# OEM Ready Program

The information presented in these guidelines is informational only and reflects Microsoft’s views as of the date of publication.  These views can and probably will change in response to changing market conditions.  Microsoft makes no warranties or guarantees, implicit or explicit, in or by virtue of this document. This document is Microsoft Confidential. Unless otherwise permitted by law, no part of this document may be copied without Microsoft’s prior written permission.

© Microsoft Corporation 2008.  All rights reserved.



Table of Contents

[OEM Ready Program 1](#_Toc197162683)

[Introduction 3](#_Toc197162684)

[Requirements and Test Tool Overview 4](#_Toc197162685)

[Definitions and Policies 6](#_Toc197162686)

[Test 1: Verify that the application’s executables follow standard User Access Control Guidelines. 8](#_Toc197162687)

[Test 2: Driving Signing and Verification 12](#_Toc197162688)

[Test 3: Verify that the application checks for the OS version in the approved way. 20](#_Toc197162689)

[Test 4: Verify that the application creates uninstall Registry entries. 25](#_Toc197162690)

[Test 5: Verify that the application passes specific AppVerifier checks. 28](#_Toc197162691)

[Test 6: Verify that the application installs to the correct folders by default. 30](#_Toc197162692)

[Test 7: Verify that the application is Restart Manager Aware. 32](#_Toc197162693)

[Appendix 1: Test Machine Setup and Tools Installation 35](#_Toc197162694)

[Appendix 2: Resources 40](#_Toc197162695)

[Appendix 3: Automating the Driver Signature Verification Test 41](#_Toc197162696)

[Appendix 4: Report Card 43](#_Toc197162697)

## Introduction

This Document

End user customers value stability, reliability, performance and quality in the Customer Systems they purchase. Microsoft focuses some of its investments to meet these requirements in the area of software applications certified and designed to run on the Windows Vista platforms for PCs. These efforts include compatibility tests for ease-of-use, better performance and enhanced security on PCs running Windows Vista software. Microsoft’s compatibility tests have been designed in collaboration with industry partners and are continuously improved in response to industry developments and consumer demand.

This document lists the requirements that Qualifying Software distributed with an OEM system needs to meet in order to comply with the OEM Ready Program. Unless otherwise noted, an application must pass all tests.

Recommend Testing Approach

For each Qualifying Software:

1. Set up the test machine (See Appendix 1: Test Machine Setup and Configuration),
2. Work through all the tests, filling in the report form (Appendix 4) as you go.
3. For feedback or support, contact [mdasw@microsoft.com](mailto:mdasw@microsoft.com)

## Requirements and Test Tool Overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Test Name | Description | Rationale | Test Tool |
| 1 | UAC Manifesting | Ensure applications follow standard User Access Control Guidelines. | Requiring proper UAC manifesting improves OS security by ensuring apps do not acquire higher privileges than necessary. | CFF Explorer |
| 2 | Driver Signing and verification | Any kernel-mode drivers installed by an app must be signed through the Windows Logo or Unclassified Driver programs & pass a driver verifier test.  Any boot drivers on OEM Systems must have embedded signatures. | Poorly written or malware drivers can severely affect the stability and security of a system.  On 64-bit versions of Windows Vista, unsigned drivers will not be able to load. Bootdrivers that are unsigned or that lack embedded signatures degrade system boot-time performance. Running all application drivers through the unclassified driver program helps improve reliability while making sure that basic security policies are met by signing drivers. | Sign Tool and Script Driver Verifier |
| 3 | OS Version Checking | An application must not fail to run if the operating system is updated. | When Windows users upgrade their operating system, they shouldn’t be hard blocked from using their current software applications. 25% of application compatibility failures are caused by incorrect version checking by applications. | App Verifier |
| 4 | Uninstall | Ensures all apps registered with add/remove programs | Supports a good uninstall experience for end-users and helps customers to easily find and remove unwanted applications. Customers should be able to easily find and remove any unwanted applications from OEM Systems. | Orca |
| 5 | Application Verifier Checks | Tests for app failures from common coding errors:  exceptions, handles, heaps, locks, memory, Thread Local Storage | App failures, crashes & hangs are top drivers of customer dissatisfaction.  Eliminating such failures improves app stability and responsiveness. App verifier catches common coding errors and prevents heap corruption (the cause of more than 15% of application failures). | App Verifier |
| 6 | Installation to Correct Folders | Applications should be installed to Program Files or the user’s AppData folder by default. | Ensuring that apps install to the correct locations helps ensure that application code is protected by the operating system.  It also helps ensure that the application is more compatible with different system configurations and migration scenarios. | App Verifier |
| 7 | Restart Manager Awareness | Verifies that all applications running on the system are able to listen and respond to shutdown messages using the Restart Manager API without User interaction while idle. | Applications that ignore restart messages degrade system shutdown performance and causes system reboots which leads to a bad user experience.  By making sure the application is restart manager aware, applications can avoid causing system reboots after installs and updates (by shutting down). | Restart Mgr Tool |

## Definitions and Policies

Application Definition

A software application qualifies for audit if:

1. It runs on the Windows Vista operating system;
2. It is distributed with OEM systems preinstalled with Windows Vista;
3. All application processing is performed on the OEM system and not from a server or other hardware; and
4. It is a stand-alone application that runs inside its own process on Windows Vista.  This includes applications that require other applications as a prerequisite.

Examples of applications that qualify for audit:

* + A Windows Vista compatible application that requires Microsoft Outlook 2007 to be installed
  + A .NET 2.0 application
  + A Windows Vista compatible application that requires optical media to run
  + A Windows Vista compatible application that requires a connection to a server
  + Applications that ship on the system, as opposed to drop-in-the-box.  This applies whether an application is fully installed or loaded.
  + Trial applications, both in trial mode and after full license.  Again, the principle is that all applications that ship on the system qualify for audit.
  + Applications from Independent Hardware Vendors (e.g., an application shipped with a graphics chip set or motherboard).

Examples of applications that do not qualify for audit:

* + Web applications accessed through a browser
  + Internet Explorer plug-ins/add-ons
  + Gadgets
  + Toolbars
  + MMC snap-ins
  + Software that does not ship on the system, such as drop-the-box software, retail software, or software download from the Internet.
  + Firmware (i.e., computer programming instructions that are stored in a read-only memory unit rather than being implemented through software).
  + Recovery images.

