Using PERL with HAL to Retrieve Web Data

Abstract: This document discusses the technique of using PERL to screen-scrape web pages and make the data thus obtained available to HAL. Includes a mini-primer on PERL and details on how to obtain and use it.

The Goal

HAL already integrates limited access to Internet data. HAL can tell you the weather forecast, sports scores, TV Listings, and traffic conditions.

However, these items all come from proprietary Internet sources, and are not adjustable by HAL users.

Many users have been asking for a long time to be able to pull other web data into a form HAL can use. This document details how to do this, in a way that anyone can use.

Speaking Text

HAL V.2.0 offers a new feature, one that has been much in demand by HAL users. A simple feature, but one that until now was missing. Simply the ability to invoke a plain-text file and have the Text-to-Speech engine in HAL read the file.

Until Version 2.0, the only way to have HAL read text is to program that text as an action item either for a rule or macro.

This opens the door to having HAL speak all sorts of data. Once we are able to do this, we will discuss ways to make that data available to other functions.

What is PERL?
PERL is an acronym for Practical Extraction and Report Language Invented in 1986 by Larry Wahl, it is primarily a language for processing text.

Running a Program

HAL has long had the ability to invoke an external program. However, Version 2 brings the ability to not only run a program, but to also include command line parameters. This makes it a little easier to invoke specific programs, but without this ability we could still perform the functions we need via DOS Batch files.

Downloading PERL
PERL sources and documentation exist at the CPAN, but for the Windows version, you can download the package at www.activestate.com. Select Languages | Downloads, then select ActivePerl | download. Select the MSI file for Windows. ActivePerl-5.6.1.635-MSWin32-x86.msi
It's currently an 8.6 MB download. Save the file in a safe spot then double-click to install.

What's a Screen Scraper?

Web pages are really just text, with some graphics to make it pretty. The text part is formatted in a structure called HTML. HTML essentially litters the text with tokens like <P> and <B> and so on, things which tell a browser how to display it. You can see what this looks like by going to a simple text web page (e.g. http://www.deco-group-partners.com) and selecting View|Source from your browser's menu.

A program does not have to be a browser to read web pages and make sense out of them. Any tool that can manipulate text in a structured manner can do so. A screen-scraper is merely a program that reads an HTML page, searches the text, manipulating the data we want, and ignoring the rest.

Tools to Operate

With the new capabilities in HAL it quickly becomes obvious that we can have HAL find and speak any data on the web. We just need a few more tools to bring it all together.

The first tool we need is something to manipulate the text and HTML tags on the web pages we are interested in. Most computer languages these days have extensive operators for processing text. Almost any language can meet this requirement.

The second tool we need is a Screen-Scraper. Ideally, our screen scraper can grab a web page, stuff it into an array or other data structure, and pass it over to our program to parse out the data we are interested in.

We can manipulate bare HTML data with any language, but highly desirable is a tool that can deal more directly with HTML tags, saving us from unnecessarily re-inventing the wheel.

PERL to the Rescue

PERL is an invisible wonder. It is so pervasive on the web as to be unseen, unnoticed, and unrecognized. It is available on almost any computer, and within some reasonable limits, code written in PERL is extremely portable from one computer to another, across laptops and mainframes, across Windows, UNIX and MAC. Most web servers on the Internet make extensive use of PERL.

This is because, among other things, PERL is free, open source software with no limits placed on it's use. It's also because PERL is extremely powerful. And finally it's because of something called CPAN, the Comprehensive Perl Archive Network.
CPAN provides for a global, organized access to all things PERL.
Beyond PERL itself, documentation and so forth, CPAN also provides a central archival and distribution point for PERL modules. PERL Modules are blocks of PERL code that are tested, debugged and encapsulated so as to hide their complexity. They are freely available, and easy to use. They allow novice PERL users to leverage the talent and expertise of more experienced programmers, and to easily build programs that contain far more elegance and complexity than they could build themselves.

