

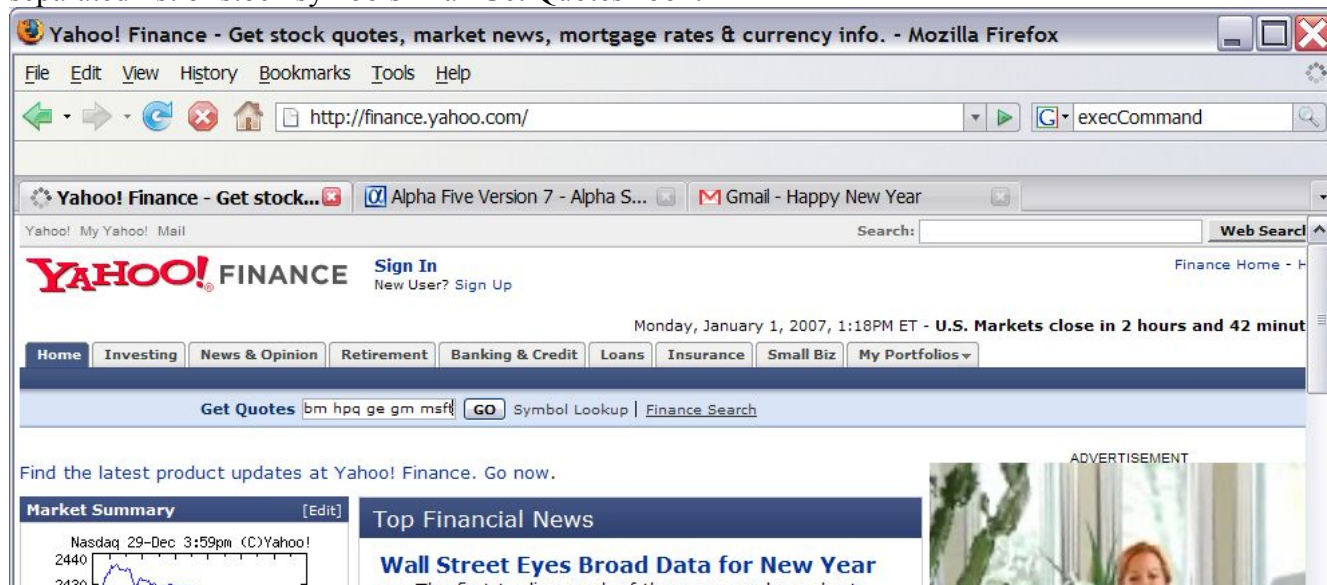
Screen Scraping with Alpha Five

by Dr. Peter Wayne

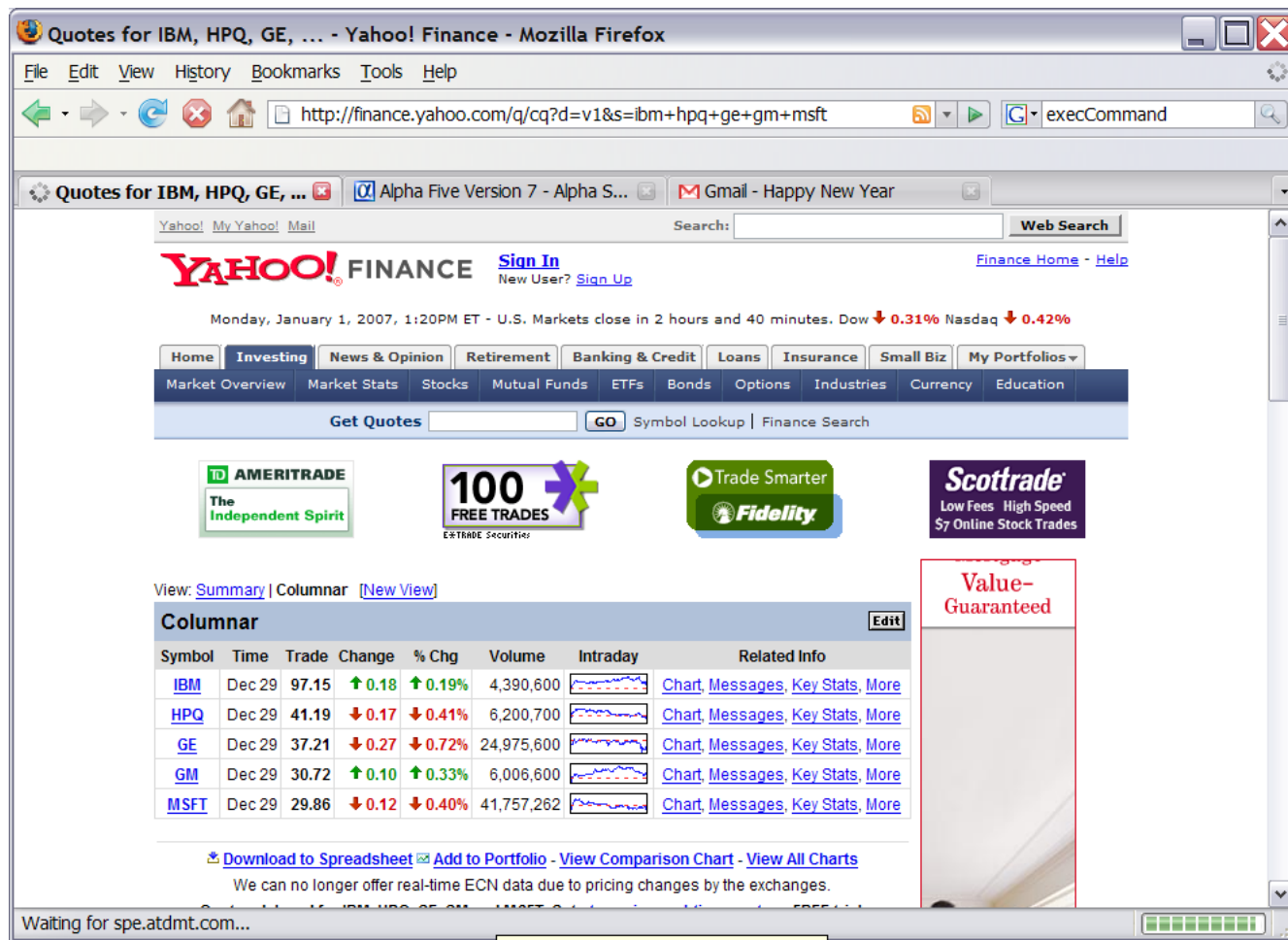
John Kukuda develops software for bail bondsmen, a demographic with which I've fortunately had no contact outside of Janet Evanovich's hilarities. I owe the idea behind this article to John, who insists he is not a programmer but seems to know his way around a keyboard. You can look at his well-designed web site at www.bonddetective.com.

