently* since the other third-party tools available today are quite good at what they're designed for. As you can see, understanding how your source data is exposed is important and not always immediately obvious, but there are usually ways to do so, and we can even try to assist you through our feedback forum. (It's important for us to understand your real-world use cases and document them where possible.)

The final concept we'll cover here is to understand how HTML pages offer opportunities to extract discrete data values. The extraction of tabular data is out-of-scope largely because it's *easy* with xSkrape: it finds candidate tables, largely based on HTML tables defined with the HTML <table> element. However, for discrete values, we might find these anywhere on a page, not just connected to <table> elements. That said, many times there is a regular pattern to how values are presented on pages, and often it *does* equate to use of <table>, or nested <span> and/or <div> elements. For a concrete example, we'll use the http://finance.yahoo.com/q?s=GS page. (Please note: this page is protected by terms of use that stipulate you cannot disseminate the data retrieved from it for commercial purposes.) This page presents some metrics that are identified by a *label* and a *value* (a specific metric's label is circled in red, value in blue):



The Goldman Sachs Group, Inc. (GS) - NYSE

205.07 0.05 (0.02%) 12:10PM EDT - NYSE Real Time Price

Prev Close:	205.02	Day's Range:	204.60 - 207.19
Open:	207.14	52wk Range:	168.02 - 218.77
Bid:	204.80 x 400	Volume:	782,889
Ask:	204.85 x 200	Avg Vol (3m):	2,485,950
1y Target Est:	219.14	Market Cap:	92.59B
Beta:	1.36	P/E (ttm):	12.14
Earnings Date:	Oct 14 - Oct 19 (Est.)	EPS (ttm):	16.89
		Div & Yield:	2.60 (1.30%)

Quotes delayed, except where indicated otherwise. Currency in USD.

If we look at the underlying page structure here, we can see that both label and value are actually contained in a <table> structure, with <th> and <td> being containing elements:

```
            ----
                <th scope="row" width="48%">Market Cap:</th>
                <td class="yfnc_tabledata1">
                    <span id="yfs_j10_gs">92.59B</span>
                </td>
            </tr>
            <tr>
                <th scope="row" width="48%">P/E <span class="small">(ttm)</span>:</th>
                <td class="yfnc_tabledata1">12.14</td>
            </tr>
            <tr>
                <th scope="row" width="48%">EPS <span class="small">(ttm)</span>:</th>
                <td class="yfnc_tabledata1">16.89</td>
            </tr>
            <tr class="end">
                <th scope="row" width="48%">Div &amp;amp; Yield:</th>
                <td class="yfnc_tabledata1">2.60 (1.30%) </td>
            </tr>
        </table>
    </div>
```

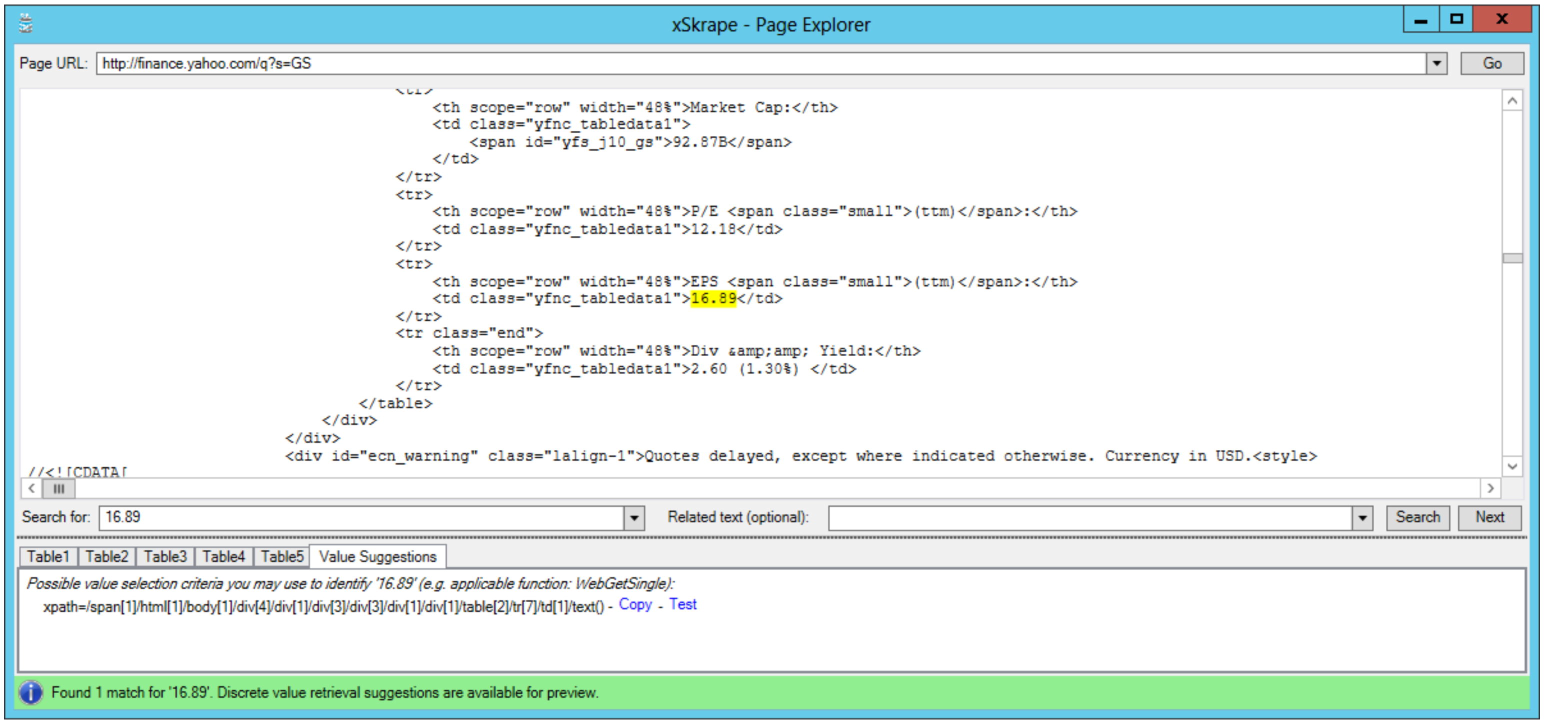
In this case, using xSkrape's *followinginnertext* based on the text "EPS" alone isn't going to work to return "16.89": the nested <span> element contains inner text "(ttm)" which is technically what *followinginnertext* would find. A way to solve this could be to expand the scope of what *followinginnertext* finds, based on the fact its parameter value is actually a regular expression. We could therefore use a value expression like this in a call to *WebGetSingle* (in this case, using Excel syntax):

=WebGetSingle("http://finance.yahoo.com/q?s=GS", "followinginnertext=EPS.+?ttm.+?:")

In this example we're finding the following text match (text matching the ".+?" is shown in green): *EPS <span class="small">(ttm)</span>:*

After that's found, we look for the following inner text, which happens to be the "16.89" which is the inner text of the <td> element that follows our matched text. *Page Explorer's* purpose is to help you identify these types of expressions to extract values of interest – but it's important to understand the underlying query options available. *Page Explorer* isn't all-seeing-all-knowing (yet!) and as such, its suggestions are useful but do not address every possible case. Page structures *do* vary from site-to-site (and often page-to-page) and so getting an understanding of what's available is important, and *Page Explorer* is one way to do that. In cases, even using a browser's "show source" functionality is a valid approach too.

A second possible approach to extract the "16.89" value would be to use an explicit XPath expression. It's not easy to determine what this should be, given the complexity of most pages. *Page Explorer* helps greatly with this, as shown here:



```
                <th scope="row" width="48%">Market Cap:</th>
                <td class="yfnc_tabledata1">
                    <span id="yfs_j10_gs">92.87B</span>
                </td>
            </tr>
            <tr>
                <th scope="row" width="48%">P/E <span class="small">(ttm)</span>:</th>
                <td class="yfnc_tabledata1">12.18</td>
            </tr>
            <tr>
                <th scope="row" width="48%">EPS <span class="small">(ttm)</span>:</th>
                <td class="yfnc_tabledata1">16.89</td>
            </tr>
            <tr class="end">
                <th scope="row" width="48%">Div &amp;amp; Yield:</th>
                <td class="yfnc_tabledata1">2.60 (1.30%) </td>
            </tr>
        </table>
    </div>
</div>
<div id="ecn_warning" class="lalign-1">Quotes delayed, except where indicated otherwise. Currency in USD.<style>
//<![CDATA[
```

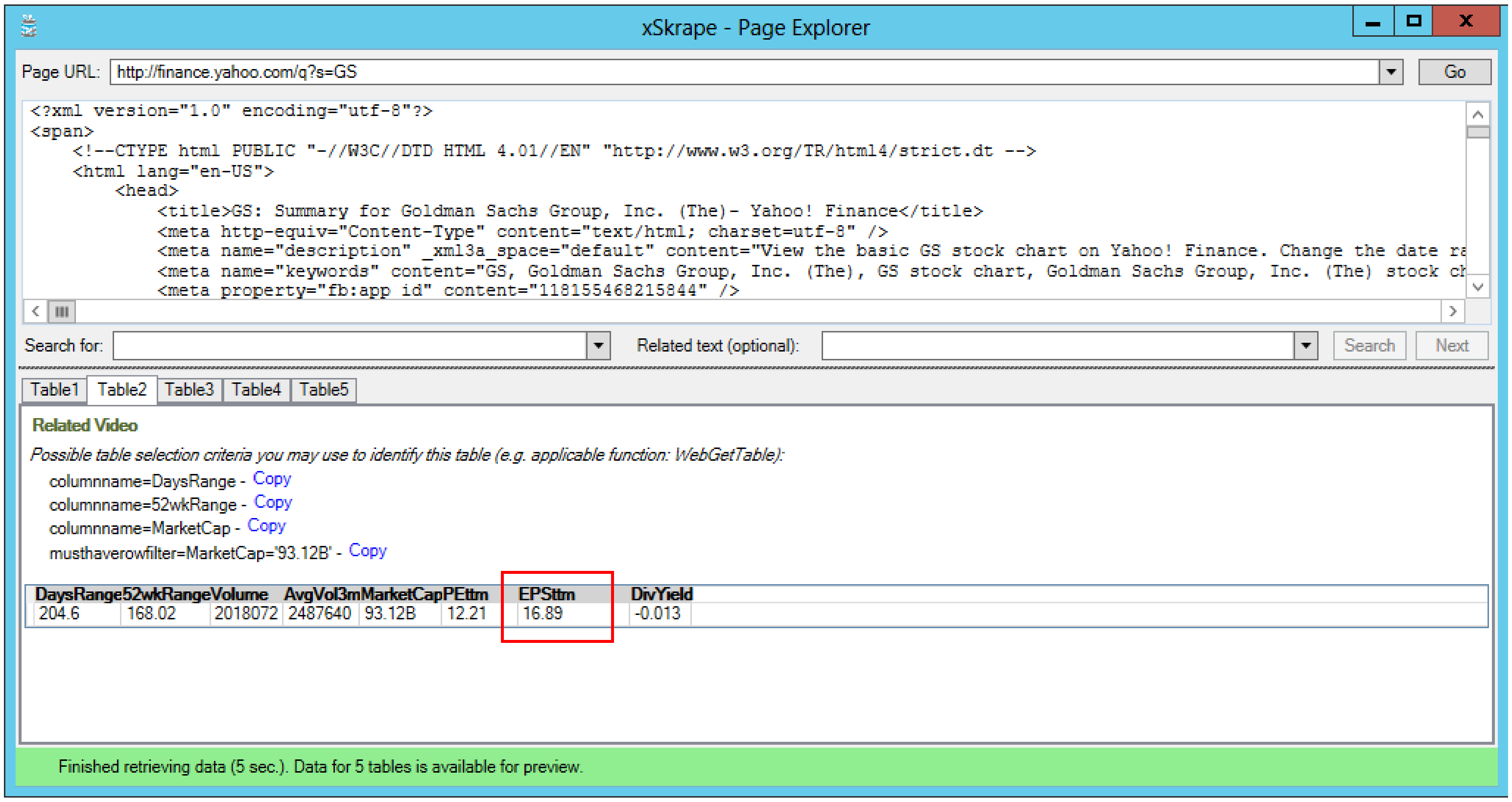
Possible value selection criteria you may use to identify '16.89' (e.g. applicable function: WebGetSingle):
xpath=/span[1]/html[1]/body[1]/div[4]/div[1]/div[3]/div[3]/div[1]/div[1]/table[2]/tr[7]/td[1]/text() - Copy - Test

Found 1 match for '16.89'. Discrete value retrieval suggestions are available for preview.

