



---

# VIM, Windows, Powershell, and Signed Code Security

---

**Published by the Open Source Software Lab at Microsoft. April 2008.**

Special thanks to Chris Travers, Contributing Author to the Open Source Software Lab. Most current version will be maintained at <http://port25.technet.com>.



---

## **Abstract:**

In the UNIX and Linux world, vi and EMACS have long held positions as the best-developed editors for handling large amounts of code or other text. More recently, an improved vi clone has emerged named VIM (Short for Vi, IMproved). VIM features syntax highlighting, a vi-like command-line interface, and many powerful features for editing large text projects. It has quickly become one of the favorite text editors outside the Windows world.

---

Information in this document, including URL and other Internet Web site references, is subject to change without notice and is provided for informational purposes only. The entire risk of the use or results from the use of this document remains with the user, and Microsoft Corporation makes no warranties, either express or implied. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

© 2008 Microsoft Corporation. This work is licensed under the Microsoft Public License. The Microsoft Public License is [available here](#).

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Microsoft, Windows, Windows XP, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

---

## 1.1 Introduction

In the UNIX and Linux world, vi and EMACS have long held the standard positions as the best-developed editors for handling large amounts of code or other text. More recently, an improved vi clone has emerged named VIM (Short for Vi, IMproved). VIM features syntax highlighting, a vi-like command-line interface, and many powerful features for editing large text projects. It has quickly become one of the favorite text editors outside the Windows world.

VIM is an engineering tool, not a lightweight text editor and hence it has a somewhat steep learning curve for beginners. The command structure is designed to help engineers process large amounts of code as quickly as possible, and so visual cues regarding how the interface works are less important. Furthermore, since VIM, like vi, is designed to be keyboard driven, visual cues for all features are not realistically possible. GUI versions of VIM are more intuitive for beginners, but the command-line interface is still considered the most efficient way to perform many tasks.

### 1.1.1 Newbie Guide to Using Vim

VIM, like vi, operates in 3 modes: command, insert, and replace. These modes are generally accessed using keyboard commands. Typically, to enter insert mode from command mode, you press 'i' and to enter replace mode you press the insert key from insert mode. To return to the command mode, press the escape key.

In command mode, the following sequences are helpful:

- /string searches for the next instance of the string
- :q exits (safe)
- :q! exits even if there are unsaved changes.
- :w writes/saves (safe)
- :w! attempts to force a write/save
- :wq saves and exits
- :n processes the next file in the queue
- :prev processes the previous file in the queue.
- :! command executes command in the shell (in Windows, cmd.exe)

Additionally, there are a number of other character editing commands indicated by the letters c (Change), d (Delete), g (chanGe case), and so forth. These can be used to create more complex commands like:

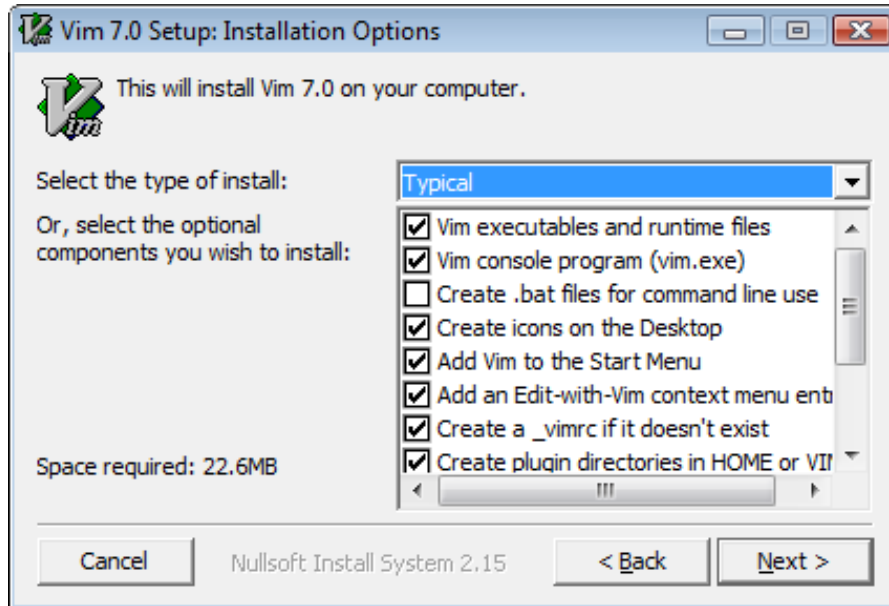
- 33dd (Delete the following 33 lines)
- d3w (delete the following three words)
- gU3w (change the following three words to upper case)
- D (delete the rest of the line)
- c3w (delete the following three words and put the interpreter in insert mode)
- p pastes the most recently deleted information after the cursor's current location.