John's clients have an abiding interest in law enforcement – they need to know when the people they have bonded are re-arrested. Monitoring arrest statistics is just the kind of information they need. John wants to direct his client's browsers to connect to a county jail's public web site, navigate through the site by entering the appropriate parameters, and then download the daily arrest statistics from the site. Had you asked me a few weeks ago, I would have thought it wasn't possible, certainly not through Alpha Five. John's attempt wasn't entirely successful but his effort convinced me it could be done.

Our example will use the Yahoo Finance web site. On this web site, you can enter a space-separated list of stock symbols in a “Get Quotes” box:



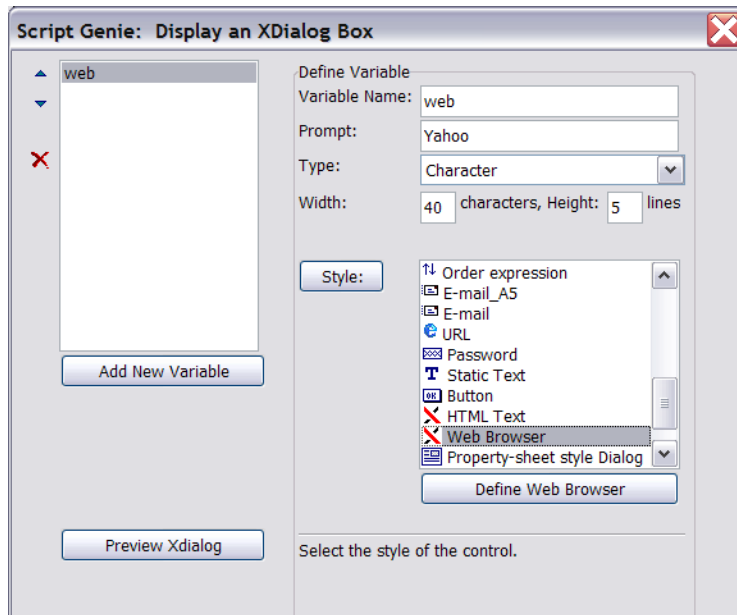
Pressing “Go” brings up a list of stock quotes:



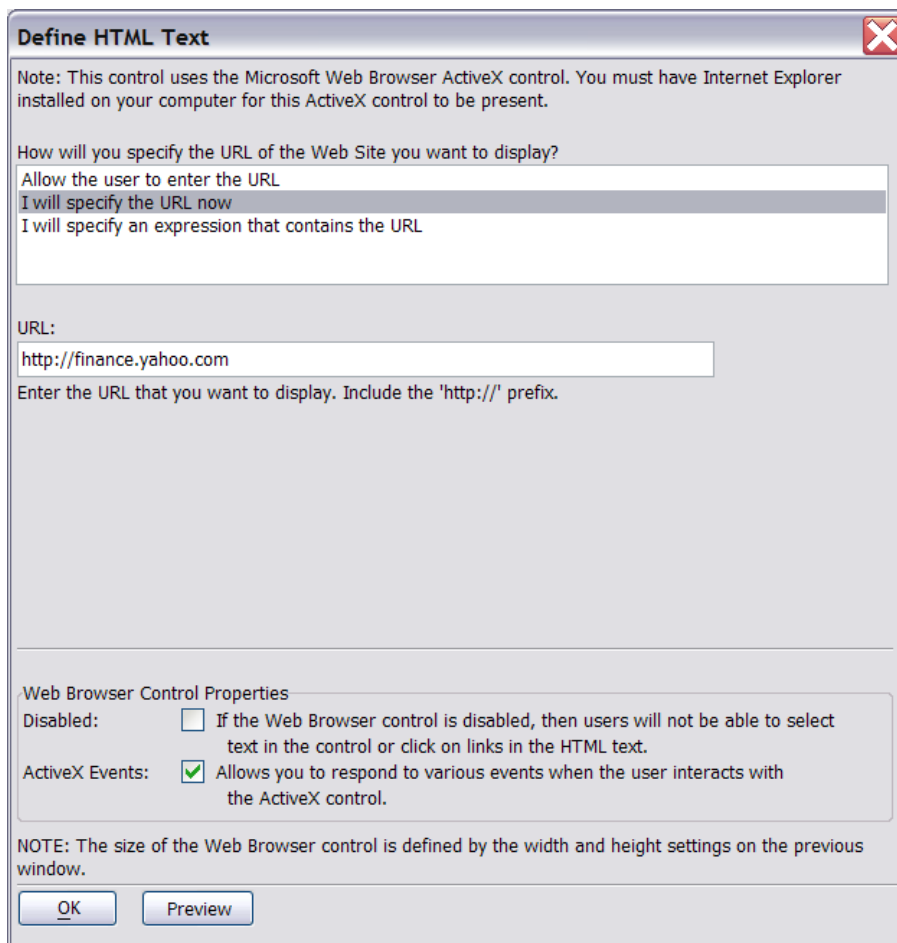
Let's see if we can submit the symbols programmatically and retrieve the quotes from Yahoo's web site.

