# Windows Vista Application Development Requirements for User Account Control Compatibility

Microsoft Corporation

Published: September 2006

Updated: February 2007

## Abstract

This document is intended to assist application developers with designing Windows Vista capable applications that are User Account Control compliant. Detailed steps about the design process are included, along with code samples, requirements, and best practices. This paper also details the technical updates and changes to the user experience in Windows Vista.

# Contents

# Windows Vista Application Development Requirements for User Account Control Compatibility

This document contains information to assist application developers with ensuring that their applications are User Account Control (UAC) compatible. Sections in this paper include:

- Why User Account Control? -- Details why UAC was developed.
- How UAC Works -- Details the UAC functionality.
- Will UAC Affect your Application? -- How to determine whether you will have to make your application UAC compatible.
- Designing Applications for Windows Vista -- How to design your application to be UAC compatible.
- Deploying and Patching Applications for Standard Users -- How to ensure that your application can be deployed for standard users.
- Troubleshooting Common Issues -- Lists common development and installation issues that arise in Microsoft .NET applications.
- References -- Includes a virtualization reference and a security settings reference.

## Why User Account Control?

Application developers have consistently created Microsoft Windows® applications that require excessive user rights and Windows privileges, often requiring that the executing user be an administrator. As a result, few Windows users run with the least user rights and Windows privileges required. Many enterprises, seeking to balance ease of deployment and ease of use with security, have often resorted to deploying their desktops as administrator due to standard user application compatibility problems.

The following list details additional reasons why it is difficult to run as a standard user on pre-Microsoft Windows Vista™ computers:

1. Many Windows applications require that the logged on user be an administrator but do not actually require administrator-level access. These applications perform a variety of administrator access checks before being permitted to run, including:

   a. Administrator access token checks.
   b. "All access" access requests in system protected locations.
   c. Data writing to protected locations, such as %ProgramFiles%, %Windir%, and HKEY_LOCAL_MACHINE\Software.
2. Many Windows applications are not designed with the concept of least-privilege and do not separate user and administrator functionality into two separate processes.
3. Windows® 2000 and Windows® XP create each new user accounts as administrators by default; therefore, key Windows components, such as the **Date and Time** and the **Power Management** control panels do not work well for a standard user.
4. Windows 2000 and Windows XP administrators must create two separate user accounts--one for administrative tasks and a standard user account to perform day-to-day tasks. Therefore, users must log off of their standard user accounts and log back in as an administrator or use Run As in order to perform any administrative tasks.

With User Account Control (UAC), Microsoft is providing a technology to simplify deploying standard user desktops in the enterprise and at home.

Building off of the Windows security architecture, as originally designed in the Microsoft Windows NT® 3.1 operating system, the UAC team sought to implement a standard user model that was both flexible and more secure. In previous versions of Windows, one access token was created for an administrator during the logon process. The administrator's access token includes most Windows privileges and most administrative security identifiers (SIDs). This access token ensures that an administrator can install applications, configure the operating system, and access any resource on the computer.

The UAC team took a drastically different approach to designing the access token creation process in Windows Vista. When an administrator user logs on to a Windows Vista computer, two access tokens are created: a filtered standard user access token and a full administrator access token. Instead of launching the desktop (the Explorer.exe process) with the administrator's full access token, the filtered standard user access token is used. All child processes inherit from this initial launch of the desktop, which helps limit Windows Vista's attack surface. By default, all users, including administrators, log on to Windows Vista as standard users.

**Note**

There is one exception to the preceding statement: Guests log onto the computer with fewer user rights and Windows privileges than standard users.

When an administrator user attempts to perform an administrative task, such as installing an application, UAC prompts the user to approve the action. When the administrator user approves the action, the task is launched with the administrator's full administrator access token. This is the default administrator prompt behavior, and it is configurable in the local Security Policy Manager snap-in (secpol.msc) and with Group Policy (gpedit.msc).

**Note**

An administrator account on a Windows Vista computer with UAC enabled is also called an administrator account in Admin Approval Mode. Admin Approval Mode identifies the default user experience for administrators in Windows Vista.

Each administrative elevation is also process specific, which prevents other processes from using the access token without prompting the user for approval. As a result, administrator users have more granular control on what applications install while greatly impacting malicious software that expects the logged on user to be running with a full administrator access token.

Standard users also have the opportunity to elevate within a task flow to perform administrative tasks by using the UAC infrastructure. When a standard user attempts to perform an administrative task, UAC prompts the user to enter valid credentials for an administrator account. This is the default standard user prompt behavior, and it is configurable in the local Security Policy Manager snap-in (secpol.msc) and with Group Policy (gpedit.msc).

## Windows Vista Updates

The following updates are reflective of the cumulative core changes in functionality that have occurred in Windows Vista.

### UAC is Enabled by Default

As a result, you might encounter some compatibility problems with different applications that have not yet been updated for the Windows Vista UAC component. If an application requires an administrator access token (this is indicative from an "access denied" error being returned when you attempt to run the application), you can run the program as an administrator by using the **Run as administrator** option on the context menu (right-click).

### All Subsequent User Accounts are Created as Standard Users

Both standard user accounts and administrator user accounts can take advantage of the UAC enhanced security. On new installations, by default, the first user account created is a local administrator account in Admin Approval Mode (UAC enabled). All subsequent user accounts are then created as standard users.

## Elevation Prompts are Displayed on the Secure Desktop by Default

The consent and credential prompts are displayed on the secure desktop by default in Windows Vista.

## Elevation Prompts for Background Applications are Minimized to the Taskbar

Background applications will automatically prompt the user for elevation on the taskbar, rather than automatically switching to the secure desktop for elevation. The elevation prompt will appear minimized on the taskbar and will blink to notify the user that an application has requested elevation. An example of a background elevation occurs when a user browses to a Web site and begins downloading an installation file. The user then goes to check e-mail while the installation downloads in the background. Once the download completes in the background and the install begins, the elevation is detected as a background task rather than a foreground task. This detection prevents the installation from abruptly stealing focus of the user's screen while the user is performing another task--reading e-mail. This behavior creates a better user experience for the elevation prompt. Information about how application developers can ensure that their applications are not minimized to the taskbar when they request elevation is available later in this document.

## Elevations are blocked in the User's Logon Path

Applications that start when the user logs on and require elevation are now blocked in the logon path. Without blocking applications from prompting for elevation in the user's log on path, both standard users and administrators would have to respond to a User Account Control dialog box on every log on. Windows Vista notifies the user if an application has been blocked by placing an icon in the system tray. The user can then right-click this icon to run applications that were blocked from prompting for elevation as the user logged on. The user can also manage which startup applications are disabled or removed from this list by double-clicking on the system tray icon.

## Built-in Administrator Account is Disabled by Default on New Installations

The built-in administrator account is disabled by default in Windows Vista. If Windows Vista determines during an upgrade from Windows XP that the built-in administrator account is the only active local administrator account, Windows Vista will leave the account enabled and place the account in Admin Approval Mode (UAC enabled). In addition, the built-in administrator account, by default, cannot log on to the computer in safe mode. Please see the following sections for more information.

**Note**

The built-in administrator account is created during setup with the user name **Administrator**.

### Non-Domain Joined

When there is at least one enabled local administrator account, safe mode will not allow the disabled built-in administrator account to logon. Instead, any local administrator account can be used to logon. If the last local administrator account is inadvertently demoted, disabled, or deleted, then safe mode will allow the disabled built-in administrator account to logon for disaster recovery.

### Domain Joined

In all cases on a domain-joined computer, the disabled built-in administrator account cannot logon in safe mode. A user account that is a member of the **Domain Admins** group can log on to the computer to create a local administrator if none exists.

**Note**

If a domain administrator account has never logged on before, then the computer must be started in **Safe Mode with Networking** since Windows Vista will not have cached the user's credentials.

**Note**

Once the computer is disjoined from the domain, it will revert back to the non-domain joined behavior previously described.

## User Account Control and Remote Scenarios

When an administrator logs on to a Windows Vista computer remotely, through Remote Desktop for instance, the user is logged on to the computer as a standard user by default. Remote administration has been modified to be restrictive over a network. This restriction helps prevent malicious software from performing application “loopbacks” if a user is running with an administrator access token.

### Local User Accounts

When a user with an administrator account in a Windows Vista computer's local Security Accounts Manager (SAM) database remotely connects to a Windows Vista computer, the user has no elevation potential on the remote computer and cannot perform administrative tasks. If the user wants to administer the workstation with a SAM account, the user must interactively logon to the computer that he/she wishes to administer.

### Domain User Accounts

When a user with a domain user account logs on to a Windows Vista computer remotely, and the user is a member of the **Administrators** group, the domain user will run with a full administrator access token on the remote computer and UAC is disabled for the user on the remote computer for that session.

## New Default Access Control List (ACL) Settings

The ACLs on certain Windows directories have been changed to enable data sharing and collaboration in data directories and outside of a user's protected directories. A user's protected directory is the user's profile (e.g. C:\Users\Denise\Pictures\), while an example of a data directory is location outside of the operating system partition on a data drive (E.G. D:\Pictures\). Because the root directory, C in this instance, is protected by more restrictive ACLs, users were previously unable to use data directories in earlier versions of Windows Vista.

These ACL changes ensure that users can share and edit files without having to provide approval to a User Account Control dialog box. Additionally, users can now make a folder private. This change ensures that users can still easily maintain data confidentiality and integrity on data drives. These private folders will still be readable by other administrators if they elevate and should be used to keep data private from standard users.

The following table lists the default ACL settings on %systemroot% and on data drives in Windows XP.

**Windows XP %systemroot% and data drive ACL settings**

| User or Group | Access Control Entry |
|---|---|
| BUILTIN\Administrators | Full control |
| NT AUTHORITY\SYSTEM | Full control |
| CREATOR OWNER | Full control |

| User or Group | Access Control Entry |
|---|---|
| BUILTIN\Users | Read<br>Special access: FILE_APPEND_DATA<br>Special access: FILE_WRITE_DATA |
| Everyone | Read |

The following table details the new Windows Vista data drive ACL settings for data drives created with format.exe.

**Windows Vista data drive ACL settings**

| User or Group | Access Control Entry |
|---|---|
| BUILTIN\Administrators | Full control |
| NT AUTHORITY\SYSTEM | Full control |
| NT AUTHORITY\Authenticated Users | Modify |
| BUILTIN\Users | Read and execute<br>Generic read, generic execute |

The following table details the new Windows Vista operating system root (%systemroot%) ACL settings.

