

Title of paper: Automating Software Validation Process using generic framework with Windows PowerShell

Nikhil Bhandari
Intuit,
Bangalore, Karnataka, India
nikhil.bhandari@gmail.com

Abstract:

In This paper describes how to automate software validation process by creating a generic framework for testing SDK (Software Development Kit) APIs with Windows PowerShell. Windows PowerShell is a command line tool as well as a very powerful scripting environment for Windows Platform. Unlike other scripting shells, PowerShell works with .Net objects. It exposes all parts of the .Net framework. All kinds of different test frameworks for Unit testing, Functional testing, Regression testing, Performance testing, Deployment testing etc. can be easily developed and integrated into a single consistent test automation framework. PowerShell is free of cost is it very easy to understand and implement so developing an Automation framework using PowerShell will reduce the cost and effort , and increase the efficiency and quality. This Test Automation delivers tremendous benefit to the QA organization. Many applications, as we know, are now days being built on top of technology frameworks using the standard ones like COM/J2EE or in-house developed frameworks in C/C++ etc. The goal is to achieve a faster time-to-market of the applications or products as developers make use of ready available plug-ins and/or skeleton code via the frameworks. This area is fast gaining importance from a testing viewpoint because, the framework is considered to be the core of the application, where bugs found early can save time and cost during the final phases of testing. Here we'll try to showcase how windows PowerShell can be used to test SDK or platform/technology frameworks (developed on Microsoft platform) that enable sharing data directly with client applications.

Keywords:

Test Automation Framework, Windows PowerShell

1 Introduction

Over time, there has been a flurry of test tools in the market catering to different areas of testing such as functional, regression and performance testing etc. Various commercially available third party tools focus certain areas of performance testing that include performance testing of Graphical User Interface (GUI) or browser based applications. Performance testing most of the times involves testing the application with a vast set of concurrent users or simultaneous clicks. However there may be certain testing needs to test the layer underneath the application layer. This involves testing the response time of queries fired to the database.

Browser based applications have application servers or COM (Component Object Model) components running underneath them. And tools such as JMeter are available to test these applications. Challenge lies in testing applications which are developed using non-application server based technologies or applications that are developed using pure C++ based methods.

Many applications, as we know, are now days being built on top of technology frameworks using the standard ones like COM/J2EE or in-house developed frameworks in C/C++ etc. The goal is to achieve a faster time-to-market of the applications or products as developers make use of ready available plug-ins and/or skeleton code via the frameworks. This area is fast gaining importance from a testing viewpoint because, the framework is considered to be the core of the application, where bugs found early can save time and cost during the final phases of testing.

This paper tries to showcase how windows PowerShell can be used to test Software Development Kit (SDK) or platform/technology frameworks (developed on Microsoft platform) that enable sharing data directly with Client applications. It is based on programming standards such as XML (eXtensible Markup Language), COM (Component Object Model), and HTTPS (for communication over the Internet). With the help of Windows PowerShell, all types of consistent test frameworks such as Performance can be easily & quickly developed as it is based on .NET and hence provides direct access to the entire CLR, plus the ability to script existing COM (ActiveX) and WMI objects.

In this session we will attempt to explore usage of Windows PowerShell in testing the SDK (Software Development Kit) of one of Intuit's flagship product built using Microsoft technologies. The requirement was to develop a common test framework which will perform all the above through a command-line interface as SDKs' tend to be non-UI based.

Quick-Books is desktop software that an end-user uses to enter and track their business data. The QuickBooks Software Development Kit (SDK) enables your application to share data directly with QuickBooks. SDK exposes APIs which can used to Add, Delete, Modify data in QuickBooks through your application.

2 What is Windows PowerShell?

Windows PowerShell is exciting, powerful and comprehensive CLI (command line interface) tool for Microsoft Windows Platform. PowerShell introduces a new, more powerful, more flexible, more consistent object-based command line tool and scripting language with a highly consistent syntax. The shell includes an interactive prompt and a scripting environment that can be used independently or in combination. In short, Windows PowerShell is scriptable automation shell. Windows PowerShell includes numerous system administration utilities, consistent syntax and naming conventions, and improved navigation of common management data such as the registry, certificates, file system. The Microsoft PowerShell features a unique object oriented syntax, extensive support for versatile .NET technology, and an adequate assortment of commands.