There are many more possibilities than this. VIM can handle parenthesis matching and many other cases. One can learn the basics in 5 minutes, and spend a lifetime perfecting the art. However, there is no doubt that once one is familiar with VIM, that these features save a great deal of time.

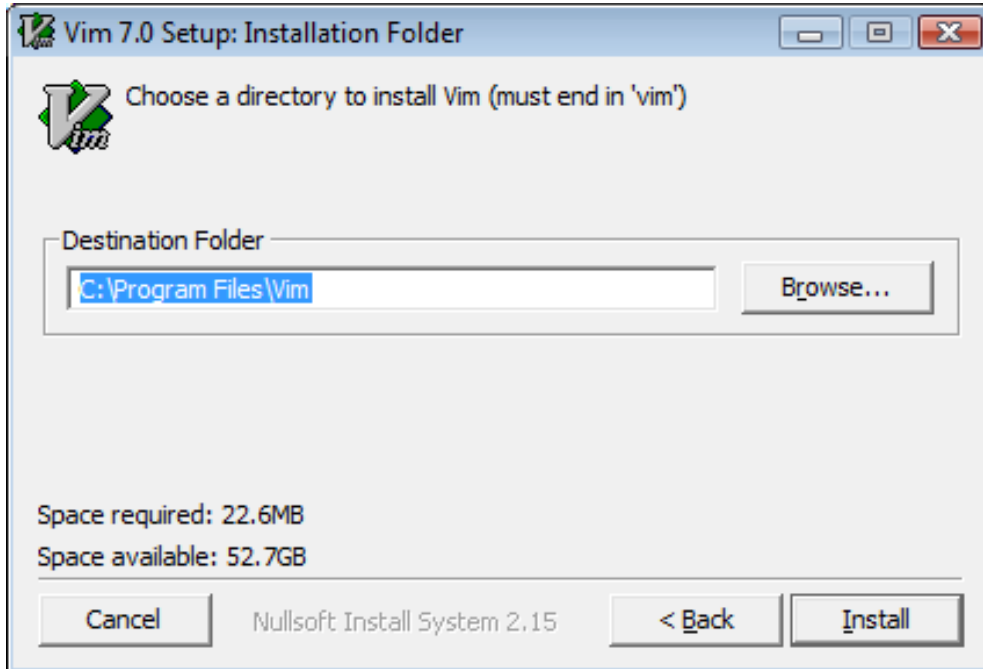
## 1.2 Installing VIM

One can obtain a VIM installer for Windows, which includes OLE support (and Visual Studio integration), a GUI application, and a console version from <http://www.vim.org/download.php#pc>.

