Reading the Feed: Creating a Simple RSS Reader

In a previous article, *Creating Really Simple RSS From A Database*, I described how to use the utilities in MKS Toolkit to build a dynamic RSS feed that took its content from a simple database. In this article, we'll look at how we can use MKS Toolkit to build a simple RSS reader that can read that feed and many others. As discussed in that earlier article, there are two current styles of RSS feeds: 0.9x and 1.0. Fortunately, the structures of these two styles are similar enough that we can build a single reader to handle both of them.

BREAKING DOWN THE STRUCTURE

Before we can begin writing our reader, we need to understand the structures of both styles. *Figure 1* and *Figure 2* show the basic structures of RSS 0.9x and RSS 1.0 feeds, respectively. As you can see, both styles use many of the same tags in the same order. For example, both styles have `<title>`, `<link>`, and `<description>` tags within the `<channel>` tag to provide general feed information and within individual `<item>` tags to provide the same information about the items contained in the feed. In fact, the tags mentioned so far are the only the tags that we will make use of in our reader. The others that may be found can be ignored for the moment, as can the additional information provided in the `rdf:about` attributes in the RSS 1.0 feed.

```
<?xml version="1.0"?>
<rss version="0.92">
  <channel>
    <title>...</title>
    <link>...</link>
    <description>...</description>
    <item>
      <title>...</title>
      <link>...</link>
      <description>...</description>
    </item>
    ...
  </channel>
</rss>
```

Figure 1: Basic Structure of RSS 0.92 Feed

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF ...>
  <channel rdf:about="...">
    <title>...</title>
    <link>...</link>
    <description>...</description>
    <items>
      <rdf:Seq>
        <rdf:li resource="...">
        ...
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="...">
    <title>...</title>
    <link>...</link>
    <description>...</description>
  </item>
  ...
</rdf:RDF>
```

Figure 2: Basic Structure of RSS 1.0 Feed

Taking another look at these structures, you can see that we can simply ignore everything before the `<channel>` tag, then get the contents of the `<title>`, `<link>`, and `<description>` tags and display their contents as the feed's general information.

Once that is done, we can simply loop through each `<item>` tag and once again, obtain and display the contents of `<title>`, `<link>`, and `<description>` tags.

Those of you that are familiar with RSS and the file formats of the feeds are saying "What do you mean ignore all that information?" Believe me, I hear you. The purpose of this article, however, is to demonstrate the power and flexibility of the MKS Toolkit tools and utilities. There are many 'features' you could add to this simple script to enhance what I am going to show you and that will make use of the additional information that RSS v1.0 contains. By all means, do so, I would love to see what you all come up with*.

BUILDING THE SCRIPT

When it comes to writing the actual MKS KornShell script that is our RSS reader, there are a few other factors to consider.

* Comments are welcome, feel free to use the `mks.public.toolkit` public newsgroups to discuss your enhancements and other MKS Toolkit related items.

First, how is this script going to be used? Because RSS feeds are always changing, the best approach is probably to schedule it using the MKS Toolkit Scheduler to run every 10 minutes or so and let you know when the feed has changed (that is, there are new items). To do this, we'll need to keep a copy of the previous version of the feed for comparison.

Second, how are the contents of the RSS feed going to be displayed? Given that the information we're presenting includes links to Web material, let's have our script create an HTML file that can be used as output.

Third, how many feeds should the script access? To keep things simple, we'll limit the script to reading a single RSS feed from the MKS Web site; however, because a single feed reader is not terribly practical, we'll design our script so that it can be easily expanded to handle multiple feeds.

With these questions answered, we can move on to the actual script. The completed `rss_read.ksh` script is shown in *Figure 3*.

