



# The Magic Of Automating With Scripting Languages

Photograph by Ethan Myerson

**T**here's something magical about test automation, and scripting languages make you the magician.

By Dion Johnson

*"I don't want the public to see the world they live in while they're in the Park (Disneyland). I want them to feel they're in another world."*—Walt Disney

Automation using scripting languages is not extremely different from

Dion Johnson is an independent consultant with more than 10 years of experience in QA, QC, requirements analysis and process improvement. You can contact him at [dionjohnson@dijohn-ic.com](mailto:dionjohnson@dijohn-ic.com).

automation using commercially available tools, but at the same time, the two approaches are in two different worlds. The root of the divide became apparent to me in late 2004 when I had the privilege of delivering a presentation to a testing industry conference at the Disneyland Hotel in Anaheim, Calif. That evening, following my talk on the topic of test automation, I reflected, along with a colleague, on the relatively large

crowds that test automation topics always seem to draw. We hypothesized that people's fascination with test automation is directly related to the sphere of mysticism that accompanies commercial automated test tools, which is why it was so fitting that the conference was held at the Disneyland Hotel. These tools have cornered the market on automation and are often viewed as magic wands which, when waved, will produce resource-saving automation magic...so buy the license, at a very hefty fee! Maybe automation is magic, but through the use of scripting languages, you can enter another world that allows you to become the magician who creates that magic.

Scripting languages are embedded

in many popular, commercially available automated test tools, but they are also available for use outside of these tools and are at your fingertips every time you sit in front of your computer. Although automation using scripting languages is a world away from the approach involving a commercially available tool, scripting-language automation offers an alternative to emptying your pockets on an expensive automated test tool simply for the convenience of a lot of functionality, the majority of which you'll probably never use.

There are many advantages to using scripting languages for test automation, and these advantages allow testers to expand the role of automation in their organizations.

The fact that scripting languages don't normally require any type of licensing may possibly be the biggest advantage to using them for test automation. Since scripting languages are typically made available with operating systems or provided as free downloads, they are ready for use at any time. Users are therefore liberated from the restrictions imposed by licensing, which results in increased availability and a simultaneous decrease in cost.

The complexity of the development environment is also reduced when you use scripting languages for automation. Scripting languages typically don't require a complex inte-

grated development environment (IDE)—a set of integrated tools for code development. This makes your environment easier to deal with, because you can usually develop and execute the scripts from a simple text editor (for example, Notepad on the Windows platform).

Another advantage is that there will be plenty of information to support

## There are a number of challenges involved with using scripting languages for test automation.

the development of scripts using scripting languages, because they are typically industry-standard languages as opposed to vendor-developed derivatives. Support will be available in the form of books, white papers, official and unofficial Web sites, and user groups.

### When to Use Scripting For Test Automation

Let's face it: For the wrong person at the wrong time, a good thing can become dangerous. The use of scripting languages for test automation can provide an automator with a wide-open field on which to play and explore. But if approached incorrectly, the automation effort can become a wild beast, and the open field will leave you vulnerable to be devoured

by that beast.

There are several indicators that may be used to determine if you are ready to explore the field of scripting-language automation. If a commercial tool is not an option, then scripting is automatically the way to go. There may be several reasons that a commercial tool is not an option; one reason may be that an organization can't afford the expensive commercial tools, or just doesn't feel that the return on investment will justify the cost. Another reason may simply be that the organization has no interest at all in automating any tests, and is content with

having testing resources continue to perform all tests manually. For whatever reason an organization might not want to purchase an expensive automated test tool, scripting provides another way of automating tests. There may be a decision to implement an official automated test effort using scripting, or, if the organization is uninterested in an official effort, a tester may decide to unofficially supplement some manual testing efforts with automated test scripts using scripting languages.

