FINAL PROJECT ISYS 540

By Jeff Baxter

4/12/10

EXECUTIVE SUMMARY

Circuit board manufacturers are paid to put capacitors, resistors, inductors and ICs onto circuit boards. Their revenue is tightly coupled to the quantity of parts that they can put on boards in an hour. The equipment they use to put parts on to boards is called placement equipment. The MyData brand of placement equipment is used by a local circuit board manufacturer (see image to the right) and this machine stores a record of each time it places a part on a board. The business owner wants a way to display how many mounts are happening on this machine each hour. He would like a chart that displays the output of the machine over time.

PROBLEM DETAILS

Here is a snippet of the data that is logged by the MyData machine:

-;16828136;14;AF 0016827949;16827949;15;AF 0016827721;16827721
>HTC;2279;1;H01;A Front;2;H01;A Front;3;H01;A Front;5;H01;A Front;6;H01;A Front
-;7;H01;A Front;8;H02;A Back
>M;774;0;537;265;7
>M;774;0;42;260

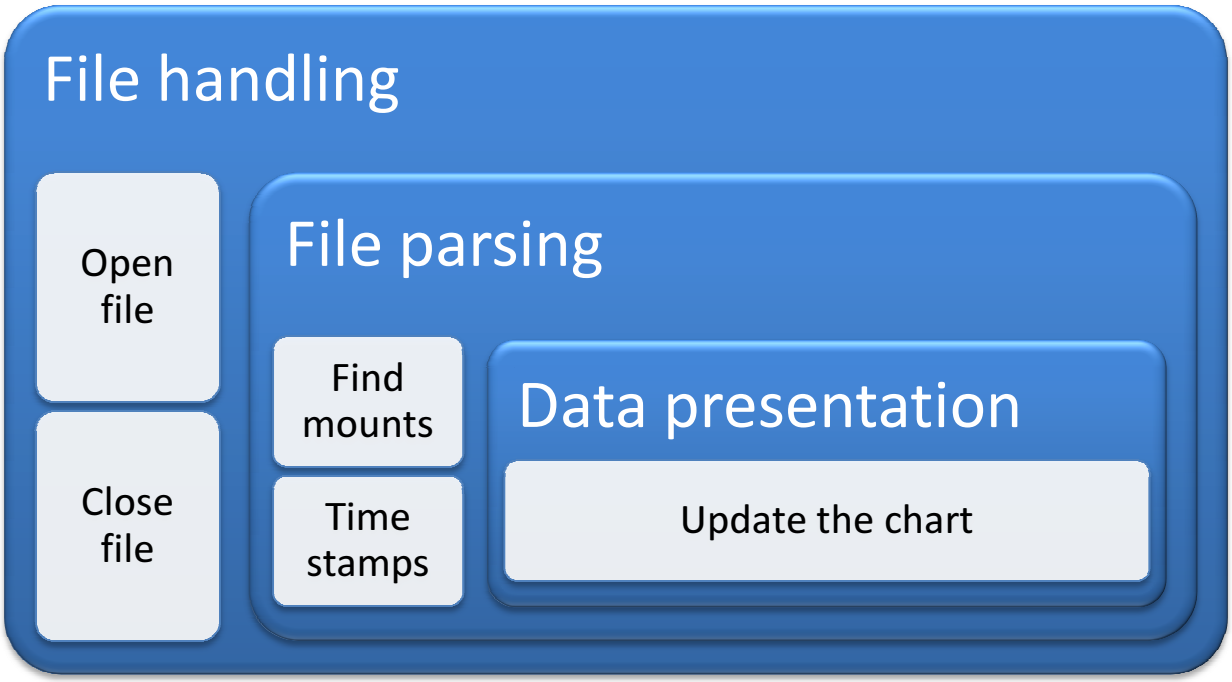
The problem is to open this data file and read in the “M” or mount lines and determine when they occurred (time is stored in unix time on this machine) and then make a chart of the output, with a count of how many mounts occurred in the last hour.

SOLUTION

The VBA script opens the data file in the directory that it is saved in and reads in every line in that file into a string. It then uses the findnext and other scripts we wrote in class to locate mount lines and their times. This order of operations is detailed in figure 1.