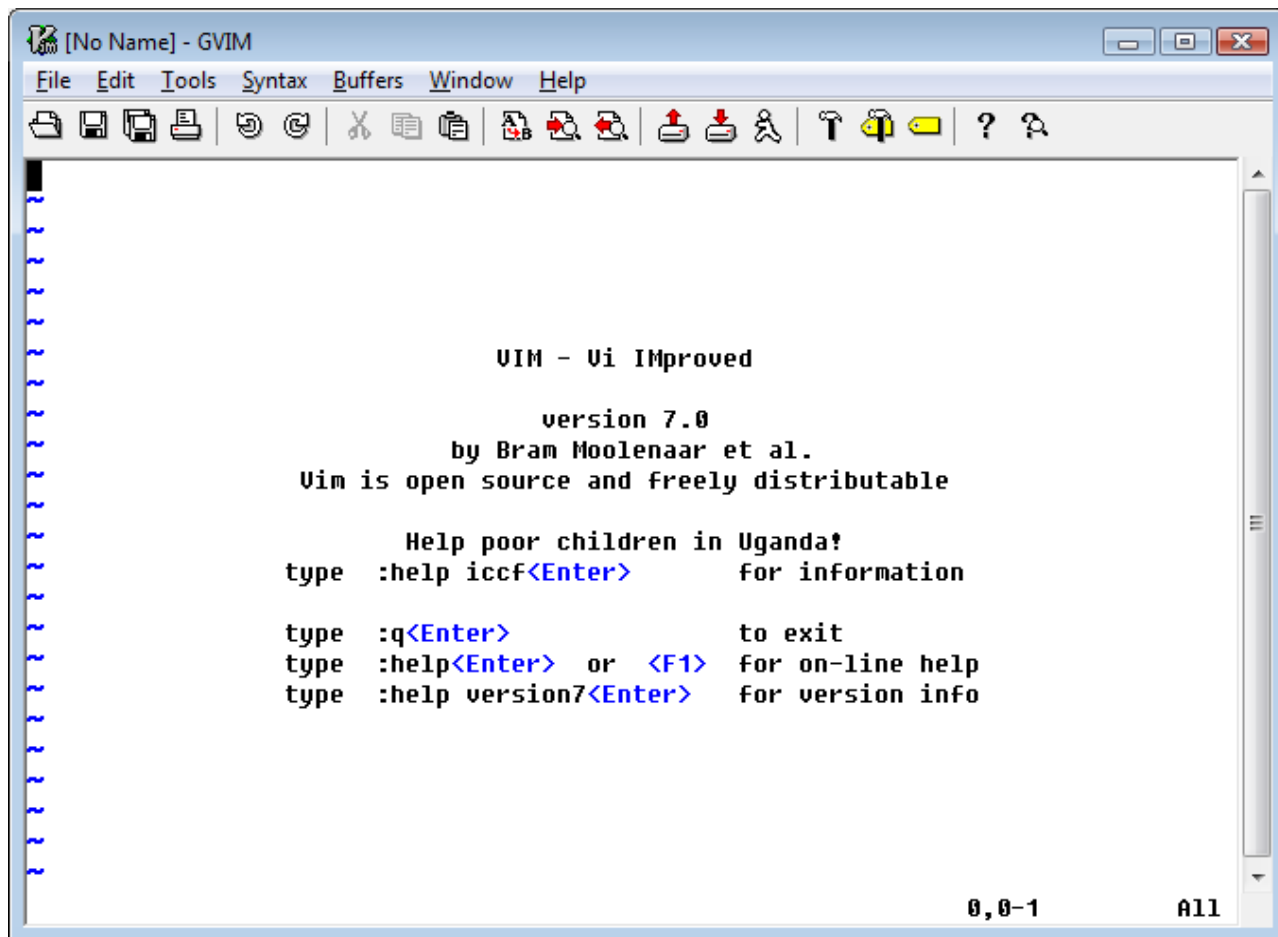
The first screen that requires user input is the Installation Option screen. In general, you can leave these at their default values. However, if you want to experiment with Visual Studio integration (which is beyond the scope of this paper), you can select the VisVim option near the end of the option list.



Typically, the application will be installed in C:\Program Files\Vim as shown on the next screen.



Once VIM has been installed, you can run it from the icon on the desktop. It brings up a screen, see the next page, and initially begins in command mode (press 'i' or the insert key to enter insert mode).



### 1.3 Installing Powershell and the VIM extensions

The Powershell syntax file can be downloaded from:  
[http://www.vim.org/scripts/script.php?script\\_id=1327](http://www.vim.org/scripts/script.php?script_id=1327).