3 How Windows PowerShell is different?

Windows PowerShell is based on .NET framework 2.0 which has deep ties into almost every aspect of the operating system - so you have direct access to the entire CLR (Common Language Runtime), plus the ability to script existing COM (Component Object Model) and WMI (Windows Management Instrumentation) objects. Everything in PowerShell is an object and has the full capabilities of the .NET Framework. A command returns a set of objects that can then be utilized by using the properties and methods of that type of object. When you want to pipe the output of one command into another command, PowerShell actually passes the objects, not just the textual output of the first command. This gives the next command in the pipeline full access to all of the objects' properties and methods.

PowerShell scripting is C# (C-Sharp)-like scripting language with support for hash tables, switch statements which can test on regular expressions, array slicing and anonymous methods (script blocks) which can be stored as data and then later executed. It also provides looping (for/foreach/while), conditional statements (if/switch), variable scoping (global/script/local) and the ability to define functions. PowerShell allows you to preview the effects that your scripts will have on the system. Commands that create or change objects support the -Confirm and -WhatIf flags. The confirm flag forces the user to -Confirm an action before it is executed and the -WhatIf flag will show you what happens without actually performing the action.

PowerShell also introduces a consistent command naming convention using a verb-noun structure. This pattern makes it much simpler to remember a few object names rather than an array of archaic commands and parameters. The command-line options are generally whole words, but can be specified as the minimum number of letters necessary to disambiguate. PowerShell has comprehensive, user-extensible tab completion features.

4 Framework Development with Windows PowerShell

This SDK functionality is tested at multiple levels, Unit, Functional, System and Performance. We wanted to have a single and generic framework which will perform all the above mentioned tasks and more specifically a command-line interface as SDK doesn't have any user interface.

Windows PowerShell is based on cmdlets (pronounced command-lets), which are small, modular commands consistently based on a verb-noun naming system. You can use each cmdlet separately, but their power is realized when you use these simple tools in combination to perform complex tasks. Windows PowerShell includes more than one hundred basic core cmdlets, and you can write your own cmdlets.

Windows PowerShell is an extremely extensible environment. There are four ways that you can extend it:

4.1 Cmdlets

Cmdlets are tiny .Net classes which are surfaced as commands. These are small executables, written in a .NET language and that implements a standard interface. Cmdlets are named using a 'verb-noun' format (e.g., Get-Help or Export-CSV). Cmdlets can take parameters or take input from the pipeline.

4.2 Scripts

PowerShell scripts are designed to string several cmdlets together to automate more complex tasks. Anything that you type at the command line can be stored as PowerShell script and reused at a later date. PowerShell can be scripted either interactively or by putting all the necessary expressions in a .ps1 file.

4.3 Providers

A set of .Net classes which expose data. These are surfaced as "Drives" which you can interact with as if they were a file-system drive. The provider represents a set of stored data. For example, the stored data can be the Windows Registry; the Windows file system, Active Directory, SQL Server, and the variable and alias data stored by Windows PowerShell.

4.4 Snap-ins

Snap-ins are assemblies containing cmdlets and can be loaded during startup time. A snap-in is a Dynamic Link Library (DLL) file. You can create your own snap-in or use snap-in created by third parties. By default, the core PowerShell snap-ins are loaded.

So with the help of PowerShell SDK, we have implemented our own cmdlets for QuickBooks called QB-Shell. The whole framework is written using these implemented cmdlets in combination with existing ones. Performance tests are automated using this framework. All the inputs and outputs use XML formats specific to QuickBooks SDK. These output XMLs are parsed using PowerShell and necessary information (like No. of Objects returned in the response) is extracted from it which is later copied to output file along with response time.

5 Power of PowerShell in Test Automation

With PowerShell more work done faster and to do that work for the same or less cost. PowerShell could be used for application testing and infrastructure testing.

Unit testing & Functional Testing can be achieved in PowerShell by creating your own cmdlet and writing a very generic framework with the help of already existing cmdlets. In unit testing, you write automated tests that exercise your code, whereas in functional testing you would be testing the functionality of the particular product.

Performance Testing is also integrated with the existing framework. In this main focus is on the time take to response is measured. All tasks related to Windows Services, Event Logs, Registry, CPU usage, System Configuration, Machine Details, BIOS information are easily achieved with PowerShell.