The first step – Microsoft's Web Browser ActiveX control

What John recognized is that Alpha Five can *send actions* to the Microsoft Web Browser ActiveX control. What? You didn't know about this control? Not only is it there, but Alpha Five even has a genie to set it up for you. Let's get started – we'll set up an Xdialog with an embedded Web Browser:



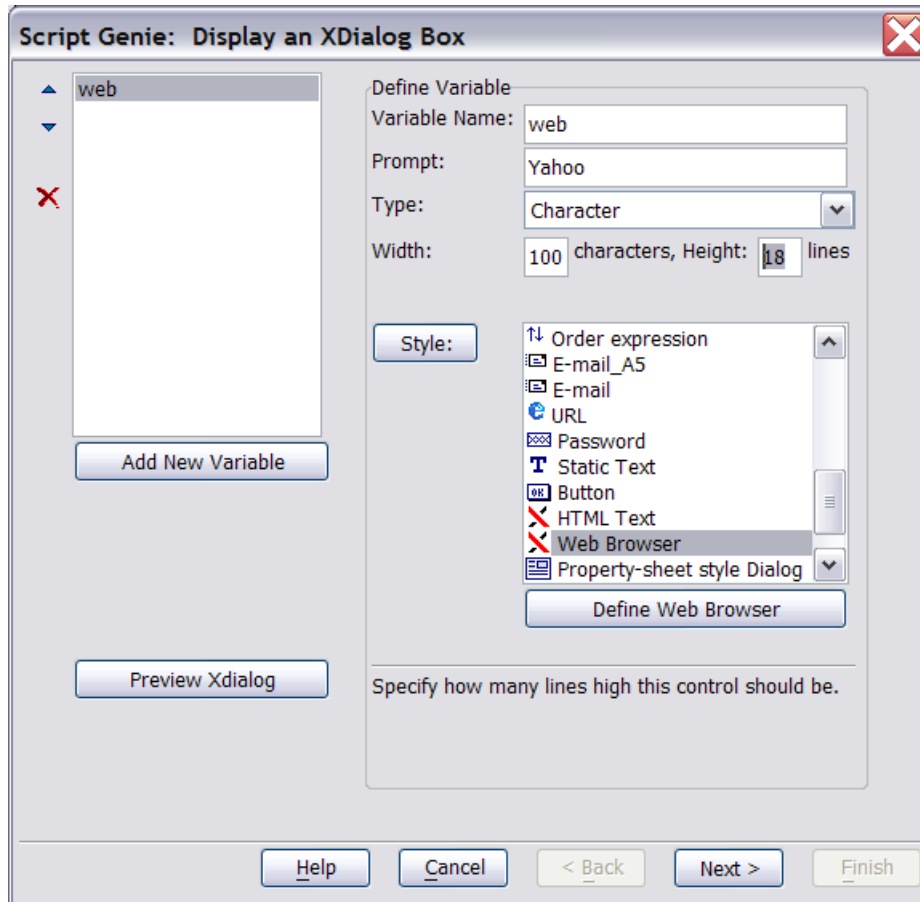
Click on “Define Web Browser”:



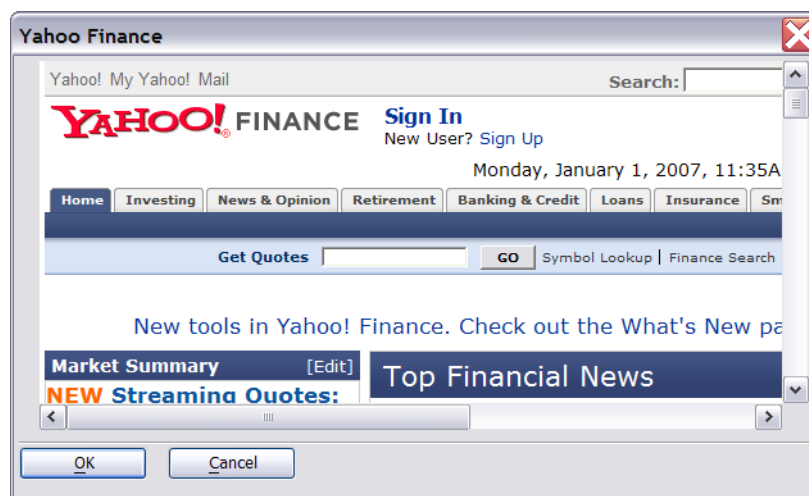
I preselected the URL of <http://finance.yahoo.com> and, *very importantly*, I also clicked on the box for

“ActiveX Events”.

Coming back to the first page of the genie, I've increased the web browser control dimensions to 100 characters wide and 18 characters in height:



Now I just quickly click on “Next” a few times to finish the rough and ready Xdialog. Running the generated code produces this:



If we inspect the generated code, we see that there are two main sections. One section describes the Xdialog and ActiveX web object itself and is relatively short:

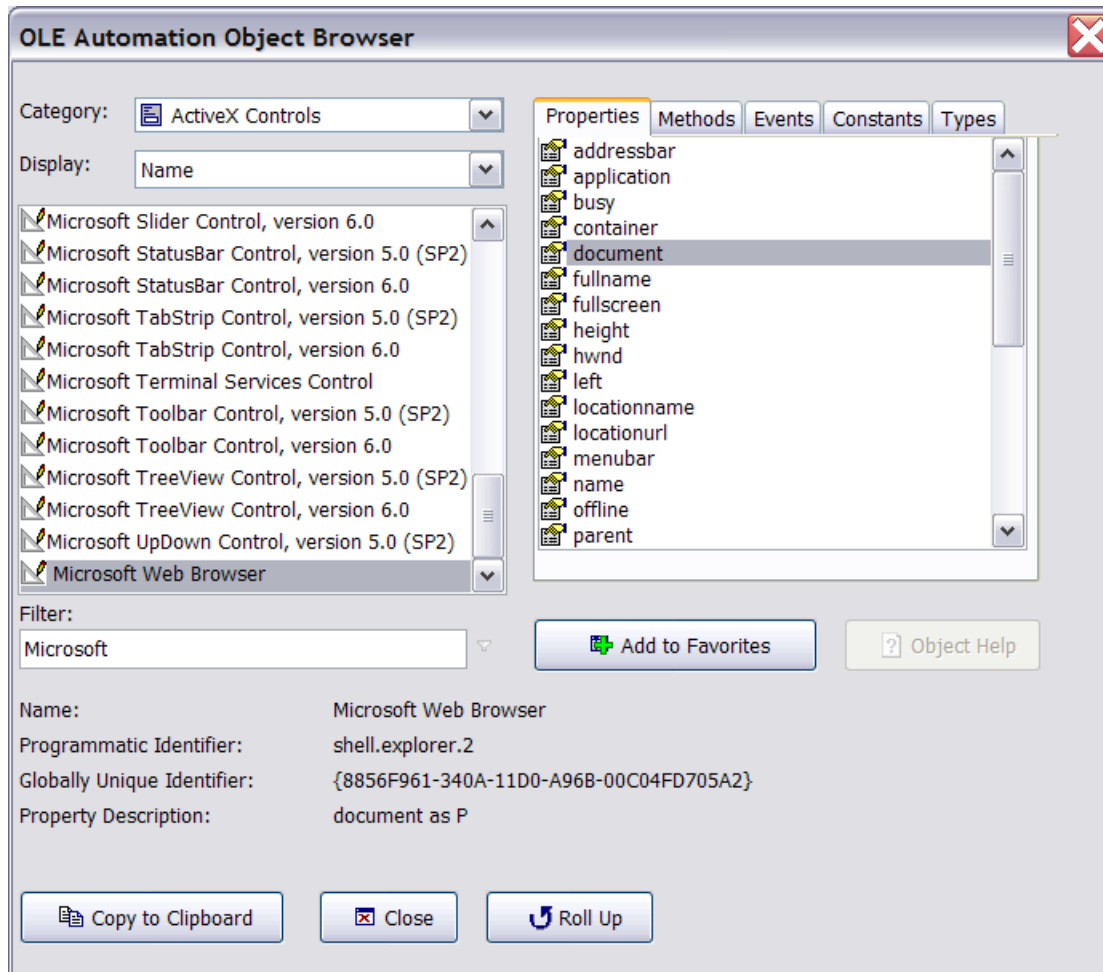
```
DIM web as P
dim web.object as p
dim web.class as c
web.class = "shell.explorer"
DIM varC_result as C
DIM web_url as c
web_url = "http://finance.yahoo.com"
ok_button_label = "&OK"
cancel_button_label = "&Cancel"
varC_result = ui_dlg_box("Yahoo Finance", <<%dlg%
{startup=init}
{region}
| {activex=100,18web?.t.};
{endregion};
{line=1,0};
{region}
<*15=ok_button_label!OK> <15=cancel_button_label!CANCEL>
{endregion};
%dlg%, <<%code%
if a_dlg_button = "init" then
    a_dlg_button = ""
    a_dlg_button = ""
    hourglass_cursor(.t.)
    if web_url <> "" then
        on error goto web_error
        web.object.navigate(web_url)
        on error goto 0
    end if
    hourglass_cursor(.f.)
end if
end

web_error:
ui_msg_box("Error", "Invalid URL.", UI_STOP_SYMBOL)
end
%code%)
```

If we run only this code, we get the full embedded web browser as shown above. But there's a lot more code generated by the genie: there's a whole **web.events** section. We'll get back to that later, but first let's talk about the web browser's properties.

Microsoft's Web Browser ActiveX Control Properties

We can learn more about the Microsoft Web Browser Control by opening Alpha Five's OLE Automation Browser, filtering on “Microsoft” and scrolling down to the Microsoft Web Browser entry. We can learn a few things from inspecting the entry:



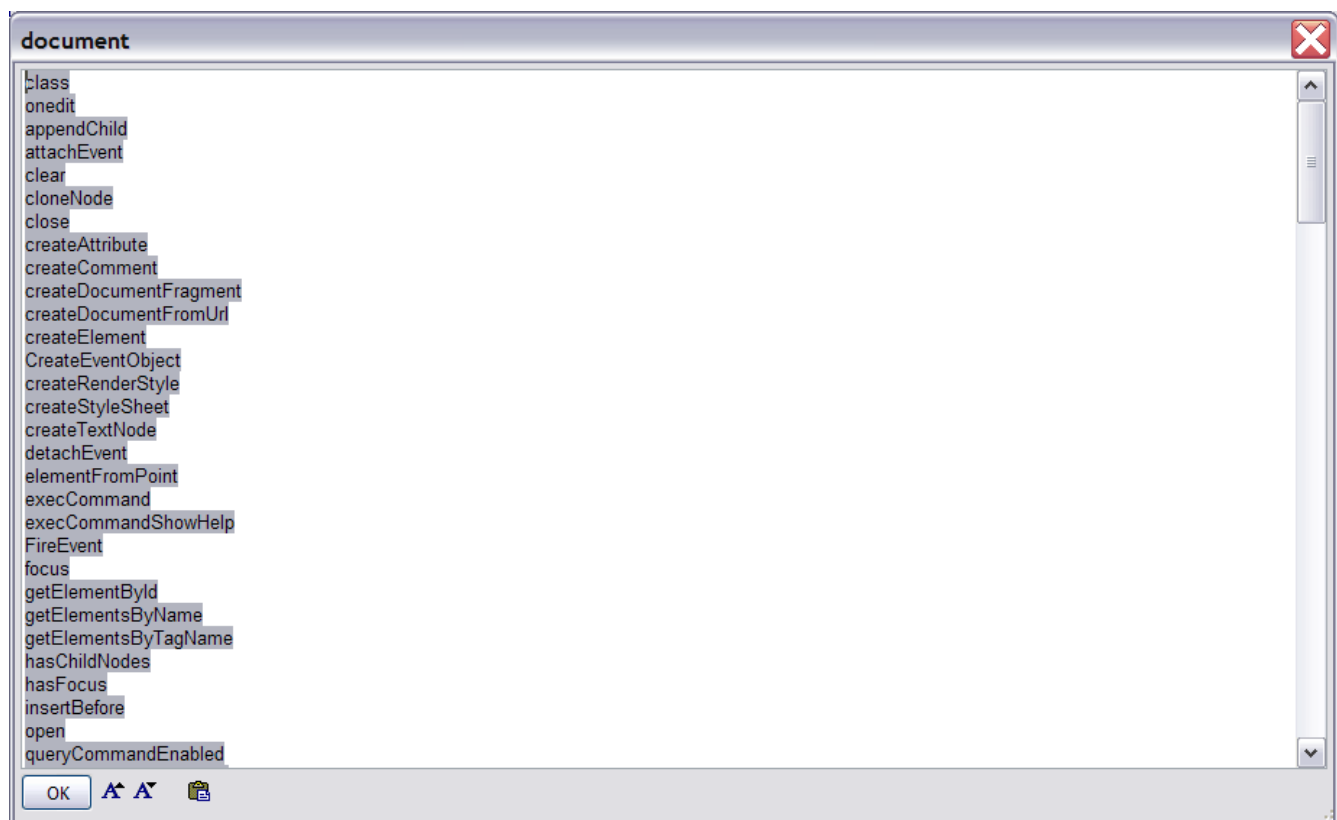
The first point of interest is that the “Programmatic Identifier” is “shell.explorer.2”, not “shell.explorer”, as provided by Alpha Five's genie. I don't know what the difference is -- “shell.explorer” seems to work – but even a cursory web search shows me that the rest of the world uses “shell.explorer.2”, so for my own peace of mind I changed the activeX control's **class** to “shell.explorer.2” in the Xbasic script.

