

The Itsy Bitsy Spider: a Simple SAS-Based Robot

Laura Tuttle*

August 4, 2001

Abstract

This paper presents a SAS-based WWW spider used to extract information from an online database. Within the context of this application, I discuss using command-line helper applications within SAS, removing HTML markup tags from input files, dynamic web-query construction, and using %call sysfunc routines to perform error-checking in a WWW information retrieval setting.

As users of the SAS language, we are fortunate to have many procedures and modules available to make common data analysis and coding tasks simple. However, every once in a blue moon, a programmer finds herself presented with a task that is extraordinarily complex or just plain bizarre, where the tools at hand don't lend themselves well to creating the application. This paper is not about the former - there is nothing terribly complex about writing a limited-scope spider. But writing a spider in SAS seemed like an odd proposition at first; wouldn't it be better suited to perl or C++? Actually, in hindsight, SAS was a good language for such a task. The macro language provides all the iterative flow control needed, and the text manipulation and data storage capabilities of the base SAS language were perfectly suited to transforming and storing the information collected.

Background: the Itsy Bitsy Spider

The Itsy Bitsy Spider was developed in order to gather data from the internet for use in academic research, at the suggestion of Professor Ingrid Werner, Fisher College of Business, The Ohio State University. Professor Werner and I study stock market microstructure - how stocks are traded given exchange trading rules; in particular, we are examining why some Nasdaq stocks trade heavily on electronic trading networks such as Island and Instinet, while others trade primarily through Nasdaq market makers. The original application we envisioned for the Itsy Bitsy Spider (IBS henceforth) was to count Yahoo!¹ Finance and Raging Bull message board posts for individual stocks, keeping the date and time of each message. The idea of using message board posting frequency as a variable in stock-trading models is a novel one, to say the least; our hope is that message board posting frequencies will be good proxies for active investor uncertainty about stock values and will help in predicting daily stock market volatility and off-exchange trading volume for individual stocks. IBS does not retain any message content: our goal is to measure which stocks are attracting the attention of a particular group of

*Email: tuttle.58@osu.edu, Department of Finance, Fisher College of Business, Ohio State University

¹Yahoo! is a registered trademark of Yahoo! Incorporated. I am not affiliated with Yahoo! in any way. While I'm on the subject of trademarks and copyrights, the content of Yahoo! message boards is not warranted in any way by Yahoo!, so obviously I can't warrant it either. Information posted to online forums and communities is in many ways public domain; however, many sites restrict the use of robots (which include spiders) in accessing their data. Read the Terms of Use for any site before implementing an application such as the one I'm presenting.

investors. Thus, we're not interested in what is being said so much as which stocks are being discussed. IBS could, however, be adapted to an application designed to analyze content for keywords, for example.

To elaborate on the scope of this project: there are over 5000 Nasdaq National Market stocks (approximately 3500 have message boards on Yahoo! Finance), and active stocks can have hundreds of thousands of posts in a year. In order to capture the date and time associated with each post, we needed to construct web-queries to return lists of messages; the HTML output file generated with each query returns 60 message headers at a time. Consequently, gathering data for a single stock can involve literally tens of thousands of web queries.

At a fundamental level, the program is very simple. It begins by determining how many messages there are for a particular stock. Then it retrieves the message headers in groups of sixty. With each query, an HTML document is returned. IBS removes the HTML tags from the input file, then reads in the records, extracting the date and time and adding those variables to its database. It continues until it has read all messages for a given stock, then proceeds to the next.

A number of challenges resulted in greater programming complexity, naturally. In particular, somewhat unpredictable results from helper applications required complex error-checking. (If you've browsed the WWW - you can probably imagine: server errors, incomplete files returned, nothing returned...) In addition, inconsistencies in database structure on the server-side required us to construct queries dynamically. All of these challenges were met within the SAS base and macro languages; IBS uses only one helper program (to actually retrieve the web document).²

IBS was implemented in October, 2000. In the three months after its implementation, it performed over 250,000 web queries and gathered nearly 12 million messages. Professor Werner and I are currently using data from IBS in an academic study on the determinants of off-exchange trading of Nasdaq listed stocks.

WWW Document Basics

Although gathering data from an online database may sound like a formidable task, in principle it is quite simple. A web page is nothing more than an ASCII file with embedded "tags" that tell a browser how to display the text. Images such as .gif and .jpg files are not contained within the HTML file itself - they are referenced within the HTML tags in such a way that the browser automatically retrieves them after seeing their references within the HTML file. Exhibit 1 shows a page of messages that IBS would retrieve³; Exhibit 2 shows a small part of the .html file that contains the information in the page, and the HTML instructions for how it is to be displayed. Trying to input text from the raw HTML file and extract the content of interest would be a nightmare; however, once the HTML tags are removed, we are left with Exhibit 3, a text file that can be processed within SAS⁴.