Versioning

Application version numbers are broken down as follows: Major.Minor.Build.Revision

For the purposes of this program, applications will be considered distinct when they have differing major or minor version numbers. Updates to the build or revision alone will not require a resubmission of the application.

Version changes that DO NOT require resubmission

1.0.0.0 to 1.0.1.1

2.1 to 2.1.1

Version changes that DO require resubmission

1.0.1.1 to 1.1.1.1

2.0.1 to 2.1

2.1 to 3

Testing within this program is also considered language neutral. Applications do not need to be resubmitted for each language as long as the major and minor version remain unchanged.

Normal Operations

Some tests require applications “Normal Operations” to be executed for the application. These tests are designed to exercise the applications under the most common usage scenarios. While these steps will vary from application to application, the guidance below dictates what should minimally be included within the normal operation step

1. Entering data. Data should be provided in a way typical for the application.
2. Changing application settings. If the application has user configurable settings, these settings should be changed as part of the normal operations.
3. Changing system settings. If the application is used to configure settings for hardware or the operating system, then these actions should be included in the normal operations.
4. Saving state. If the user has the ability to save their work (documents, projects, playlists, game progress, etc.) in the application, then this should be included in the normal operations.
5. Loading state. If the application has the ability to save its state (documents, media, projects, etc.) then reloading this state should be part of the normal operations. Additionally, if the application is able to load files of a common type (jpg, pdf, doc, mp3, etc.), loading a files should be part of the normal operations regardless of whether the application also has the ability to create these files.
6. Processing. If the application completes processing jobs (such as scanning for viruses, or encoding a video), then this should be included in the normal operations.

## Test 1: Verify that the application’s executables follow standard User Access Control Guidelines.

Criteria

*In order to pass this test:*

Every executable file included with the application (that is, all files ending with the extension **.exe**) must contain a manifest that defines the execution level.

Rationale

A user’s Windows experience is more secure when applications run with only the permissions needed.

Verification Process

For each Qualifying Software:

1. Browse the application’s install paths for all executables ending with the **.exe** extension.
2. For **each** of these application’s executables above:  
   1. Open **CFF Explorer** (See Appendix 1: Test Machine Setup and Configuration).
   2. Navigate to the executable, and open it.
   3. In the tree structure on the left, highlight the **Resource Viewer**.
   4. Expand the **Configuration Files** tree.
   5. Click on the resource named **1** (This is the most common name, but may vary based on language).
   6. The embedded manifest execution level will have this format:

<security>

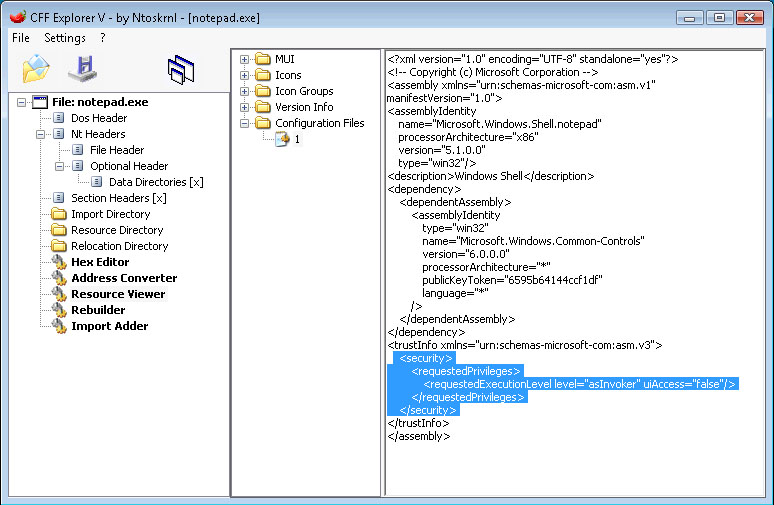
<requestedPrivileges>

<requestedExecutionLevel level="asInvoker" uiAccess="false"/>

</requestedPrivileges>

</security>

Example



Requirements

* Each manifest must have the **requestedExecutionLevel** tag set to the appropriate level of execution required for that executable. Levels of execution are:   
  + level="asInvoker"
  + level="requireAdministrator"
  + level="highestAvailable"
* The **requestedExecutionLevel** tag in each manifest must have the attribute **uiAccess** set to **false**.
* The application’s main executable **requestedExecutionLevel** tag must have **level="asInvoker"** in order to pass this test case. That means that the application’s main executable itself must be able to be run as Least-Privilege.
* If any of the application’s executables use an external manifest, the execution level resource for each must be set appropriately. To verify the external manifest:  
  + Browse the application’s directories for **<executable\_name>.exe.manifest**.
  + Open the manifest in Notepad.
  + The execution level resource must be set as follows in order to pass this test case:  
      
    <security>

<requestedPrivileges>

<requestedExecutionLevel level="asInvoker" uiAccess="false"/>

</requestedPrivileges>

</security>

Notes

* Any application that is required to run with the user logged in as an administrator or requires an elevated privilege to run properly must receive a waiver from Microsoft in order to pass this requirement. In this case the **requiredExecutionLevel** tag must be set to **requireAdministrator** or **highestAvailable**. The process for applying for these waivers is to be determined.
* Applications that require input from the UI of any other window - regardless of the relationship to the application - may set the **uiAccess** attribute to **true** and will need to apply for the waiver from Microsoft in order to pass this requirement. The process for applying for these waivers is to be determined.

Additional Information

Applications must follow the best practices for working with User Access Control (UAC) as described on MSDN:

<http://msdn.microsoft.com/windowsvista/default.aspx?pull=/library/en-us/dnlong/html/AccProtVista.asp>

Applications that attempt to circumvent UAC by modifying the Access Control Lists (ACLs) of the system will fail this requirement.

## Test 2: Driving Signing and Verification

This test comprises of three sub-tests.

Test 2.1:  
Verify that all kernel-mode drivers are signed.

Criteria

*In order to pass this test:*

All kernel-mode drivers installed by the application must contain a valid Microsoft signature and should not cause Windows to alert the user with a pop-up message regarding the installation of an unsigned driver.