Data-driven testing is a partial solution to improving test script maintainability and lowering the cost of adding test cases. The way to reduce the cost of testing is to increase the 'test cases to test script' ratio, that is, write fewer (and leaner) test scripts that can be used to run a great many more test cases each

Deployment Testing is another area where PowerShell make it really easy to implement. You can quickly check the structure of the deployed application by creating a script that verifies your release went as expected - check running processes, services, database metadata, database content, application file versions, etc. You can even check the contents of configuration files since PowerShell does a nice job of navigating XML and plain text files.

Infrastructure testing can be really handy with PowerShell. Simply write a script that verifies the hardware, operating system, running services, running processes, etc. match your expectations. PowerShell allows you to execute the script on a Remote machine. So, most of the Administrative tasks are easily controlled through PowerShell.

6 Architecture

The SDK functionality is tested at multiple levels, Unit, Functional, System and Performance. We wanted to have a single framework which will perform all the above mentioned tasks and more specifically a command-line interface as SDK doesn't have any UI.

The whole framework is written using these implemented cmdlets in combination with existing ones. Unit, Functional, Performance & Deployment tests are automated using this framework. All the inputs and outputs use XML formats.

Unit testing is implemented by integrating NUnit framework with PowerShell framework. The Functional tests comprises of executing particular tests and comparing the Actual response output with the Reference response.

Performance tests calculate the response time taken for each of the Request query to execute and based on the expected response time tells whether performance is degraded or not. These performance tests are carried out on various company file databases ranging from small to huge. In deployment testing, the product installed in attended mode. After installation, registry entries are validated, status of windows services is checked, status of

product specific process in verified, folder structure and DLL files are verified, and information from the log files is extracted and so on.

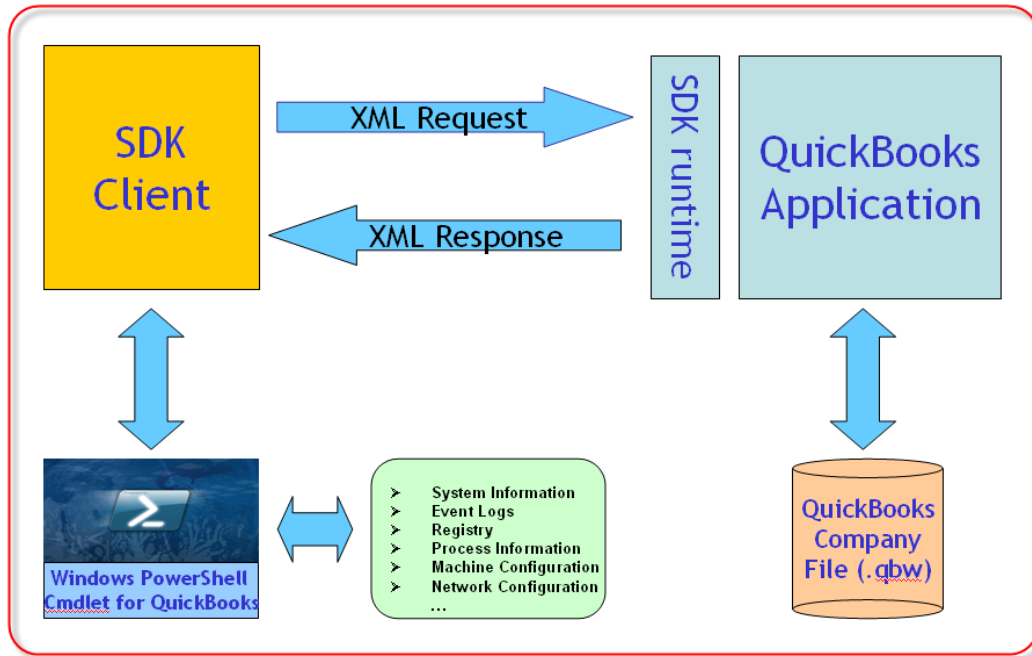


Figure 1 : Architecture

7 How tests are executed?

An extensible test framework was developed using Windows PowerShell. This test framework was designed in such a way that it could be used across company files. A company file in general terms is a database. All the company files were segregated into different groups based on the number of customers, vendors and Employees they possess. In this way, the team was able to realize the performance for different kinds of company files.