The second unusual task within a spider is the actual retrieval of the HTML file from the server.

²It is fairly simple to retrieve a web document within SAS. However, using a helper application reduced coding and allowed for greater error handling. In particular, a failed document request could be easily detected and dealt with by SAS when using a command line program to do the actual 'go for' work.

³Source: Yahoo! Finance, <http://messages.yahoo.com/>.

⁴Discussing the HTML tag removal is beyond the scope of this article. However, the code for the %dehtml macro is included in the simplified IBS program in the appendix. This macro has been customized for Yahoo! Finance message board output, but with slight changes should work for most similar applications where data is returned in HTML tables. Coding this macro was easily the most difficult part of writing IBS; fortunately, for most applications there is no need to reinvent the wheel. There are numerous command-line HTML tag strippers in the public domain - I used one initially in IBS, but switched to my own to improve program efficiency. Check CNET's shareware download library for several examples of public-domain or shareware HTML strippers.

Although SAS includes tools to access resources over the internet, the task at hand was greatly simplified by using a command-line utility to do the actual document retrieval. The X command allows the programmer to call a command-line program from within SAS; in the case of IBS, the command line program is a .bat file that calls Qweb.exe⁵ to retrieve an HTML document. In this case, the advantage to using a helper application stems from the possibility of document requests returning server errors and incomplete files. Coding for these exceptions is a nontrivial task; even with a helper application, some error checking within the spider is required. However, the error-checking only involves testing for existence of a file and checking to be sure the file contains the correct HTML termination string (showing the file is complete.) Dealing with these problems is the focus of the next section.

Challenges of WWW Data Retrieval

One of the real challenges in writing a program for data retrieval over the internet is dealing with the uncertain outcome of an HTTP server request. Almost everyone has encountered HTTP 404 errors or has seen web pages that began loading and never finished. IBS faces those challenges as well. If the document retrieval step stalls, the spider will also stall; incomplete documents and embedded non-ASCII characters are also potential problems. My solution is logic within the spider to ascertain whether the document retrieval step has returned a file at all, and if it has, logic to test whether the file is complete.

The document retrieval step begins with construction of a text file, getmsg.bat. This is an ASCII file that, when executed at the command line, calls qweb.exe to retrieve a specified URL. For example, this file might contain:

```
c:\ibs\qweb messages.yahoo.com/bbs?.mm=FN&action=l&mid= &board=4687950&sid=4687950&tid=xlrx&start=15311  
c:\ibs\board.htm  
  
exit
```

This command instructs qweb.exe to retrieve the document at

```
messages.yahoo.com/bbs?.mm=FN&action=l&mid=&board=4687950&sid=4687950 &tid=xlrx&start=15311
```

and save it as c:\ibs\board.htm. After executing this battery file with the X command in SAS, board.htm may or may not exist, and might be incomplete if it does exist. If we have failed to retrieve the complete file, we need to execute the battery file getmsg.bat again.

Two macros accomplish these goals. The first, %cleanup, tests for the existence of three files: getmsg.bat, board.htm, and board.txt (the battery file, the HTML file, and the text file produced by the HTML stripper, respectively.) If these files exist, they are deleted. By running this macro before constructing the getmsg.bat battery file and executing it from the command line, we can be certain that if it exists, the file board.htm was created successfully by qweb.exe.

The second macro is %errcheck. This macro executes the getmsg.bat battery file using the X command; it then checks for existence of the file board.htm. If the HTML file doesn't exist, it executes getmsg.bat again until the file is successfully created. Once the file exists, %errcheck inputs the file looking for the text string "<html>", the text within an HTML tag that marks the end of the formatted text. If it finds this text, the file is complete. If the text "<html>" is not found, %errcheck sets the success code for file existence to zero and executes getmsg.bat again.

⁵Copyright 1995, Prime Data Corp. Visit QFactory online at <http://www.q32.com>. For UNIX, the CURL7.1 package performs the same task; visit the publisher at <http://www.haxx.se/>.

The final problem, embedded non-ASCII characters, is a bit more tricky. For IBS, the presence of these characters is only a problem if they occur within message numbers, dates or time stamps. Consequently, the datastep that processes message text removes non-ASCII text from the retained fields. Programmers contemplating a WWW data retrieval application need to keep this potential pitfall in mind; a non-ASCII stripper could be incorporated into the HTML stripper without much difficulty.