The second point is that there are scads of properties and events defined for the Web Browser object. The genie has already provided us with stubs for the events, but what about the properties? Some of them look very promising. In particular, what about the **document** property? Let's examine it

by placing this code on a button in our Xdialog:

```
<*15=ok_button_label!OK> <15=cancel_button_label!CANCEL>;
<Inspect Document!document>
{endregion};
%dlg%, <<%code%
if a_dlg_button="document" then
  a_dlg_button=""
  showvar (properties_enum (web.object.document) , "document")
end if
```

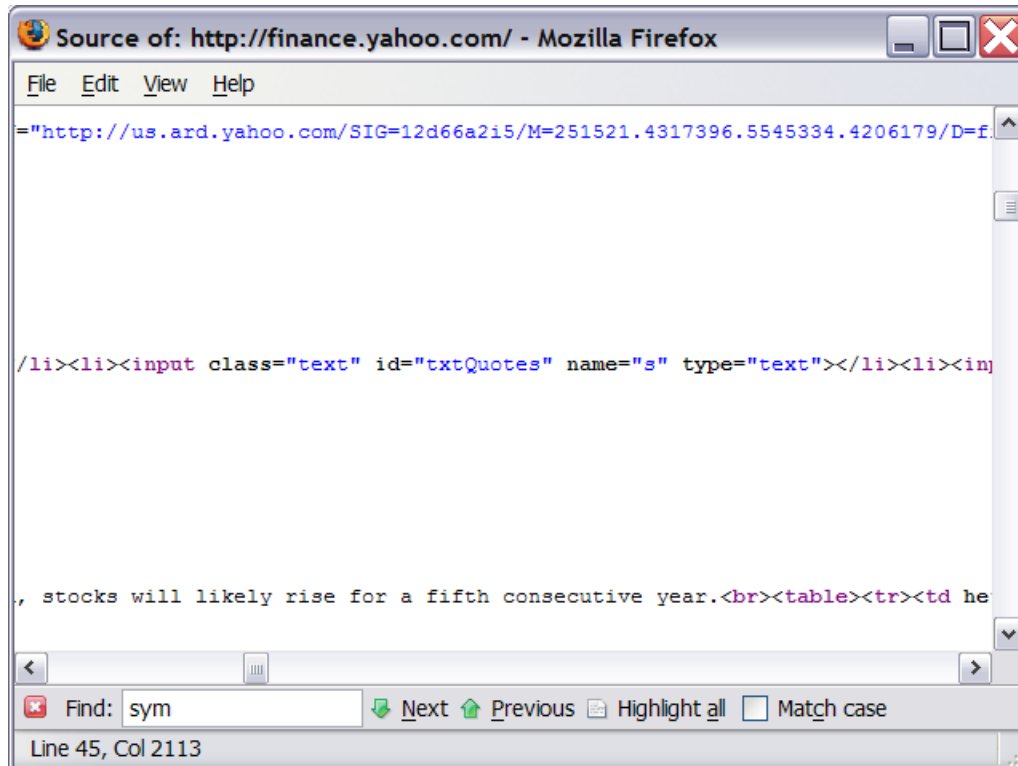
The sections in bold are new. Notice that the ActiveX object's properties are accessed through its *object pointer*. If I now run the Xdialog and press the **Inspect Document** button, I get this screen:



My, this looks very promising. Some of these I've never heard of, like **FireEvent**, but at least half of them are well documented Javascript commands for inspecting and manipulating HTML pages, and the rest can be found at Microsoft's web sites.

Probably the most useful HTML document method is **getElementById**. Any object on a web page is accessible through the Document Object Model or DOM, but sometimes the path to an object can be through a succession of long succession of confusing **childNodes**. If the object has a unique identifier, though, it can be found immediately and unambiguously through **getElementById**. The

good news is that most new web sites have attached unique identifiers to all input fields and buttons, if only to simplify the jobs of their Javascript programmers! If you inspect the source HTML of Yahoo Finance, you'll see a section like this:



There's an input text box for entering symbols whose unique identifier is **txtQuotes**. Similarly, the “Go” button on the web page has an **id** of **q**. So we can get our stock quotes from Yahoo Finance by putting values into **document.getElementById(“txtQuotes”)** and pressing the button known as **document.getElementById(“q”)**. Here's our code:

```
<*15=ok_button_label!OK> <15=cancel_button_label!CANCEL>;
<Get Quotes!quote>
{endregion};
%dlg%,<<%code%
if a_dlg_button="quote" then
  a_dlg_button=""
  doc=web.object.document
  doc.getElementById("txtQuotes")="ibm dell msft hpg stx symc nok vz mot t q"
  doc.getElementById("q").click()
end if
```

