

webDOMinator^{1.0}

USER GUIDE / REFERENCE

COPYRIGHT (C) 2007 NATHANIEL D. GIBSON

Table of contents

webDOM Overview	1
webDOM Basics	3
Explaining The Interface	
Simple View	
Advanced View	
How it Works	
Step-by-Step – Dominating your First Web Site	
IMPORTANT: Securing webDOM from viruses	
Advanced Usage	
Automation	
Information List Auto-Population	
Page Action Scripts for Community Profiles	
Building a webDOM Site Profile	
Quick Action Scripting	
Script Scheduling	
siteBore – the built in webDOM spider	
siteBore Overview	
Interface	
Advanced Usage of siteBore	
Minion Code Reference / Examples	
Basic Rules & Operations	
Variables	
Comparators	
Operators	
Structures	
Functions	
Creative Use of webDOM	
Appendix A – <i>Glossary</i>	
Appendix B – <i>Minion Function Reference</i>	
Index	

webDOMinator 1.0 Overview

webDOMinator, (or webDOM for short,) is a robust web-based desktop application that offers task automation on any internet user interface available over a web browser. Not only does it offer built-in automation of certain tasks specific to social networking sites (like myspace and facebook) but it goes even further to include it's own simple scripting language and a powerful spidering function, allowing you to automate just about any task you can think of. This expandability of function through scripting and spidering allows webDOM to have very strong web-marketing implications and applications. You can literally expand your market reach by thousands of people in your target market each day. Some good other uses of webDOM might be: Data Mining, Direct and Passive Marketing, Social Network Profile Management, Information consolidation and management, Automated Stock Trading, etc. webDOM can even simulate interaction with web 2.0 pages that use AJAX to hide their code. This simulation allows you to get past most of the “security” that web pages have built in against automated programs of it's kind.

webDOM uses the built-in DOM (Document Object Model) of a browser object and gives you direct control over the contents and interaction with a website you're viewing through webDOM. You can programatically change the property of any HTML element on the active page, delete or add new elements, even simulate user clicks on clickable elements like buttons and links. webDOM's creative use of the DOM interface allows for unlimited manipulation of web pages in a super-scripting environment. To find out more about webDOM's extensive scripting and control, go to the *Advanced Usage* section of this guide.

This guide is broken up into three major sections which include a User Manual for new users of webDOM, a set of code Examples that showcase some of the Minion code functions, and an extensive reference of the Functionalities of both the webDOM program itself and the Minion scripting language. If you are interested in finding a community about webDOM scripting and more scripting examples, please go to webDOM.com/community and sign in using the login information that came with your registration e-mail.

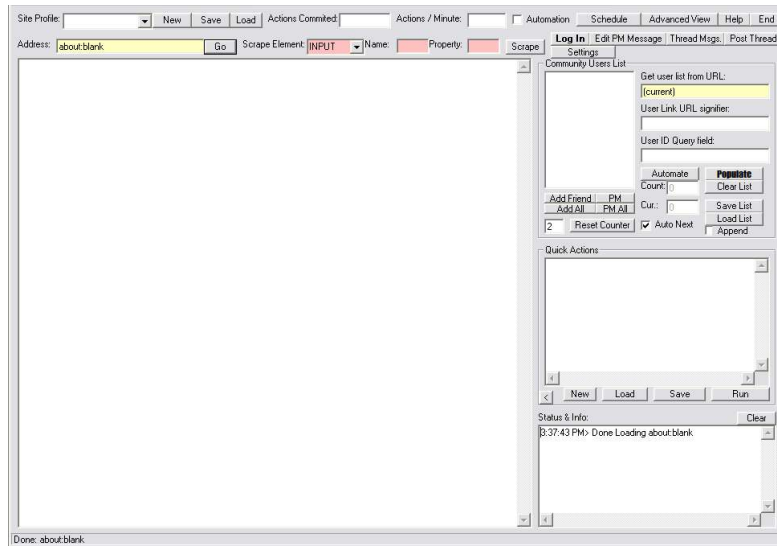
NOTE: All words referenced in the Glossary appear in the main text as **bold** words.