**Note: This test only applies to applications which install Kernel Drivers.**

Rationale

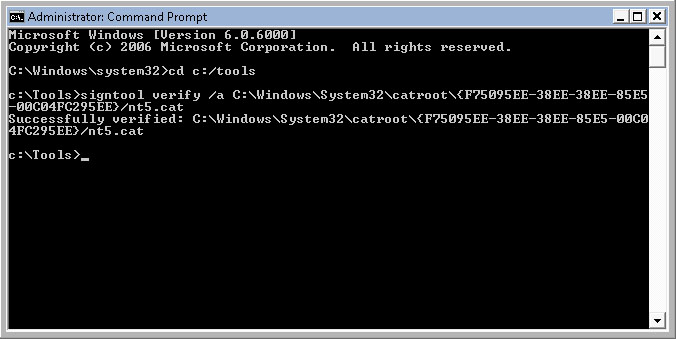
Poorly written or malware drivers can severely affect the stability and security of a system. On 64-bit versions of Windows Vista, unsigned drivers will not be able to load.

Verification Process

For each Qualifying Software:

1. Open an elevated command window (click **Start** / **All Programs** / **Accessories**, then right-click on **Command Prompt** and select **Run as administrator**).
2. For **each** Kernel Mode Driver installed by the application, use **SignTool**(See Appendix 1: Test Machine Setup and Configuration) to verify the signature of each driver in the following manner:  
   1. If the driver's signed catalog file (**.cat**) is not installed in   
       **%WINDIR%\System32\Catroot**   
        
      type:   
        
      **signtool verify /c <Name\_of\_Catalog\_File> <Driver\_Name>**
   2. If the driver's .cat file is installed in   
       **%WINDIR%\System32\Catroot**   
        
      type :  
       **signtool verify /a <Driver\_Name>**
   3. To pass this test, all drivers must have a verified signature.

Example



Requirements

In a Windows Logo Program (formally WHQL) signature, the command window output under the Signing Certificate Chain must display **Microsoft Windows Hardware Compatibility Publisher** in the **Issued to:** line and **Microsoft Windows Hardware Compatibility PCA** in the **Issued by:** line. If so, the kernel-mode driver has passed the WHQL Certification and has obtained a valid Microsoft signature from the WHQL program. The Certification Path will also have these names under the signature Root Authority.

In the Unclassified Driver Signature program signature the command window output under Signing Certificate Chain must display **Microsoft Windows Driver Reliability Publisher** in the **Issued to:** line and **Microsoft Windows Driver Reliability PCA** in the **Issued by:** line. That means that the kernel-mode driver has passed the Driver Reliability Certification and has obtained a valid Microsoft signature from the Unclassified Driver Program. The Certification Path will also have these names under the signature Root Authority.

Any kernel-mode drivers installed by the application must have a Microsoft signature obtained through the either of the above programs in order to pass this test case.

Notes

1. This test is also covered by the Windows Vista Logo Program. For more information, see [Requirements for the Windows Vista Logo Program for Software](https://partner.microsoft.com:443/US/40018567).
2. To speed up the verification process, use the script contained in Appendix 3 to automate the procedure.

Test 2.2:  
Driver Verify with Switches.

Criteria

*In order to pass this test:*

All drivers that make up the application must pass the Driver Verification process.

Rationale

To ensure the quality of the Windows Vista experience, drivers must pass the Driver Verification process.

Verification Process

For each Qualifying Software:

* 1. Boot the system and log in as an administrator.
  2. Open a command window, and enter verifier /standard /all to enable the Driver Verifier with the standard settings applied to all drivers.
  3. Shut down and restart the system.
  4. If the system crashes or fails, reboot into safe mode by holding F8 during the boot process, and save the kernel memory dump (c:\windows\memory.dmp) for further analysis.
  5. If the system doesn’t crash, the system has passed the test.
  6. Open a command window, and enter verifier /reset to clear all the Driver Verifier settings.
  7. Shut down and restart the system.

Requirements

In order to pass this test, the system must not crash or fail when the above Driver Verifier option is used.

Test 2.3:  
Boot drivers contain Embedded Signatures.

Criteria

*In order to pass this test:*

All boot-start drivers installed by the application, for both 32-bit and 64-bit versions of Windows Vista should have embedded signatures in the driver image file.

Rationale

Embedded signatures in boot-start driver image files optimize performance by eliminating the need to locate the appropriate catalog file when the operating system loader verifies the driver signature.

Verification Process

For each Qualifying Software:

1. Open an elevated command window (click **Start** / **All Programs** / **Accessories**, then right-click on **Command Prompt** and select **Run as administrator**).
2. For **each** boot-start driver installed by the application, use **SignTool**(See Appendix 1: Test Machine Setup and Configuration) to verify the signature of each driver in the following manner:  
     
   **signtool verify /kp /v <driver.sys>**  
   Where <driver.sys> is the name of the boot-start driver.
3. To pass this test, SignTool must report all drivers as successfully verified.

Notes

SignTool arguments:  
  
**verify**  
Verifies the signature in the specified file.

**/kp**

Verifies that the kernel policy has been met.

**/v**

Specifies the verbose option, which displays successful execution and warning messages.

**FileName**

Specifies the name of the binary file that contains the signature to be verified.

Additional Information

An MSDN topic that explains usage of **SignTool**:

<http://msdn2.microsoft.com/en-us/library/aa906251.aspx>

An MSDN topic describing the Kernel-Mode Code Signing Policy (Windows Vista):

<http://msdn2.microsoft.com/en-us/library/aa906239.aspx>

A Whitepaper covering the use of Digital Signatures for Kernel Modules:

<http://www.microsoft.com/whdc/winlogo/drvsign/kmsigning.mspx>

## Test 3: Verify that the application checks for the OS version in the approved way.

Criteria

*In order to pass this test:*

An application must continue to run if the Operating System (OS) version number increases, unless the End User License Agreement prohibits use on future operating systems.

If the application is supposed to fail, it must do so gracefully by sending a message to the user and writing a message to the NT event log.

Applications must use the version-checking APIs (GetVersionEx) to check the OS version. They must not read the following the registry keys for the OS version:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion

Rationale

When Windows users upgrade their operating system, they shouldn’t be blocked from using their current applications simply because the OS version has increased.