The *xpath=* suggestion shown here *will* return "16.89", but it assumes the value can be found in six levels of <div> element nesting, in a table element, and in a specific row and column. This may be acceptable if you're confident the page structure will be static, but all it would take is introducing a seventh level of <div> nesting (or removing one) to break the validity of the XPath expression. This is where the first approach of *followinginnertext* could be considered more "robust," even though that is still subject to breakage if the label text were to change for example from "EPS" to "Earnings per Share". Using *xpath=* tends to be a good choice when dealing with native XML since the inherent structure can often be considered a data *contract* that is less likely to change, whereas HTML defines a user-interface which can change for a variety of reasons.

The above example demonstrates some important features of *Page Explorer*: entering a "sample URL" (in this case we're using "GS" as the stock symbol although in our final solution, that value is likely to be *parameterized*), and a value of interest from the *sample* leads to possible suggestions. This is the reason we offer "Also Launch Browser" as an option when clicking on the "Go" button, so you can more readily *see* what the value of interest is, side-by-side with the raw text as delivered by the HTML request. The option offered to see the raw text transformed "as XML" is also useful since that's what xSkrape does internally too, where applicable. In doing so, HTML is guaranteed to be loadable into an XML DOM (document model), and therefore queryable via XPath.

It should be noted that in the above example, there's a *third* way we can get our data value of interest. Because the value is actually part of a <table>, we could use *WebGetSingleFromTable*, or *WebGetTable* with a SQL query over *that* if using SQL Server. The table we'd be pulling from is shown here in *Page Explorer*:


xSkrape - Page Explorer

Page URL: http://finance.yahoo.com/q?s=GS

```
<?xml version="1.0" encoding="utf-8"?>
<span>
    <!--CTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dt -->
    <html lang="en-US">
        <head>
            <title>GS: Summary for Goldman Sachs Group, Inc. (The)- Yahoo! Finance</title>
            <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
            <meta name="description" _xml3a_space="default" content="View the basic GS stock chart on Yahoo! Finance. Change the date ra
            <meta name="keywords" content="GS, Goldman Sachs Group, Inc. (The), GS stock chart, Goldman Sachs Group, Inc. (The) stock ch
            <meta property="fb:app id" content="118155468215844" />
```

Search for: Related text (optional): Search Next

Table1 Table2 Table3 Table4 Table5

Related Video

Possible table selection criteria you may use to identify this table (e.g. applicable function: WebGetTable):

- columnname=DaysRange - Copy
- columnname=52wkRange - Copy
- columnname=MarketCap - Copy
- musthaverowfilter=MarketCap='93.12B' - Copy

DaysRange	52wkRange	Volume	AvgVol3m	MarketCap	PEttm	EPSttm	DivYield
204.6	168.02	2018072	2487640	93.12B	12.21	16.89	-0.013

Finished retrieving data (5 sec.). Data for 5 tables is available for preview.

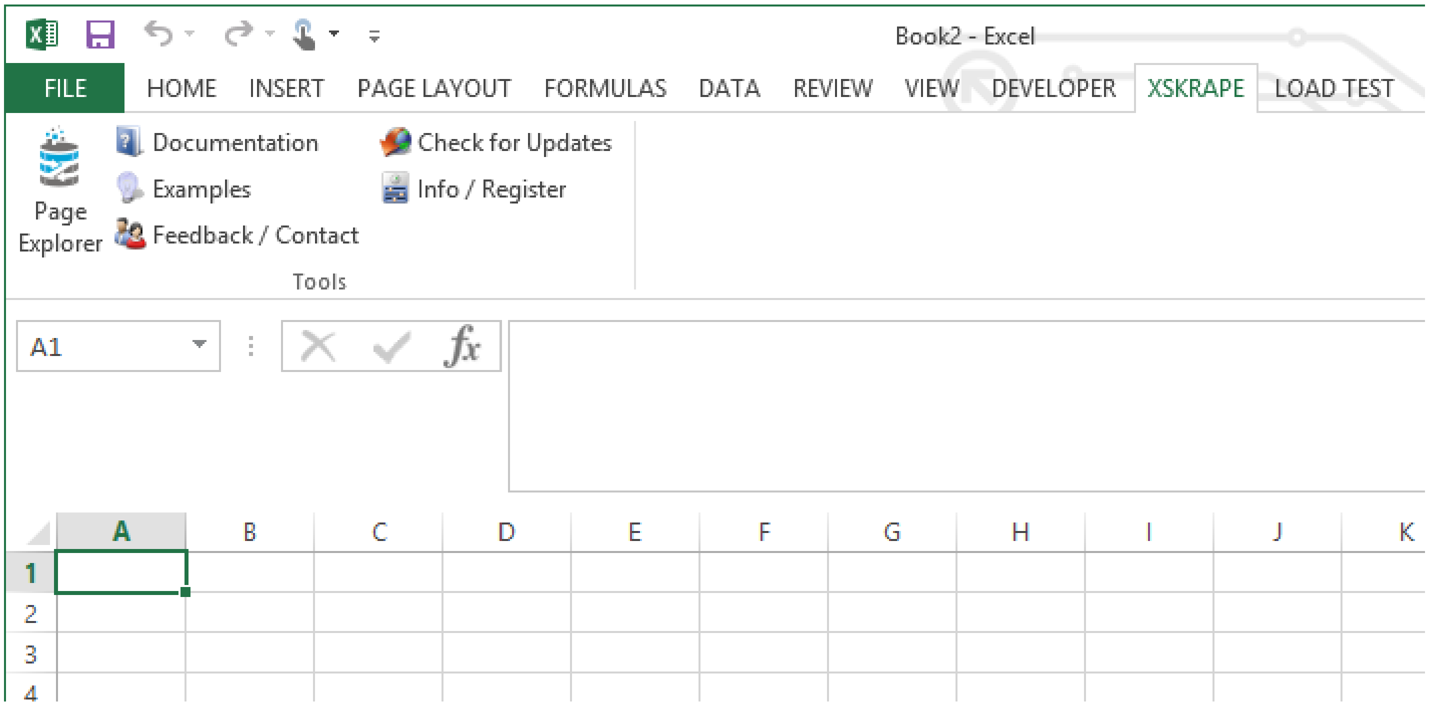
The *WebGetSingleFromTable* call would look like this, identifying both the table of interest, and the value of interest (in this case, we can safely use "*FIRST*" since it's apparent this table will only ever have one row):

=WebGetSingleFromTable("http://finance.yahoo.com/q?s=GS", "columnname=DaysRange", "FIRST(EPSttm)")

The topic of parameterization varies in implementation depending on whether you're using Excel or SQL Server but the concept is the same: you'll often be interested in changing the actual URL (or other parameters to your function calls) based on possible parameters you could pass. In our previous example of http://finance.yahoo.com/q?s=GS, substituting a different symbol for "GS" enables the pulling of values for *any* stock symbol (but for the same metric, assuming the page structure does not change for different symbols).

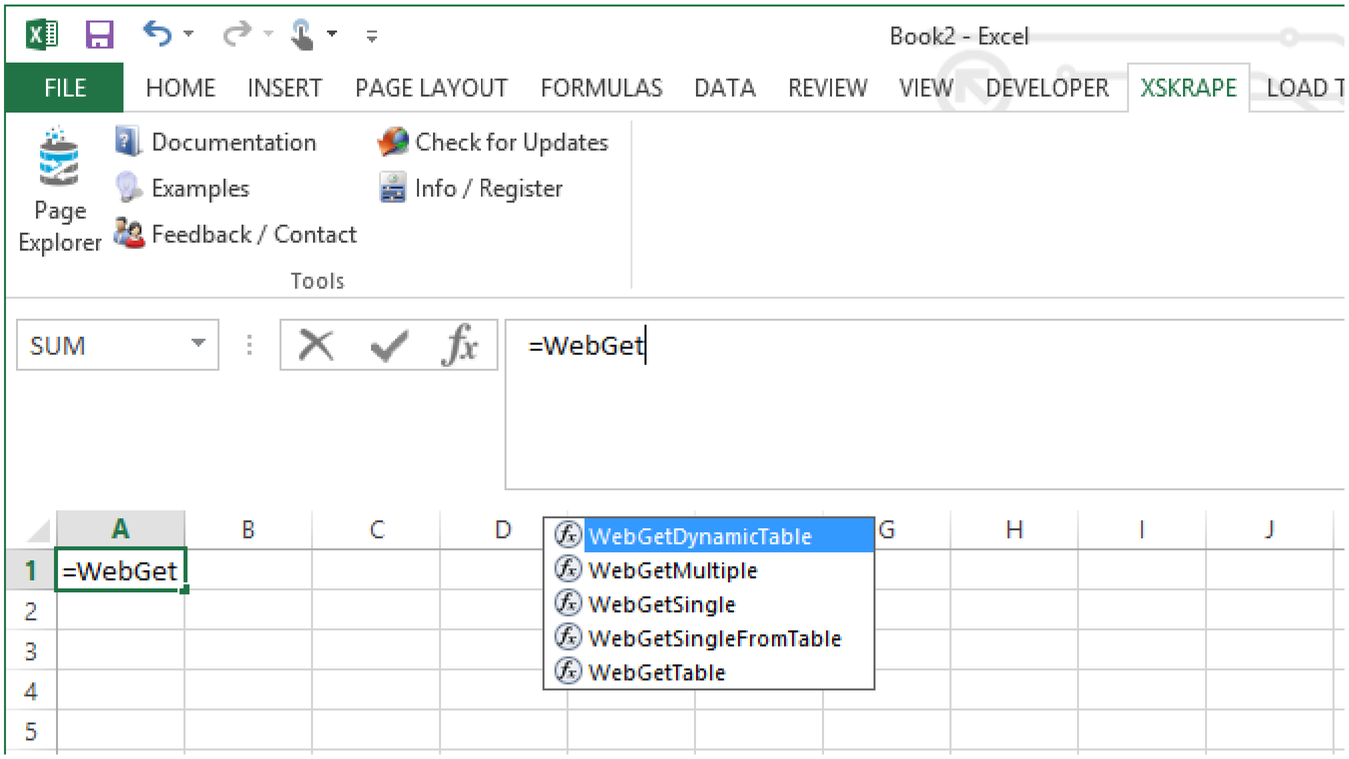
xSkrape for Excel - Desktop Add-in

The current version of xSkrape is available as an add-in for Excel 2010 – 2016. The add-in includes two main elements: new Excel functions, and *Page Explorer* as a menu option, as shown here:

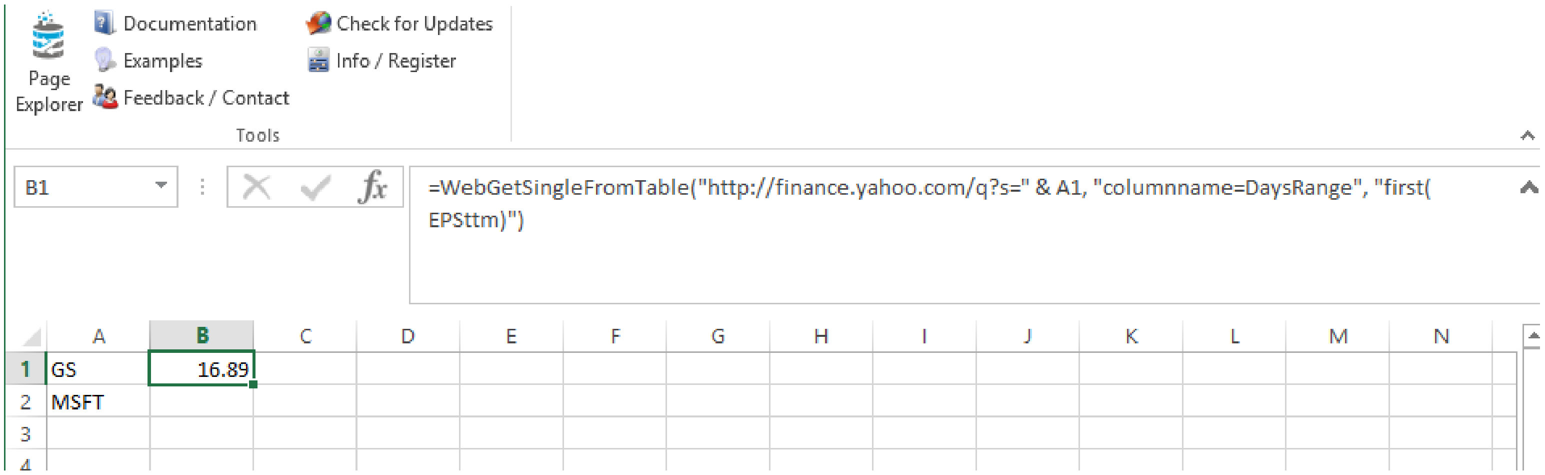


Other menu options either navigate to the xSkrape web site to provide documentation, feedback, etc. or allow you to provide a registration key if upgrading from an evaluation version for example.

xSkrape functions are documented [on-line](). You can see them appear in the auto-completion list as you type their names:



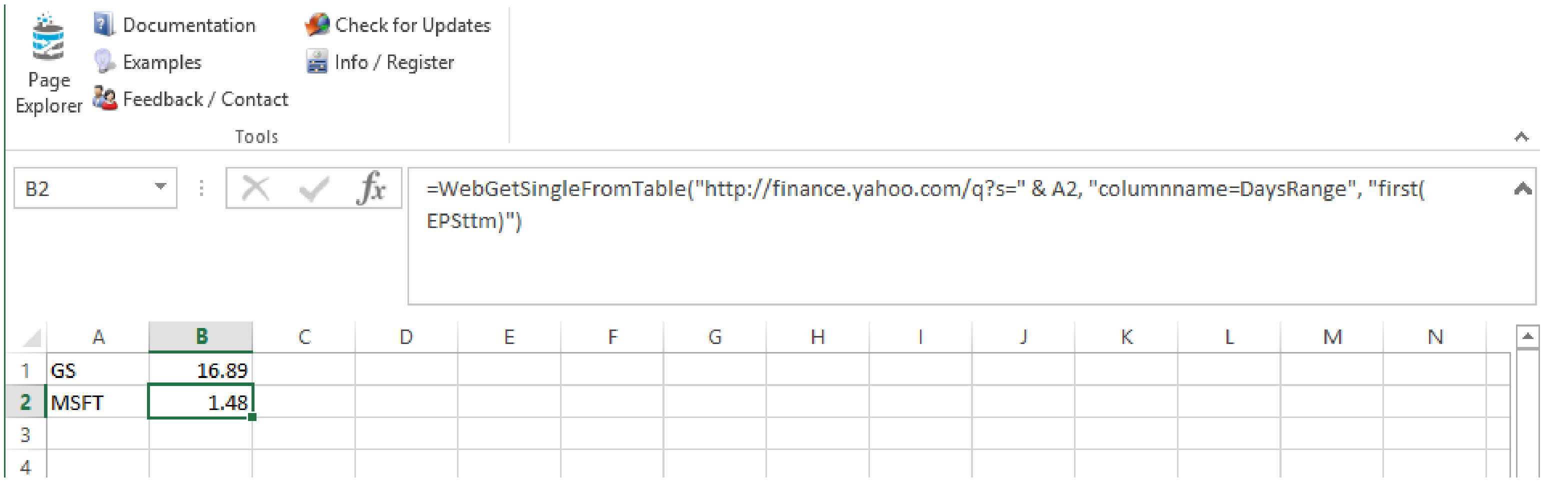
Parameterization in Excel is as simple as using cell references. For example, we could use Column A as a stock symbol, with column B representing a value retrieved from a page. In this case, we've established a cell reference to "A1":



```
=WebGetSingleFromTable("http://finance.yahoo.com/q?s=" & A1, "columnname=DaysRange", "first(
EPSttm)")
```

	A	B
1	GS	16.89
2	MSFT	
3		

Now if we copy the formula in B1 to B2, Excel automatically handles the updated cell reference, and we get a different value retrieved based on the symbol in A2:



```
=WebGetSingleFromTable("http://finance.yahoo.com/q?s=" & A2, "columnname=DaysRange", "first(
EPSttm)")
```

	A	B
1	GS	16.89
2	MSFT	1.48
3		
4		

These general principles apply to other xSkrape functions and can be used to build sophisticated calculations as shown in our Excel examples.

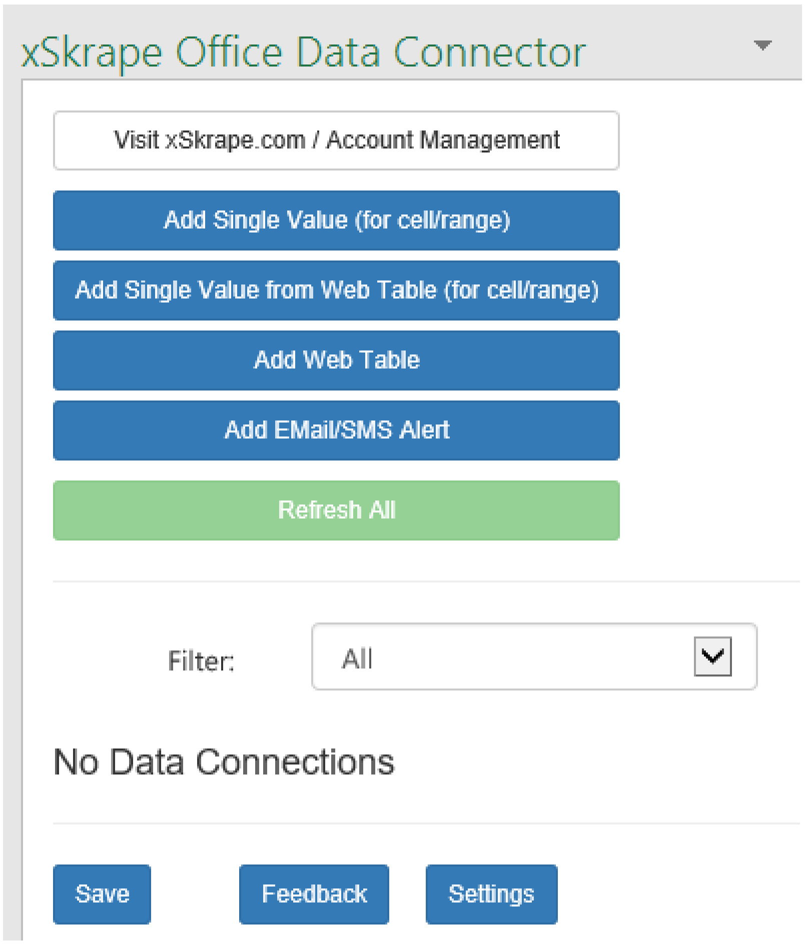
xSkrape for Excel - Web Add-in

The xSkrape web add-in has been introduced in the first half of 2016. Being web-based, we are able to offer it in the Microsoft Office Store. However: its functionality is very similar to that of the Excel Desktop add-in. Instead of using functions, we offer data connectors that do *the same thing*.

The add-in interacts with our cloud services that perform requests on your behalf. This means you can use our REST-based API directly too. (Documentation to follow.) The usage model is based on “free” client software, where we charge “per transaction” as work is performed on our server. You can purchase credits directly from us on an as-needed basis, or get them monthly using a subscription from the Microsoft Office Store. If you install the Trial version from the Office Store, trial expiry only means you may lose your “free credits” offered when you sign up: you can still use your xSkrape.com account and your client connection to it, simply by topping up your credits.

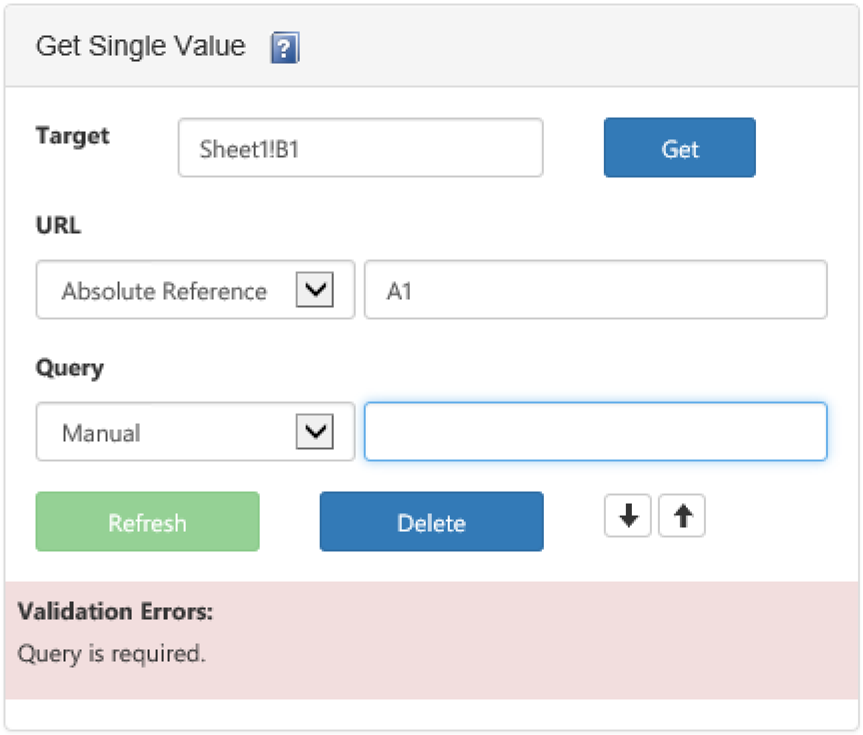
The web add-in offers one piece of novel functionality that the rest of xSkrape does not have (i.e. it’s not in the desktop add-in, or xSkrape for SQL or SSIS). It’s the “Send Email/SMS” connector, and it will soon be augmented by a “Send Push Notification” connector that will work with iOS and Android.

The add-in is implemented as a task pane.


xSkrape Office Data Connector
Visit xSkrape.com / Account Management
Add Single Value (for cell/range)
Add Single Value from Web Table (for cell/range)
Add Web Table
Add EMail/SMS Alert
Refresh All
Filter:
All
No Data Connections
Save
Feedback
Settings

The four blue buttons add new connectors. The settings for the connectors is stored in the current document, and is committed to the document when you refresh data or click “Save”. The Filter drop-down restricts the view to either all connector types or a specific one. “Refresh All” fires a Refresh on all visible connectors, in the order they are shown. Having the order be enforced allows construction of sequenced actions that have dependencies.

Looking at the “Single Value” connector:


Get Single Value

Target: Sheet1!B1

URL: Absolute Reference — A1

Query: Manual

Validation Errors:
Query is required.