Another indicator may be the size of a company's automation effort. If the person doing the automating is not proficient in scripting languages, but the automation effort is small (a small number of tests, simple functionality, etc.), it may be suited for

TABLE 1: SCRIPTING-LANGUAGE ANALYSIS

	Free	Mature & Libraries	Line Interpreter	Regex	Templating	Java Embed	COM	OO
Ruby	✓	*	✓	✓	✓	?	✓	***
Python	✓	**	✓	✓		✓	✓	**
Perl	✓	***		✓	✓		✓	*
TCL	✓	***/*	✓	✓	✓	?	?	*
VBScript & WSH	✓	***		✓			✓	**

scripting. When resources are proficient in terms of scripting, it may be a little easier for testers to build and maintain an automated test framework using scripting languages, so the probability of success is increased regardless of the size of the overall automation scope.

One last indicator is when a project requires large numbers of people to create or execute automated scripts, a situation that will normally require multiple tool licenses. Many commercial automated test tools offer a server or floating license, but this only allows for a small, finite number of concurrent users of the tool. Scripting offers an alternative to this expensive option. You can choose to use scripting-language automation either in lieu of purchasing an expensive automated test tool, or to supplement the automation being conducted with an automated test tool so that fewer licenses will be required.

Although these indicators are useful, it is still advisable to perform a cost-benefit analysis for determining whether it is more cost-effective to rely solely on a commercial automated test tool or to use scripting languages. Items to consider in a cost-benefit analysis include:

- Resource skills level
- Script development costs
- Script maintenance costs
- Training costs
- Cost of tools
- Cost of licenses
- Number of licenses required

Scripting-language automation starts out with an advantage, in that there are no tool or licensing costs associated with its use. The choice will therefore be a matter of determining whether or not the money saved in tool and license costs is outweighed by higher development, maintenance and training costs. If it is not, consider scripting as your main approach or as a supplemental approach.

If the prospective development, maintenance and training costs seem too high, you may want to go with a commercial automated test tool. For more detailed information on conducting a cost-benefit analysis, refer to Douglas Hoffman's paper entitled "Cost-Benefit Analysis of Test Auto-



mation" (see References, below).

### How to Implement Scripting-Language Automation

Implementation of an automated testing approach using scripting languages can initially seem like much too big of a task, but the key to making it possible is to break the implementation down into small, feasible subtasks. The three main subtasks are:

- Choose a scripting language.
- Learn the basics of the language.
- Learn the appropriate object model and application object properties.

Upon completing these steps, you will then be equipped to develop automated scripts using scripting languages.

Several factors can be used for determining which scripting language to use for test automation. One factor may be simple convenience. For example, if your application is on a Windows platform, you may decide to use VBScript, simply because it comes with the latest releases of the operating system. Or maybe you have some resources who are proficient in Perl,

in which case you may want to use the Perl scripting language for your automated testing solution. You may also decide to pick a scripting language that has a certain level of features that is desired.

In his handbook entitled "Homebrew Test Automation," Brett Pettichord presents an analysis of several of the industry's leading scripting languages. Table 1 is a chart from that handbook.

The columns in the table may be defined as follows:

- Free—The scripting language is free.
- Mature & Libraries—Ranks the maturity level of the language and its libraries.
- Line Interpreter—The language provides the ability to type and test a single line of code without executing an entire script.
- Regexp—The language supports the use of regular expressions (wild cards) for pattern matching.
- Templating—The language supports the ability to interpolate variables in strings.

- Java Embed—The language supports the ability to run inside of a Java Virtual Machine (JVM).
- COM—The language supports the ability to call COM objects.
- OO—Ranks the ability of the language to support object-oriented programming.

If most of the information described in this list makes little or no sense to you, it will probably be sufficient to choose a language based on convenience or on resource proficiency,

and keep moving. Or, for further information regarding the chart, refer to [www.pettichord.com/homebrew\\_automation.html](http://www.pettichord.com/homebrew_automation.html).

Scripting-language automation can be very exciting, and a great resume builder, but before aiming your sights too high in the clouds, it is important to step back and take some time to understand some basic scripting concepts.