**Windows Vista %systemroot% ACL settings**

| User or Group | Access Control Entry |
|---|---|
| BUILTIN\Administrators | Full control |
| NT AUTHORITY\SYSTEM | Full control |
| BUILTIN\Users | Read and execute |
| NT AUTHORITY\Authenticated Users | Modify<br>Append data |
| Mandatory Label\High Mandatory Level | No write |

### New UAC Security Settings and Security Setting Name Changes

The new security settings and security setting name updates are detailed in the Reference section of this document.

# How UAC Works

This section describes the architectural and functional components of UAC for application developers.

## New Technologies for Windows Vista

The following sections detail new technologies for Windows Vista, including the ActiveX® Installer Service, installer detection, standard user patching with Windows Installer 4.0, Security Center integration, User Interface Privilege Isolation, and virtualization.

### ActiveX Installer Service

The ActiveX® Installer Service enables enterprises to delegate ActiveX control installation for standard users. This service ensures that routine business tasks are not impeded by failed ActiveX control installations and updates. Windows Vista also includes Group Policy settings that enable IT professionals to define Host URLs from which standard users can install ActiveX controls. The ActiveX Installer Service consists of a Windows service, a Group Policy administrative template, and some changes in Internet Explorer. The ActiveX Installer Service is an optional component, and will only be enabled on client computers where it is installed.

### Installer Detection

Installation programs are applications designed to deploy software, and most write to system directories and registry keys. These protected system locations are typically writeable only by administrator users; this restriction means that standard users do not have sufficient access to install most programs. Windows Vista heuristically detects installation programs and requests administrator credentials or administrator approval in order to run with access privileges. Windows Vista also heuristically detects updater and un-installation programs. A design goal of UAC is to prevent installations from being executed without the user's knowledge and explicit consent since installations write to protected areas of the file system and registry.

**Important**

When developing new installation programs, much like developing programs for Windows Vista, be sure to embed an application manifest with an appropriate requestedExecutionLevel element. See the Step Six: Create and Embed an Application Manifest with Your Application section for more information. When the requestedExecutionLevel is present in the embedded application manifest, it overrides Installer Detection.

Installer Detection only applies to:

1. 32-bit executables

2. Applications without a requestedExecutionLevel

3. Interactive processes running as a standard user with UAC enabled

Before a 32-bit process is created, the following attributes are checked to determine whether it is an installer:

- Filename includes keywords like "install," "setup," "update," etc.
- Keywords in the following **Versioning Resource** fields: Vendor, Company Name, Product Name, File Description, Original Filename, Internal Name, and Export Name.
- Keywords in the side-by-side application manifest embedded in the executable.
- Keywords in specific StringTable entries linked in the executable.
- Key attributes in the resource file data linked in the executable.
- Targeted sequences of bytes within the executable.

**Note**

The keywords and sequences of bytes were derived from common characteristics observed from various installer technologies.

Ensure that you thoroughly review the entirety of this document, including the Step Six: Create and Embed an Application Manifest with Your Application section.

**Note**

The **User Account Control: Detect application installations and prompt for elevation** setting must be enabled for installer detection to detect installation programs. This setting is enabled by default and can be configured with the Security Policy Manager snap-in (secpol.msc) or with Group Policy (gpedit.msc).

General information and an overview of the Microsoft Windows Installer can be found at MSDN (http://go.microsoft.com/fwlink/?LinkId=30197).

## Patching Applications in a UAC Environment

Microsoft Windows Installer 4.0 was designed with UAC in mind in order to make application installations and patching easier. With the introduction of Windows Installer 4.0, patches can be applied to applications without reinstalling a newer version of the application. This method is ideal when an application is deployed in a per-computer install and patches need to be deployed by a user without requiring an administrator access token. For information about how to create and apply patches and updates to applications, see MSDN (http://go.microsoft.com/fwlink/?LinkId=71492).

## Security Center Integration

When UAC is disabled on a Windows Vista computer, the Security Center creates an alert and prompts the user to re-enable UAC. Security Center displays this alert once the computer has been restarted after the UAC setting change.

## User Interface Privilege Isolation

User Interface Privilege Isolation (UIPI) is one of the mechanisms that helps isolate processes running as a full administrator from processes running as an account lower than an administrator on the same interactive desktop. UIPI is specific to the windowing and graphics subsystem, known as USER, that supports windows and user interface controls. UIPI prevents a lower privilege application from using Windows messages to send input from one process to a higher privilege process. Sending input from one process to another allows a process to inject input into another process without the user providing keyboard or mouse actions.

Windows Vista implements UIPI by defining a set of user interface privilege levels in a hierarchical fashion. The nature of the levels is such that higher privilege levels can send window messages to applications running at lower levels. However, lower levels cannot send window messages to application windows running at higher levels.

The user interface privilege level is at the process level. When a process is initialized, the User subsystem calls into the security subsystem to determine the desktop integrity level assigned in the process's security access token. The desktop integrity level is set by the security subsystem when the process is created and does not change. Therefore, the user interface privilege level is also set by the User subsystem when the process is created and does not change.

All applications run by a standard user have the same user interface privilege level. UIPI does not interfere or change the behavior of window messaging between applications at the same privilege level. UIPI comes into effect for a user who is a member of the administrators group and may be running applications as a standard user (sometimes referred to as a process with a filtered access token) and also processes running with a full administrator access token on the same desktop. UIPI prevents lower privilege processes from accessing higher privilege processes by blocking the behavior listed below.

A lower privilege process cannot:

- Perform a window handle validation of a higher privilege process.
- SendMessage or PostMessage to a higher privilege application window. These application programming interfaces (APIs) return success but silently drop the window message.
- Use thread hooks to attach to a higher privilege process.
- Use Journal hooks to monitor a higher privilege process.
- Perform dynamic link-library (DLL) injection to a higher privilege process.

With UIPI enabled, the following shared USER resources are still shared between processes at different privilege levels:

- Desktop window, which controls the screen surface
- Desktop heap read-only shared memory
- Global atom table
- Clipboard

Painting to the screen is another action that is not blocked by UIPI. Painting to the screen refers to the process of using the Paint method to display content on an external output—a monitor, for example. The USER/graphics device interface (GDI) model does not allow control over painting surfaces; therefore, it is possible for a lower privilege application to paint over the surface region of a higher privilege application window.

**Note**

Because the Windows Shell (the Explorer.exe process) is running as a standard user process, any other process running as standard user can still send the Windows Shell keystrokes. This is the primary reason why an administrator account in Admin Approval Mode is prompted for elevation consent when the user initiates an administrative action, such as double-clicking on a setup file or clicking on a button marked with an elevation shield icon.

## Virtualization

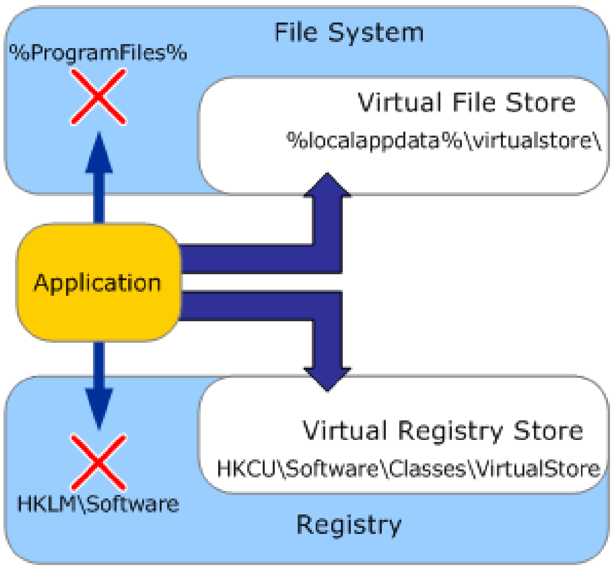
**Important**

Virtualization is implemented to improve application compatibility problems for applications running as a standard user on Windows Vista. Developers must not rely on virtualization being present in subsequent versions of Windows.

Prior to Windows Vista, many applications were typically run by administrators. As a result, applications could freely read and write system files and registry keys. If standard users ran these applications, they would fail due to insufficient access. Windows Vista improves application compatibility for standard users by redirecting writes (and subsequent file or registry operations) to a per-user location within the user's profile. For example, if an application attempts to write to C:\Program Files\Contoso\Settings.ini, and the user does not have permissions to write to that directory, the write will get redirected to C:\Users\Username\AppData\Local\VirtualStore\Program Files\contoso\settings.ini. For the registry, if an application attempts to write to HKEY_LOCAL_MACHINE\Software\Contoso\ it will automatically get redirected to HKEY_CURRENT_USER\Software\Classes\VirtualStore\MACHINE\Software\Contoso or HKEY_USERS\UserSID_Classes\VirtualStore\Machine\Software\Contoso.

The following figure details the virtualization process in Windows Vista. In this example, Denise is an administrator in Admin Approval Mode and Brian is a standard user. Virtualization is comprised of two components: file virtualization and registry virtualization.

**Virtualization process**



**Important**

While developing Windows Vista programs, to reduce the complexity of virtualized files and registry keys, be sure to embed an application manifest with an appropriate requestedExecutionLevel in order to turn off file and registry virtualization.

Virtualization is only enabled for the following:

- 32-bit interactive processes
- Administrator writeable file/folder and registry keys

Virtualization is disabled for the following:

- 64-bit processes
- Non-interactive processes
- Processes that impersonate
- Kernel mode callers
- Executables that have a requestedExecutionLevel

Virtualization and roaming:

- Virtualization files/folders and registry keys do not roam
- Associated with global objects that do not roam

### File Virtualization

File virtualization addresses the situation where an application relies on the ability to store a file, such as a configuration file, in a system location typically writeable only by administrators. Running programs as a standard user in this situation might result in program failures due to insufficient levels of access.

When an application writes to a system location only writeable by administrators, Windows then writes all subsequent file operations to a user-specific path under the Virtual Store directory, which is located at %LOCALAPPDATA%\VirtualStore. Later, when the application reads back this file, the computer will provide the one in the Virtual Store. Because the Windows security infrastructure processes the virtualization without the application's assistance, the application believes it was able to successfully read and write directly to Program Files. The transparency of file virtualization enables applications to perceive that they are writing and reading from the protected resource, when in fact they are accessing the virtualized version.