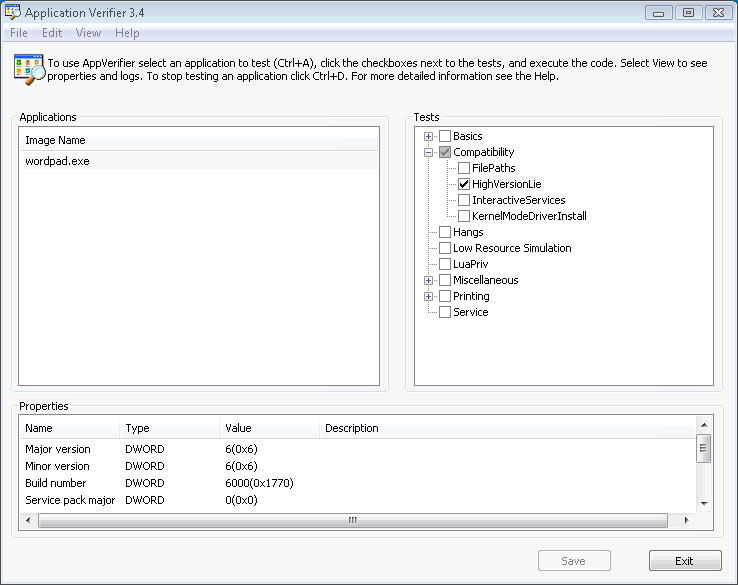
Verification Process

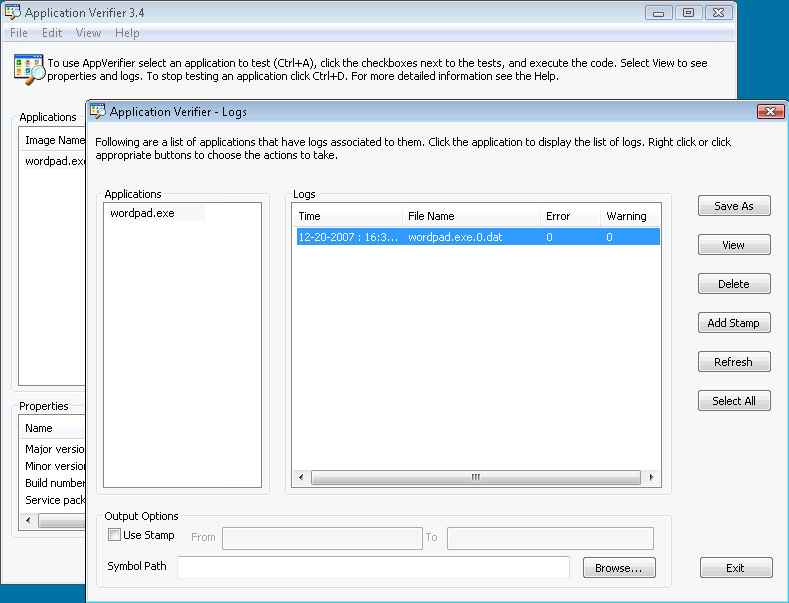
For each Qualifying Software:

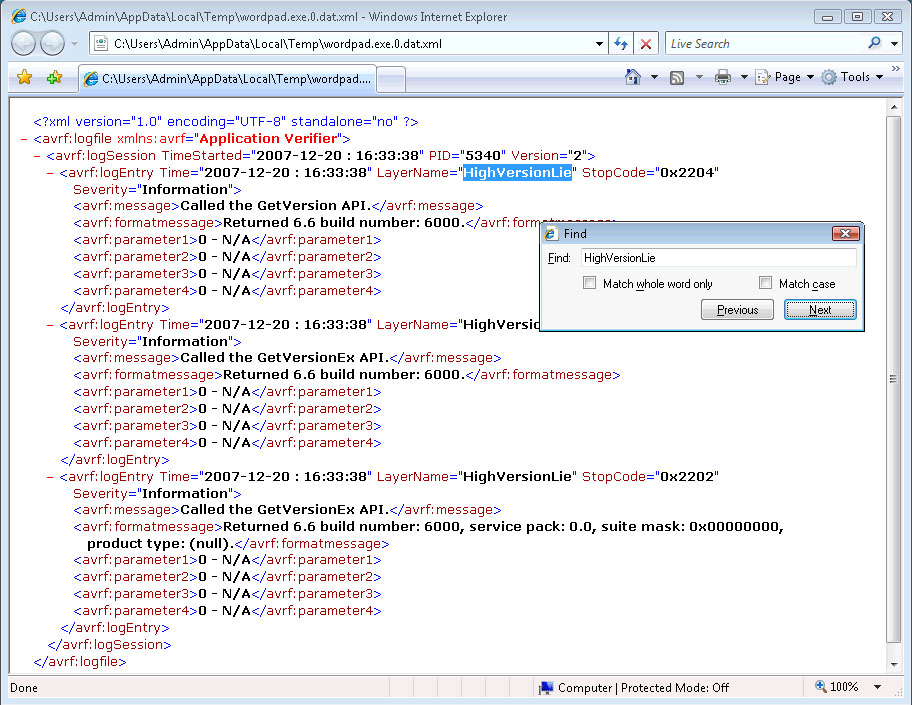
1. Browse the application install paths for all executables which end with the **.exe** extension.
2. For **each** of the application’s executables above:  
   1. Add the executable to AppVerifier (Select **Add Application** from the **File** menu) and open the **Property Window** from the **View** menu.
   2. Uncheck the **Basic** tests.
   3. Check **HighVersionLie** from **Compatibility** tests.
3. Highlight **HighVersionLie**.
4. Double-click **Major version** and enter **6**.
5. Double-click **Minor Version** and enter **0**.
6. Double-click **Build number** and enter the build number of Windows Vista you are using to perform the tests.
7. Double-click **Service pack major number** and enter **0**
8. Double-click **Service pack minor number** and enter **0**

Launch executable with **AppVerifier** running.

* 1. Perform a normal operation to determine whether the application is running.  
       
     **Note: The steps involved in normal operation must be submitted as part of the application testing submission**.
  2. Exit the application, and view **AppVerifier** Logs for the executable
     1. Search each log for **HighVersionLie**
  3. Change the **major version number** to **10** in **AppVerifier** and repeat steps d – f.