To install it, simply save it in C:\Program Files\vim\vim70\syntax\.

You will also need the indent file from [http://www.vim.org/scripts/script.php?script\\_id=1815](http://www.vim.org/scripts/script.php?script_id=1815) (save this to C:\Program Files\vim\vim70\indent) and the file type extension from [http://www.vim.org/scripts/script.php?script\\_id=1816](http://www.vim.org/scripts/script.php?script_id=1816) (save this in the C:\Program Files\vim\vim70\ftplugin\ directory).

The current links to the Powershell downloads can be found at:  
<http://www.microsoft.com/technet/scriptcenter/hubs/msh.msp>.

At the time of this writing, the link for Vista users is separate than the link for users of other operating systems.

Additionally, two lines need to be added to the following file: C:\Program Files\vim\vim70\filetype.vim. I added these lines (starting at line 1994 in the file:

<http://port25.technet.com>

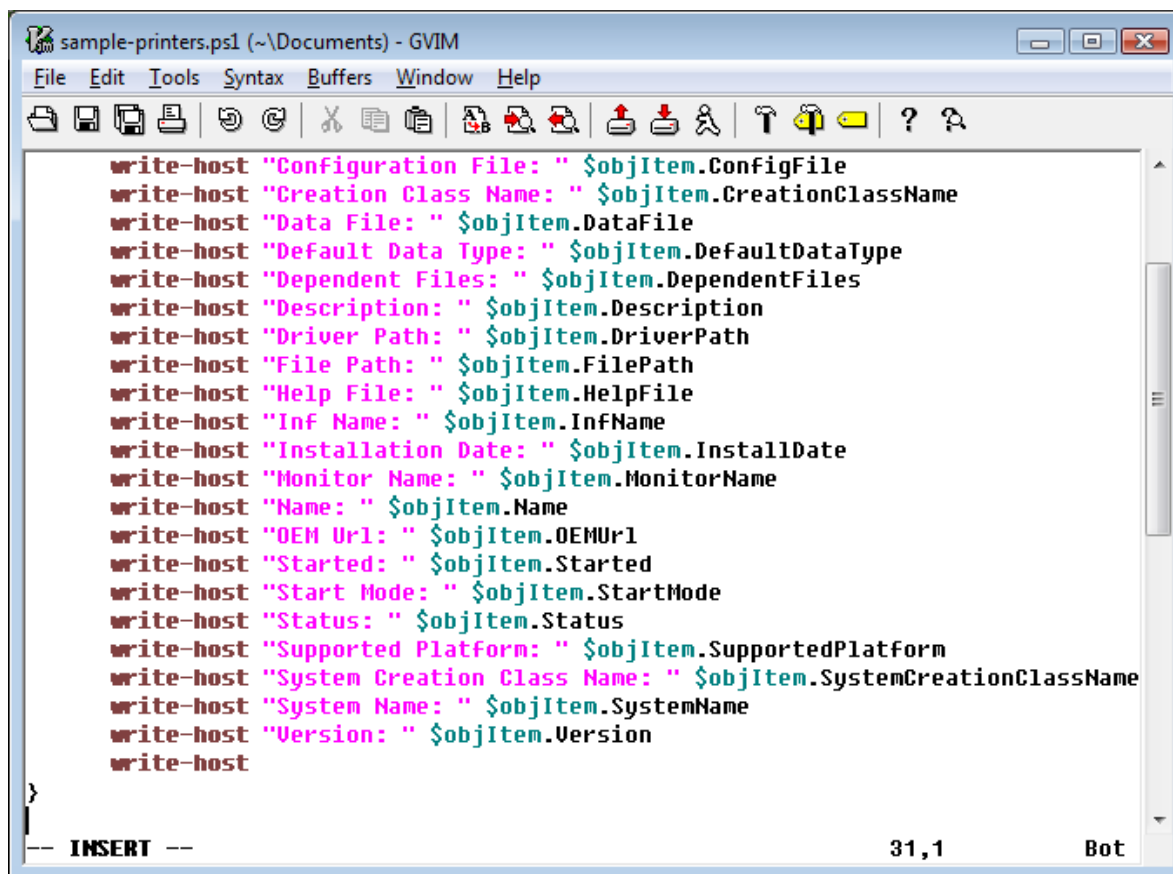
```
" Powershell
au BufNewFile,BufRead *.ps1,*.pscl          setf ps1
```