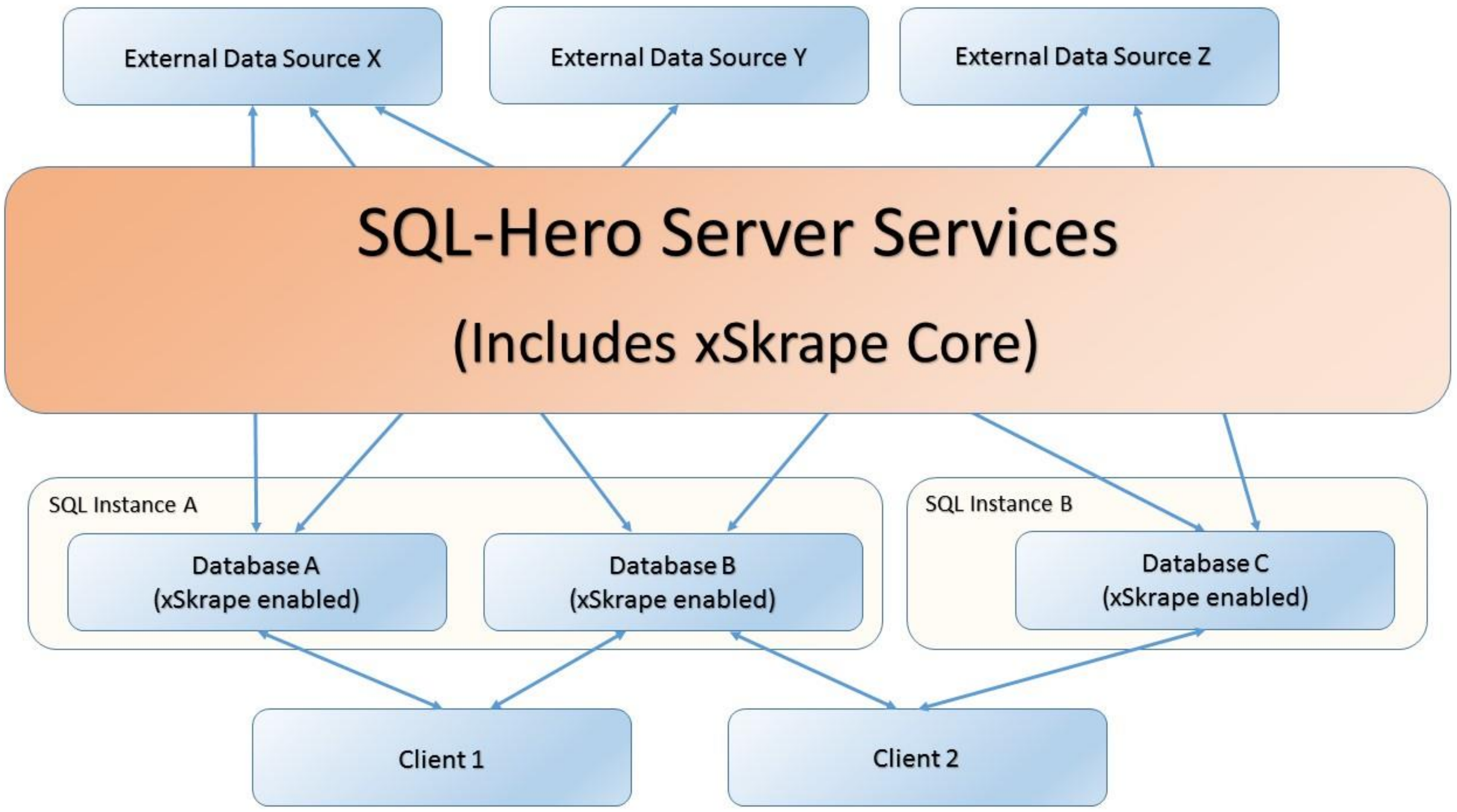
"Target" is the cell location for the result of the data connector, on refresh. It can include a sheet name separated from the cell reference with a "!", or it can include just a cell reference in which case the current worksheet is assumed.

"URL" can be an actual web address (e.g. starting with http or https), or a XS.QL expression. Any validation errors will disable the Refresh button. Up/Down arrows let you move the connector ahead of or behind other connectors to establish the desired refresh order.

The actual meaning behind each parameter (URL, Query) can be found by looking at the very same "function" that's documented for the Desktop add-in – or any of the other "flavors" of xSkrape.

xSkrape for SQL Server – Query Engine

xSkrape for SQL Server is actually packaged in the SQL-Hero product which has been available for some time. The reason this packaging decision was made is largely based on architecture requirements. SQL-Hero already has the concept of a web services API layer (hosted in Microsoft IIS), and this is what xSkrape for SQL relies on for its "magic." Why? Most xSkrape logic is contained in .NET code that would not be easy to deploy into SQL Server using SQL CLR Integration: the links to system (and third-party) libraries goes deep and the process would turn into a nasty rabbit hole of dependencies. xSkrape's answer to this is to still use SQL CLR Integration, but only using a *very* thin layer that communicates with the SQL-Hero services to do the hard work. This approach is illustrated here:



This has some implications for xSkrape for SQL users:

1. You need to install SQL-Hero server components on at least one machine that has IIS 7+. The process of installing SQL-Hero is out of the scope of this xSkrape document, but more details can be found in our SQL-Hero documentation.
2. You do need to enable CLR integration for SQL instances you wish to use xSkrape functions on.
3. You need to manage access licenses (except for Enterprise users – no such management is required): each machine that will contact the services layer needs to be registered and requires a SQL-Hero developer (or higher) license. (Two free client access licenses are included in the SQL-Hero Advanced edition which is the edition that includes the server components.)

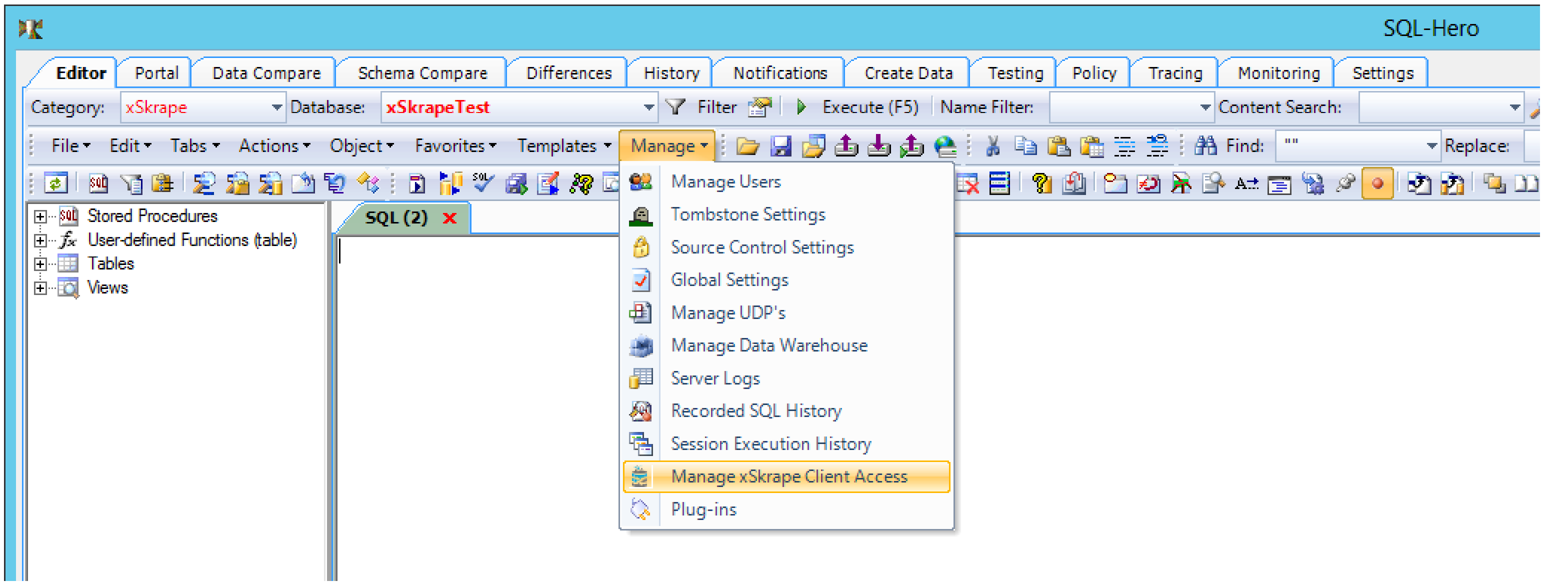
Step #2 can be achieved in different ways. The easiest is to use the *xSkrape Script Generator* program (xSkrapeForSQLScriptGen.exe) which is included in the SQL-Hero program files directory. This program guides you through basic tasks done as part of xSkrape for SQL Server including enabling databases for xSkrape usage and creating the actual SQL wrapper objects that let you access tabular data in a familiar way: using SQL queries! We provide more detailed documentation on the xSkrape Reference page.

The "enabling" of a database for xSkrape use means a few things:

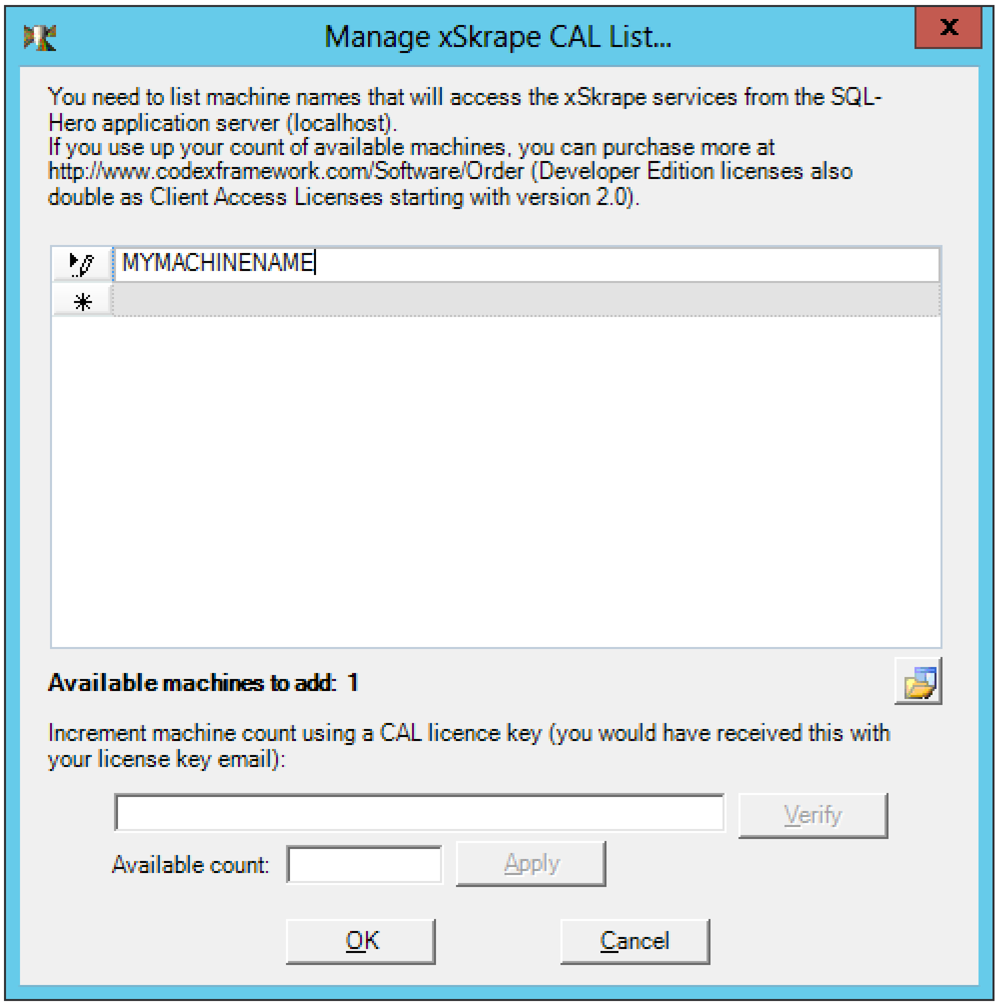
- The *instance-level* "clr enabled" setting is set to "true". If you don't want to do this (or are not able due to policies or privileges), it rules out using xSkrape for SQL Server.
- Your target database which will "receive" the xSkrape CLR objects is set to "TRUSTWORTHY ON". Again, if this isn't an option for you, xSkrape can't be used.
- xSkrape CLR assemblies are added to the target database.
- xSkrape CLR UDF's are added to the target database.