**Note**

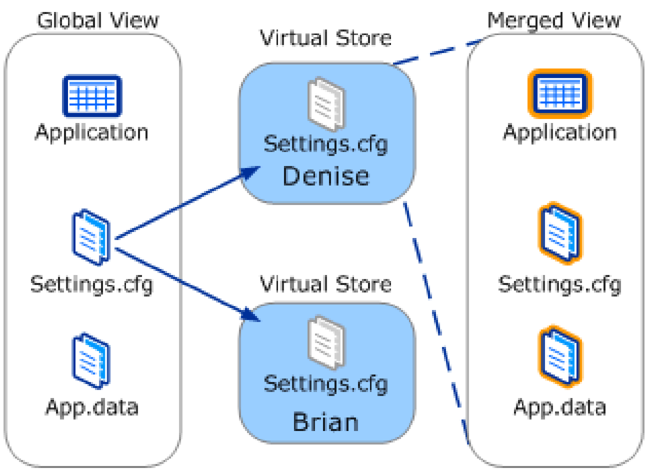
When you enumerate resources in folders and in the registry, Windows Vista will merge global file/folder and registry keys into a single list. In this merged view, the global (protected) resource is listed along with the virtualized resource.

**Important**

The virtual copy will always be present to the application first. For example, config.ini is available in \Program Files\ApplicationName\config.ini and %LOCALAPPDATA%\VirtualStore\config.ini, and the config.ini in the virtual store will always be the one read, even if \Program Files\ApplicationName\config.ini is updated.

The following figure details how global and merged views for virtualized resources are displayed for different users.

**Virtualized resources and views**



The following is an example of the file virtualization process:

Syed Abbas, a sales representative at Woodgrove Bank, is running as a standard user on a computer that he shares with other sales representatives. Syed often uses a spreadsheet application to update and save a file under the Program Files\SalesV1\ directory: \Program Files\SalesV1\SalesData.txt. Although Program Files\SalesV1\ is protected, the file will be saved successfully from the spreadsheet application's point-of-view because of Windows Vista file virtualization. To achieve this, the file write is redirected to Users\Username\AppData\Virtual Store\Program Files\SalesV1\SalesData.txt. When Syed opens Windows Explorer and browses to the Program Files directory, he will see the global view of the SalesData.txt file.

**Note**

For Syed to discover his virtualized files, he must navigate to the virtual store with the **Compatibility files** button on the Explorer toolbar.

However, after Stuart Munson, another sales representative, logs on to the same workstation Syed uses, he will NOT see the file SalesData.txt in the Program Files\SalesV1\ directory. If a different user uses the computer and writes to the \Program files\SalesV1\SalesData.txt file, that write will also be virtualized to that user's virtual store. The files Syed updates and saves will remain independent of the other virtualized files on the computer.

### Registry Virtualization

Registry virtualization is similar to file virtualization but applies to registry keys under HKEY_LOCAL_MACHINE\SOFTWARE. This feature permits applications that rely on the ability to store configuration information in HKEY_LOCAL_MACHINE\SOFTWARE to continue to when they are run under a standard user account. The keys and data are redirected to HKEY_CLASSES_ROOT\VirtualStore\SOFTWARE. As in the file virtualization case, each user has a virtualized copy of any values that an application has stored in HKEY_LOCAL_MACHINE.

#### Registry Virtualization Details

- Can be turned on/off on individual keys in the Software hive
- New FLAGS option in reg.exe for key level virtualization control: Allows recursive enable/disable of virtualization and control of “open access right policy”
- ZwQueryKey: Programmatically query the virtualization flags for a key.
- Virtualization happens on top of WoW64 redirection
- Enabled both in the 64-bit and 32-bit registry views: HKEY_USERS\UserSID_Classes\VirtualStore\Machine\Software and HKEY_USERS\UserSID_Classes\VirtualStore\Machine\Software\SYSWOW3264
- Most pre-Windows Vista 32-bit apps will use the 32-bit view

### Virtualization Recommendations

Virtualization is intended only to assist in application compatibility with existing programs. Applications designed for Windows Vista should NOT perform writes to sensitive system areas, nor should they rely on virtualization to provide redress for incorrect application behavior. When updating existing code to run on Windows Vista, developers should ensure that, during run-time, applications only store data in per-user locations or in computer locations within %allusersprofile% (CSIDL_COMMON_APPDATA) that have access control list (ACL) settings properly set.

**Important**

Microsoft intends to remove virtualization from future versions of the Windows operating system as more applications are migrated to Windows Vista. For example, virtualization is disabled on 64-bit applications.

The following list details other file and registry virtualization recommendations:

- Add an application manifest with an appropriate requestedExecutionLevel for your interactive applications. This will turn virtualization off for the manifested application.
- Do not use the registry as an inter-process communication mechanism. Services and user applications will have different views of the registry key.
- Test your application on Windows Vista: Ensure that processes running as standard user do not write to global namespaces like %systemroot%.
- For filter driver developers: Check your altitude range (http://go.microsoft.com/fwlink/?LinkId=71503). See File System Filters and fltmc.exe (http://go.microsoft.com/fwlink/?LinkId=71504). These must be higher than FSFilter virtualization.
- Remember that virtualized resources are per-user copies of global resources.

## Access Token Changes

When a user logs on to a Windows Vista computer, Windows looks at the administrative Windows privileges and Relative IDs (RIDs) that the user account possesses to determine if the user should receive two access tokens (a filtered access token and a full access token). Windows will create two access tokens for the user if either of the following is true:

1. The user’s account contains any of the following RIDs:
   - DOMAIN_GROUP_RID_ADMINS
   - DOMAIN_GROUP_RID_CONTROLLERS
   - DOMAIN_GROUP_RID_CERT_ADMINS
   - DOMAIN_GROUP_RID_SCHEMA_ADMINS
   - DOMAIN_GROUP_RID_ENTERPRISE_ADMINS
   - DOMAIN_GROUP_RID_POLICY_ADMINS

- DOMAIN_ALIAS_RID_ADMINS
- DOMAIN_ALIAS_RID_POWER_USERS
- DOMAIN_ALIAS_RID_ACCOUNT_OPS
- DOMAIN_ALIAS_RID_SYSTEM_OPS
- DOMAIN_ALIAS_RID_PRINT_OPS
- DOMAIN_ALIAS_RID_BACKUP_OPS
- DOMAIN_ALIAS_RID_RAS_SERVERS
- DOMAIN_ALIAS_RID_PREW2KCOMPACCESS
- DOMAIN_ALIAS_RID_NETWORK_CONFIGURATION_OPS
- DOMAIN_ALIAS_RID_CRYPTO_OPERATORS

2. The user's account contains any Windows privileges other than those of a standard user account. A standard user account contains only the following Windows privileges:
   - SeChangeNotifyPrivilege
   - SeShutdownPrivilege
   - SeUndockPrivilege
   - SeIncreaseWorkingSetPrivilege
   - SeTimeZonePrivilege

**Note**

What Windows privileges the filtered access token contains are based on whether the original access token contained any of the restricted RIDS listed above. If any of the restricted RIDs were in the access token, all of the Windows privileges are removed except:

SeChangeNotifyPrivilege

SeShutdownPrivilege

SeUndockPrivilege

SeReserveProcessorPrivilege

SeTimeZonePrivilege

If no restricted RIDs were in the access token, only the following Windows privileges are removed:

SeCreateTokenPrivilege

SeTcbPrivilege

SeTakeOwnershipPrivilege

SeBackupPrivilege

SeRestorePrivilege

SeDebugPrivilege

SeImpersonatePrivilege

SeRelabelPrivilege

The first access token, called the filtered access token, has the previous RIDs (if present) marked as USE_FOR_DENY_ONLY in the access token and the administrative Windows privileges, not listed previously, removed. The filtered access token will be used by default when the user launches applications. The unmodified full administrator access token is then attached to the filtered access token and is used when requests are made to launch applications with a full administrator access token.

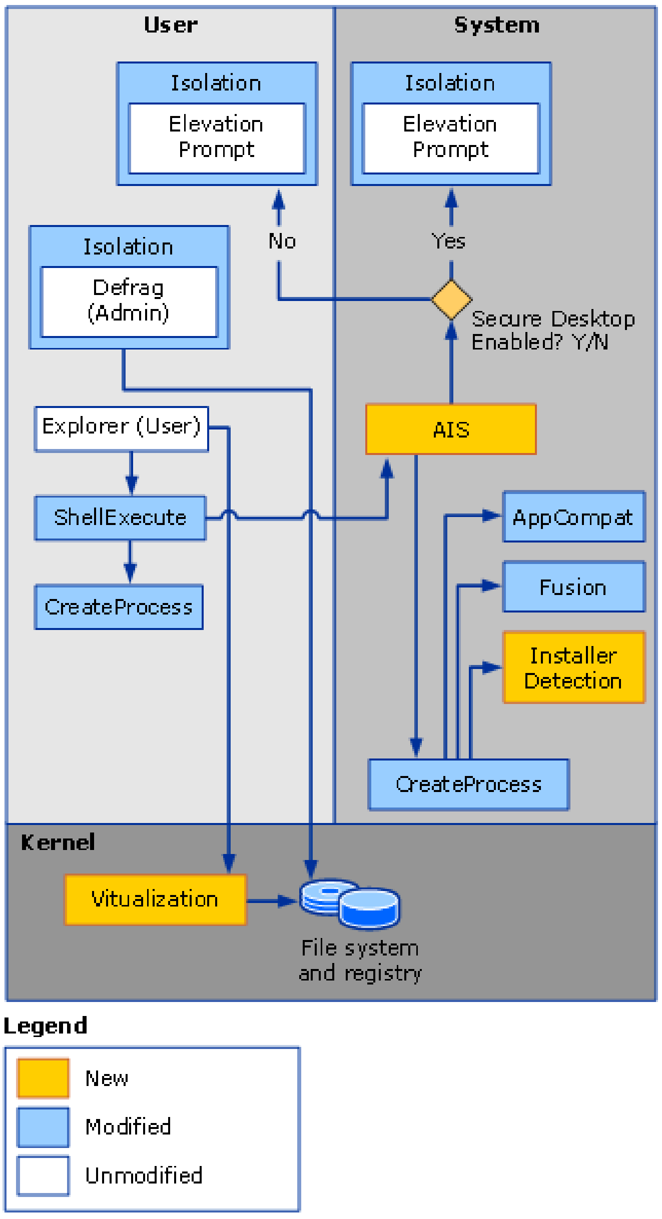
More information on RIDs can be found at MSDN (http://go.microsoft.com/fwlink/?LinkId=71494).