Once again, the bolded lines are new. Pressing the “Get Quotes” button on our Xdialog enters the symbols into the input field on Yahoo Finance and clicks the “Go” button on the web site. Yahoo Finance responds just as if a user were keyboarding and mouse-clicking through the site:

Symbol	Time	Trade	Change	% Chg	Volume	Intraday	Relate
IBM	Dec 29	97.15	↑ 0.18	↑ 0.19%	4,390,600		Chart , Messages
DELL	Dec 29	25.09	↓ 0.15	↓ 0.59%	13,942,030		Chart , Messages
MSFT	Dec 29	29.86	↓ 0.12	↓ 0.40%	41,757,262		Chart , Messages
HPQ	Dec 29	41.19	↓ 0.17	↓ 0.41%	6,200,700		Chart , Messages
STX	Dec 29	26.50	↑ 0.03	↑ 0.11%	2,350,100		Chart , Messages
SYMC	Dec 29	20.85	↓ 0.43	↓ 2.02%	7,395,948		Chart , Messages
NOK	Dec 29	20.32	↓ 0.17	↓ 0.83%	4,311,800		Chart , Messages
VZ	Dec 29	37.24	↓ 0.08	↓ 0.21%	9,488,500		Chart , Messages

This may be nifty, but what we want to do next is to *capture* the quotes so that you can put them in your Alpha Five database.

Where are the quotes? Well, the quotes themselves are HTML, and true to good web design, Yahoo Finance creates unique **ids** around each quote. Inspecting the page source now shows us segments like

```
<span id="yfs_l10_ibm">97.15</span>
```

So at this point you should be thinking it will be real simple to get a quote. We need to get the values insides the `..` HTML markers; that value is the **innerText** property of the **span**. Let's modify our Xdialog's **quote** event to return the list of stocks and quotes:

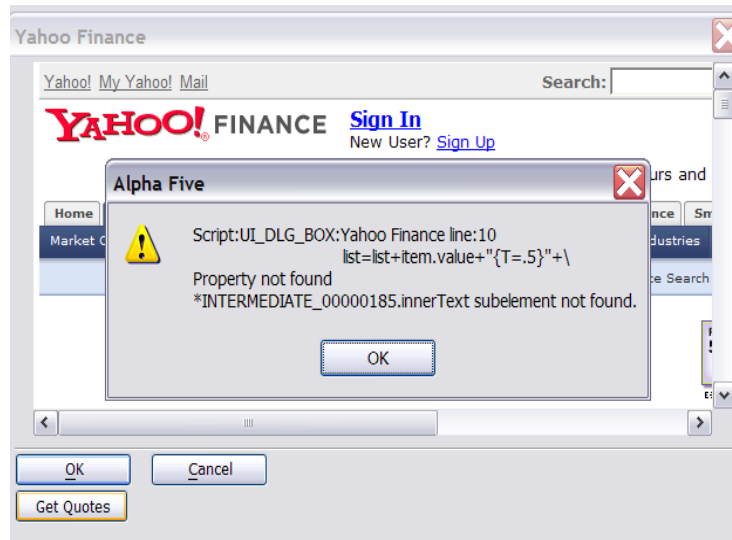
```
if a_dlg_button="quote" then
  a_dlg_button=""
  symbols="ibm dell msft hpq stx symc nok vz mot t q"
  doc=web.object.document
  doc.getElementById("txtQuotes")=symbols
  doc.getElementById("q").click()
  list=""
  symbols=stritran(symbols," ",crlf())
  for each item in symbols
    list=list+item.value+"{T=.5}"+\
      doc.getElementById("yfs_l10_"+trim(item.value)).innerText+\
      "{T=1}"+doc.getElementById("yfs_t10_"+trim(item.value)).innerText+crlf()
  next
```

```

ui_dlg_box("quotes", <<%dg%
[%O={@@}% .34, 10^#list];
%dg%)
end if

```

I really, really thought this would work, and you can only imagine my disappointment when an ungrateful Alpha Five spat this at me:



What's the matter? After much experimenting I realized that Alpha Five is trying to read the return values from the web page *before they are available*.

First "solution"

One solution is simply to delay Alpha Five by a fixed amount, giving Yahoo time to respond. A 3 to 5 second delay seems to work nearly all the time:

```

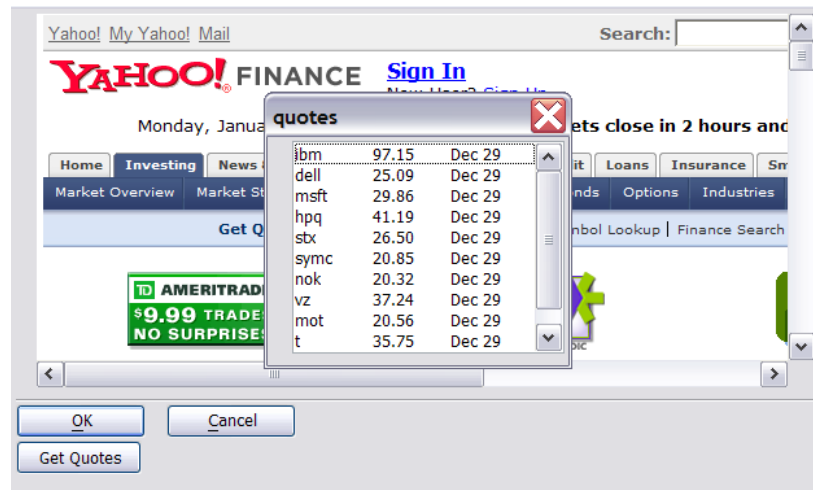
if a_dlg_button="quote" then
  a_dlg_button=""
  symbols="ibm dell msft hpq stx symc nok vz mot t q"
  doc=web.object.document
  doc.getElementById("txtQuotes")=symbols
  doc.getElementById("q").click()
  xbasic_wait_for_idle(5)
  list=""
  symbols=stirtran(symbols, " ",crlf())
  for each item in symbols
    list=list+item.value+"{T=.5}"+"\
    doc.getElementById("yfs_l10_"+trim(item.value)).innerText+"\
    "{T=1}"+doc.getElementById("yfs_t10_"+trim(item.value)).innerText+crlf()
  
```

```

next
ui_dlg_box("quotes",<<%dg%
[%O={@@}%.34,10^#list];
%dg%)
end if

```

Yup, this one works for me:



Although it worked, it clearly is not a satisfactory solution. First, it obligates a 5-second delay, even when it isn't necessary; but more significantly, if Yahoo Finance is very busy or the internet connection is very slow, then 5 seconds may not be enough. We need to know *exactly when* the document has been completely downloaded. Oops – isn't that the definition of an event?