Grasping concepts such as how scripting languages differ from compiled languages is necessary for under-

standing how to effectively execute scripts once they are developed. Object-oriented programming concepts will be useful for efficient script development, while language-specific syntax will be necessary for understanding how to write scripts that are able to invoke and control your application under test (for instance, in VBScript, `CreateObject` is an important statement for invoking your application under test, while `GetElementById` is an important statement for control-

ling an element inside that application). For more information, refer to my paper entitled “Test Automation Using Scripting Languages” (see References).

An object model is simply a group of related, hierarchical objects that define an application and work together to complete a set of application functions. Much like an anatomical drawing tells you how different parts of the body work, an object model provides information on how the different objects in an appli-

cation work. It details an object’s hierarchical position, properties, and the actions that may be performed on the object. Upon studying the appropriate object model, information may be obtained about the objects that are specific to the application under test, and scripting may begin.

Let’s briefly examine the case of a browser-based application; the View Source option may be used to obtain object properties manually.

## A Sample Web Page

By viewing the source of the simple Web page illustrated in Figure A, we’ll see the following HTML code:

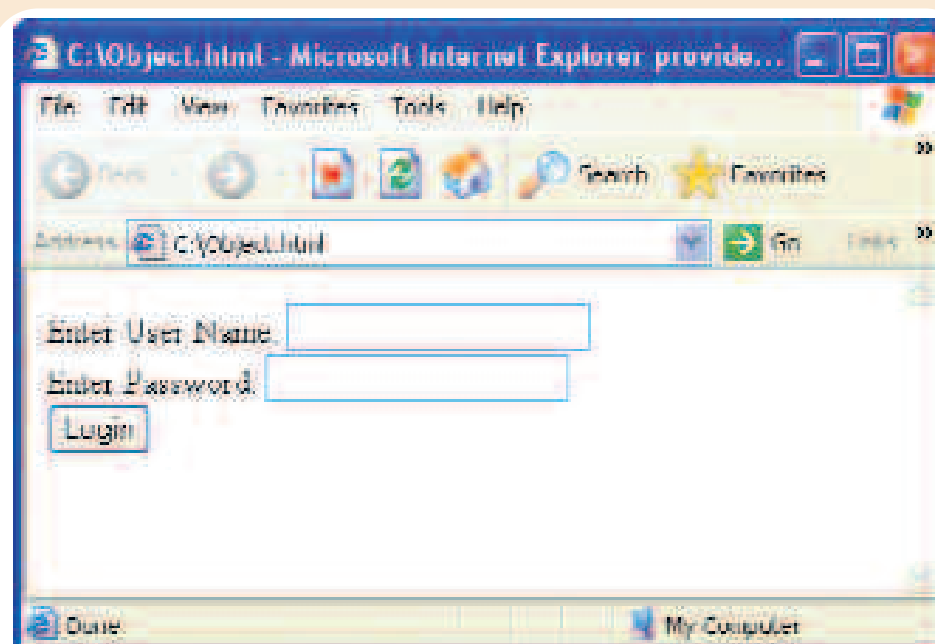
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">