More information on Windows privileges can be found at MSDN (http://go.microsoft.com/fwlink/?LinkId=71495).

## UAC Architecture

The following diagram represents the process flow for executable launches in Windows Vista.

**UAC architecture**



The following is a description of the process flow displayed in the UAC architecture diagram and how UAC is implemented when an executable attempts to launch.

### Standard User Launch Path

The Windows Vista standard user launch path is similar to the Windows XP launch path, but includes some modifications.

1. ShellExecute() calls CreateProcess().
2. CreateProcess() calls AppCompat, Fusion, and Installer Detection to assess if the application requires elevation. The executable is then inspected to determine its requestedExecutionLevel, which is stored in the executable's application manifest. The AppCompat database stores information for an application's application compatibility fix entries. Installer Detection detects setup executables.
3. CreateProcess() returns a Win32 error code stating ERROR_ELEVATION_REQUIRED.
4. ShellExecute() looks specifically for this new error and, upon receiving it, calls across to the Application Information Service (AIS) to attempt the elevated launch.

### Elevated Launch Path

The Windows Vista elevated launch path is a new Windows launch path.

1. AIS receives the call from ShellExecute() and reevaluates the requested execution level and Group Policy settings to determine if the elevation is allowed and to subsequently define the elevation user experience.
2. If the requested execution level requires elevation, AIS launches the elevation prompt on the caller's interactive desktop (based on Group Policy), using the HWND passed in from ShellExecute().
3. Once the user has given consent or valid administrator credentials, AIS will retrieve the corresponding access token associated with the appropriate user, if necessary. For example, an application requesting a requestedExecutionLevel of highestAvailable will retrieve different access tokens for a user that is only a member of the Backup Operators group than for a member of the local Administrators group.
4. AIS reissues a CreateProcessAsUser() call, supplying the administrator access token and specifying the caller's interactive desktop.

## Will UAC Affect your Application?

Whether or not your application will be affected by UAC depends on the application's current state. In a number of cases, no changes will be necessary to comply with Microsoft Windows® Security requirements. However, some applications, including line of business (LOB) applications, may require changes to their install, function, and update processes to properly work in a Windows Vista UAC environment.

**Note**

If an application works well as standard user on Windows XP, then it will work well as a standard user on Windows Vista.

## Why Do I Need to Remove My Application’s Administrative Dependencies?

One fundamental step toward increasing the security of the overall computing environment is to allow users to run without using their administrator access token. If an application only operates or installs when the user is an administrator, users are being forced to run applications with unnecessary elevated access. The fundamental problem is that, when users are always forced to run applications using elevated access tokens, deceptive or malicious code can easily modify the operating system, or worse, affect other users.

Microsoft’s goal is for customers to understand that applications should not unnecessarily run as an administrator and for users to question any time they are asked to approve an application’s request to run as an administrator. UAC is a fundamental component for helping to achieve this goal.

### Reducing Your Application's Total Cost of Ownership

The standard user account is very attractive to information technology (IT) administrators interested in increasing security and control over their managed computers while reducing total cost of ownership (TCO). Because a standard user account cannot make system changes, there is a direct relationship to the reduction of TCO and better management of application installation and system-wide modifications. The standard user account is also attractive to home users since many parents share a computer with their children. Microsoft Windows Vista includes integrated parental controls, which are only successfully implemented by creating children's user accounts as standard users. Standard user accounts also cannot change or delete files created by other users. They cannot read files in other users’ profiles, infect system files, or alter system-shared executables, either accidentally or deliberately. Standard user accounts result in an overall improvement in computer security and parental controls.

## Secure by Default

At Microsoft, the tenets of Microsoft's Trustworthy Computing Initiative have been ingrained into software development. Consequently, improved security has been an integral part of the Windows Vista development process.

The security pillar of Trustworthy Computing encompasses three fundamentals: secure by design, secure by default, and secure in deployment. How you and other independent software vendors (ISVs) develop your applications to contribute to the overall security of the operating system will be a key success factor for achieving Trustworthy Computing in Windows Vista.

The goal of the remainder of this guide is to help assist application developers with learning how to do the following:

- Write applications that do not require the user to be an administrator to perform routine tasks.
- Create installation packages with Windows® Installer 4.0 UAC patching technologies that deploy well to the standard user desktop in enterprises and also update correctly in the home.
- Identify standard user and administrative functionality and extrapolate administrative tasks for UAC compatibility
- Write application user interfaces that utilize the UAC functionality

It is essential for the success of UAC that application developers embrace the philosophy of least-privilege and design their applications to function correctly when running with a standard user account.

One of the goals of the Windows Vista release is to evangelize and encourage the principle of designing for standard users and administrators in Admin Approval Mode to all developers. Achieving this goal will assist in the prevention of various attacks against individual applications and mitigate the possibility that such attacks will compromise the security of the system. Although these goals can be accomplished in some degree today by requiring administrators to use two accounts, they tend to fail for the following reasons:

- It is nearly impossible to control a user that has a full administrator access token. Administrators can install applications and run any application or script that they wish. IT managers are always seeking ways to create "standard desktops" where users log on as standard users. Standard desktops greatly reduce help desk costs and reduce IT overhead.
- There is substantial overhead when switching between accounts whenever the user wishes to perform an administrative operation.

- After users perform administrative operations, they may forget to switch back to a standard user account, or they might decide that it is too much effort to switch back.

As a result, users may decide to always logon with their administrator accounts, thus defeating the security measures. To help mitigate this, UAC introduces the concept of Admin Approval Mode.

In the enterprise, Admin Approval Mode will be used as a bridge technology for migration to Windows Vista. Ideally, enterprises will run all users as standard users and disable the elevation prompt for standard users. This setup enables a managed standard desktop where installations are deployed with a software deployment technology, such as Microsoft Systems Management Server (SMS).

**Important**

Microsoft still recommends that members of the Domain Admins group continue to maintain two separate user accounts in Windows Vista: a standard user account and a domain administrator user account. All domain administration should be done with the domain administrator account. To further enhance security, consider deploying a smart card (http://go.microsoft.com/fwlink/?LinkId=71505) solution in domain environments.

The following are Windows Vista design goals for Admin Approval Mode:

- Eliminate the need for two separate accounts for users who are members of the administrators group: This goal is accomplished by running programs only with a standard user access token, unless the user provides approval to use the full administrator access token.
- Protect processes running with a full administrator access token from being accessed or modified by processes running as a standard user.
- Provide for a seamless transition between administrator and standard user workspaces.

Currently, many Windows applications must be run as an administrator but do not actually perform administrative operations. These applications are a byproduct of the Microsoft Windows® 9x operating systems philosophy: “everyone is an administrator.”

The following are examples of problematic applications:

- Applications that unnecessarily write to HKEY_LOCAL_MACHINE or to system files within the file system.
- An ActiveX® installation to facilitate a LOB application with a Web interface.
- Applications that unnecessarily request access to resources that require a full administrative access token.

The next section presents new technologies for Windows Vista that impact ISVs.

## How Do I Determine If My Application Has Administrative Dependencies?

To assist developers, ISVs, and organizations in evaluating their applications, Microsoft provides the Microsoft Standard User Analyzer. The Standard User Analyzer can be used to help identify an application's non-UAC–compliant. Microsoft recommends that developers run this tool to identify issues with running the application under a standard user account. These tests should be performed, even if the application already installs and runs properly under a standard user account on Windows XP. The application may perform operations, such as attempting to write to system registry locations, and make decisions based on the system's behavior, such as looking for an error response. Windows Vista may behave differently than earlier versions of the Windows operating system due to the addition of new application compatibility support. Therefore, it is recommended that all applications be tested with the new version of the Standard User Analyzer, which can be downloaded from Microsoft (http://go.microsoft.com/fwlink/?LinkId=71359).

The Standard User Analyzer will record all administrative operations encountered by an application, including registry/file system access and elevated API calls. This data is stored in a log file and is displayed within the tool. The Standard User Analyzer identifies the following common dependencies, in addition to many others:

- Dependency on objects that restrict the requested access to trusted users only.

For example, HKEY_LOCAL_MACHINE only grants KEY_WRITE to administrators and SYSTEM—an application that requests KEY_WRITE to HKEY_LOCAL_MACHINE will not work with UAC enabled.

- Use of Windows privileges that have security ramifications, such as SE_DEBUG_PRIVILEGE, which allows the debugging of other users' processes and is granted only to administrators.

## What Are the Requirements If I Have a Legitimate Administrator Application?

For applications that, by design, perform legitimate administrative operations, Microsoft has implemented an extension to the trustInfo section of the current Windows XP application manifest schema. You can use these new attributes to indicate to the computer that you have a legitimate administrative application; Windows Vista will automatically ask the user for approval to launch the application with a full administrator access token. For information about how to extend the application manifest, see the Create and Embed an Application Manifest with Your Application section within this document.

## Designing Applications for Windows Vista

The following list represents a workflow for designing your application for Windows Vista:

1. Test your application for Windows Vista application compatibility
2. Classify your application as a standard user, administrator, or mixed user application
3. Redesign your application's functionality for UAC compatibility
4. Redesign your application's user interface for UAC compatibility
5. Redesign your application's installer
6. Create and embed an application manifest with your application
7. Test your application
8. Authenticode sign your application
9. Participate in the Windows Vista Logo program

### Step One: Test Your Application for Windows Vista Application Compatibility

Testing for Windows Vista and UAC application compatibility can be easily performed by installing the Standard User Analyzer. The Standard User Analyzer is a free download on the Microsoft Web site (http://go.microsoft.com/fwlink/?LinkId=71359).

To utilize the Standard User Analyzer's graphical log display, you must install the Microsoft Application Verifier. The Application Verifier is a free download on the Microsoft Web site (http://go.microsoft.com/fwlink/?LinkId=71506).

The following procedure illustrates how to use the Standard User Analyzer to identify pre-Windows Vista administrative applications that do not run correctly on Windows Vista.

**Important**

There are two approaches you can take to utilize Standard User Analyzer: launch your application as standard user or launch your application elevated as an administrator.

Launch your application as standard user. In this instance, the Standard User Analyzer is running in diagnosis mode. The application will fail at the first error it encounters and the Standard User Analyzer will report why it failed.

Launch your application elevated as an administrator. In this instance, the Standard User Analyzer is running in prediction mode. The application will be able to run through its course and the Standard User Analyzer will predict and give an overview of the errors the application might encounter if it is run as standard user.