The Web Browser's events

Remember how the genie generated a whole section of code for web browser events? Here's a snippet of the generated code:

```

dim web.events as c
web.events = <<%code%
function beforenavigate2 as v (pDisp as P,URL as A,Flags as A,TargetFrameName as
A,PostData as A,Headers as A,Cancel as I)
end function

function clienttohostwindow as v (CX as N,CY as N)
end function
(... 35 functions in all!)
%code%

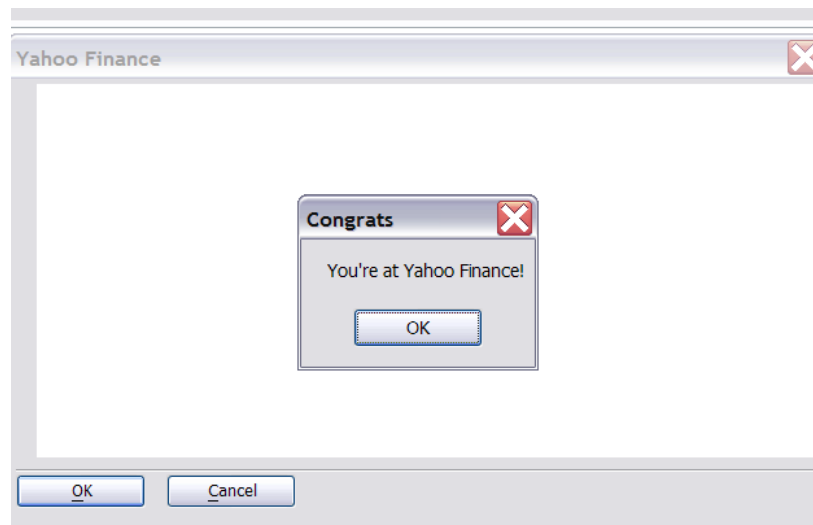
```

I have only the vaguest idea what some of these events do, but I can make a guess. Probably the

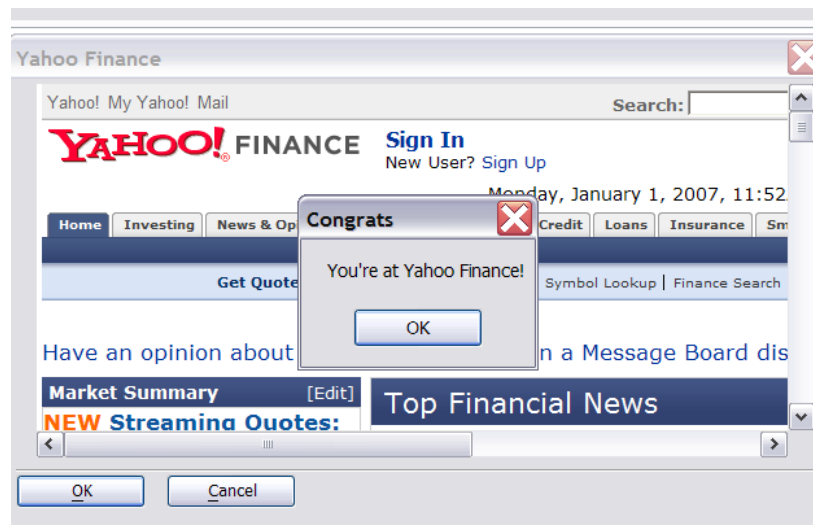
navigatecomplete2 event fires when the browser window has finished navigating to the specified URL. Let's try it, by putting something in the event:

```
function navigatecomplete2 as v (pDisp as P,URL as A)
    ui_msg_box("Congrats","You're at Yahoo Finance!")
end function
```

And if I run our Xdialog now, I get:



followed by:



Indeed, the message pops up four or five times. I don't know why **navigatecomplete2** keeps firing, but it's probably related to the way Yahoo Finance's web page keeps updating itself; the event only fires once for simple, static web pages like www.learnalpha.com.

The Web Events Function Anomaly

Now here's the part that I never would have gotten if not for the help of Selwyn Rabins (I wrote to Bill Gates at the same time, but Selwyn responded sooner). Change the `navigatecomplete2` code to this:

```
function navigatecomplete2 as v (pDisp as P,URL as A)
    ui_msg_box("Congrats","You're at "+web_url)
end function
```

By all rights *this code shouldn't run*. In Xbasic, functions cannot see local variables declared outside the function unless they are explicitly passed to the function, and `web_url` is a local variable defined in the main code, external to this function. *But Selwyn told me it does run*, and it runs because **the ActiveX object's event functions can “see” and manipulate local variables declared outside the function**. This transparency is strictly one-way: a local variable declared *within* the function remains invisible outside the function. I don't even have to supply the arguments to the function (`pDisp` and `URL` in `navigatecomplete2`); Alpha Five's interface to the ActiveX events provides the arguments for me.

Now that we have access to the ActiveX object's events, I can program for those events just like I can program for Xdialog events. I wasn't sure which event to use, but `documentcomplete()` seemed like a logical choice, and for once, I scored bingo on the first deal!

```
web.events=<<%code%
function documentcomplete as v (pDisp as p, url as A)
if "finance.yahoo.com/q"$url then ' this is how yahoo finance prefixes a ticker
symbol query
    showQuotes(web.object)
end if
end function
%code%
```

And the `showQuotes` function is defined as:

```
function showQuotes as v (webObj as p)
doc=webObj.document
list=""
for each item in symbols
    list=list+item.value+"{T=.5}"+\
    doc.getElementById("yfs_l10_"+trim(item.value)).innerText+\
    "{T=1}"+doc.getElementById("yfs_t10_"+trim(item.value)).innerText+crlf()
next
ui_dlg_box("quotes",<<%dlg%
[%O={@@}%34,10^#list];
%dlg%)
end function
```

There are a few more cosmetic changes in the script, which is available for download at www.learnalpha.com.

