A Not So Tangled `web`

One of the more useful utilities included in MKS Toolkit is the **`web`** command. With this single tool, you can retrieve a web page from a specified web server, put a web page on a server, post information to a form on a web page, and many other activities. In this article, we'll take a look at just three of these many uses.

URL WATCHING

I think I can safely say that all of us have web pages in our **Bookmarks** or **Favorites** folder that we visit regularly just to see if the content on that page has changed. For example, perhaps you regularly check the home page of you favorite news or sports site to see if there are any major stories. Wouldn't it be nice if you could convince the companies responsible for those pages to send you an e-mail telling you when their pages changed? To some extent, the growing popularity of RSS (Really Simple Syndication) does this and for those sites that do publish RSS feeds you can use the tips and tricks within the article *Creating a Simple RSS Reader* to gather that information. RSS is not yet mainstream though and by using **`web`** and other MKS Toolkit utilities, you can create a script that will read a web page, compare it to the last version it read, and send you that e-mail if things have been updated.

The technique is similar to the one I used for the RSS reader. However, as you can see in *Figure 1*, unlike the RSS reader which dealt with a single a web page (the RSS feed), our `urlwatch.ksh` script uses a text file (`url_watch_list.txt`) to store the list of URLS to keep an eye on.

```
1   for web_url in $(cat url_watch_list.txt)
2   do
3      new_page=new_${web_url//[\/:\\]/_}
4      old_page=old_${web_url//[\/:\\]/_}
5      web get $web_url $new_page
6      if ! cmp -s $new_page $old_page
7      then
8         echo "Web page: $web_url has changed." |
9         smtpmail -s "$web_url has changed" nobody@mks.com
10        mv $new_page $old_page
11     fi
12  done
```

Figure 1: The urlwatch.ksh Script

Using an MKS KornShell **`for`** loop, `urlwatch.ksh` cycles through each site specified in the `url_watch_list.txt` file. The script uses the shell's parameter substitution capabilities to replace each `:` or `/` in the URL with a _ character and then prefixing either `new_` or `old_` to create two file names based on the URL for the site. See *Figure* 1 lines 3 and 4. These file names are stored in the variables `new_page` and `old_page`, respectively, and are used to store the current and previous versions of the web page we are interested in.

In lines 5 and 6, the script uses the **`web`** utility to retrieve and store the current web page being processed and compares it to the previous version retrieved. If the two versions of the page are identical, here we use the **`cmp`** utility to check for any differences, nothing is done and `urlwatch.ksh` moves on to the next URL. If they differ, however, the script uses the MKS Toolkit **`smtpmail`** utility to send an e-mail message letting you know this. For a little more detail on this type of communication refer to lines 8 and 9 in the script. The **`echo`** command in line 8 provides the content for the body of the mail message while line 9 is the actual **`smtpmail`** command where we used `nobody@mks.com` to stand in for your own e-mail address. The **`-s`** option specifies the subject line for the message. You should note that you may need to specify additional options for **`smtpmail`** to work on your system.

Finally, in line 10, the current version of the web page is renamed so that it will be the version of the page that is compared the next time this site is checked.

There are two ways that we could make use of this script. First, we could simply run it manually every time we wanted to check to see if any of the sites listed in the `url_watch_list.txt` file have changed. While this is still an improvement over visiting each of the pages in your browser and glancing over each to see if there is anything new, we can do better. By using the MKS Toolkit Scheduler to run the script at regular intervals (for example, once an hour) and because the script itself handles e-mailing you when it finds a web page that changed, you do not need to worry about setting up the scheduler to redirect the script's output.

STUFFING THE BALLOT BOX

The web is full of forms for people to fill out. Whether it is a simple poll about their favorite type of pie, a newspaper collecting feedback from its readers, or a contest that can win you valuable prizes, web sites everywhere are constantly looking for your input. For most of these situations you can simply fill out the form and move on; however, in some cases such as contests that allow multiple entries, you might want to fill the form out over and over again to maximize your chances. I can think of few more tedious tasks than entering the same information repeatedly. Once again however, the **web** utility can come to our rescue with its ability to pass information directly to a form on a web page.