webDOM Basics

Explaining the Interface

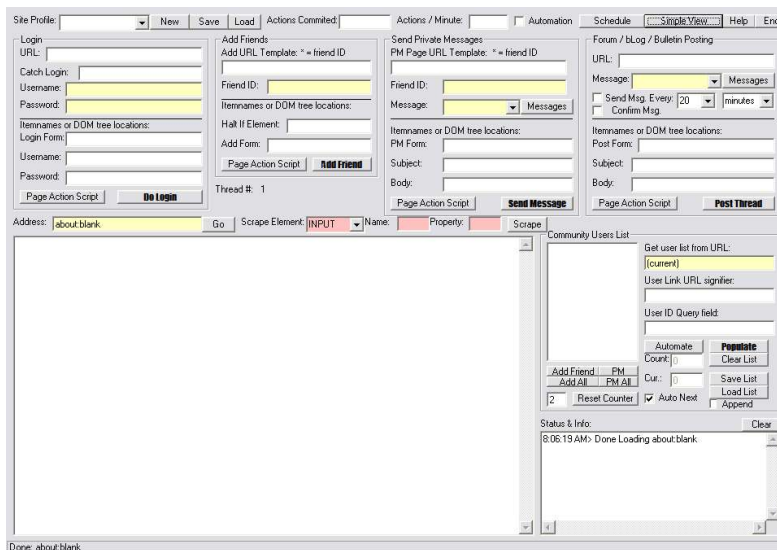
webDOM can tend to be a bit daunting to look at when you first open up your copy. Most applications currently have a less busy looking interface, but you'll soon find that what's provided in webDOM's **simple view** is just enough for you. Most all of the things you can do in webDOM are visually accessible as soon as you open a new **thread**.

Threads are one instance of a webDOM form within the main webDOMinator application. Threads have two different views, **advanced view** and simple view. Simple view looks like this:



Simple view can be broken up into 5 major sections: The browser area, the **Community Users List**, the **Quick Actions** scripting area, the **Status & Info** Section, and the **Profile** loading & Automation area on top.

Let's take a look at the **Advanced View**:



Advanced View is the same as simple view but it includes a forms for you to quickly program the major actions above the browser and community users list. There is a section for each major action one can do on a normal member based website: Log In, Add Friends, Send Private Messages, and Post Threads to any sort of bulletin system. In advanced view, Quick Action Scripting is not accessible.

As previously stated, looking at either of these views can be daunting at first to one who's never opened webDOMinator, but you'll quickly find that all of the pieces are a necessary part of your internet automation toolbox. We will now look further into the previous sections mentioned in the captions above.

The Browser Area: This is simply a web browser that uses a version of internet explorer for it's base. There is an address bar, and a Scrape tool which will be explained later in the *Scraping* part of this section. Notice the Browser does not have the standard buttons a normal browser has, like back, forward, refresh, and such. This is because you will usually be doing these types of things programmatically rather than manually. If you wish to do these actions manually, the good old backspace, shift-backspace, and F5 buttons will do the trick when you are focused on the browser section.

The Community Users List: This acts as a list of User ID's which gathers user ID's for taking further action on. The way it works, is it goes through the active browser page, or whatever page you tell it to use and, based off of what it's programmed to find, finds all of the User ID's linked to on the page. The User ID Link signifier tells the system what to look for in a link that would carry a user ID in it. Then the Query field signifier tells the system how to extract that ID from the link. Once those two fields are filled in properly, all you have to do is go to a page with a list of users on it, and then click the "Populate" button.

The Quick Actions Section: There is a whole chapter of this guide committed to scripting using Minion code. The quick action section is a live testing ground for a Minion script. One can save, load, and run the scripts written in this section.

The Status & Info Section: This section outputs status and information including: what page was loaded when, debug information for the Minion scripts, and **scrape** information output from the Scraping tool.