Anatomy of the Itsy Bitsy Spider

IBS is written in SAS's macro language; it consists of five macros, with %spider being the main program. Three of the other four macros IBS calls (%dehtml, %cleanup, and %errcheck) have already been discussed. The fourth, %boardnum, is executed once for each message board to be cataloged; it retrieves the unique identifier number for the message board associated with a topic.

The simplified version of IBS that appears in the appendix of this paper⁶ is set up to retrieve the message numbers, dates and times associated with a single stock ticker in the Yahoo! Finance message boards. The Yahoo! message boards were actually the second database that IBS cataloged; the code that reads the text file returned by IBS naturally must be tailored to the application at hand. However, most online databases follow similar formats, so the discussion that follows is less database-specific than it might seem at first glance.

Flow charts for each of the macros within IBS appear in the appendix; the remainder of the discussion will walk through the run-time logic for the %spider macro itself.

Most online databases have some sort of main page where one can search for information by topic. Many limited-range spider applications will begin with a list of key words (in our case, stock ticker symbols); the first step is to identify the area of the database that contains information on this topic. In our case, this task is simplified by the organization of Yahoo!'s message boards by ticker symbol. Thus, the first step for IBS is to retrieve the web page at the URL:

```
http://messages.yahoo.com/?action=q&board=AAPL
```

where in this case, the ticker symbol is AAPL (Apple Computers). This request returns the latest page of messages for Apple; the unique message board identifier numbers can be extracted from the HTML tags in the page that link to previous groups of messages⁷ This function is performed by the %boardnum macro; it assigns the unique board number to the macro variable &boardnum.

Once the board number is identified, the queries to retrieve particular groups of messages can be constructed. Since the %boardnum macro returned the most recent batch of messages, we can strip the HTML tags with %dehtml and read in the text file. To construct a query to retrieve the next group of messages, we need to construct a query such as:

```
http://messages.yahoo.com/bbs?.mm=FN&action=1 &mid=&board=4686874&sid=4686874&tid=aapl&start=178896
```

The parts of this query that change are the &boardnum, &sid, &tid, and &start variables. The &boardnum and &sid variables are typically equal; &tid is the ticker symbol of the stock and &start is the lowest message number that will appear in the next batch. Since messages are returned in groups of forty, it is tempting to use a %do loop to iteratively construct queries. In practice, this doesn't work very well: messages will occasionally be deleted due to obscenity or gaps will appear in the message

⁶If you would like an electronic copy, email me at tuttle.58@osu.edu.

⁷The board number must be extracted from the raw HTML file, since it appears in the HTML tags themselves. Exhibit 2 contains part of a raw HTML file; the number that appears immediately after &sid is the unique board identifier.

database for reasons unknown. Consequently, the first section of the %dehtml macro extracts the &start variable associated with the next group of messages and assigns it to the SAS macro variable &next⁸.

Once all the variables we need to construct the query are identified, IBS enters a %do %while loop that terminates with the condition &stop=1. As IBS iteratively retrieves batches of messages, it scans for message number 1⁹. When it finds the first message, it uses call symput to set the macro variable &stop=1, which causes the %do %while loop to terminate. Until that condition is met, IBS:

- Writes the text file getmsg.bat. This file instructs Qweb.exe to retrieve the URL with the next batch of messages.
- Calls the %errcheck macro, which uses the X command to execute getmsg.bat, checking to be sure the HTML file has been returned successfully.
- Calls the %dehtml macro to extract the next value of &start and remove all HTML markup tags.
- Reads in the message list.
- Appends the message list to its database.
- Calls the %cleanup macro to delete the files associated with the current group of messages: getmsg.bat, board.htm, and board.txt.
- Checks to see if the macro variable &stop =1. If so, the simplified version of IBS attached to this paper terminates.

Conclusion

In this paper, I've presented a SAS-based spider that can be used to retrieve information from certain online databases. The logic contained in the code is not terribly complex; the difficulties that a programmer faces in such an application relate more to the mundane problems encountered in performing HTTP queries, such as server errors, incomplete files, and embedded non-ASCII characters in the files returned.

Applications for a spider such as IBS are not limited to the admittedly limited-interest case of counting messages in an online forum. IBS could potentially interface with any keyword-based search mechanism; for example, counting news stories mentioning a particular company or event, or collecting historic stock-price information from one of the many online services that provide such data.

⁸In the HTML document in Exhibit 1, there is a link to "Previous 40"; that link contains the value for &next, which must be extracted before the HTML tags are removed.

⁹The stop criterion can be anything based on variables retrieved by the spider, such as a date prior to some cut-off date.