Downloading Mechanize

Located on the CPAN site, Mechanize.PM is easily downloaded from
http://search.cpan.org/author/PETDANCE/WWW-Mechanize-0.33/lib/WWW/Mechanize.pm
It should be placed in C:\Perl\lib\WWW\ for access by your scripts.

Testing PERL

Once you've installed PERL on your computer, you might wish to test it before trying anything more complex. To do so, open a DOS CMD window and cd C:\Perl. At the C:> prompt type the following:

```
C:\Perl>copy con helloworld.pl
print "Howdy, world!\n";
^Z
        1 file(s) copied.

C:\Perl>perl helloworld.pl
Howdy, world!

C:\Perl>
```

If you get the shown response, Perl is working

Mechanize PERL Module

Two CPAN modules are of interest to our purpose.
Mechanize allows you to go to a URL and explore the site, following links by name, taking cookies, filling in forms and clicking "submit" buttons.

TokeParser is an HTML aware text parser. The initial examples in this document only use Mechanize, but for more complex tasks, TokeParser will become extremely valuable.

Mechanize takes a URL as a argument and returns the HTML text of the web page in a variable named {content}.
Once we have the contents of the web page in a variable within our program, extracting the data we want becomes a matter of simple text manipulation.

Getting Started With PERL

The first thing you must do is download and install PERL. You can, of course, do this from CPAN, but a company named ActiveState has created a Windows Installer package for the less technical users that automates what would otherwise be a tedious process. All you need do is download the file .msi and then double-click on it.

The installation will create a directory tree on your hard drive that contains the components of the program, starting with the top level C:\Perl directory. Note that immediately under C:\Perl is C:\Perl\lib where libraries are stored.

Anatomy of a PERL Screen-Scraper

We are going to build a simple screen scraper program. The first step is to declare the external libraries we are using. The USE statement tells the PERL interpreter to look in C:\Perl\lib\WWW\ for the file Mechanize.pm and import it.

Next, we create a local instance of the variables and data structures defined in Mechanize.pm, using the my statement.

```
#!
# External libraries
use WWW::Mechanize;

# Create local instances
my $agent = WWW::Mechanize->new();
```

Once we have the data structures set up, the next step is to open the file we intend to write our data out to. This is accomplished by the open statement. An item of note here is that if the open fails for any reason (file in use, disk full, whatever) the program will terminate with a message.

```
open(EX, "> Temperature.TXT ")
   || die "ERROR! Unknown problem opening TXT file.";
```

In three lines of code, we now have our screen-scraper initialized and our destination file open. Now we need only provide a URL and the program will go retrieve the HTML text.

The $agent-get function takes the URL as an argument and returns the data in an element named content.

```
# Go grab the web page into content
$agent->get("http://weather.yahoo.com/forecast/USCA0195_f.html");
# Change URL to match desired zip code
```

Regular Expressions

Regular expressions ("regex's" for short) are sets of symbols and syntactic elements used to match patterns of text.
Regular expressions figure into all kinds of text-manipulation tasks. Searching and search-and-replace are among the more common uses, but regular expressions can also be used to test for certain conditions in a text file or data stream.

The remaining lines of the program are regex and print statements writing the results of the regex to the output file. Actually, we only use two regex statements, Match (m) and Substitute (s). The first looks at the content variable and returns a string matching the match statement. First we look for a statement beginnint 'at:' and containing a space, one or two digits, a colon, two more digits, another space, two letters, another space and finally three letters. This contains the timestamp information.

We then look for the PST or PDT field and replace that with the words Pacific Time and Pacific Daylight Time.