The Profile Loading & Automation Section: This section serves as a place to load your webDOM profiles. A profile is a copy of the automation settings given for any specific website. Profiles can be linked specifically to real online profiles, like a myspace profile for instance. This allows for automated logins, etc. Overall Profiles can be for any website, whether it has a login or not. With profiles, you can create a new one, save, or load. The automation piece of this section refers to the four buttons, (which in Advanced View turn into four sections,) that make up Logging in, sending messages, adding friends, and posting bulletin board or blog posts. The other piece of the automation section that is present in both views is the Automation Checkbox, and the Schedule button. These give you the ability to create a Automation script schedule for each thread loaded in webDOM. More of the automation section programming and scheduling will be explained later on in this text.

How webDOM Works

webDOM is a multi-faceted application based off of a simple idea. Simulate real user interaction with a webpage using automated processes. If the webpage doesn't know that a robot is viewing it, then it can't do anything about said robot like denying access. The

structure of webDOM's thread forms are also a key piece in this automation process. With a single form, you can perform any actions you need automatically as well as scrape information from a website to use in programming webDOM to use a different social networking site. For example: If you come across some new social networking site, you simply open up webDOM's advanced view in a new thread, fill out the four programming forms that drop-down, and then start populating your user list of your target market. To read more about how to create a profile, see the ***Building a Website DOM Profile*** section.

WebDOM allows you to store as many profiles for websites as you want, along with your username and password so that later, you can schedule things like adding friends or sending messages. You can either make new profiles yourself or download them from webDOMinator website.

Currently webDOM does not perform CAPTCHA (security code images), but if programmed, the automation can allow you to bypass almost all of the other “anti-robot security” featured on some of the largest social networking websites.

Enough technical talk, let's see in a working step-by-step example, how webDOM can be used to perform some of the simple tasks the program is capable of.

Step-by-Step: Dominating your first web site

1. The first thing that you're going to need is a user account on the website that you want to dominate. Go to the website and set up your account. *If you wish to do these steps right now, make sure it's a website that you already have a **webDOM site profile** for. (a webDOM profile is not the same thing as your user profile on your target website.)*
2. Open the webDOMinator application
3. Click the “Load” button and browse to the profile file for the site you wish to dominate
4. You'll need to enter your user information for the first time and save your profile again. You can do this by clicking the “Settings” button located underneath the “Log In” button on the top right of the webDOM form, and then enter your username and password and clicking save.
5. Click “Log In” and webDOM should automatically log in for you.
6. After you've logged in, use the browser to browse to a page that has users you wish to add as friends or send messages to.
7. Click the “Populate” button in the “Information List” and you should see user ID's or user names show up in the list box.
8. Continue to browse to different pages you want users from and click Populate until you have a list of a decent size. *There is a way to automate this process, which you can read about in the **Information List Auto-Population** section of this guide.*
9. If you want to use this list again, you can save it by clicking the “Save List” button and choosing a file name then clicking “Save”.
10. Once you're ready to use the list, click the “Add All” button to automatically add everyone on the list as a friend. To stop the adding process, just click the “Add All” button again.
11. If you want to send Private Messages to the people on your list, click on the “PM

Message” button on the top right of the webDOM form, and enter in a subject and some message text, and then click “PM All” in the Information List section.

Important: Securing your computer against viruses

webDOMinator uses internet explorer as the browser element it controls. While webDOM is in use, it's best to set internet explorer to either A. Not show photos, because they are one way in which people can insert viruses into your computer... or B. Secure the ActiveX controls to only run scripts that are signed with Authenticode, or turn ActiveX off altogether if you don't need it. You can do this by opening up Microsoft Internet Explorer and then selecting “Tools” > “Options” and then going to the security tab, or advanced options tab.

Advanced Usage

Information List Auto-Population

It may seem that it's quite a task to create a large list of user ID's for a web site, considering that some web sites only show 10 users per page, or even 30. This is why Automation was added to Populating the information list. With auto-population, it's possible to automatically add thousands of people to your information list by entering in some simple pieces of information. All you have to do is tell webDOM how to navigate to the next pages it needs for populating more users into the list, and automation will do the rest of the work for you.