Note that the script drops / creates objects such that any upgrade of xSkrape you wish to apply can be done using the same process as *enabling* the database. It also means the step of upgrading must be done *manually*, and keeping the CLR objects "in-sync" with the server-side components is something you need to pay attention to.

Managing access licenses can be done using the SQL-Hero desktop application, from the “Manage xSkrape Client Access” menu option:



You can enter as many machine names as you have available licenses for:


Manage xSkrape CAL List...

You need to list machine names that will access the xSkrape services from the SQL-Hero application server (localhost).
If you use up your count of available machines, you can purchase more at http://www.codexframework.com/Software/Order (Developer Edition licenses also double as Client Access Licenses starting with version 2.0).

MYMACHINENAME

Available machines to add: 1

Increment machine count using a CAL licence key (you would have received this with your license key email):

You can also import a list from a text file, or supply a CAL license key to increment your available machine count. It’s important to note the machine count is tied to the number of SQL machines, *not* clients that connect to SQL Server. For example, if you have one production, one staging and one development SQL server instance hosted on 3 different boxes, you’d need three

CAL's. It would not matter how many users connect to production or any of the other environments (i.e. it could be hundreds or thousands). One CAL is included per SQL-Hero Developer Edition license, meaning in the above scenario since two free CAL's are included in an Advanced Edition license, you'd probably want to purchase one additional Developer Edition license and use the CAL key included in the registration e-mail to increment your available machine count to three, then list the three SQL Server machine names for your three environments. (Enterprise Edition users have no CAL management required – they have effectively an unlimited number of available machines.)

The architecture we're using relies on CLR user-defined functions to return data to your applications. The implication for the WebGetTable function is it must have a tabular structure that's rarely if ever shaped exactly like your source data. If for example one of your sources has six columns and another one needs eight – we need a way to return data using the *same* number of columns, since UDF's can't change their returned table structure dynamically. Our way to do that is using row and column numbers, as shown here:

```
create function dbo.WebGetTable (@serviceUrl nvarchar(1000), @parseUrl nvarchar(MAX), @tableCriteriaQuery
nvarchar(MAX), @additionalConfig nvarchar(2000))
returns table (RowNumber int, ColumnNumber int, Value nvarchar(MAX))
```

This is what we'd consider "raw data." You would for example see results like this:

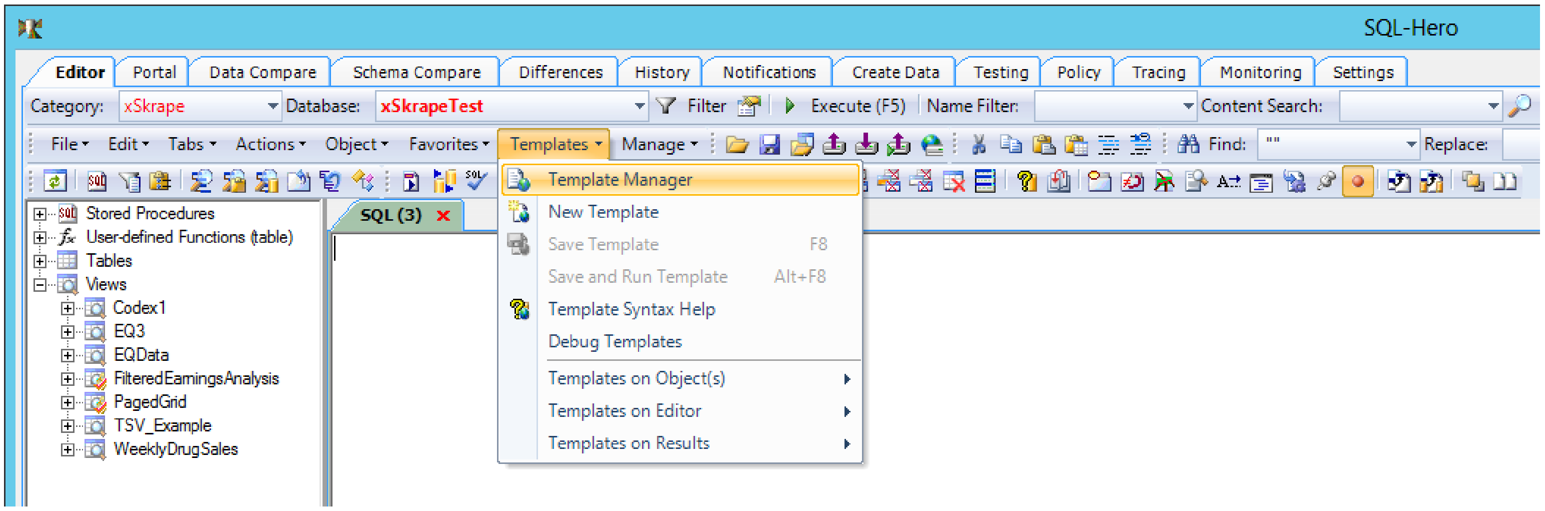
SELECT * FROM [dbo].WebGetTable('http://localhost', 'http://earthquake.usgs.gov/fdsnws/event/1/query?starttime=2015-04-18&endtime=2015-04-29&minmagnitude=4.0&format=csv', '', 'Timeout=90')

RowNumber	ColumnNumber	Value
0	0	4/28/2015 4:50:14 PM
0	1	-5.0631
0	2	129.7692
0	3	202.58
0	4	4.5
0	5	Mb
0	6	
0	7	75
0	8	3.264
0	9	1.02
0	10	Us
0	11	us20002a0r
0	12	7/2/2015 5:23:46 PM
0	13	206km SE of Saparua, Indonesia
0	14	Earthquake
1	0	4/28/2015 4:27:27 PM
1	1	-26.0954
1	2	179.7
1	3	489.94
1	4	4.4
1	5	Mb
1	6	
1	7	89
1	8	3.776
1	9	0.82
1	10	Us
1	11	us20002c6e
1	12	7/2/2015 5:23:46 PM

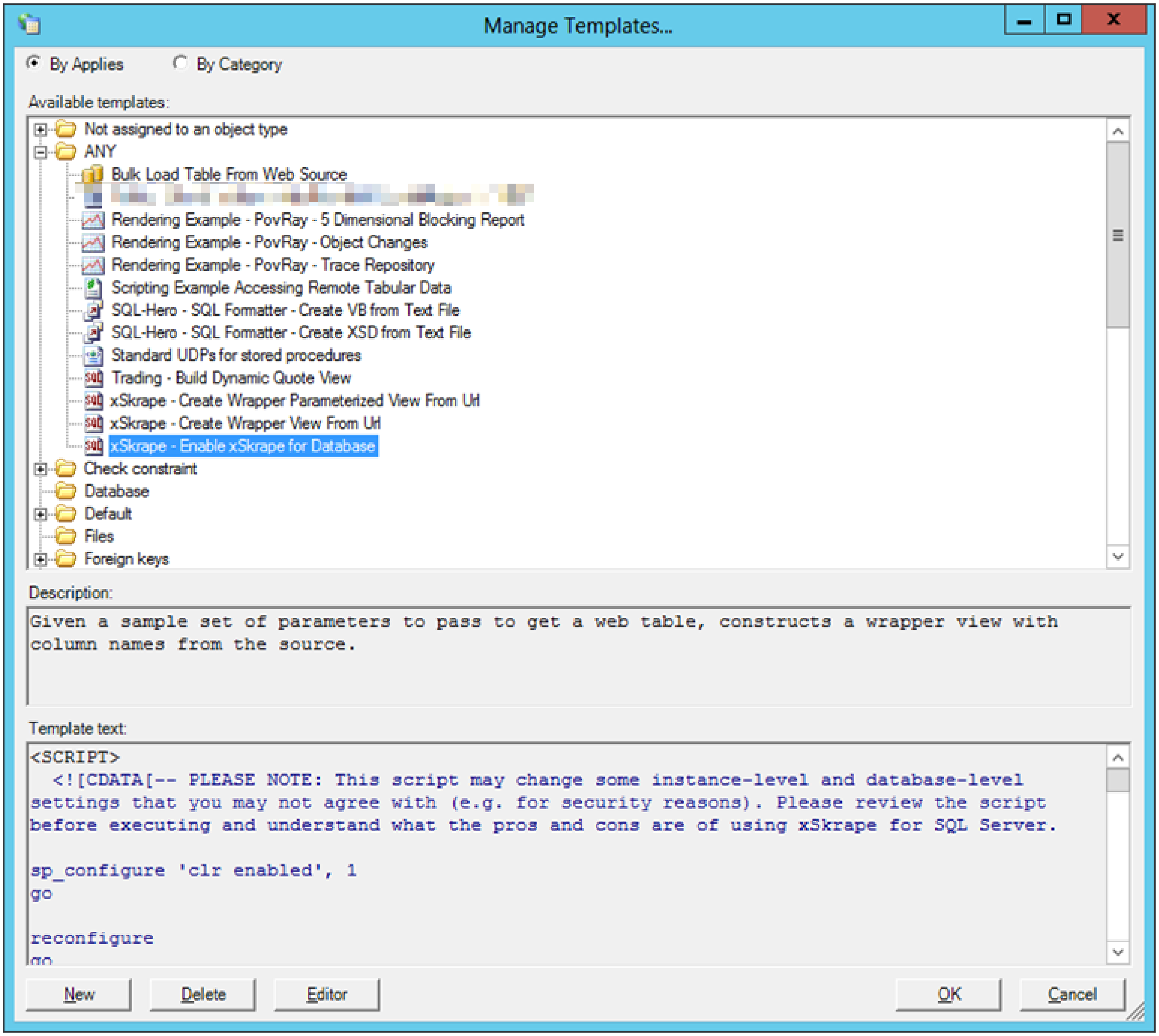
1	13	South of the Fiji Islands
1	14	Earthquake
2	0	4/28/2015 4:27:16 PM
2	1	15.827
…	…	…

This is technically viable information if we hard-code our queries using the numeric column numbers, and we'd likely want to *pivot* to turn row numbers into actual rows. xSkrape addresses this using *code generation* to create views or table-valued user-defined functions that: a). convert column numbers into textual column names, b). pivot based on row number, and c). return data in native SQL types that match what's present in the source data.