Once the bugs are fixed and resolved, perform this procedure once more as a standard user without the Standard User Analyzer to ensure your application is working as expected on Windows Vista.

**To identify application compatibility problems for pre-Windows Vista applications**

1. Log on to a Windows Vista computer as an administrator in Admin Approval Mode.
2. Click **Start**, click **All Programs**, and then click **Standard User Analyzer**.
3. In the **Standard User Analyzer**, for **Target Application**, specify the full directory path for an application to test or click the **Browse** button to locate the program's executable file with Windows Explorer.
4. Click **Launch** and then click **Continue** at the **User Account Control** dialog box.
5. After the test application launches, perform standard administrative tasks in the application, and then close the application when you have completed.
6. In the **Standard User Analyzer**, examine the output on each tab. Use this data to identify the program's application compatibility issues.

## Step Two: Classify Your Application as a Standard User, Administrator, or Mixed User Application

Administrative applications in Windows Vista often have a mixture of both administrative and standard user functionality. As a result, a number of options must be considered when deciding how your application will work in Windows Vista. The administrative functionality can be removed completely or separated from the standard user account functionality by prompting the user for approval.

## Questions to Help Classify Your Application

Answer the following questions to determine whether your application will require any redesign for Windows Vista compatibility:

- Does your application run as a standard user?
- Can the administrative functionality be fixed to no longer require an administrator access token?
- Can the administrative sections be removed from the program's functionality?

### Does Your Application Run as a Standard User?

To answer this question, ensure that the application can be fully used by standard users. If any part of your application requires the user to be an administrator, the answer to this question is “No.”

How to verify that the application can be used by standard users:

- Thoroughly test the application as both a standard user and an administrator. Verify that the user interactions are all exactly the same for both standard users and administrators.
- Check where the settings are stored in the registry. If any settings are stored in HKEY_LOCAL_MACHINE, the application or control panel will most likely require an administrator access token.
- If any of the settings are per-computer, the application or control panel will require an administrator access token.
- If any of the settings do anything in other users’ profiles, the application or control panel will require an administrator access token.

### Can the Administrative Functionality be fixed to no Longer Require an Administrator Access Token?

If your application or control panel has settings or interactions that require a full administrator access token, can it be changed to work correctly as a standard user? Specifically, can the program store information in per-user locations instead? If it cannot, the answer to this question is "No."

A good example of the kind of feature/setting that can be fixed is Calc.exe (the Windows Calculator). In Windows XP, the setting of whether Windows Calculator was in “Scientific” versus “Standard” mode was a per-computer setting. This setting meant that a full administrator access token was needed to change the setting. In Windows Vista, this setting is stored in the user’s profile.

How to verify that administrative sections can be removed from the program’s functionality:

- Thoroughly test the application as both a standard user and as an administrator. Can the experience be the same for both types of users?
- Is it possible to lower the ACLs required to write to the HKEY_LOCAL_MACHINE key?

**Note**

This course should not be taken lightly. Be cautious not to compromise the overall security of the computer by lowering the control afforded by the ACL.

- Is it possible to change the user interface to set per-user state rather than global state (and do not expose global state modification through the user interface)?

### Can the Administrative Sections be Removed from the Program’s Functionality?

Does your program absolutely have to have this administrative functionality? If you cannot remove the administrative features/functionality, the answer to this question is "No."

To determine whether the administrative sections can be removed from the program's functionality, do the following:

- Test the application as a standard user and as an administrator. What is the user scenario for retaining this feature?
- Is this setting/feature exposed elsewhere in the application? Perhaps the functionality in the application is redundant.

## Analyzing the Answers to Classify Your Application

### If You Answered "Yes" to any of the Preceding Questions

Make the necessary changes in the application or control panel (if any) to eliminate those items that require the user to have a full administrator access token.

The following list details benefits of having a true standard user application:

- Your feature is equally usable for all users. This is the ideal state since most features should not require a full administrator access token.
- Your users will never see an elevation prompt with your features.
- Your features are much more secure by never requiring the administrator access token.

#### If You Answered "No" to all of the Preceding Questions

The application must be modified to make the feature work with UAC.

### Verify the Application or Control Panel Works with UAC:

Finally, test the application as a standard user and as an administrator. Ensure that other options (the previous questions) cannot be used for this particular application.

## Step Three: Redesign Your Application's Functionality for UAC Compatibility

Use the information in this section once you have classified your application and determined whether it must be redesigned for UAC.

### Windows Vista Application Run-time Requirements

A large component of redesigning your application for Windows Vista will be examining your application's user access model at its core.

#### Requirements for all Windows Vista Applications

#### Specify a requestedExecutionLevel

For UAC to operate properly, the operating system has to be able to identify what code needs elevated privilege and what code does not.

In Windows Vista, these changes require that applications be marked with information that allows the operating system to determine in what context the application should be launched. For example, standard user applications need to be marked to run as the invoker and accessibility-enabled applications need to be identified by the system.

#### Do not register components with Rundll32

Some applications use the Windows Rundll32 executables to run components. However, this method is not compliant with Windows Vista development requirements. Calling directly into Rundll32 results in UAC compatibility issues. When an application relies on the Rundll32 executables to perform its execution, Rundll32 calls AIS on behalf of the application to initiate the UAC elevation prompt. As a result, the UAC elevation prompt has no knowledge of the original application and displays the application requesting elevation as “Windows host process(Rundll32).” Without a clear description and icon for the application requesting elevation, users have no way to identify the application and determine whether it is safe to elevate it.

If your application calls into Rundll32 to run components, use the following workflow to redesign the execution call.

1. Create a new separate executable file for your application.
2. In the new executable file, call the exported function in your DLL that you would have specified with Rundll32. You may need to LoadLibrary the DLL if it does not have a .lib.
3. In a resource file, create and add a new icon for the executable. This icon will be displayed in the User Account Control dialog box when the application requests elevation.
4. Provide a short, meaningful name for the executable. This name will be shown in the User Account Control dialog box when the application requests elevation.
5. Create and embed an application manifest file for the executable and mark it with the requested execution level of requireAdministrator. This process is detailed in the Create and Embed an Application Manifest with Your Application section.
6. Authenticode sign the new executable. This process is detailed in the Authenticode Sign Your Application section later in this document.

Finally, following the un-installation of an application, the user should be able to reinstall it without errors.

### Requirements for Standard User Applications

Here is a summary of things to remember when designing applications that operate correctly under a standard user account. Developers should keep these requirements in mind during the design phase of their applications.

#### Setup

- Never perform administrative actions (such as completing the setup process) on first run; these actions should be done as part of the initial setup process.
- Never write directly to the Windows directory or subdirectories. Use the correct methods for installing files, such as fonts.
- If you need to automatically update your application, use a mechanism suitable for use by standard users, such as Windows Installer 4.0 User Account Control patching to accomplish the update.

#### Saving State

- Do not write per-user information or user-writable information to Program Files or Program directories.

- Do not use hard-coded paths in the file system. Take advantage of the KnownFolders API and ShGetFolder to find where to write data.

### Run and Test Under a Standard User Account

If you are writing a non-administrative application, such as a LOB application or a user application, such as a game, you must always write application data to a location that standard users can access. The following are some of the recommended requirements:

- Write per-user data to the user profile: CSIDL_APPDATA.
- Write per-computer data to Users\All Users\Application Data: CSIDL_COMMON_APPDATA.
- The application cannot depend on any administrative APIs. For example, a program that expects to successfully call the SetTokenInformation() Windows function will fail under a standard user account.

### Be Fast User Switching (FUS) Aware

Applications will more commonly be installed by a user other than the user who will run the application. For example, in the home, this means that a parent will install the application for a child. In the enterprise, a deployment system, such as SMS or Group Policy advertisement, will install the application using an administrator account.

If the per-user settings do not exist at first run, rebuild them. Do not assume that the setup process took care of the settings.

## Requirements for Administrator Applications

### Use the HWND Property to be acknowledged as a Foreground Application

Background applications will automatically prompt the user for elevation on the taskbar, rather than automatically switching to the secure desktop for elevation. The elevation prompt will appear minimized on the taskbar and will blink to notify the user that an application has requested elevation. An example of a background elevation occurs when a user browses to a Web site and begins downloading an installation file. The user then goes to check e-mail while the installation downloads in the background. Once the download completes in the background and the install begins, the elevation is detected as a background task rather than a foreground task. This detection prevents the installation from abruptly stealing focus of the user's screen while the user is performing another task--reading e-mail. This behavior creates a better user experience for the elevation prompt.

However, some foreground applications currently prompt as background applications on Windows Vista. This behavior is the result of an absent parent HWND. In order to ensure that Windows Vista acknowledges your application as a foreground application, you must pass a parent HWND with a ShellExecute, CoCreateInstanceAsAdmin, or managed code call.

The UAC elevation mechanism uses the HWND as part of determining whether the elevation is a background or foreground elevation. If the application is determined to be a background application, the elevation is placed on the taskbar as a blinking button. The user must click on the button, as with any application requesting foreground access, before the elevation will continue. Not passing the HWND will result in this occurring even though the application might actually have foreground.

The following code sample illustrates how to pass HWND with ShellExecute:

```
BOOL RunAsAdmin( HWND hWnd, LPTSTR lpFile, LPTSTR lpParameters )
{
    SHELLEXECUTEINFO   sei;
    ZeroMemory ( &sei, sizeof(sei) );

    sei.cbSize          = sizeof(SHELLEXECUTEINFOW);
    sei.hwnd            = hWnd;
    sei.fMask           = SEE_MASK_FLAG_DDEWAIT | SEE_MASK_FLAG_NO_UI;
    sei.lpVerb          = _TEXT("runas");
    sei.lpFile          = lpFile;
    sei.lpParameters    = lpParameters;
    sei.nShow           = SW_SHOWNORMAL;

    if ( ! ShellExecuteEx ( &sei ) )
    {
        printf( "Error: ShellExecuteEx failed 0x%x\n", GetLastError() );
        return FALSE;
    }
    return TRUE;
}
```

The following code sample illustrates how to pass HWND with CoCreateInstanceAsAdmin by using the elevation moniker. It assumes that you have already initialized COM on the current thread. More information about the elevation moniker is available in Step Four of this document.

```
HRESULT CoCreateInstanceAsAdmin(HWND hwnd, REFCLSID rclsid, REFIID riid, __out 
void ** ppv)
{
    BIND_OPTS3 bo;
    WCHAR  wszCLSID[50];
    WCHAR  wszMonikerName[300];

    StringFromGUID2(rclsid, wszCLSID, sizeof(wszCLSID)/sizeof(wszCLSID[0]));
    HRESULT hr = StringCchPrintf(wszMonikerName, 
sizeof(wszMonikerName)/sizeof(wszMonikerName[0]),
L"Elevation:Administrator!new:%s", wszCLSID);
    if (FAILED(hr))
        return hr;
    memset(&bo, 0, sizeof(bo));
    bo.cbStruct = sizeof(bo);
    bo.hwnd = hwnd;
    bo.dwClassContext  = CLSCTX_LOCAL_SERVER;
    return CoGetObject(wszMonikerName, &bo, riid, ppv);
}
```

BIND_OPTS3 is new in Windows Vista. It is derived from BIND_OPTS2. It is defined as follows:

```
typedef struct tagBIND_OPTS3 : tagBIND_OPTS2
{
    HWND hwnd;
} BIND_OPTS3, * LPBIND_OPTS3;
```

The only addition is an HWND field, hwnd. This handle represents a window that becomes the owner of the elevation UI when secure desktop prompting is enabled.