<head>
</head>
<body>
<FORM>
  <P>
    <LABEL for="Name">Enter User Name:
  </LABEL>
    <INPUT type="text"
id="UserName"><BR>
    <LABEL for="Password">Enter Password:
  </LABEL>
    <INPUT type="text"
id="Password"><BR>
    <INPUT type="submit" value="Login"
id="LoginButton">
  </P>
</FORM>
</body>
```

This code reveals information about properties that will be used for performing actions on these objects. For example, the HTML code reveals that the object next to the “Enter User Name” text has an ID equal to “UserName.” From the object model for browser-based applications, I know that objects with an INPUT type equal to “text” can have data entered into them. This information will collectively allow me to enter data into the “Enter User Name” object by instructing the script to input data into the object

**The complexity of the development environment is reduced when you use scripting languages.**

**FIGURE A: SAMPLE WEB PAGE**



with an ID equal to "UserName."

For more information on the use of object models for automated scripting, refer to "Test Automation Using Scripting Languages" (see References, below).

## Script Implementation Challenges

As with anything good, there are some challenges involved with using scripting languages for test automation that must be considered and mitigated to increase the probability of success.

The challenge of dealing with a non-intuitive development environment is offset by the advantage of being able to develop scripts in a text editor. While this advantage provides simplicity, it also presents a hurdle in that it's not very intuitive. There are no text-coloring features that help you to distinguish language keywords and methods from other text. Also, there's no functionality that provides design-time syntax help for adding functions to scripts. The solution to this challenge may be to use a freeware or inexpensive shareware script editor and debugger. For VBScript, Microsoft offers a script debugger at its MSDN Web site, and you may also be able to locate one at [www.download.com](http://www.download.com).

Another challenge in using scripting languages is that they require a higher degree of technical expertise. The most obvious solutions to this include taking formal training, using Internet tutorials, and reading books and white papers specific to your scripting language of choice. By familiarizing yourself with the available information, you'll have a better idea where to look when a specific scripting need arises.

In addition, it may be useful to join a user group. There are plenty of free and low-cost user groups that can provide support for your efforts. These groups often work faster than the sup-

port team for an expensive automated test tool, because many people are reading and investigating your request.

One final challenge presented by scripting languages is that they have no built-in mechanism for learning application objects. Commercial tools provide an automatic way of identifying properties of objects that are being acted on in an automated test script. When using scripting languages, you either have to have prior knowledge of the object from the application's object model, or, in the case of an Internet application, you'll need to check the source code to get the object properties. One solution for this is to create a custom function for learning application objects.

**If you decide a commercial tool is not an option for your organization, then scripting automatically becomes the way to go.**

## Challenges And Advantages

Automation through scripting languages presents some challenges to overcome, but it does offer several advantages. There will never be a world in which expensive automated test tools aren't used, because these tools provide a level of convenience and dependability that is often worth the price. What this article is proposing, however, is that testers be allowed to make the decision as to whether or not an expensive automated test tool is the best approach, rather than having that decision made for us.

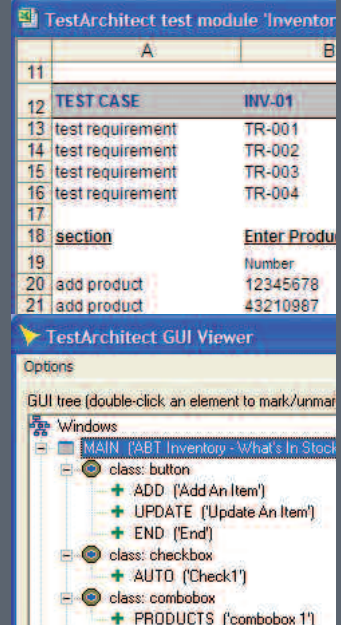
Scripting-language automation provides us with additional options—options that help us pursue the goal that all of us in the IT world strive for: a higher-quality product. ☒

## REFERENCES

- Clark, Susanne, et al., "VBScript Programmer's Reference," Wrox, 1999
- Hoffman, Douglas, "Cost-Benefit Analysis of Test Automation," 2000, [www.stickyminds.com](http://www.stickyminds.com)
- Johnson, Dion, "Designing an Automated Web Test Environment," 2001, [www.stickyminds.com](http://www.stickyminds.com)
- Johnson, Dion, "Test Automation Using Scripting Languages," 2004, [www.stickyminds.com](http://www.stickyminds.com)
- Pettichord, Brett, "Homebrew Test Automation," 2004, [www.pettichord.com](http://www.pettichord.com)

Test earlier. Test faster.  
Automate smarter.

## TestArchitect™ 2



### Advanced Software QA

- test design
- test automation
- test management

### Automation that is

- maintainable
- scaleable
- reusable

### Global management

- shared repository
- distributed teams

### TestArchitect is for

- Business Analysts
- Test Engineers
- Automation Engineers
- Build Engineers
- Managers and Leads

**Download evaluation copy and whitepaper**

**LogiGear®**

Tel +1 800 322 0333  
Fax +1 650 572 2822  
[sales@logigear.com](mailto:sales@logigear.com)  
[www.logigear.com](http://www.logigear.com)