Example

Example (continued)



Requirements

1. The **AppVerifier** logs must show that **GetVersion** or **GetVersionEx API** were used when checking the version of Windows. The installer, and each executable that ships as part of the application, must check the version of Windows in this way.

Notes

1. The **AppVerifier** logs may show **HighVersionLie**; but contain no data, which indicates the application did not attempt to check the OS version. This will pass this test case.
2. You can obtain the Windows Vista build number by typing **winver** into the Command prompt.
3. **GetVersion** has been superseded by **GetVersionEx**. All new software should use **GetVersionEx**.

Additional Information

For more information, see the [MSDN reference topic for GetVersionEx](http://msdn2.microsoft.com/en-us/library/ms724451.aspx).

## Test 4: Verify that the application creates uninstall Registry entries.

Criteria

*In order to pass this test:*

The application must create the correct registry settings to allow it to be uninstalled.

Rationale

A broken uninstall process can frustrate the user, and may prevent a successful re-install.

Verification Process

For each Qualifying Software

1. Launch the **Orca** tool (See Appendix 1: Test Machine Setup and Configuration).
2. If MSI based install:

Open the application’s **MSI installer package** in Orca

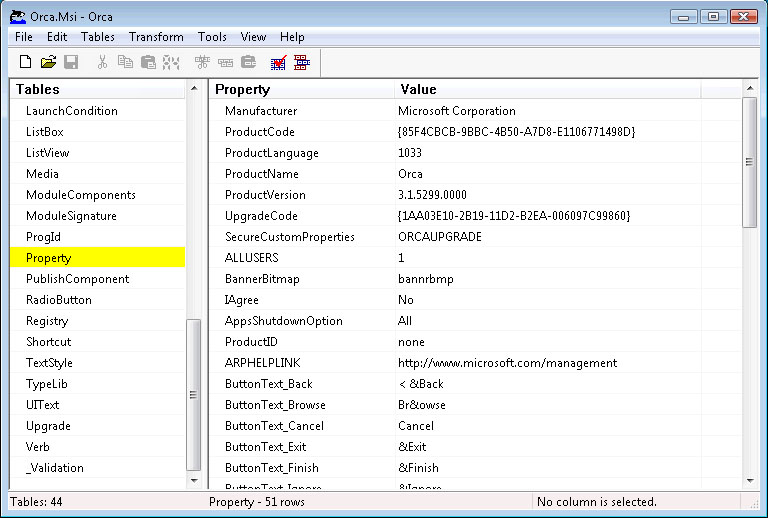
1. In the table tree on the left select **Property**.
2. In the right hand pane look at **ProductCode**.
3. Make note of the **ProductCode**.

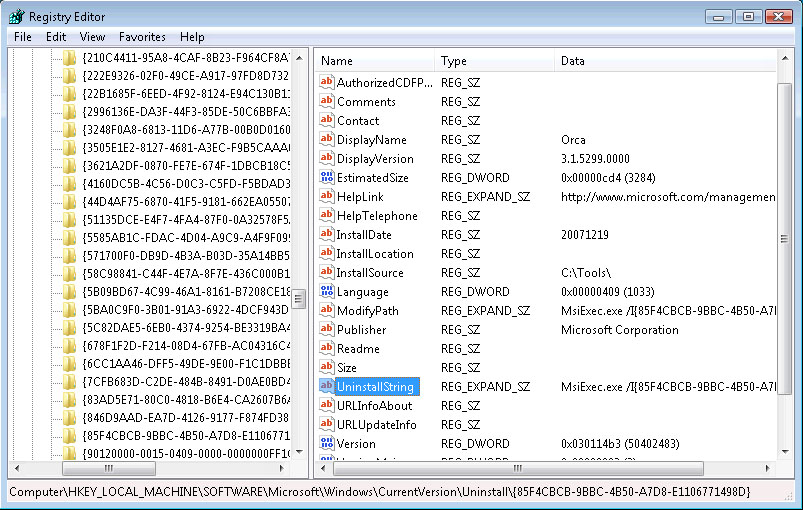
**If not MSI based install, proceed to step 3**

1. Once the installation is complete, open the registry by typing **RegEdit** from the command line.
2. Navigate to the following registry key:
3. HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\
4. If MSI install locate the **ProductCode** key from step 2, and select it, otherwise select the appropriate application based on application name.

1. Make note of all the following entries in that key:  
   1. **DisplayName**
   2. **InstallLocation**
   3. **Publisher**
   4. **UninstallString**
   5. **VersionMajor**
   6. **VersionMinor**

Example





Requirements

In order to pass this text case, all the registry entries listed above must match the information for the application and vendor. Registry keys set to **null** are not permitted.

Notes

* 1. The uninstall registry key location for win32 applications installed on x64 versions of Windows Vista is: HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall
  2. The uninstall registry key location for native 64-bit applications installed on x64 versions of Windows Vista is: HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\

## Test 5: Verify that the application passes specific AppVerifier checks.

Criteria

*In order to pass this test:*

The application’s executable files must run when attached to AppVerifier running specific checks without breaking into the debugger.

**Note: This test case is not applicable to fully Managed applications.**

Rationale

This test looks for memory leaks, unhandled exceptions and other issues which may cause an application fail.

Verification Process

For each Qualifying Software:

1. Log on to machine as an administrator.
2. Launch the Windows Debugger(See Appendix 1: Test Machine Setup and Configuration) to the machine.  
   1. Open an elevated command window (click **Start** / **All Programs** / **Accessories**, then right-click on **Command Prompt** and select **Run as administrator**).
   2. Change directories to where Debugging Tools for Windows is installed.
   3. Start the Windows Debugger by typing **windbg –I (case sensitive)**(**Note: For more detailed information see Windbg and AppVerifier help**).
3. Browse the application install paths for all executables which end with an **.exe** extension.
4. For **each** of the application’s executables above:

Attach the executable to AppVerifier and check the following AppVerifier Checks:

* + 1. **Exceptions**, **Handles**, **Heaps**, **Locks**, **Memory** and **TLS** from the **Basics checks**.
    2. **DangerousAPIs** and **DirtyStacks** from the **Miscellaneous checks**.