As an example of the sort of thing you might use this for, let's look at the web page for VTech's $1,000,000 Baseball Giveaway (http://www.vtechbaseballgiveaway.com/ddigital/enter.php). This is a contest where you can win prizes if one of your entries is drawn and a Toronto Blue Jays player hits a home run. To find out what information needs to be fed to the web page, you need to look at the HTML source for that page. Specifically, you are looking for the source for the form to be filled out. Within the `<form>` tag itself, you are looking for `<select>` and `<input>` tags with `name=` attributes. The `name` values given by these attributes identify the fields of the form to be filled out. Looking over the source for this page reveals that the form to be filled out has the following fields:

```
title                   city                    phone_business
name_first              province                phone_fax
name_last               postal_code             email
street_address          phone_home              promo
```

All of these are fairly obvious, with the possible exception of `promo` which is the name for the check box on the web page that states whether or not you want to receive additional promotional material from VTech.

For fields defined with `<input>` tags, any value can be posted. For fields defined with `<select>`, only one of the values specified with an `<option value="…">` tag can be posted. For example, you could post anything for the `city` field and have it be accepted whereas you must use on the one of the specified two-letter provincial abbreviations for the `province` field.

So, if I wanted to submit an entry for my friend, Noah Boddy, I could use this **web** command:

```
web get
"http://www.vtechbaseballgiveaway.com/ddigital/enter.php?title=Mr.&name_first=Noah&
name_last=Boddy&street_address=100%20Nowhere%20Lane&city=Nowheresville&province=ON&
postal_code=K6V%203J6&phone_home=613-555-1234&phone_business=613-555-
6789&email=nobody@mks.com" filled_in_page.htm
```

This command posts the information and returns the filled-in web page in a file named `filled_in_page.htm`. Note that I used `%20` in place of spaces and quoted the URL to ensure that it is not misinterpreted. As an alternative I could put all of this URL from the `?` to the end on a single line in a text file (let's call it `vetch_url.txt`) and instead issued the **web** command:

```
web post "http://www.vtechbaseballgiveaway.com/ddigital/enter.php" vtech_url.txt
```

in either case, this information passed along by **web** is entered into the contest database. And if I just wanted to submit a single entry for Noah, that would be that. However, since the contest allows each individual an entry every twenty-four hours, I can use the MKS Toolkit Scheduler to run either of these commands once per day and thus maximize Noah's chance to win a prize.

Now, of course, as I write this, the 2004 baseball season is drawing to a close, so the specific contest being used here is also coming to an end, but like many of the techniques described in these articles, you can apply what is discussed here to your own situation. In particular, this usage of **web** can be handy for testing web-based applications that relay upon form input.

LINK CHECKING

The final application of the **web** utility that we are going to look at is using the command in checking hyperlinks on a web site and reporting back any problems found. However, unlike the other uses discussed here, I won't be going into much detail on how to do so. Why is that? Because MKS already offers a superb, well-documented MKS KornShell script named `weblink.ksh` in the MKS Toolkit Resource Kit. The Resource Kit can be downloaded by licensed users of an MKS Toolkit product free of charge from http://www.mkssoftware.com/reskit and contains many other additional samples of using MKS Toolkit to create specific or general solutions.

But let's get back to `weblink.ksh`. This script, which was created by an MKS developer a few years back primarily uses the **web** utility not only to retrieve the web pages whose links are being checked but also as the means for checking whether a given link is valid. Of course, before those links can be checked, they need to be found within the page itself. To do so, `weblink.ksh` uses a combination of the MKS Toolkit **htsplit** and **awk** utilities to look for and extract the links needing to be checked. This is all fully explained in the comments found in the `weblink.ksh` script itself.

Finally, unlike many of my examples in these articles, `weblink.ksh` is designed to work like a fully functioning MKS Toolkit utility. It offers error checking and handling, command line option processing, and many other features. Thus, even if you are not interested in the specific task of link checking, it is worth taking a look at `weblink.ksh` and the other scripts in the MKS Toolkit Resource Kit just to see how such techniques can be handled.

CONCLUSIONS

In this article, we have looked at three general uses for the MKS Toolkit **web** utility. In truth, there are many more. Essentially, any time you have a task to accomplish which requires retrieving and sending information to a web site, **web** should be one of the first tools you look at in crafting your solution.

For more information about the MKS Toolkit products please visit http://www.mkssoftware.com and to view the full reference pages for the commands mentioned in this document visit http://www.mkssoftware.com/docs/cmd_index.asp.