All the required software on the test machine was installed using the existing TCoE (Testing Center of Excellence). Every test had three runs for measuring the response time of the XML request. Average of all the three runs was compared with the response time of the last build. Any deviation in the response was investigated thoroughly for no further degradation in performance. All the response times of different kinds of requests are written to an excel file. The tests were run in both Single & Multi User mode.

7.1 XML Technology

A request XML is a file which would contain the input query. This query when triggered through PowerShell would get the information about the time it took to retrieve information from the database. The response information and the time would be stored in the response XML file. The PowerShell script developed for this would retrieve only the response time for all the requests send to the database. Another script was developed that would collect the average response times of all the requests.

7.2 Test Inputs

The input file (.xml) contains the Test Configuration which includes the locations of Company file, Request XML files, Output directory, Output XML file and so on. Some of the entries like Company file is dynamically added to the input XML file which gives us the flexibility to run the tests against in the multiple company files.

7.3 Test Outputs

The output file (.xml) contains the test result (passed/failed) along with the response time for each of the request XML and No. of Object returned. Also the machine configuration, status of the QuickBooks related processes, Company file information, size of the response file and other details are captured in the output file. Event logs and Registry values are also captured and kept in separate log file.

7.4 Test Report

Test Report also automatically generated using Windows PowerShell by using COM functionality of Microsoft Excel. The information available from the Output XML file is gathered and composed into a well formed excel sheet which will be sent across to the management team. The average response time of three Iteration is compared against the benchmarked numbers and the X factor is calculated which tell us whether performance is improved or degraded. Then the Average Speed-Up and Slow-Down is calculated based on the X factor. This report contains results for Single User and Multi User mode

8 Benefits

8.1 Availability

PowerShell is available for free and runs on Windows XP, Windows Server 2003, Windows Vista and later operating systems. Windows PowerShell requires the .NET Framework version 2.0. PowerShell installation can be done in unattended mode.

8.2 Consistency of Implementation

Windows PowerShell cmdlets consistently use a verb-noun (that is a verb followed by a hyphen followed by a noun) naming scheme. All cmdlets support a set of common parameters. Consistency is maintained even if the new cmdlets is developed by a developer. While implementing your own cmdlet, PowerShell gives a framework that provides the basic feature and forces developers to be consistent about many aspects of the interface.

8.3 Straightforwardness

PowerShell is self-explanatory. PowerShell scripting is very easy to understand and learn. It provides a consistent help with examples. In fact certain build in cmdlets that you use everyday, like foreach-object, where-object, compare-object, sort-object, measure-object and select-object. Even implementing cmdlets and providers are not that difficult, one has to follow the framework provided by PowerShell and customize according to your need. Working with PowerShell is just like working on any other command-line tool like cmd.exe

8.4 COM Access

Windows is built on COM, and COM objects provide significant functionality for files, folders, and much more. PowerShell includes an adaptation layer that permits you to utilize COM components. The new-object cmdlet, when used with the Com-Object parameter, allows you to create a new COM object. For COM objects, the get-member command lists all attributes and methods.

8.5 WMI Access

The overall WMI architecture is like a database in that you connect to the WMI service on a particular computer, optionally specifying credentials, and retrieve the objects using a query. Windows PowerShell provides full read access to Windows Management Instrumentation (WMI). An important area where WMI access fills a gap in the current PowerShell cmdlets is access to remote machines. PowerShell provides a Get-Wmiobject cmdlet that can retrieve WMI class instances

8.6 Security

For security purposes, PowerShell defaults to a very restrictive execution policy that says the shell can only be used interactively, which occurs when you type in commands directly and have them execute immediately. This helps ensure that PowerShell can't be used to run script-based viruses by default. Execution policies define the restrictions under which PowerShell loads configuration files and runs scripts. The four execution policies are Restricted, AllSigned, RemoteSigned, and Unrestricted.

8.7 Admin related tasks

All administrative tasks are easily achieved with the help of PowerShell. This includes manipulating registry entries, starting & stopping services or process, event logging, machine configurations, remote machine administration, networking activities and the list goes on and on.

9 Conclusion

My last point is that automation possibilities are always changing. You need to keep on top of new technologies and techniques so that when the time comes to save some time through automation you have a better idea of all your options.

References

- [1] Andrew Watt: Professional Windows PowerShell: Wrox publications
- [2] Don Jones and Jeffery Hicks: Windows PowerShell TFM: Sapien Press
- [3] Microsoft Windows PowerShell - Step by Step: Microsoft