Scripting an Internet Explorer window

devblogs.microsoft.com/oldnewthing/20141020-00

October 20, 2014

Raymond Chen

Today's Little Program takes a random walk through MSDN by starting at the `Create-Process` page and randomly clicking links. The exercise is not as important as the technique it demonstrates.

```
function randomwalk(ie, steps) {
 for (var count = 0; count < steps; count++) {
  WScript.StdOut.WriteLine(ie.document.title);
  var links = ie.document.querySelectorAll("#mainSection a");
  do {
   var randomLink = links[Math.floor(Math.random() * links.length)];
  } while (randomLink.protocol != "http:");
  WScript.StdOut.WriteLine("Clicking on " + randomLink.innerText);
  randomLink.click();
  while (ie.busy) WScript.Sleep(100);
 }
}
```

(I'm assuming the reader can figure out what language this script is written in. If you have to ask, then you probably won't understand this article at all. I am also not concerned with random number bias because Little Program.)

To talk a random walk through MSDN, we ask for all the links in the `mainSection` element. Note that I'm taking an undocumented dependency on the structure of MSDN pages. This structure has changed in the past, so *be aware that the script may stop working at any time if the MSDN folks choose to reorganize their pages*. I'm not too worried since this is a demonstration, not production code. In real life, you are probably going to script a Web page that your team designed (as part of automated testing), so taking a dependency on the DOM is something the QA team can negotiate with the development team. (If your real life scenario really is walking through the MSDN content, then you should use the MSDN content API. Here's sample code.)

Anyway, we grab a link at random, but throw away anything that is not an http: link. This avoids us accidentally navigating into a mailto: link, for example.

We then invoke the `click()` method on the link to simulate the user clicking on it. We could also have just navigated to `randomLink.href`, but I'm using the `click()` method because it is more general. Your script may want to tick some checkboxes and then click the Submit button, and those actions can't be performed by navigation.

We then wait for the Web page to settle down. I'm lazy and am simply using a polling loop. If you want to be clever, you could listen on the `onreadystatechange` event, but this is just a Little Program, so I'm content to just poll.

Once we have settled on the new page, we loop back and do it again.

Now we just need to drive this helper function.

```
var ie = new ActiveXObject("InternetExplorer.Application");
ie.visible = true;
ie.navigate("http://msdn.microsoft.com/ms682425");
// Wait for it to load
while (ie.busy) WScript.Sleep(100);
randomwalk(ie, 10);
ie.Quit();
```

We create our own instance of Internet Explorer so we can change its carpet without getting anybody upset, navigate it to the `CreateProcess` page, and wait for the page to load. We then use our `randomwalk` function to click on ten successive links, and then when we're done, we bring in the demolition crew to destroy the browser we created.

For extra evil, you could commandeer an existing Internet Explorer window rather than creating your own. (Now you're barging into somebody's house and rearranging the furniture.)

```
var shellWindows = new ActiveXObject("Shell.Application").Windows();
for (var i = 0; i < shellWindows.Count; i++) {
 var w = shellWindows.Item(i);
 if (w.name == "Windows Internet Explorer") {
  randomwalk(w, 10);
  break;
 }
}
```

Making the appropriate changes to `randomwalk` so as not to be MSDN-specific is left as an exercise.

Raymond Chen

Follow