An Alternative: http_post_page2()

After I completed the first version of this article, Selwyn Rabins pointed out to me that **http_post_page2()** can be used to interact with web sites, without the overhead of displaying an ActiveX browser window. I could never get **http_post_page2()** to work, but after Selwyn sent me a working example, I discovered that I had made the classic “newbie” error of assuming the function documentation was correct. Yes, Virginia, **http_post_page2()** works, but it doesn't work if the optional 4th parameter, the **timeout**, is provided. Or at least it doesn't work in v7 and the beta version of v8 that I have; it may work by the time v8 is released.

The syntax of **http_post_page2()** is:

```
Page_Text as C = http_post_page2( URL as C [, Body as C [, Include_Headers as L [, Timeout as N ]])
```

We can use it to interrogate Yahoo Finance as follows:

```
dim url as c="http://finance.yahoo.com"
dim dlg_title as c="Yahoo Finance"
dim symbols as c="ibm,dell,msft,hpg,ctx,smc"
symbols=comma_to_crlf(symbols)
dim list as c=""

ui_modeless_dlg_box(dlg_title,<<dlg%
{font=Tahoma,10}
{units=f}
{can_exit=Close}
{region}
Enter symbols,;
\ (one on a line)\:;
[%MW%.20,6symbols];
<Get Quotes!Read>;
{endregion}|
{region}
Quotes:;
[%O={@@}%.34,10^#list];
{endregion};
{line=2,0};
{region}
<Close!Close>
{endregion};
```

```

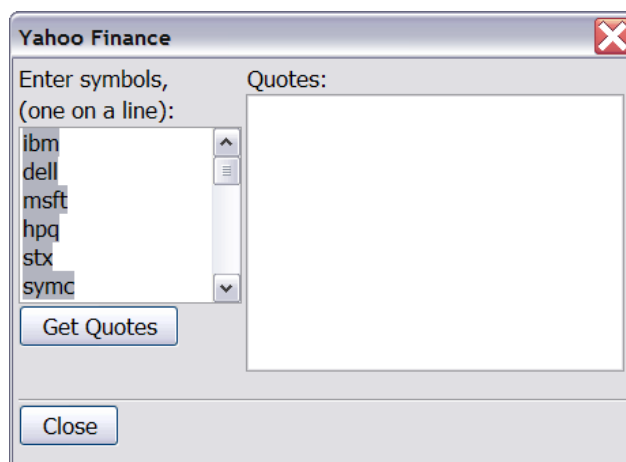
%dlg%,<<%code%
if a_dlg_button="Read" then
  flat_symbols=stritran(symbols,crlf(),"+")
  postBody="d=v1&s="+flat_symbols
  hourglass_cursor(.t.)
  yahooResult=http_post_page2("http://finance.yahoo.com/q/cq",postBody)
  hourglass_cursor(.f.)
  showQuotes(yahooResult,dlg_title, list, symbols)
end if
if a_dlg_button="Close" then
  ui_modeless_dlg_close(dlg_title)
end if
%code%)

function showQuotes as v (yahoo_result as c, dlg_title as c, byRef list as
c,symbols as c)
list=""
for each item in symbols
  startAmt="<span id=\"yfs_l10_\"+item.value+\">"
  endspan="</span>"
  startDate="<span id=\"yfs_t10_\"+item.value+\">"

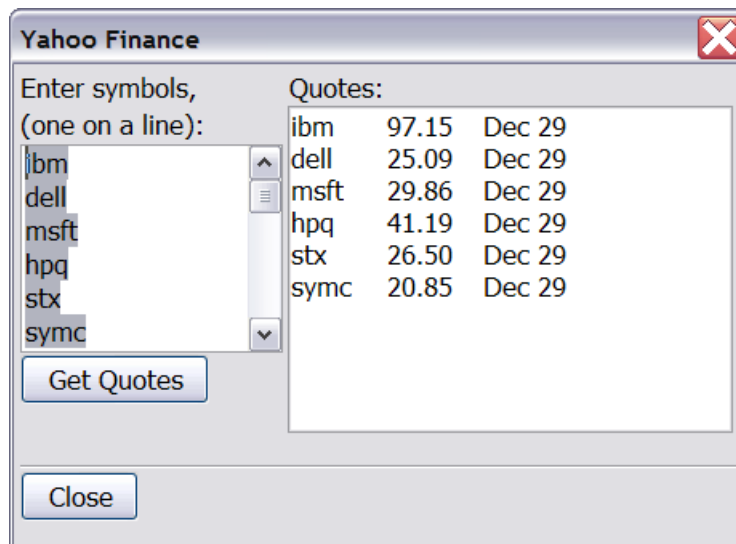
list=list+item.value+"{T=0.5}"+extract_string(yahoo_result,startAmt,endspan)+\
  "{T=1}"+extract_string(yahoo_result,startDate,endspan)+crlf()
next
end function

```

This script produces the following Xdialog:



Pressing the **Get Quotes** button gives us:



It's a lot faster than ActiveX, and it doesn't require programming for ActiveX events. It *does* require that you spend some time trying to figure out how queries are created on the web sites with which you want to interact, and sometimes that can be quite difficult; they're not all as transparent as the Yahoo site.

Recap and limitations

So what have we accomplished? We can use Alpha Five to navigate to a site, enter input and read data back. We can, if we desire, continue to send requests back to the remote server and read the returned HTML. Alternatively, we can use **http_post_page2()** on many sites to accomplish the same data transfer, without the overhead of the Web Browser ActiveX control. There is an inherent fragility in both approaches – if Yahoo Finance changes the **ids** they use for input or results, then we have to write a new script -- but that's the limitation of screen scraping and automated http transfer, not a limitation of Alpha Five.