```
1   rm -f rss_feed.htm
2   RSS_URL=http://www.mkssoftware.com/rss.asp
3   new_rss_feed=new_${RSS_URL//[\/:\\]/_}
4   old_rss_feed=old_${RSS_URL//[\/:\\]/_}
5   web get $RSS_URL $new_rss_feed
6   if cmp -s $new_rss_feed $old_rss_feed
7   then
8      exit
9   fi
10   msgbox -fq "New RSS Items" "New items in $RSS_URL RSS feed."
11   htsplit -xc $new_rss_feed |
12      awk '
13         /<channel/ {
14            main_title=get_contents("title")
15            main_link=get_contents("link")
16            main_desc=get_contents("description")
17            print "<H1><A HREF=\"" main_link "\">" main_title "</A></H1>"
18            print "<P>" main_desc "</P>"
19            print "<BLOCKQUOTE>"
20         }
21         /<items>/,/<\/items>/ {
22            next
23            }
24         /<item/ {
25            title=get_contents("title")
26            link=get_contents("link")
27            desc=get_contents("description")
28            print "<H3><A HREF=\"" link "\">" title "</A></H3>"
29            print "<P>" desc "</P>"
30            getline
31            while ($0 !~ /<\/item/) {
32               getline
33            }
34         }
35         END {
36            print "</BLOCKQUOTE>"
37         }
38         function get_contents(tag) {
39            getline
40            while ($0 !~ "<"tag) {
41               getline
42            }
43            str=""
44            getline
45            while ($0 !~ "</"tag) {
46               str=str " " $0
47               getline
48            }
49            sub(/^ */,"",str)
50            return str
51         }' >> rss_feed.htm
52   mv $new_rss_feed $old_rss_feed
```

Figure 3: The rss_read.ksh Script

Getting The Feed

The first step in the script is to actually go out and get the RSS feed. To do so, we'll use the MKS Toolkit **web** utility. Once we have obtained the feed, we can compare it to the previous version and if there are no differences in the feed, we can exit quietly. Otherwise, the script will continue on to parse and display the feed. Lines 2-9 are the section of `rss_read.ksh` that accomplishes these tasks.

By using a variable to specify the URL of the RSS feed, the script can be easily modified to handle different feeds. Similarly, by using variables to identify the files which contain the old and new versions of the feed and basing the names of those files on the feed's URL, we set the stage for the script handling multiple feeds. Specifically, lines 3 and 4 create the file names for storing the old and new versions of the feed by taking the URL and, using shell variable substitution, replacing any `/`, `\`, or `:` characters with `_` and prefixing it with `old_` or `new_` as appropriate. This creates a unique pair of file names for any given URL.

Line 5 retrieves the RSS feed using **web** and stores it in the file identified by `$new_rss_feed`. This is the new version of the feed. Line 6 then compares this new version with the old version of the feed. The MKS Toolkit **cmp** utility performs the comparison and because the **-s** option is specified, returns nothing but an exit status indicating whether or not the two files are the same. If the two files are the same, the feed has not changed and there is nothing to do but exit the script.

If the two files are different (or there is no old version of the feed), the script assumes that the change is due to new information in the feed. Line 10 uses the **msgbox** command to announce the arrival of new feeds to the user. When you click **OK**, the script continues. This line can be commented out or modified if a less interactive method of user notification is desired.

Parsing the Feed

Now that we have retrieved the new RSS feed and informed the user that new items are available, we can parse the feed and generate the HTML code that displays the feed's information.

There are two key components to parsing the feed: the **htsplit** utility and an **awk** script. The **htsplit** utility, provided in the MKS Toolkit, breaks HTML input (or XML input when the **-x** option is specified) into tokens. Each token is either a tag (including attributes) or a single word. In our script, we feed the output generated by the **htsplit -x** command into an **awk** script.

The **awk** script is really the heart of our RSS reader. This script not only reads and processes the contents of the RSS feed but it also generates the HTML code to display it. As you may remember, in the *Breaking Down the Structure* section, we discussed a general approach to processing the RSS feed that works equally well for both RSS 0.9x and RSS 1.0 feeds. The **awk** script simply implements that approach.

Because much of the **awk** script's processing involves retrieving the contents of RSS tags, the key to the script's inner workings is the `get_contents()` function defined in lines 38-51. This function, when passed the name of a tag (for example, `link`) uses **getline** to read through the RSS feed a token at a time (because the input to **awk** is the output from **htsplit**) until it reaches a token which matches `<`*tag* where *tag* is the specified tag name. Once this opening tag is found, `get_contents()` continues reading tokens (and now concatenating them in the variable `str`) until a corresponding closing tag (`</`*tag*) is found and the concatenation ends. The variable `str` is returned as the contents of the specified tag.

For example, `get_contents(link)` reads through the feed until it finds a token matching `<link`. At which point, it begins accumulating and concatenating tokens until `</link` is found. The concatenated tokens are returned as the contents of the `link` tag.

As you may have noticed, we leave the closing > off many of the tag names throughout this script. We do this so that we can match a tag that is specified either with or without attributes. For example, item tags in RSS 0.9x are simply `<item>`, whereas in RSS 1.0, they are `<item rdf:about="…">`. Using `<item` matches both.

Now that we understand how this key function works, we can look at the rest of the **awk** script. Lines 13-20 handle what happens when the `<channel>` tag is encountered. When this happens, the script uses the `get_contents()` function to retrieve the values of the `<title>`, `<link>`, and `<description>` tags and, using the **awk print** statement, adds the HTML formatting for this general information about the feed. The `<BLOCKQUOTE>` tag is also added to the output so that individual items will be indented under the feed information.

The next part of the script (lines 21-23) basically ignores the `<items>` tag and anything within it. This tag only appears in RSS 1.0 feeds and does not provide us with any useful information for our reader. Also, by explicitly ignoring the `<items>` and `</items>` tags, it avoids any possible confusion with `<item>` and `</item>`.

Lines 24-33 handles the `<item>` tag. When the script encounters this tag, the processing is similar to that for the `<channel>` tag. It uses `get_contents()` to retrieve the contents of `<title>`, `<link>`, and `<description>`. It then uses print to add the appropriate HTML formatting to these values in the output.

Finally, the END section of the **awk** script simply adds the `</BLOCKQUOTE>` tag to the output to close the `<BLOCKQUOTE>` tag opened earlier.

Once the HTML output has been generated by the **awk** script, it is appended (using the >> redirection operator) to `rss_read.htm` file. As you may or may not have noticed, the `rss_read.htm` file is deleted in line 1. This ensures that the **awk** script will be appending output to an empty file. Why did we do it this way instead of using the > redirection

operator to replace an existing file rather than append to it? The reason is future expansion. If, in the future, we place a loop around lines 3-52 to handle multiple RSS feeds, we do not need to change how the output is handled.

Finally, once we have created the HTML file, we rename the `$new_rss_feed` file to `$old_rss_feed`, so that the next time the script is run, we can compare the newly received feed with the one we just processed to see if there are any new items.

You can now use your browser to view the `rss_read.htm` file and read the items in the feed, following the links provided if you want to read the full item.

So there you have it, one full pass through a given RSS feed. In order to make the best use of the `rss_read.ksh` script, can now use the MKS Toolkit Scheduler to have the script run every 10 minutes to make sure that you catch new items in the feed as soon as possible. You can read all about the MKS Toolkit Scheduler from the *MKS Toolkit Scheduling Solutions Guide* available from the **START** menu, MKS Toolkit -> Documentation group.

IS THAT ALL?

As it is currently set up, our RSS reader only reads a single RSS feed which we specified in a variable. In truth, though, it is rare that you'll only want to read a single feed. One way to extend the reader's usefulness is to let the user specify one or more URLs to read feeds from on the command line and then use a MKS KornShell **`for`** loop to process each URL individually:

```
for RSS_URL in $*
```

With this method, you would include the feeds to be read as part of the command line when you schedule the script.

As an alternative, you could use a file named `rss_feeds.lst` to specify the feeds to be read. Each line of this file would contain a URL. In this case, you would modify the loop to be similar to

```
for RSS_URL in `cat rss_feeds.lst`
```

Also, when handling multiple feeds, you should only use the comparison between new and old versions to determine whether to issue a notification of new items. All feeds should be processed completely to ensure that they are included.

You could also change the output produced by changing the HTML tags added to the output to other HTML tags or to something else altogether.

Try using the MKS Toolkit **`web`** utility to post the HTML file produced by this script to a Web site or perhaps use the **`mapimail`** or **`smtpmail`** tools to email the results to your inbox.

There are many different enhancements you could add to this script and hopefully many techniques used here that you can apply to your own scripting solutions.

CONCLUSIONS

So, there you have it: a very simple RSS reader that can be easily expanded to produce more sophisticated results. In addition, you have learned a few basic techniques about parsing XML files (which is what RSS feeds are) that you can apply to reading other XML files.

Feel free to comment on this or any other MKS article in the `mks.public.toolkit` public newsgroup. Suggest your own enhancements and contribute your scripts to the MKS Toolkit resource kit.

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.