The following code sample illustrates how to pass HWND in managed code to ensure that parent dialogs are aware of the HWND and its use.

```
System.Diagnostics.Process newProcess = new System.Diagnostics.Process();
System.Diagnostics.ProcessStartInfo info = new
System.Diagnostics.ProcessStartInfo(“D:\SomeProgram.exe”);
info.UseShellExecute = true;
info.ErrorDialog = true;
info.ErrorDialogParentHandle = this.Handle;
newProcess.StartInfo = info;
newProcess.Start();
```

### Do Not Prompt for Elevation in the User's Logon Path

Applications that start when the user logs on and require elevation are now blocked in the logon path. Without blocking applications from prompting for elevation in the user's log on path, both standard users and administrators would have to respond to a User Account Control dialog box on every log on. Windows Vista notifies the user if an application has been blocked by placing an icon in the system tray. The user can then right-click this icon to run applications that were blocked from prompting for elevation as the user logged on. In addition, the user can manage which startup applications are disabled or removed from this list by double-clicking on the tray icon.

### Do Not Use Runas to Launch an Elevated Process

The **Run as…** option from Windows XP and Windows Server 2003 has been replaced with **Run as administrator** on the context menu (available when you right-click an executable) in Windows Vista. When a standard user selects the **Run as administrator** option, the user is presented with a list of active administrators on the local computer. Standard users with higher privileges, such as members of the Backup Operators group, are also displayed. When an administrator selects the **Run as administrator** option, a User Account Control dialog box immediately prompts the user to continue before running the application.

Users must use the **runas** command at the command prompt in order to run an application as another user.

**Important**

Be aware that **runas** does not provide the ability to launch an application with an elevated access token, regardless of whether it is a standard user with privileges like a Backup Operator or an administrator. The **runas** command grants the user the ability to launch an application with different credentials. The best method to use to launch an application with a different account is to perform the action programmatically by using a service and not rely on the user to run the component as a different user. If your program programmatically uses the **runas** command, ensure that it is not intended to launch an elevated process.

If your application will require the user to run parts of the application with a different user account, ensure that the **runas** command with the command prompt option is exposed. The following table details the available parameters for the **runas** command.

**Runas parameters**

| Parameter | Description |
|---|---|
| /noprofile | Specifies that the user's profile should not be loaded. This enables the application to load more quickly, but can cause some applications to malfunction. |
| /profile | Specifies that the user's profile should be loaded. This is the default setting. |
| /env | Use the current environment instead of the user's. |
| /netonly | Use this parameter if the credentials specified are for remote access only. |

| Parameter | Description |
|---|---|
| /savecred | Use credentials previously saved by the user. This option is not available on Windows XP, Home Edition, and will be ignored. |
| /smartcard | Use this parameter if the credentials to be provided are from a smart card. |
| /user | The user's user name. The user name should be provided in the form of USER\DOMAIN or USER@DOMAIN. |
| /showtrustlevels | Displays the trustlevels that can be used as arguments for the /trustlevel parameter. |
| /trustlevel | One of the levels enumerated in /showtrustlevels. |
| program | Command line for an executable. |

Examples:

runas /noprofile /user:mymachine\Denise cmd

**Note**

Enter the user's password only when prompted. The /profile parameter is not compatible with the /netonly parameter. The /savecred parameter is not compatible with the /smartcard parameter.

**Requirements for Console Applications**

A console application presents its output on the console window and not with a separate user interface. If an application needs a full administrator access token to run, then that application needs to be launched from an elevated console window.

You must do the following for console applications:

1. **Mark that your application "asInvoker":** You can do this by authoring the manifest of your application in which you set RequestedExecutionLevel == asInvoker. This setup allows callers from non-elevated contexts to create your process, which allows them to proceed to step 2.

2. **Provide an error message if application is run without a full administrator access token:** If the application is launched in a non-elevated console, your application should give a brief message and exit. The recommended message is:
    - “Access Denied. Administrator permissions are needed to use the selected options. Use an administrator command prompt to complete these tasks.”

The application should also return the error code ERROR_ELEVATION_REQUIRED upon failure to launch to facilitate scripting.

### Requirements for Scripts

Scripts may be considered as a group of applications run in a predefined order and the results of one being channeled into other.

In order to make your scripts UAC compliant, examine the logic of your scripts and add “tests” to ensure the person running the script has sufficient privileges to do that task. This check should be done before performing an action in the script.

### Requirements for Bulk Operations

If your application performs a task that consists of actions on multiple objects, and some of them might require the user's administrator access token, then show the elevation prompt the first time the access token is needed. If the user approves the elevation, then perform the rest of the tasks. Otherwise, terminate the batch operation. This behavior would be consistent with the current multi-select/copy/delete operation.

### APIs that Help Identify an Administrator

- IsUserAnAdmin()
- GetTokenInformation()

### Registry/Handle Access Permissions that are Inherently Different between Administrators and Standard Users

- MAXIMUM_ALLOWED
- KEY_WRITE
- DELETE (when applied to registry keys)
- Other HKEY_LOCAL_MACHINE (HKLM) type keywords (opened with MAXIMUM_ALLOWED on XP):
- SHELLKEY_HKLM_EXPLORER
- SHELLKEY_HKLM_SHELL

**Other APIs that Are Re-directed to HKEY_LOCAL_MACHINE Registry Values and Virtualization will Apply**

- WritePrivateProfileString(,,,"system.ini");
- CheckSectionAccess("system.ini",…);

## Step Four: Redesign Your Application's User Interface for UAC Compatibility

Use the guidelines in this section to develop your application's user interface for UAC compatibility. Closely adhering to these guidelines in your application's development will ensure that your application will have a consistent and predictable user experience in Windows Vista.

- Impact of UAC on the Windows user experience
- Goals of the UAC user experience
- Elevation prompt
- User experience process flow
- Elevation entry points
- User interface implementation
- When to add a shield icon to your application's user interface
- Key decisions for designing administrator-only applications

**Important**

Simply refractoring your application's user interface will not fulfill the requirements for UAC compatibility. Your application's core functionality must comply with the Windows Vista standard user model requirements. These requirements were detailed in the previous step, Step Three: Redesign Your Application's Functionality for UAC Compatibility.

### Impact of UAC on the Windows User Experience

The biggest and most immediate impact on the user experience will be felt by administrators. Administrator users will now need to provide permission to accomplish administrative tasks. Coupled with that, standard users will now gain the ability to perform administrative tasks within the currently logged in session by providing valid administrator credentials.

## Goals of the UAC User Experience

The overall goal for UAC user experience is to provide predictability in the user experience:

- For an administrator, this means that the user always know when he/she will need to give permission to run an elevated task.

This is the act of requesting the user's own administrator access token so that he/she can make administrator-required changes.

- For standard users, this means that they will know when they:
    - Will need to provide administrator credentials (home and unmanaged environments) for administrative tasks.
    - OR When they cannot complete a task (managed environments where elevation is explicitly disallowed) and must contact the help desk.

### Design Goals

The following list comprises the UAC design goals.

#### Eliminate Unnecessary Elevation

Users should have to elevate only to perform tasks that require an administrator access token. All other tasks should be designed to eliminate the need for elevation. Pre-Windows Vista software often requires an administrator access token unnecessarily by writing to the HKEY_LOCAL_MACHINE or HKEY_CLASSES_ROOT registry sections or to the Program Files or Windows system folders.

#### Be Predictable

Administrators need to know which tasks require elevation. If they cannot predict the need for elevation accurately, they are more likely to give consent for administrative tasks when they should not. Standard users need to know which tasks require an administrator to perform or cannot be performed at all in managed environments.

#### Require Minimal Effort

Tasks that require a higher privileged access token should be designed to require a single elevation. Tasks that require multiple elevations quickly become tedious.

#### Revert to Standard User

Once a task that requires a higher level of access token is complete, the program should revert to the standard user state.

## Elevation Prompt

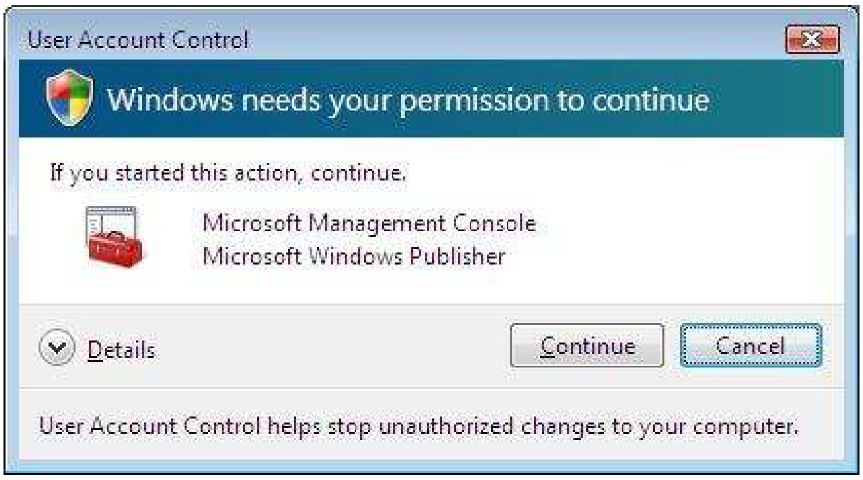
The elevation prompt is built upon an existing Windows user interface. The elevation prompt displays contextual information about the executable requesting elevation, and the context is different depending on whether the application is Authenticode signed. The elevation prompt is seen in two variations: the consent prompt and the credential prompt.