You can perform this code generation using one of two approaches that yield the same end-result: a). within the SQL-Hero desktop application, or b). using the xSkrape Script Generator tool. From SQL-Hero, one can invoke xSkrape templates from the Template Manager:



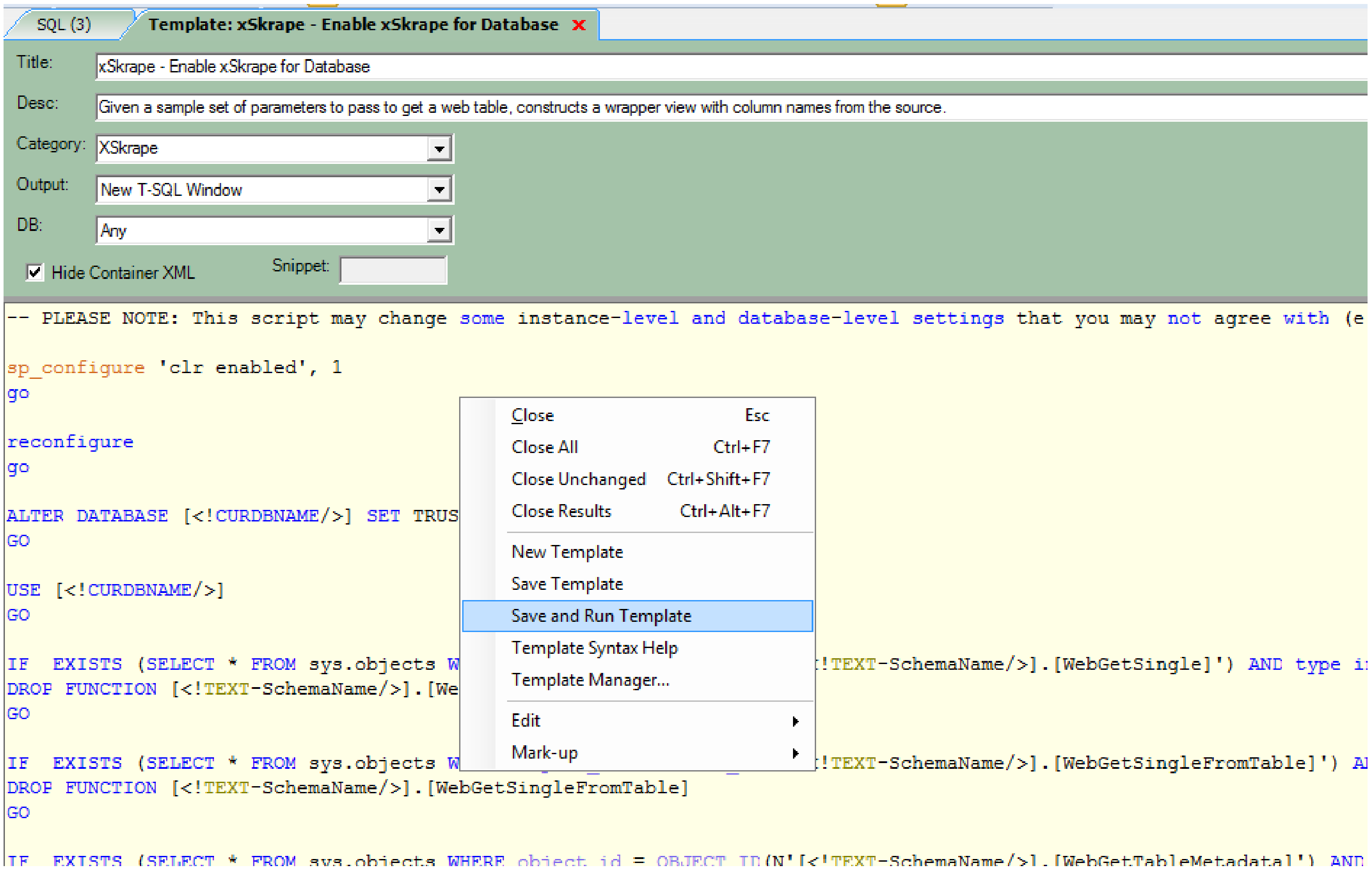
… Followed by the specific xSkrape template of interest:


Manage Templates...
By Applies
By Category
Available templates:
Not assigned to an object type
ANY
Bulk Load Table From Web Source
Rendering Example - PovRay - 5 Dimensional Blocking Report
Rendering Example - PovRay - Object Changes
Rendering Example - PovRay - Trace Repository
Scripting Example Accessing Remote Tabular Data
SQL-Hero - SQL Formatter - Create VB from Text File
SQL-Hero - SQL Formatter - Create XSD from Text File
Standard UDPs for stored procedures
Trading - Build Dynamic Quote View
xSkrape - Create Wrapper Parameterized View From Url
xSkrape - Create Wrapper View From Url
xSkrape - Enable xSkrape for Database
Check constraint
Database
Default
Files
Foreign keys
Description:
Template text:
New
Delete
Editor
OK
Cancel

Given a sample set of parameters to pass to get a web table, constructs a wrapper view with column names from the source.

```
<SCRIPT>
  <![CDATA[-- PLEASE NOTE: This script may change some instance-level and database-level
settings that you may not agree with (e.g. for security reasons). Please review the script
before executing and understand what the pros and cons are of using xSkrape for SQL Server.

sp_configure 'clr enabled', 1
go

reconfigure
go
```

Clicking on “Editor”, the selected template is opened in an editor window, from where it can be run (either from the menu or as a context menu option as shown here):



Title: xSkrape - Enable xSkrape for Database

Desc: Given a sample set of parameters to pass to get a web table, constructs a wrapper view with column names from the source.

Category: XSkrape

Output: New T-SQL Window

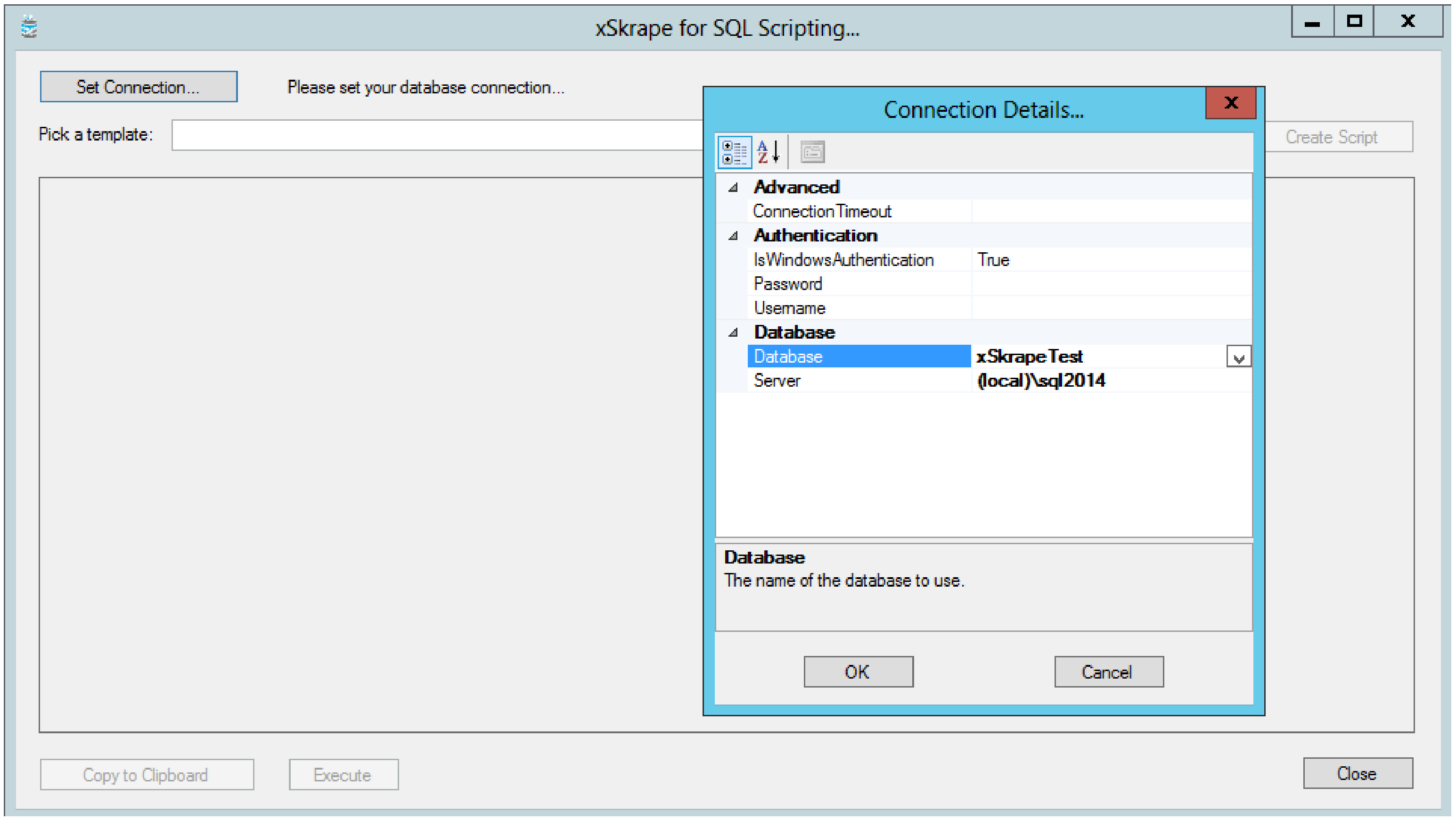
DB: Any

Hide Container XML: ✓ Snippet:

```
-- PLEASE NOTE: This script may change some instance-level and database-level settings that you may not agree with (e

sp_configure 'clr enabled', 1
go

reconfigure
go

ALTER DATABASE [<!CURDBNAME/>] SET TRUS
GO

USE [<!CURDBNAME/>]
GO

IF  EXISTS (SELECT * FROM sys.objects W                    !TEXT-SchemaName/>].[WebGetSingle]') AND type i
DROP FUNCTION [<!TEXT-SchemaName/>].[We
GO

IF  EXISTS (SELECT * FROM sys.objects W                    !TEXT-SchemaName/>].[WebGetSingleFromTable]') A
DROP FUNCTION [<!TEXT-SchemaName/>].[WebGetSingleFromTable]
GO

IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[<!TEXT-SchemaName/>].[WebGetTableMetadata]') AND
```

Since not everyone is a SQL-Hero client tool user, *Script Generator* offers a streamline approach for running the same xSkrape templates. There are four or five basic steps to using *Script Generator*:

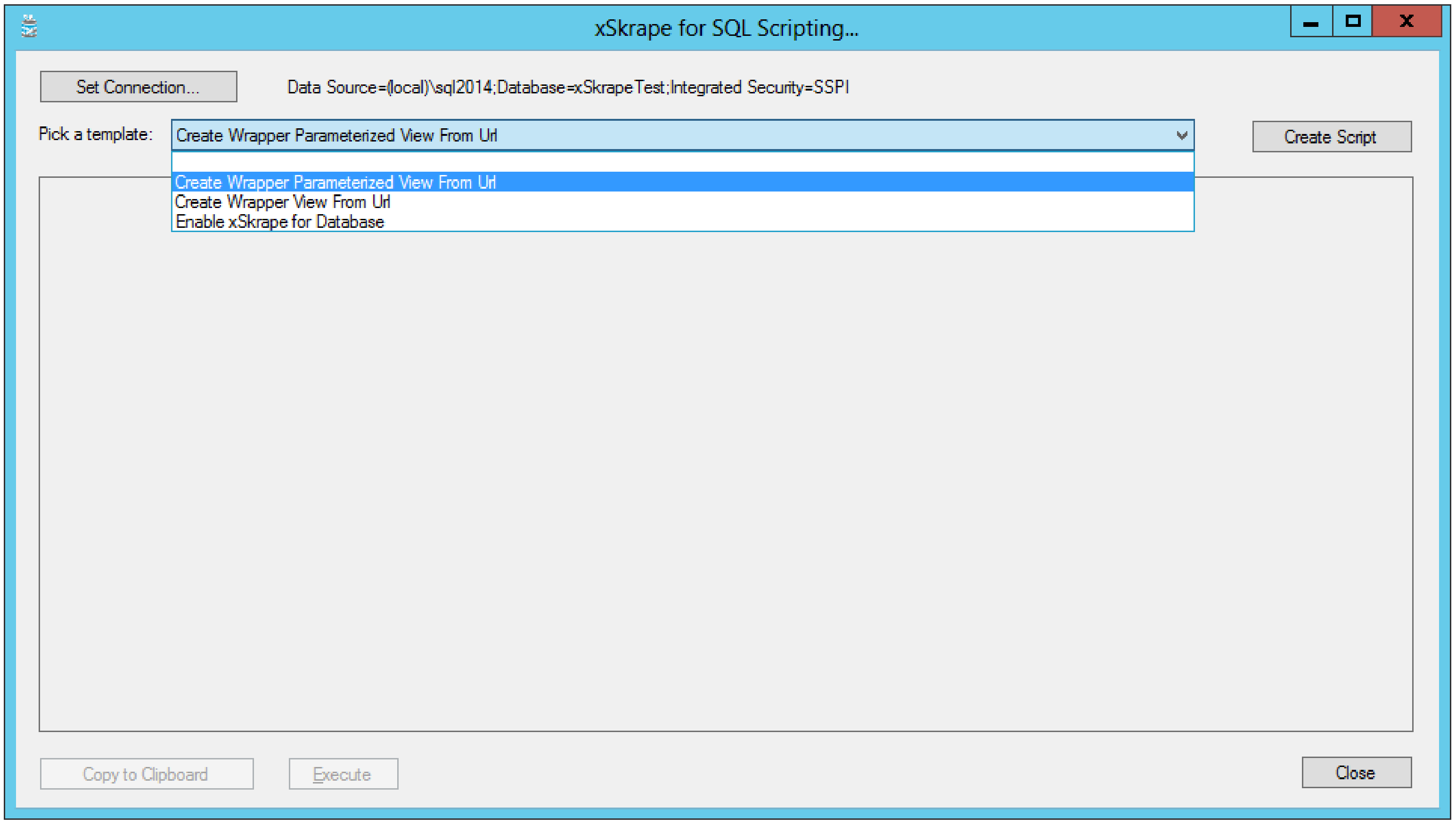
Step 1 – Set a database connection. For example:



Advanced	
ConnectionTimeout	
Authentication	
IsWindowsAuthentication	True
Password	
Username	
Database	
Database	**xSkrapeTest**
Server	**(local)\sql2014**

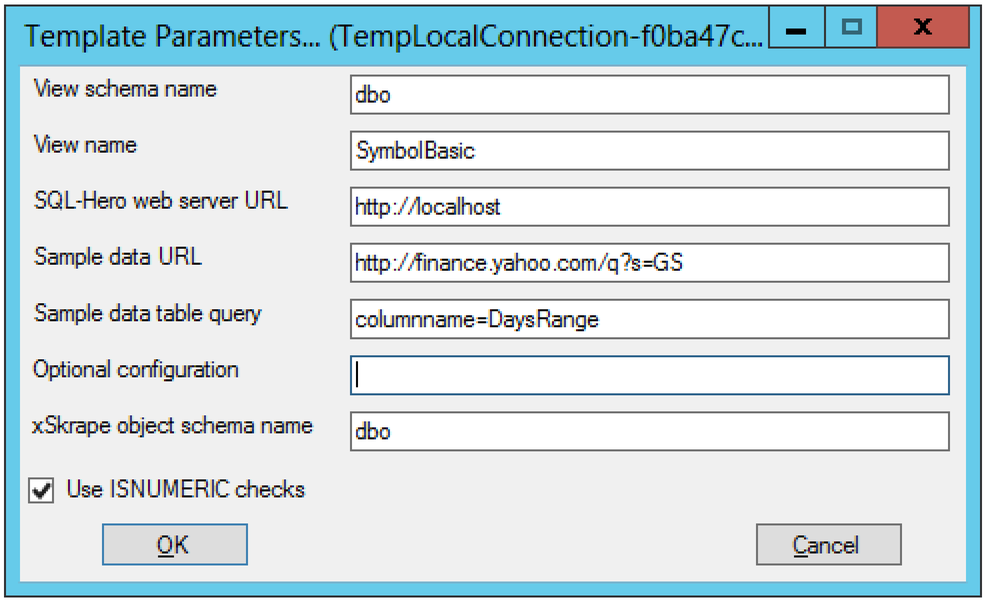
Database
The name of the database to use.

Step 2 – Pick your template and click on "Create Script":


xSkrape for SQL Scripting...
Set Connection...
Pick a template:
Create Script
Create Wrapper Parameterized View From Url
Create Wrapper View From Url
Enable xSkrape for Database
Copy to Clipboard
Execute
Close

Data Source=(local)\sql2014;Database=xSkrapeTest;Integrated Security=SSPI

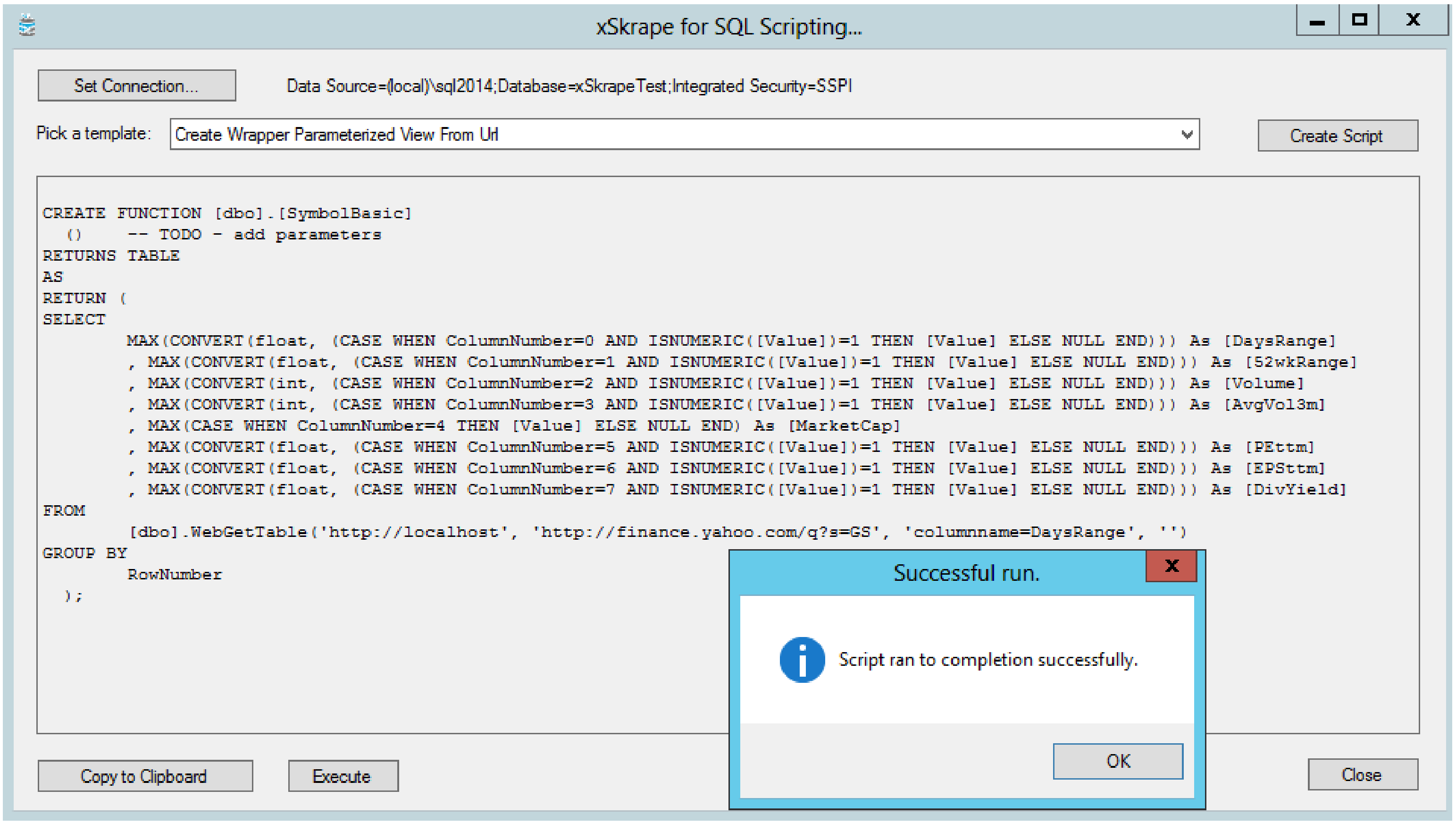
Step 3 – Enter parameters for the script. When creating wrapper objects, provide values that would produce *prototype results*. For example, here we're supplying values that hard-code a stock symbol (GS), but in reality we'll *parameterize* this value *after* the code has been generated. Sometimes you might be getting data that you don't need to parameterize: in those cases, a basic wrapper *view* is a reasonable choice. If you do want to parameterize, the *parameterized view* template is a good choice. The "SQL-Hero web server URL" is displayed when you install the SQL-Hero server components and is typically of the form "http://machinename:46837" where "machinename" matches the name of the machine on which the components were installed and "46837" is the default port number that the installer uses. This is the endpoint that the SQL CLR functions use to communicate with the SQL-Hero web services which are doing the actual "heavy lifting" of running your requests. (This is a consideration as it relates to credentials under which your requests will be run: in this case, the identity of the Application Pool that hosts your SQL-Hero web services.)



View schema name	dbo
View name	SymbolBasic
SQL-Hero web server URL	http://localhost
Sample data URL	http://finance.yahoo.com/q?s=GS
Sample data table query	columnname=DaysRange
Optional configuration	
xSkrape object schema name	dbo

☑ Use ISNUMERIC checks

Step 4 – Execute or copy the generated T-SQL. In this case we executed it, but notice there is "TODO" text in the body of the function.



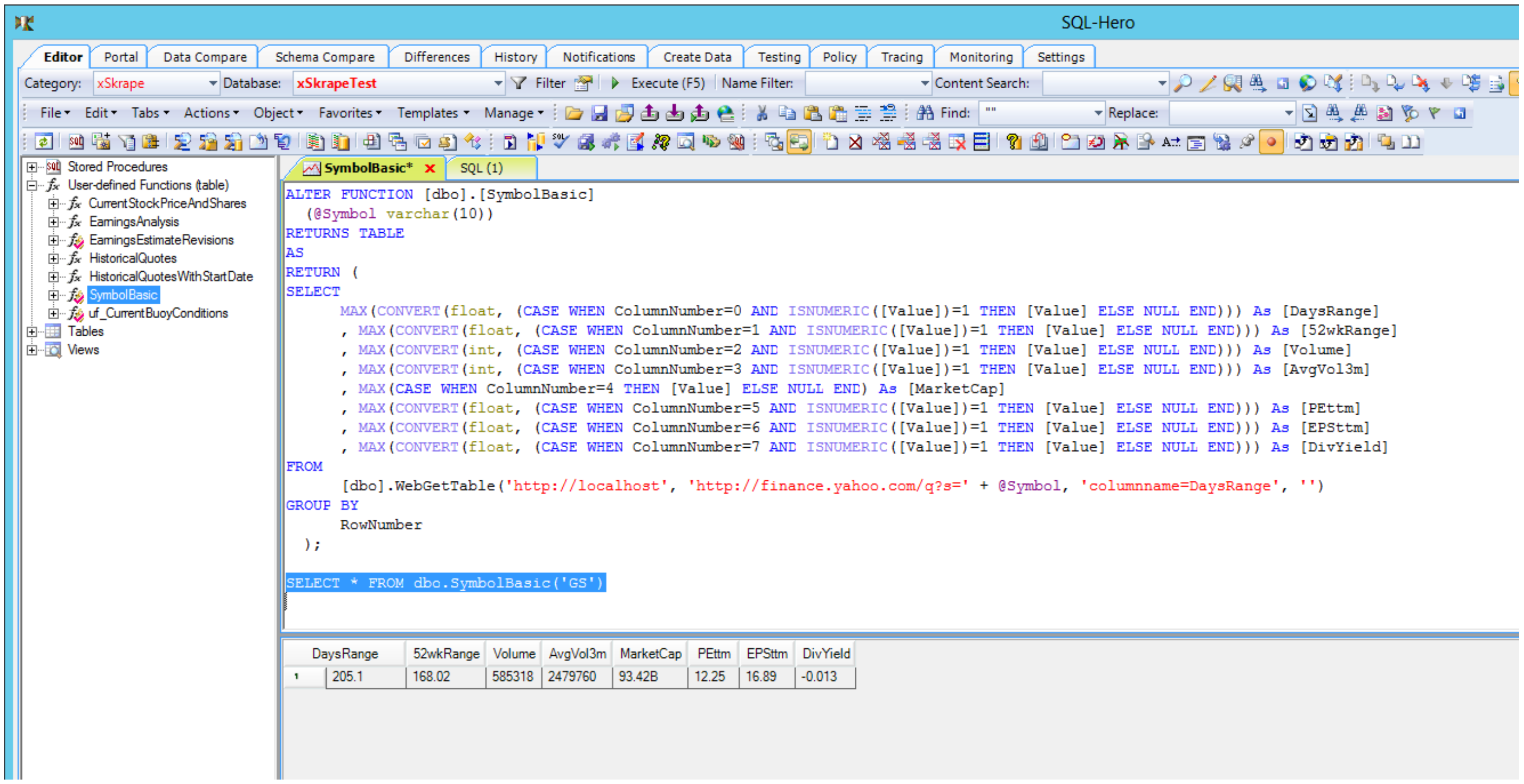
Data Source=(local)\sql2014;Database=xSkrapeTest;Integrated Security=SSPI