Step By Step on how to start Auto-Population

Building a webDOM Profile for a Site

1. First, find the site that you wish to build a webDOM profile for. It's good to try to find sites that are smaller because they usually don't implement CAPTCHA codes. Try to make the site you find fit your target market or include your target market.
2. Create an account on that website first. You can't use webDOM with that site unless you have a user account. If the site does not include user accounts, but still includes a way for you to post comments or leave messages, don't worry about this step.
3. After you have your website, and you have your account there, open up webDOM.
4. In the new form that appears, first enter in a handle for that site profile... for instance the handle for myspace.com is just "myspace".
5. Click the "Advanced View" button on the top right of the form. This will show you the programmable sections for performing each action.
6. Program webDOM to use the Log In of that site by filling out the Log in section.
 1. Use the Address bar on top of the browser to browse to the website's log in page, and also enter it in the text box that reads "URL" under the log in section.
 2. Enter the username and password you have set up for your profile on that website.
 3. **Scrape** the page for "FORM" elements, and enter in either the name of the login form element, or it's DOM location (i.e: DOM.FORM.1), if the login form does not have a name property.
 4. **Scrape** the page for "INPUT" elements, and then find the name or DOM location of the individual input elements for both the username and password text boxes.
 5. Click the "Do Login" bolded button at the bottom of the login section to see if your automated login works. If not, you may need to add a **Page Action Script**.
7. Program webDOM to automatically populate User ID's by filling in the two important fields in the "Information List" section.
 1. Navigate to a page that has users listed using the browser.
 2. Scrape all "A" elements from the page and scroll through the list of links to find one that links to a user profile.
 3. Using that link, read through the link URL and find the place where you can see the user ID. If it's a part of the query part of the URL (i.e: <http://profile.myspace.com/viewprofile?friendid=64583>) then you would use "friendid" and enter it into the "Link sub-field" text box in the information list

section. If it's not part of the query, but listed as if it were a page or a directory (i.e: <http://www.tagged.com/users/webDOMmaster>) then you would use enter in “users” into the “Link sub-field” text box.

4. Now you have to tell webDOM what it should look for in the links on a page to find the link that has your user ID's in it... for instance, if the user profile URL had “profile.myspace.com” in it, and no other links besides user profiles have that, you enter that into the “User Link URL signifier” text box.
8. At this point, you should save your profile. Just click “Save” on the top left of the form, located next to the “Load” button. This will open up a save box, just enter in a name for the file that makes sense, like if you're making a profile for “froppe.com”, enter in “froppe” and click save.
9. Now you are halfway done. The next thing to do is set up the ability to add friends using the “Add Friends” section of the webDOM form.
 - 1.

Scraping:

In order to build a webDOM profile, it's important for you to understand the scrape tools which you'll need to use to gather information about each part of the profile and which elements to reference for which actions, etc. This sounds complicated, but is actually quite easy. Your

To scrape a web page...

Minion Code Reference

Minion coding has not only a simple way to access and reference DOM (Document Object Model) elements and webDOM form information, but also contains some very useful functions for interacting with these elements. Before you get started using the Minion coding language, you should already know something about the structure of HTML documents and at least know what the Document Object Model is. The most important thing you need to know after you know those two things is how to **scrape** lists of elements off of the page you're viewing through the webDOM browser.

Document Object Model elements can be accessed using the following code:

```
dom.(element type).(element identifier).(element property)
```

Where element type is the type of element like “input” for text boxes or “a” for links, element identifier is either the name of that element or the sequential numeric ID representing that element, and element property is the property to be accessed like “value” or “id” or “innerHTML”.

Much like the javascript coding language, you can use a series of “.” period symbols to access more precise data in the structure of the web page. Unlike javascript however, webDOM sees the DOM structure as directly accessible through element names, (or other identifiers depending on the element), as well as a sequential number in a list of the element type you're trying to access. This gives you the ability to access elements that are not identified or named another way.

The elements you can access are: a, input, form, textarea, div, td, table, select, option, and basically any standard HTML element you want to access.

The properties of these elements are much like

Example 1.

Accessing the DOM structure using element names

```
dom.input.q.value
```

this would access the value property of the input element named “q”

Example 2.

Accessing the DOM structure using the element type and its numeric list value.

```
dom.input.3.value
```

if “q” were the third input element in the DOM structure, you would be accessing the same exact property of the same element.

Example 3.