### Consent Prompt

The consent prompt is displayed to administrators in Admin Approval Mode when they attempt to perform an administrative task. This is the default user experience for administrators in Admin Approval Mode and can be configured in the local Security Policy Manager snap-in (secpol.msc) and with Group Policy.

The following figure is an example of a User Account Control consent prompt.

**User Account Control consent prompt**



User Account Control

Windows needs your permission to continue

If you started this action, continue.

Microsoft Management Console
Microsoft Windows Publisher

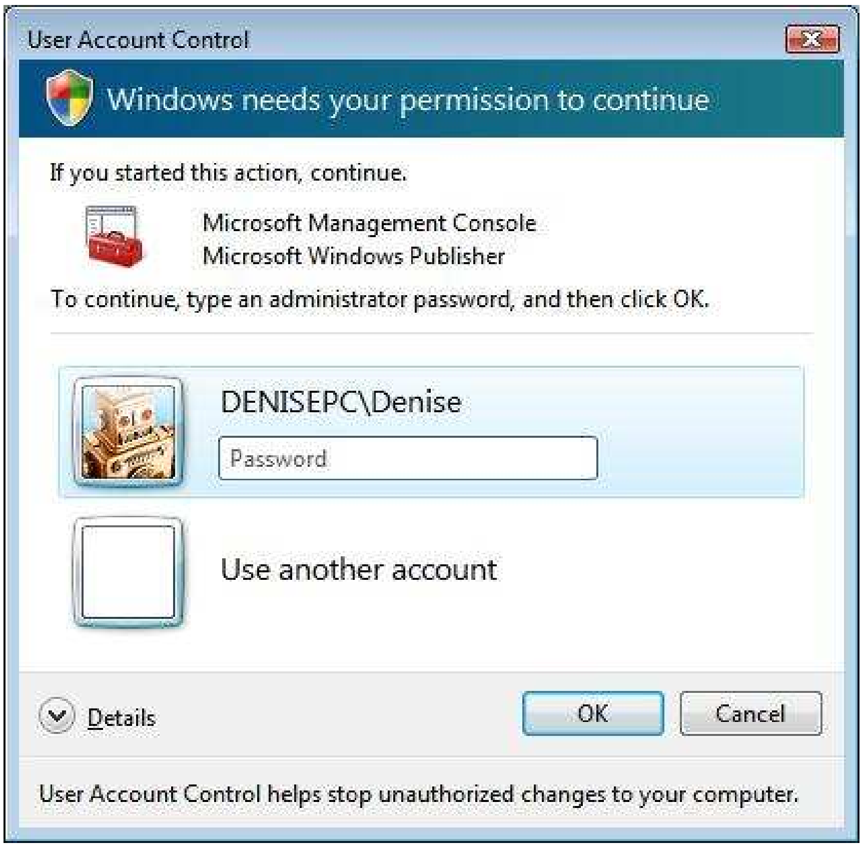
User Account Control helps stop unauthorized changes to your computer.

### Credential Prompt

The credential prompt is displayed to standard users when they attempt to perform an administrative task. This is the default user experience for standard users and can be configured in the local Security Policy Manager snap-in (secpol.msc) and with Group Policy.

The following figure is an example of a User Account Control credential prompt.

**User Account Control credential prompt**


User Account Control

Windows needs your permission to continue

If you started this action, continue.

Microsoft Management Console
Microsoft Windows Publisher

To continue, type an administrator password, and then click OK.

DENISEPC\Denise

Password

Use another account

User Account Control helps stop unauthorized changes to your computer.

**Default Elevation Prompt Consent Policy for Windows Vista**

The following table outlines the default prompt style for each user account type in Windows Vista.

**Default elevation prompt behavior**

| User Account Type | Elevation Prompt Setting |
|---|---|
| Standard user | Prompt for credentials |
| Administrator account in Admin Approval Mode | Prompt for consent |

## User Experience Process Flow

The UAC user experience process flow consists of three distinct components:

1. Elevation entry point (for example, a control or link that displays the UAC shield icon).
2. Elevation prompt (a request for consent or for administrator credentials).
3. Elevated process.

The following example workflow summarizes how the preceding components are related:

1. An administrator in Admin Approval Mode logs on to a Windows Vista computer.
2. The user then decides to add another administrator user for the computer.
3. The user clicks **Start**, clicks **Control Panel**, and then clicks the link in the **Security** section entitled **Allow a program through Windows Firewall**, which is displayed inline with a shield icon.
4. A consent prompt appears requesting the user for approval.
5. The user clicks **Continue** and the elevated process is created.
6. In **Windows Firewall Settings**, the user modifies the Windows Firewall settings and then clicks **OK**, which terminates the elevated process.
7. The user continues to work on the computer as a standard user.

**Note**

Elevation entry points do not remember state (e.g. when navigating back from a shielded location or task), as well as the entry point will not remember that elevation has occurred. As a result, the user will need to re-elevate to enter the task/link/button again.

## Elevation Entry Points

For entry points, the shield icon will be attached to certain controls (e.g. buttons, command links, hyperlinks) to indicate that the next immediate step requires elevation.

### Shield Icon

The shield icon is the primary user interface decoration for a UAC elevation point. This icon signifies security related activities in Windows Vista and previous versions of Windows, and this relationship is continued in Windows Vista.

The following figure is an example of the shield icon.

### Shield icon

The shield icon will play a critical part in all three components of the UAC user experience.

When viewing the system with Windows Explorer, any application that is marked to request an administrator access token when it is launched will automatically be decorated with a shield glyph over its icon. This permits users to know which applications, when launched, will request elevation.

Shield icon properties:

- Consistent appearance throughout the entire UAC user experience.
- Does not reflect any visual state (e.g. active, hover, disabled, etc.).
- Does not remember state.

There are three consistent control styles that an entry point marked with a shield icon can take within the user experience:

- UAC button
- UAC hyperlink
- UAC command link

These styles apply to all scenarios where these user interface elements can appear such as Wizards, property pages, control panels, explorers, etc. Each of the styles implies that an elevation prompt will immediately be displayed after the user clicks a UAC user interface control.

A fourth UAC user interface entry point, the UAC icon overlay, is also discussed in this section. Whether an executable receives an icon overlay or not is not controlled by the application developer. Windows Vista overlays a shield icon on applications' icons for executables that have requestedExecutionLevel set to requireAdministrator.

### UAC Shield Button

The UAC shield button should be used in any user interface button that, when pressed, will require the elevation prompt to prompt the user for approval or credentials.

UAC shield buttons can be used as commit buttons (e.g. **Next** in a Wizard) or as a button to display an additional settings user interface (e.g. Change **Settings** in a property dialog).

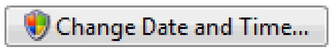
The UAC shield button consists of two user interface components:

- Shield icon
- Text label

The UAC shield button is packaged in a manner so that developers can use it in the place of a normal button. The UAC button also supports rendering the shield icon on the left or right side of the text label. In addition, developers will have the option to hide/show the shield icon while the UAC button is displayed.

The following screenshot is an example of a UAC shield button.

**UAC shield button**



**UAC Hyperlink**

The UAC hyperlink should be used in any user interface hyperlink that, when clicked, will require the elevation prompt to prompt the user for approval or credentials.

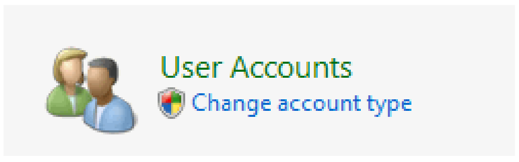
A UAC hyperlink consists of the following components:

- Shield icon
- Hyperlink control

The UAC hyperlink is not packaged with the shield icon for a developer to use. Developers will need to get the shield icon resource and render it next to the hyperlink.

The following screenshot is an example of a UAC hyperlink.

**UAC hyperlink**



**UAC Command Link**

The UAC command link should be used in any user interface button that, when clicked, will require the elevation prompt to prompt the user for approval or credentials.

UAC command links should only be used as commit buttons (e.g. “Do this option” in a dialog box).

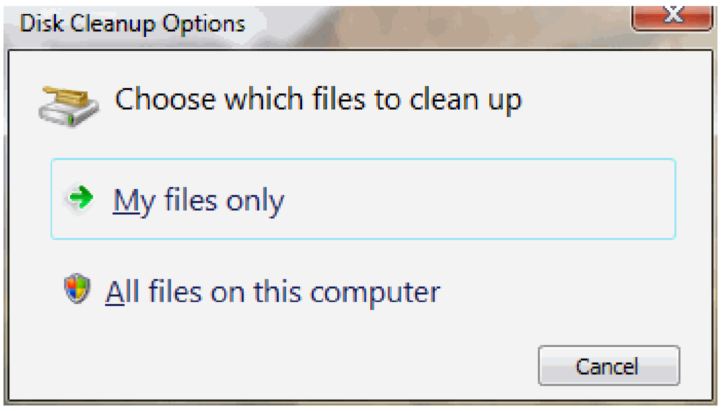
The UAC command link consists of the following components:

- Shield icon
- Standard command link components
- Link text
- Note text

The UAC command link is packaged in a way where a developer can use a UAC command link in the place of a normal command link. The UAC command link supports rendering the shield icon on the left or right side of the command link.

The following is an example of a UAC command link.

**UAC command link**

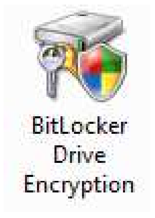


**Icon Overlays**

In Windows Vista, if an executable file requires elevation to launch, then the executable's icon should be “stamped” with a shield icon to indicate this fact. The executable's application manifest must designate a requestedExecutionLevel of requireAdministrator to designate the executable as requiring a full administrator access token. The shield icon overlay will also be automatically placed on executables that are deemed to require elevation, as per the installer detection heuristics. For example, a file named setup.exe will automatically receive a shield icon overlay, even if the executable does not have an embedded application manifest.

The following figure is an example of a UAC icon overlay.

**UAC icon overlay**



**Note**

Guidance about how to create and embed an application manifest with an executable is provided in the Create and Embed an Application Manifest with Your Application section of this document.

## User Interface Implementation

### Shield Icon Implementation and APIs

This section provides preliminary information on the icons and APIs available to developers as they migrate or implement new administrative application functionality.