1. For **each** of the application’s **main** executables:  
   1. Launch the application executable.
   2. Perform the normal operations fro the executible.  
         
      (**Normal operation steps are required to be submitted along with the application for testing**).
   3. Close the application executable.

Requirements

In order to pass this test, all the application’s executables ending with an **.exe** extension must run without breaking into the debugger while the executable is attached to AppVerifier configured as above.

Notes

1. Save all the AppVerifier logs which generated errors. This will enable the ISV to gather the needed data to fix those errors that broke into the debugger.
2. When applicable copy and save the stack trace from the Windows Debugger (**windbg**). This will give the ISV more information as to where the break resides.
3. Every error that caused the application to break into the debugger must be fixed in order to pass this test.
4. This test case is not applicable to fully managed applications.

## Test 6: Verify that the application installs to the correct folders by default.

Criteria

*In order to pass this test:*

Applications should be installed to Program Files or the user’s AppData folder by default.

For per-machine installations, user data must be written at first run and not during the install. Since the install potentially elevates to a different user account during the per-machine install, there is no correct user location to store data at install time.

Rationale

Users should have the flexibility to install applications where they need them and should have a consistent and secure experience with the default location of files.

Verification Process

For each Qualifying Software:

1. Use an imaging tool to take a snapshot of the machine.
2. Install the application using custom values and selecting to install all components.
3. Once install is complete, use an imaging tool to take a snapshot of the machine.
4. Compare images and ensure that application is only installing to folders indicated below.

Requirements

In order to pass this test, the application only installs files to a subdirectory of Program Files or the user’s AppData directories.

Notes

1. Per-machine installs must install to **Program Files** by default in order to pass this test case.
2. Per-machine installs have no correct user **AppData** directory because elevation may be needed: user data must be written when the application runs for the first time, and not during installation.
3. Default install directories are:   
   1. Native x86 and x64 applications

|  |  |
| --- | --- |
| Program Files | %ProgramFiles% |
| User’s AppData | %APPDATA% |
| ProgramData | %ALLUSERSPROFILE% |

* 1. Win32 applications on x64

|  |  |
| --- | --- |
| Program Files | %ProgramFiles(x86)% |
| User’s AppData | %APPDATA% |
| ProgramData | %ALLUSERSPROFILE% |

1. If the application installs drivers and/or Windows redistributive components, they may be installed to existing Windows directories.

## Test 7: Verify that the application is Restart Manager Aware.

Criteria

*In order to pass this test:*

The application must deal appropriately with Shutdown and (optionally) Restart system messages.

**Note: The application must be idle and not performing any operations during this test.**

Rationale

To avoid disrupting a Windows Restart or Shut Down request, an application must close when the system instructs it to do so, and if expected, it must restart automatically.

Verification Process

For each Qualifying Software:

1. Launch the application.
2. Open the **Windows Task Manager**.
   1. Click on the **Processes** tab
   2. From the **View** menu, select **Select Columns..**
   3. Select the **PID (Process Identifier)** check box and click **OK**.
   4. Click on **Image Name** to list all the current processes by name.
   5. Find the application image name and make a note of its PID.
3. Open a command window.
4. Change directories to the directory that contains the **Restart Manager** tool (it is contained within the **Logo Testing Tools for Windows** directory).
5. From the command window send a shutdown message to the application through the RMShutdown API in the following manner:

|  |  |
| --- | --- |
| If the application is designed to shut down and restart, type: | rmtool.exe –p dwPID –S –R |
| If the application is not designed to restart after shutdown, type: | rmtool.exe –p dwPID –S |
| If your application is a service and is designed to restart after shutdown, type: | rmtool.exe –f $dir\<service> –S –R |
| If your application is a service and is not designed to restart after shutdown, type: | rmtool.exe –f $dir\<service> –S |

Where dwPID is the application's Process ID, obtained via Task Manager, and $dir is the path to the service executable or dll.

6. Open Event Viewer by typing eventvwr from the command line or from the Administrator Tools.

1. Expand **Windows Logs**.
2. Click on **Application**.
3. There will be a **RestartManger (Information)** message for each Shutdown and Restart action performed by Restart Manager on each executable.
   * 1. The shutdown message will contain the executable name and Process ID in the log.
     2. The restart message will contain the executable name and Process ID in the log, with the Process ID being different then the shutdown log. This log will have a later time stamp.

Requirements

In order to pass this test:

* 1. Using the Restart Manager tool, it is possible to shut down the idle application quietly and without user interaction via the RMShutdown API, and – if designed to do so – it restarts automatically.
  2. If the application is a service, it is possible to use the Restart Manager tool to shut it down while idle, quietly and without user interaction, via the RMShutdown API. If the service is designed to restart after shutdown, it restarts.
  3. There must be both an Information message with **Source** listed as **RestartManger** for each shutdown and restart action performed by Restart Manager in the Windows Application Logs on each executable.

Notes

* + - 1. The application must be idle and must not be running or performing any operations while performing this test case.
      2. All applications must listen and respond to shutdown messages using the Restart Manager API quietly, without user interaction while idle in order to pass this test case.
      3. The application or service must not cause an Access Violation and it must shutdown/restart safely.
      4. Restart Manager shutdown messages are:

|  |  |
| --- | --- |
| WM\_QUERYENDSESSION with LPARAM = ENDSESSION\_CLOSEAPP(0x1) | GUI applications must respond (TRUE) immediately to prepare for a restart. |
| WM\_ENDSESSION with LPARAM = ENDSESSION\_CLOSEAPP(0x1) | The application must shutdown within 5 seconds (20 seconds for services). |
| CTRL\_SHUTDOWN\_EVENT | Console applications must shutdown immediately. |

* + - 1. If the application or service is not designed to restart after shutdown, then there will only be an Information message with **Source** listed as **RestartManger** for each shutdown action performed by Restart Manager in the Windows Application Logs.

## Appendix 1: Test Machine Setup and Tools Installation

Follow this procedure to create a standard test machine setup.