The first step is to open the file that contains all the placement data. This data always has the same name and internal formatting, because it is created by the placement equipment. That is why it is unnecessary to ask for the file name when the macro runs. It opens the file with a file handle

Figure 1 – Code layer summary

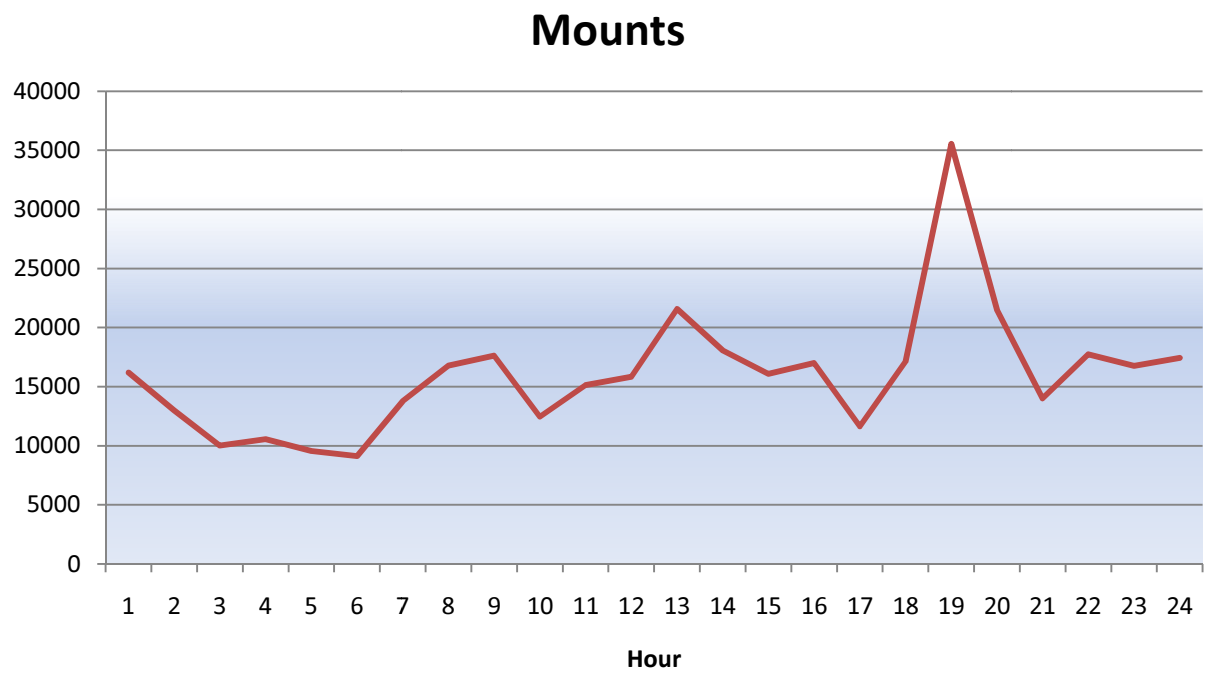


number and then reads in one line at a time into a very large string variable. This string variable can be very large because the files that we are reading in are large. For convenience, the indicator bar at the bottom of the screen updates what line we are on in the file, so that the user does not think that the software has hung during the long file import times.

Once the file is entirely inside a string I close the file and then begin the process of finding the mount events in the string. Here I used the findnext text parser that we wrote for parsing HTML to find the appropriate string of characters. I also instated several copies of the findnext module because I needed to look for more than the mount characters alone. I also needed to know when a mount occurred to see what hour "bucket" it fell into.

I would determine this using a two step method. First of all, I looked through the string to find an entire unix time stamp. Various other operations print an entire time stamp to the file, but not a mount operation. The mount operation only writes the increment of time it took to do the mount. So, I would look through the string looking for the next unix time stamp, and use it to update the running clock. Every time there was a mount operation I would add the duration to the clock to find out what time the mount occurred. For example, if we mount the first part at 1:00:00 pm and then the next fifty parts take 200 milliseconds each, then the clock should be at 1:00:01 pm. This is often not the case because of rounding error in the mount durations. That is why it is necessary to update the clock any time there is another full Unix time stamp.

Once we have determined when a mount occurred, then the hour it falls in gets its mount count incremented by one. This mount count is the data that the chart displays, in mounts per hour. This is a copy of that chart:



The chart above is a summary of the placements that occurred over a month-long period created by running this piece of software. This allows the owners of the business to understand which shifts are producing the most placements and make business decisions based on that output.

CHALLENGES

I ran into some challenges with the unix time converter. I struggled to get the time zone right and also to convert unix time appropriately. I believe that I have all of those issues worked out and that the data truly corresponds to the correct hour.

This project was helpful to me to get a real chance to use some of the nifty parsing tools that we wrote in class. The only down side with this method was that it took over night to run on the large data file that I had. I think the data file had hundreds of thousands of lines in it, and this caused the software to take forever to import. I wish there was a way to import an entire file directly into a string. That would have greatly streamlined the whole thing.