Setting a property of a DOM element.

```
dom.input.q.value=dubstep dj
```

When you run this piece of code on the proper web page, (A lot of websites use “q” as their search input name), you will actually see this change reflected on the page... in other words, you'll see the search text box have the text “dubstep dj” in it.

Example 4.

Setting a DOM element property equal to the value of another element property on the same page.

```
dom.input.q.value=dom.a.1.innerHTML
```

Let's say in this example that the first link on the page was a link containing the words “foo bar”, then this script would make our search text box say “foo bar” because the innerhtml property of the link holds this.

DOM interaction functions:

remove – remove a DOM element. Usage – remove:(dom element)

example: remove:dom.textarea.1 – removes first textarea in the page.

Remove can come in handy for certain situations where there is an ajax element you don't want to load, or you want to change the structure of a form to be submitted.

click – Clicks on a DOM element. Usage – click:(dom element)

example: click:dom.a.Add Friend – clicks on the A element with the innerhtml value of “Add Friend”

Click will perform a click action as if you were actually clicking on some element. It currently works with all A and INPUT elements, so you can click on a button, a text box, and any link on the page.

focus – Focuses on a DOM element. Usage – focus:(any dom element)

type – Types a phrase using the keyboard input directly. Usage – type:(phrase)

submit – Submits a DOM Form element. Usage – submit:(dom form element)

example: submit:dom.form.1 – submits the first form on the page.

navigate – Navigates the browser object to a new address. Usage – navigate:(URL)

File I/O and shell:

open – Opens a File. Usage – open:(access mode):(file handle):(file path)

example: open:w:foo;c:\foo.txt

write – Writes to an opened file. Usage – write:(file handle):(information)

example: write:foo:Hello World – writes “Hello World” to the file with the handle “foo”

read – Reads one line from an opened file. Usage – read:(file handle):(read variable)

example: read:foo:var.line

Read variables can be a var, a global, or even a DOM element property. The file handle is whatever file handle you used to open the file you're reading from.

close – closes a file handle. Usage – close:(file handle)

scrapetofile – Scrapes the page information listed in the scrape box for the current thread into a file as a list. Usage – scrape:(file path)

shell – Run a regular shell command . Usage – shell:(shell command)

example: shell:iexplore ^dom.a.34.href – run the internet explorer with the argument of the url from the 34th link on the current page.

Shell will automatically run commands as if you typed them into the command prompt

shell32 – Run a 32 bit shell command. Usage – shell32:(file name)

example:

shell32:http://www.google.com/?q=^dom.a.3.innerhtml – open the file or URL shown in this example... in this case google.

Shell32 only needs a file name and it will use the windows default program to open that file if it's located on your computer or even if it's an external URL.

webDOM based actions:

nosubmit – This is mainly useful on page action scripts where you are adding a friend or sending a private message. It's meant to be used whenever you don't want a form to auto-submit, like an add friend form or a send message form. It allows you to perform other actions and even submit the form later if you want.

refresh – Makes the web browser refresh the page it's currently on.

back – Makes the web browser go backwards in it's history.

forward – Makes the web browser go forward in it's history.

wait – Waits for (n) seconds. Usage: wait:(n seconds)

pagewait – Waits for the page to finish loading before performing the next command

login – Performs the webDOM programmed log in action for whatever profile is loaded

add_all – Starts the Add All automation for the current list of information or users.

pm_all – Starts the Private Message All automation for the current list of info or users.

end – Ends the current running script.

populate – Populates the information list with the information from the current web page that the profile you're using is programmed to grab.

clearidlist – Clears the Information list.

loadprofile – Loads a profile into the current working Thread.

Debug and Status output:

cleardebug – Clears the debug output.

debug – Makes all of the scripts that run output debug information, for when you're trying to debug your scripts.

echo – Echo's the information requested. Usage – echo:(information)

example: echo:dom.a.4.innerhtml – echoes the html inside of the fourth link on the page you're currently on.

Variables:

var

global

Operators structures and comparators:

loop-endlop

if-else-endif

String Concatenation:

To concatenate a string, you must use the character “^” as your concatenation element.

String concatenation in minion coding is not like