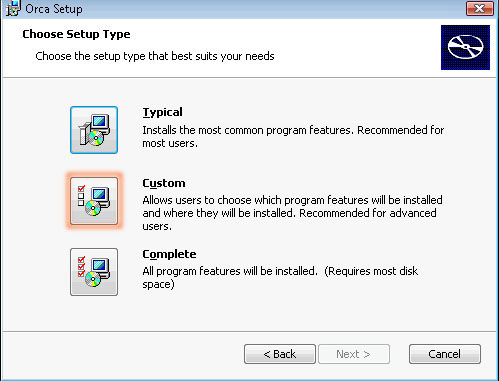
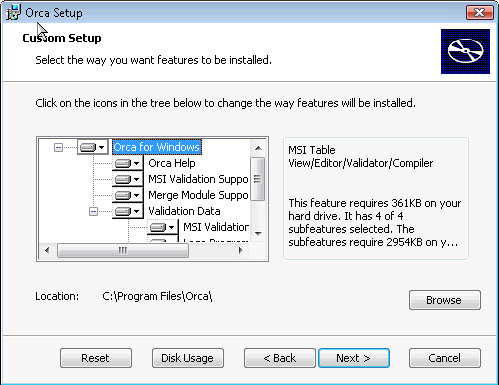
**Please note: This procedure may take some time, and requires internet access to download specific tools. Please ensure you have the ability make a disk image of your finished installation, in order to avoid having to repeat the process.**

****Configuring the Test Machine****

1. Install Windows Vista with all Windows components and options installed. Log into machine as the local administrator.
2. Install the Windows Vista SDK from:  
     
   <http://www.microsoft.com/downloads/details.aspx?FamilyID=e6e1c3df-a74f-4207-8586-711ebe331cdc>

Select all the components, with the exception of the Mobile options, on a machine that is being used to perform the tests.

* 1. Copy the files **capicom.dll**, **orca.msi** and **signtool.exe** from **<SDK\_Install\_Dir>\ Windows\v6.0\Bin\** directory to a temporary location.
  2. Copy the **Schema.msi** Windows Install Package from **<SDK\_Install\_Dir>\ Windows\v6.0\Bin\msitools\Schemas\MSI\** directory to a temporary location.

1. Create a directory called **Tools** on the system’s boot drive (typically, **C:**).
2. Copy **Schema.msi**, **orca.msi**, **signtool.exe** and **Capicom.dll** to the **Tools**directory from the temporary location in step 3.
3. Install **Orca** from **Tools\orca.msi** with all components installed (you will need to select a custom install, and ensure than all components are selected).  
     
     
     
     
     
   

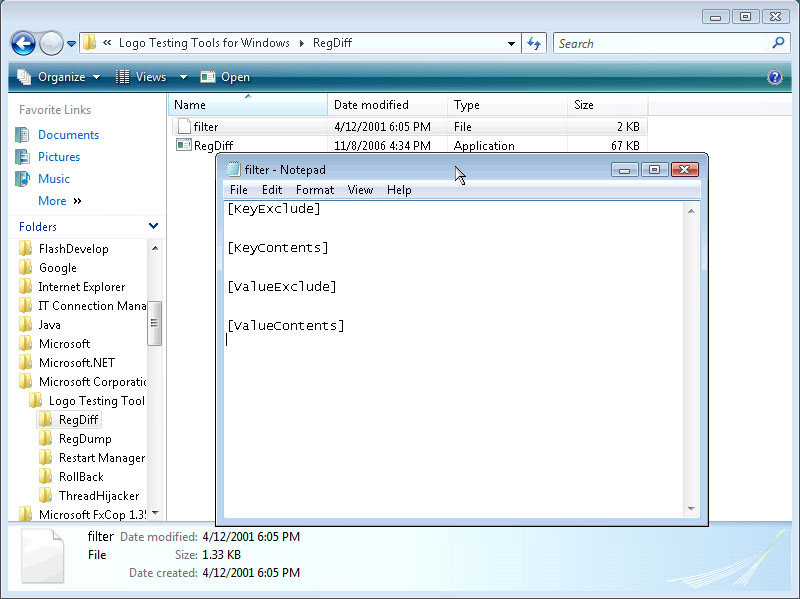
1. Install Vista Logo Tools from  
   <http://download.microsoft.com/download/d/2/5/d2522ce4-a441-459d-8302-be8f3321823c/LogoToolsv1.0.msi>.
2. Go to the directory **<Vista\_Logo\_Tools\_Installed\_Dir>\RegDiff**
   1. Open the file **filter** in Notepad.
   2. Select all the text in the file, and delete it.
   3. Copy and paste the following information as-is:

[KeyExclude]

[KeyContents]

[ValueExclude]

[ValueContents]

* 1. Save as **filter.txt**. You may need to alter the file permissions to be able to replace the original file.  
       
       
     

1. Install the latest released version of AppVerifier from:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=bd02c19c-1250-433c-8c1b-2619bd93b3a2&DisplayLang=en>

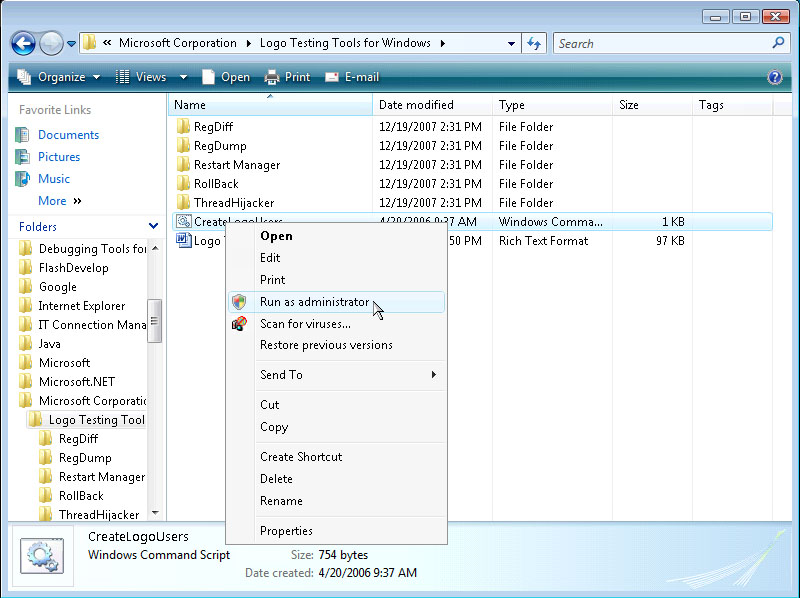
**Note: For native x64 applications install x64 versions of AppVerifier.**

1. Install **Debugging Tools for Windows** from:  
     
   <http://www.microsoft.com/whdc/devtools/debugging/default.mspx>. .