Pick a template: Create Wrapper Parameterized View From Url

```
CREATE FUNCTION [dbo].[SymbolBasic]
  ()     -- TODO - add parameters
RETURNS TABLE
AS
RETURN (
SELECT
        MAX(CONVERT(float, (CASE WHEN ColumnNumber=0 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [DaysRange]
        , MAX(CONVERT(float, (CASE WHEN ColumnNumber=1 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [52wkRange]
        , MAX(CONVERT(int, (CASE WHEN ColumnNumber=2 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [Volume]
        , MAX(CONVERT(int, (CASE WHEN ColumnNumber=3 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [AvgVol3m]
        , MAX(CASE WHEN ColumnNumber=4 THEN [Value] ELSE NULL END) As [MarketCap]
        , MAX(CONVERT(float, (CASE WHEN ColumnNumber=5 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [PEttm]
        , MAX(CONVERT(float, (CASE WHEN ColumnNumber=6 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [EPSttm]
        , MAX(CONVERT(float, (CASE WHEN ColumnNumber=7 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [DivYield]
FROM
        [dbo].WebGetTable('http://localhost', 'http://finance.yahoo.com/q?s=GS', 'columnname=DaysRange', '')
GROUP BY
        RowNumber
  );
```

Script ran to completion successfully.

Step 5 – Customize the generated object. This step is technically optional. Here we've removed the "TODO", added a stock symbol parameter, and placed it in the proper place within the function to remove the hard-coded "GS" value:



```sql
ALTER FUNCTION [dbo].[SymbolBasic]
  (@Symbol varchar(10))
RETURNS TABLE
AS
RETURN (
SELECT
      MAX(CONVERT(float, (CASE WHEN ColumnNumber=0 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [DaysRange]
      , MAX(CONVERT(float, (CASE WHEN ColumnNumber=1 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [52wkRange]
      , MAX(CONVERT(int, (CASE WHEN ColumnNumber=2 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [Volume]
      , MAX(CONVERT(int, (CASE WHEN ColumnNumber=3 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [AvgVol3m]
      , MAX(CASE WHEN ColumnNumber=4 THEN [Value] ELSE NULL END) As [MarketCap]
      , MAX(CONVERT(float, (CASE WHEN ColumnNumber=5 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [PEttm]
      , MAX(CONVERT(float, (CASE WHEN ColumnNumber=6 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [EPSttm]
      , MAX(CONVERT(float, (CASE WHEN ColumnNumber=7 AND ISNUMERIC([Value])=1 THEN [Value] ELSE NULL END))) As [DivYield]
FROM
      [dbo].WebGetTable('http://localhost', 'http://finance.yahoo.com/q?s=' + @Symbol, 'columnname=DaysRange', '')
GROUP BY
      RowNumber
  );

SELECT * FROM dbo.SymbolBasic('GS')
```

	DaysRange	52wkRange	Volume	AvgVol3m	MarketCap	PEttm	EPSttm	DivYield
1	205.1	168.02	585318	2479760	93.42B	12.25	16.89	-0.013

You can use any SQL editor you like for Step 5 such as SQL-Hero or SQL Server Management Studio. Here we've shown the "tweaked" object, plus the results that we get when we run it for a specific symbol. Notice:

- The output includes column *names*, not numbers
- The column data types are floating point, integer, or string, depending on the characteristics of the source data
- Row numbers are eliminated in favor of actual rows

The actual way these benefits work is the template engine calls the *WebGetTableMetadata* xSkrape CLR function, using the same parameters that would be passed to *WebGetTable*. The structure of the result set can be inspected and decisions made about column names and data types, at code generation time. This is important to understand: if your source data structure changes, the wrapper object structure *may* be invalidated and you may get unexpected results.

You *can* adjust the generated code however you like. In fact, you don't *have to* use the code generator at all, but it is a convenient way to get at very least a starting point. The generated objects can be used however you see fit inside larger SQL queries. Our Isolating and analyzing a specific table from an HTML page example demonstrates this nicely, with a fully functional stock screener tool built in less than 100 lines of T-SQL.

xSkrape for SQL Server Integration Services

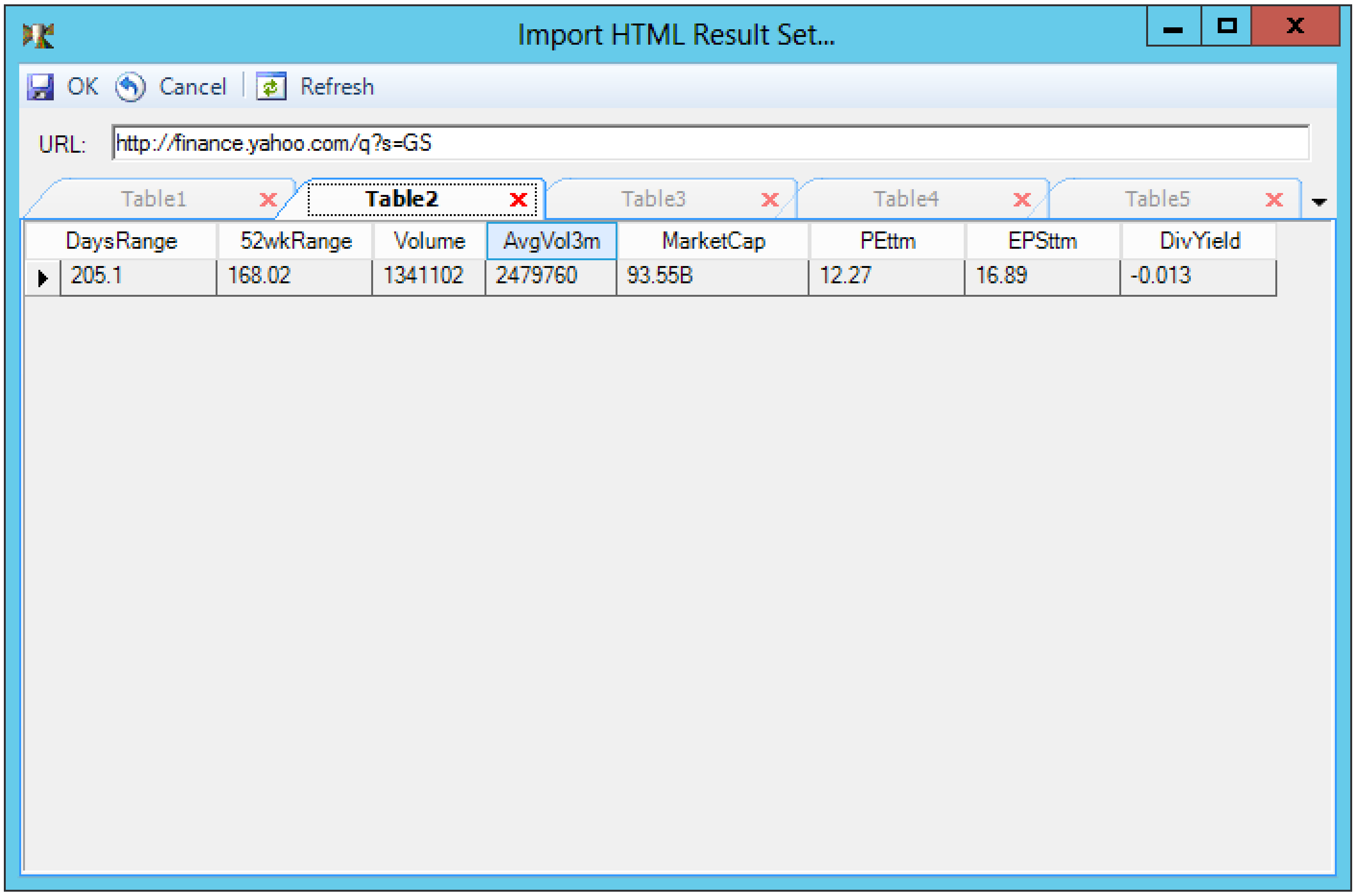
xSkrape for SSIS is really just a subset of several tasks and data flow components we offer as part of the SQL-Hero product package. The parsing logic of xSkrape is exposed in the "Dynamic Data Import" (DDI) task and "Dynamic Data Import" data source. The "fundamentals" for this really belong to SSIS itself: if you have an understanding of SSIS, using these components will be fairly straightforward. The DDI task can be placed in a control flow to move data from an external source to a table in a target database. The DDI data source can expose an external source as a stream of data to be included in a larger data flow process. Expect new components in up-coming releases, along with continual improvement on the existing ones – including even more documentation.

SSIS component licensing requires a Developer Edition license (or higher) for any machine that's running these components. In addition, many components (including the DDI ones) expect to "talk" to SQL-Hero server components, installed on typically one machine per organization. Server components are part of the Advanced Edition.

xSkrape – Within SQL-Hero

The genesis of xSkrape was from work done for SQL-Hero. Examples that have been available since the 1.0 release include:

- The "Scripting Example Accessing Remote Tabular Data" template which is included as an example of using the C# scripting feature in the template language, where the RelationalParser class is used to find "best match" tabular data and return it back in an enumerable class with dynamic items (i.e. properties corresponding to column names of the matched table). This is using the core xSkrape parsing engine and is still available to SQL-Hero users, inside the template / code generation engine. And wherever the template engine is exposed, this functionality follows, so for example in the SHCommand.exe command-line automation tool.
- The BULK_IMPORT template tag. This tag has a great deal of "smarts" – in fact, the SSIS "Dynamic Data Import" task we publish is actually just a wrapper for BULK_IMPORT.
- The "Import HTML Result Set" option, available within the SQL Editor tool window allows you to identify all tabular data from a URL and selectively import these into the current editor window as result sets (which can further be manipulated with result set commands):



URL: http://finance.yahoo.com/q?s=GS

DaysRange	52wkRange	Volume	AvgVol3m	MarketCap	PEttm	EPSttm	DivYield
205.1	168.02	1341102	2479760	93.55B	12.27	16.89	-0.013

Final thoughts

Although we've given a lot of examples using Finance and Trading as a topic area, you can see that the underlying concepts are not specific to any industry. (We just happen to have a lot of interest in Finance and the tools lend themselves very well to quantitative analysis.) The possibilities are as diverse as pulling in your competitor's public web-site pricing, to joining your internal data with public data sources in intuitive ways and with familiar tools, to streamlining your current usage of Excel by combining data from different sources, to doing “minimal effort” data imports from CSV / Excel files, to… your imagination is the only limit, really! If you *do* hit a limit, give it to us as a challenge to solve. Chances are, we’ll address it for you *and* the world! Ultimately our goal is to deliver *data solutions*, with less and less emphasis on the technology involved.

What did you think of this overview? We’d love to know! Feel free to drop us a line with comments or questions at mailto:info@codexframework.com.