Once this was complete, and VIM restarted, the editor could perform proper syntax highlighting for Powershell files.

## 1.4 VIM and Powershell

For purposes of this paper, the print driver enumeration sample was used from <http://www.microsoft.com/technet/scriptcenter/scripts/msh/default.mspx?mfr=true>.

As you can see from the screen shot, syntax handling is done properly.



```
sample-printers.ps1 (~\Documents) - GVIM
File Edit Tools Syntax Buffers Window Help
write-host "Configuration File: " $objItem.ConfigFile
write-host "Creation Class Name: " $objItem.CreationClassName
write-host "Data File: " $objItem.DataFile
write-host "Default Data Type: " $objItem.DefaultDataType
write-host "Dependent Files: " $objItem.DependentFiles
write-host "Description: " $objItem.Description
write-host "Driver Path: " $objItem.DriverPath
write-host "File Path: " $objItem.FilePath
write-host "Help File: " $objItem.HelpFile
write-host "Inf Name: " $objItem.InfName
write-host "Installation Date: " $objItem.InstallDate
write-host "Monitor Name: " $objItem.MonitorName
write-host "Name: " $objItem.Name
write-host "OEM Url: " $objItem.OEMUrl
write-host "Started: " $objItem.Started
write-host "Start Mode: " $objItem.StartMode
write-host "Status: " $objItem.Status
write-host "Supported Platform: " $objItem.SupportedPlatform
write-host "System Creation Class Name: " $objItem.SystemCreationClassName
write-host "System Name: " $objItem.SystemName
write-host "Version: " $objItem.Version
write-host
}
-- INSERT --                               31,1                               Bot
```

## 1.5 Handling Signed Code

Powershell uses digital signatures verified using X.509 certificates in order to verify the authenticity of the script author. This is a useful tool in ensuring that scripts were written by appropriate individuals, and that they were not altered by third parties prior to being run. The certificate management is best handled using Certificate Server and integrated with Active Directory for large networks, though individual developers may find OpenSSL or makecert (from the .Net SDK) helpful.

While it is possible to manage the code signing from within the editor, there are a number of good reasons not to:

- 1) Shell escapes operate on the last saved version. It is logically cleaner and less error prone to close the editor and sign. This avoids signing a file just to discover that one has not saved the last of the changes, for example.
- 2) Any editing of the file after signing invalidates the signature.

However, you can access the full capability of Powershell (including signing capabilities by escaping to the shell using the command:

```
:! powershell
```

In general, I would consider handling such signing tasks from within the text editor to be somewhat dangerous because the signing process must occur after all changes have been committed.

Note that after each change, you must sign your code. I am not aware of any facility in VIM to automate this, but such scripts could potentially be written. Note that people do use similar facilities in VIM to build and debug C code so this is not out of the question.

## **1.6 Final Thoughts**

VIM is a powerful text-editing tool, though like all engineering tools, there is a learning curve. It is extensible and people have already contributed Powershell support.

VIM offers a number of benefits and challenges in a Powershell environment mostly because Powershell is still new and so are the Powershell extensions for this tool. While the benefits may not be large for small scripts, editing large scripts, code refactoring, and other more complex tasks are better handled by VIM in my experience than just about any other editor (with the possible exception of Emacs).

## **1.7 About the Author**

Chris Travers is the owner of Metatron Technology Consulting, a firm devoted to helping businesses leverage open source software on any platform. He has written many other papers covering open source databases, web servers, and applications.