Lastly, we grab the actual temperature reading and pad a few words around it and we're done.

```
# Find the word AT: and grab the timestamp following it
# match the string    'at: nn:nn a|pm PS|DT-->'
$agent->{content} =~ m/at:\s\d{1,2}:\d\d\s\S\S\s\S\S\S/;
$t = $&;                              # place matched data into a string

$t =~ s/://;                                   # remove the colon
$t =~ s/PST/Pacific Time/;                     # expand abbreviation
$t =~ s/PDT/Pacific Daylight Time/;            # expand abbreviation

print (EX $t);                        # 'at 9:54 pacific time'

# Skip ahead past the '22'
$agent->{content} =~ m/22/;

# Find the end of the line and grab the temperature
$agent->{content} =~ m/-->\n\d\d</;
$t = $&;                              # place matched data into a string

$t =~ s/\n//;                                  # Remove the newline
$t =~ s/-->/ the temperature was /;            # replace the arrow
chop($t);                                      # Chop off the back arrow.

print (EX $t);  print (EX " degrees, according to yahoo");
close(EX);
```

Using Temperature with HAL

Once the PERL program is complete, test run it from the command line as we did the helloworld program. Then type out the file Temperature.TXT. This should look like this:

```
C:\Perl>perl temperature.pl

C:\Perl>type temperature.txt
at 9:54 pm Pacific Time the temperature was 53 degrees,
according to yahoo
```

We can create several programs using Temperature.pl as a template, and retrieve all sorts of interesting weather data one item at a time. But making them available to HAL still requires some macros in HAL to operate them.

At it's simplest, what is required is a macro that will recognize a voice command, in this case, "What is the outside temperature?" and invoke the program. This is where the new HAL v. 2 feature of being able to provide arguments to external programs becomes helpful. We tell HAL to run the PERL interpreter located in C:\perl\bin\ and pass the argument C:\perl\temperature.pl to it.

MACRO Weather Temperature Yahoo {681} "What is the outside temperature?"
Run Program C:\perl\bin\perl.exe {C:\perl\temperature.pl}
TTS: "Let me ask my computer buddy over at yahoo weather"
In 10 seconds Speak Text File C:\Program files\HAL\temperature.txt
In five minutes Run Program C:\bin\notemperature.bat

HAL then responds:
at 9:54 pm Pacific Time the temperature was 53 degrees, according to yahoo

Unfortunately, HAL makes no provision to notify us when the program is complete. The time to access the web is variable and could be very short, or it could be fairly long, and we have no way to know when it is done. This causes us a problem, because we don't know when to read the file. Fortunately, waiting 10 seconds and then just blindly reading the file seems to work most of the time.

Is it live or is it Memorex?

The only other remaining problem is knowing whether the data is real or not. That is, since the data stays around in a text file long after it's spoken, we could, for example, fail to reach yahoo, and then read yesterday's file and not know it. There is an easy fix though. Simply create a file named noyahoo.txt. In it place a line of text something like "I'm sorry, but I can't reach yahoo right now".

What we want to have happen is that sometime after we read out the temperature, we copy our noyahoo.txt file onto temperature.txt, This way, if the temperature program fails, HAL will tell us he couldn't get the data.

Unfortunately, HAL does not allow us to execute MSDOS commands. So we have to create a batch file which we can then have HAL run as a program. This is very easy to do, and for those not familiar with the process, see the example. I should point out that I use a special directory to hold small utilities and batch files, called C:\bin. It's a good idea to keep these sorts of things together. Then add one more action to our Macro (shown in italics) to invoke the batch file in five minutes.

Creating a batch file to overwrite the temperature file

```
C:\>cd \bin
C:\BIN>copy con notemperature.bat
copy noyahoo.txt temperature.txt
^Z
        1 file(s) copied.

C:\BIN>
```

What did he say?

Why wait five minutes to change the file? Actually, for the purposes we've discussed herein, we could invoke the batch file immediately. However, that deprives us of a valuable option, the option to ask for a repeat. I find that when others are in the house, invariably just as HAL begins to speak, someone will try to interrupt. Or the dog will bark. Or the phone ring. Or the doorbell. Or SOMETHING!

In an earlier document[1], I described the process of enabling remember and repeat important announcements on command. I won't detail that process here, but rather refer the reader to the earlier document. Adding a flag, rule and a few more actions, HAL can easily repeat the reading on command.

[1] http://www.deco-group-partners.com/HAL-Announcements.pdf