Confirm that you wish to update the currently installed tools.  
  
**Note: For native x64 applications install x64 version of Debugging Tools for Windows.**

1. Install **CFF Explorer** to the Tools directory:  
     
   <http://ntcore.com/exsuite.php>  
     
   Extract and install the appropriate version (x64 or x86).

Note: this is not Microsoft software

1. Navigate to the **Logo Testing Tools for Windows** directory, right-click on **CreateLogoUsers** and select **Run as administrator**.  
     
   This script will create the necessary accounts required for testing:  
   1. Least-Privilege User – logouser1
   2. Least-Privilege User – logouser2
   3. Least Privilege User – vistalogolongusernam (Long User Name)
   4. Administrator – logoadmin
   5. Power User – logopuser
   6. Backup Operator – logobouser  
        
        
      

**Note: The test machine must not be registered to a domain, or this step will fail.**

1. Use an imaging tool to take a snapshot of the machine. This will allow you to re-create the test platform without having to repeat steps 1 to 12.   
     
   Microsoft’s own imaging tool for Vista is called **ImageX:**

<http://technet2.microsoft.com/WindowsVista/en/library/bb068119-1ba6-48c7-9ad7-3ed3f72592e91033.mspx?mfr=true>

## Appendix 2: Resources

**Windows Vista SDK**

<http://www.microsoft.com/downloads/details.aspx?FamilyID=e6e1c3df-a74f-4207-8586-711ebe331cdc>

**Windows Logo Program**

[Requirements for the Windows Vista Logo Program for Software](https://partner.microsoft.com:443/US/40018567)

**User Account Control Guidelines**

<http://msdn.microsoft.com/windowsvista/default.aspx?pull=/library/en-us/dnlong/html/AccProtVista.asp>

**Windows Installer Information**

<http://go.microsoft.com/fwlink/?LinkId=3985>

**Also see:**

[Sending Messages to Windows Installer Using MsiProcessMessage](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/sending_messages_to_windows_installer_using_msiprocessmessage.asp)

[Returning Error Messages from Custom Actions](http://msdn.microsoft.com/library/en-us/msi/setup/returning_error_messages_from_custom_actions.asp)

[Deferred Execution Custom Actions](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msi/setup/deferred_execution_custom_actions.asp)

[Changing the System State Using a Custom Action](http://msdn.microsoft.com/library/en-us/msi/setup/changing_the_system_state_using_a_custom_action.asp)

[Organizing Applications into Components](http://msdn.microsoft.com/library/en-us/msi/setup/organizing_applications_into_components.asp)

[Changing the Component Code](http://msdn.microsoft.com/library/en-us/msi/setup/changing_the_component_code.asp)

[What happens if the component rules are broken?](http://msdn.microsoft.com/library/en-us/msi/setup/what_happens_if_the_component_rules_are_broken.asp)

**Privacy Information**

<http://www.antispywarecoalition.org/documents/index.htm>

**Developer Portal**

<https://winqual.microsoft.com/member/wer/user/default.aspx>

## Appendix 3: Automating the Driver Signature Verification Test

To automate the process of using **SignTool** to verify drivers (**Test 2: Verify that all kernel-mode drivers are signed**), copy and paste the following text into a new file ending with extension .CMD, and execute it from the command window.

@REM Check for embedded signatures in boot drivers

@REM Boot-start drivers without embedded signatures impact Vista boot performance

@echo off

setlocal enabledelayedexpansion

set SC\_QUERY="sc.exe query type= all"

for /F "tokens=1,2" %%i IN ('%SC\_QUERY%') DO (

if "%%i" == "SERVICE\_NAME:" (

@REM @echo %%j

call :chk\_START\_TYPE %%j

)

)

GOTO :EOF

@REM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END MAIN PROCESSING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@REM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*chk\_START\_TYPE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

:chk\_START\_TYPE

set bootdrv=0

set SERV\_NAME=%1

@REM @echo %SERV\_NAME%

for /F "tokens=1,2,3,4" %%m IN ('sc qc %SERV\_NAME%') DO (

if "%%m" == "START\_TYPE" (

if "%%p" == "BOOT\_START" (

@echo ---- FOUND BOOT-START DRIVER: %1

set bootdrv=1

)

)

if "%%m" == "BINARY\_PATH\_NAME" (

if !bootdrv! EQU 1 (

if "%%o"=="" (

echo Unable to find BINARY\_PATH\_NAME for %SERV\_NAME%

) else (

call :chk\_SIGNATURE %%o

)

)

)

)

GOTO :EOF

@REM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*chk\_SIGNATURE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

:chk\_SIGNATURE

set drv\_path=%1

set drv\_path=%SystemRoot%%drv\_path:~11%

@echo !drv\_path!

if EXIST !drv\_path! (

call %~dp0signtool verify /kp !drv\_path!

if !ERRORLEVEL! EQU 1 (

@echo THIS BOOT DRIVER DOES NOT HAVE EMBEDDED SIGNATURE, WILL SLOW BOOT PERFORMANCE!!!

@echo USE SIGNTOOL.EXE WITH YOUR CROSS-CERTIFICATE TO EMBED SIGNATURE INTO THE BINARY

@echo.

@echo.

@echo.

)

)

GOTO :EOF

@REM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Appendix 4: Report Card

To be submitted to Microsoft

|  |  |
| --- | --- |
| **Submission Details** | |
| Application name |  |
| Version number |  |
| Email contact |  |
| Submission date |  |

|  |  |  |
| --- | --- | --- |
| **Test Results** | | |
| **Test Number** | **Pass/Fail** | **Comments** |
| Test 1 |  |  |
| Test 2.1 |  |  |
| Test 2.2 |  |  |
| Test 2.3 |  |  |
| Test 3 |  |  |
| Test 4 |  |  |
| Test 5 |  |  |
| Test 6 |  |  |
| Test 7 |  |  |
| **Normal Operation** | | |
| Please describe the steps taken to demonstrate "normal" operation of the application. |  | |