**Shield icon implementation and APIs**

| Icon | API |
|---|---|
| Shield | User resource: IDI_SHIELD |
| Button | Button_SetElevationRequired(hwndButton) |
| Syslink / Hyperlink | Layout IDI_SHIELD next to syslink |
| Command link | Load IDI_SHIELD and set as command link icon |
| Context menu | Icon support in DefCM for static commands |

**How do I…**

- Add a shield icon to the user interface?

- Add a shield icon to a button?
- Add a shield icon to a Windows Installer button?
- Add a shield to a "Next" button control on a Wizard?
- Add a shield icon to a task dialog button?
- Elevate a modal dialog?

### Add a Shield Icon to the User Interface

#### Add a small icon:

```
#include <shellapi.h>
SHSTOCKICONINFO sii;
sii.cbSize = sizeof(sii);
SHGetStockIconInfo(SIID_SHIELD, SHGSI_ICON | SHGSI_SMALLICON, &sii);
hiconShield  = sii.hIcon;
```

#### Add a large icon:

```
SHSTOCKICONINFO sii;
sii.cbSize = sizeof(sii);
SHGetStockIconInfo(SIID_SHIELD, SHGSI_ICON | SHGSI_LARGEICON, &sii);
hiconShield  = sii.hIcon;
```

#### Add an icon of custom size:

```
SHSTOCKICONINFO sii;
sii.cbSize = sizeof(sii);
SHGetStockIconInfo(SIID_SHIELD, SHGSI_ICONLOCATION, &sii);
hiconShield  = ExtractIconEx(sii. ...);
```

**Note**

Generally, you should not add the shield icon directly to your user interface. Using one of the proceeding methods of embedding the shield icon in a control is recommended. Additionally, simply adding a shield icon in your user interface will not ensure UAC compatibility. You must also refractor the entirety of your application's user experience (add a requestedExecutionLevel, fix any standard user application compatibility problems, and ensure that the user interface is user friendly and UAC compatible).

### Add a Shield Icon to a Button

The standard button control (PUSHBUTTON, DEFPUSHBUTTON) has been enhanced to allow you to add an icon along with the displayed text, without requiring the BS_ICON or BS_BITMAP styles to be set.

To display the shield icon, call the following macro (defined in commctrl.h):

```
Button_SetElevationRequiredState(hwndButton, fRequired);
```

**Note**

hwndButton is the HWND of the button; fRequired determines whether to show (TRUE) or hide (FALSE) the UAC shield icon.

### Add a Shield Icon to a Windows Installer Button

Windows Installer dialogs authored using the internal table support can add a shield to the last button of the user interface dialog sequence by setting the ElevationShield attribute on the control.

### Add a Shield Icon to a "Next" Button on a Wizard

**Important**

Displaying the UAC shield icon the "Next" button is only supported in AeroWizards (PSH_AEROWIZARD).

To display the shield icon on the "Next" button for a specific page in an AeroWizard, use the following code:

```
case WM_NOTIFY:
    if (reinterpret_cast<NMHDR*>(lParam)->code == PSN_SETACTIVE)
    {
        // Show next button
        //
        // Note new wParam flag -- when PSWIZBF_ELEVATIONREQUIRED flag
        // is specified, it indicates that the next page will require
        // elevation, so if the next button is being shown, show it as
        // a shield button.

        SendMessage(GetParent(hwndDlg),
                    PSM_SETWIZBUTTONS,
                    PSWIZBF_ELEVATIONREQUIRED,
                    PSWIZBF_NEXT);

        // return 0 to accept the activation
        SetWindowLong(hwndDlg, DWLP_MSGRESULT, 0);
    }
    break;
```

### Add a Shield Icon to a Task Dialog Button

**Caution**

A task dialog button should never require a UAC shield icon. The “press” action on a task dialog button is expected to commit/cancel and dismiss the task dialog. It would be unexpected for such a button to then display the elevation prompt to the user.

### Elevate a Modal Dialog

You should use the elevation moniker to elevate a COM object representing the modal dialog. Use the following tasks to elevate using a modal dialog:

- Move the dialog box into a COM object.
- Expose a ShowDialog() method.
- Use the CoCreateInstanceAsAdmin() API to create the COM object and call ShowDialog().

This API will run an instance of the COM object as administrator after going through the elevation process.

**Note**

A version of this API that is more complicated to call is available. A simplified version will be available in a later version of Windows Vista.

### User Education and Assistance Guidelines

When a user interface has been refractored and put behind a button, ISVs should evaluate whether a change to the button name is warranted. Microsoft strongly advises against using **Advanced** as a button label for elevation tasks. Instead, use more descriptive and understandable labels like **Change settings** or a term that suggests what is behind the button.

### Guidelines for Administrator-only User Interface

If an application will always be launched by an administrator, then you do not need to add additional shields within the application's user interface. This is because the application will be elevated and everything it does will be elevated. Therefore, the application does not need further elevation.

**Note**

If you have links to other administrator user interface in your administrator-only user experience, the user interface will launch its target elevated. Therefore, you do not need to put any shields in an application that is solely administrative.

## When to Add the Shield Icon to Your Application's User Interface

### An Administrative Choice Application

### An Elevated Process or COM Object

An elevated process or COM object launches without requiring elevation. Those items in the user interface that require an administrator access token are decorated with a shield icon to identify this requirement. The shield icon decoration indicates to the user that using that feature will require administrator approval. When the application detects that one of these buttons has been selected, it has the following two choices:

- The application launches a second program using ShellExeucute() to perform the administrative task. This second program would be marked with a requestedExecutionLevel of requireAdministrator, thus causing the user to be prompted for approval. This second program would be running with a full administrator access token and would be able to perform the desired task.

OR

- The application launches a COM object using CoCreateInstanceAsAdmin. This API would launch the COM object with a full administrator access token following approval, and this COM object would be able to perform the desired task.

This method provides the richest user experience and is the preferred method of dealing with administrative functionality.

The following list details requirements for an elevated process or COM object:

- The application should implement the shield decoration and its required architecture.
- The developer must determine where the shield should go within the user interface.
- The developer must do the architectural work to separate the business logic into a COM object from the user interface object.
- The developer must call into the UAC elevation process when the OnClick event for the shield icon is detected.

The following list details benefits of properly designing an elevated process or COM object:

- This is the best overall user experience for both user types. The user interface will launch, viewable to everyone, and all UAC functionality on that user interface will be accessible to everyone. Only when an administrator task is required does the user attempt to elevate to complete the task.
- Doing this work now will make you fully UAC compliant moving forward.

- The user interface/COM separation is good architectural practice.

Clicking on a shield icon causes the application to launch either an elevated program or an elevated COM object to perform the task.

### Administrator-only Application

In this instance, the application's initial launch requires administrator approval. This method is called "prompt before launch." Once launched, the application is running with a full administrator access token and can therefore perform the desired administrative tasks. This method requires the least amount of work for the developer, and the application's manifest is marked with a requestedExecutionLevel of requireAdministrator.

**Important**

While this does require the least amount of work for the developer, please note that, just like other administrative applications in Windows Vista, administrators will have to elevate in order to use this application and that standard users will be unable to use the application unless they have access to administrator credentials for an administrator account on the computer.

The following list details requirements for administrator-only applications:

- The application manifest should contain a requestedExecutionLevel marking set to requireAdministrator.
- The user is prompted for administrator approval prior to Windows launching the application with a full administrative access token.

The following list details benefits of properly designing an administrator-only application:

- The operating system does not have to "guess" if your setup application is an administrative application.
- Standard users will automatically be given a hint that the operation is an administrative operation. For example, when you see the icon for an application marked requireAdministrator, the icon has a shield embedded in the icon.
- On Windows Vista, if you mark your application as requireAdministrator you *know* that, once it is launched, it will be running with a full administrator access token. Users must elevate to run the application (either as an administrator in Admin Approval mode or by using **Run as administrator**).

**Note**

Marking an application requireAdministrator does NOT silently elevate the application. The user will still have to give elevation consent to start the application. There is no way to mark an application in Windows Vista to silently elevate.

The following list details points of consideration for designing an administrator-only application:

- This user experience means that all users will see an elevation prompt (either the credential prompt or the consent prompt) prior to the user interface even being visible. That also means no one is able to simply view the current settings until after authenticating with administrator credentials
- If you are marking requireAdministrator on a setup application, you should be aware that the user that is running the setup is different from the user that may use the application. Therefore, you should not modify HKEY_CURRENT_USER (HKCU) and other per-user settings, such as writing to the user profile, during your administrative setup.

**Important**

You must assume that the user running the administrative application is different from the normal user on the computer.

Executables that require an administrator access token are marked with a shield icon overlay.

### Mixed Application

A mixed application is one that can be run by users—all users of the computer (standard users, administrators in Admin Approval Mode, and those in between like Backup Operators). This is also a "prompt before launch" application. The application will run with the invoker's access token and will launch normally for standard users (no elevation prompt).The program must then modify its behavior at run time to disable those features that would not be available to the user based on the access token obtained.

A mixed application does not have the ability to obtain additional administrative privileges once launched; therefore, it does not provide the flexibility of the elevated process or COM object method described previously. This is most useful for applications that require an access token above that of a standard user, but less than that of a full administrator.

For example, the Microsoft Management Console (MMC) is marked highestAvailable. If a true standard user runs the MMC, MMC will launch as a standard user application without any elevation attempt or prompt. If the user has a filtered access token, such as an administrator in Admin Approval Mode or a Backup Operator, the operating system will prompt the user for consent to launch MMC with the user's "highest" available privilege. In the case of a standard user who has Backup Operator privileges, after elevation, MMC will be launched with standard user + Backup Operator, but nothing more. If an administrator launches MMC, after elevation, MMC will be running as a full administrator application.

The benefit of properly designing a mixed application is that the application is available to all users of the system, even though some functionality may be disabled.

The following list details points of consideration for designing mixed applications:

- The developer must dynamically change the behavior of the application based on the user's available administrative Windows privileges and user rights.
- The standard user is prevented from ever being able to act on the administrative-level functions on the user interface. There is no potential for prompt elevation once the program is running (the administrators must elevate before opening the user interface).

**Note**

There is one workaround for the previous bullet point. An administrator can launch an elevated command prompt on the standard user's computer and run the application from the command prompt. For example, right-click the command prompt, select **Run as administrator**, and then type "applicationname.exe" in the command prompt.

The user experience is branched between the standard user and the administrator in Admin Approval Mode.

#### Example Mixed Application: Backup Application

The application could be launched by a member of the Backup Operators group. The program would then verify that the highest level of administrative Windows privileges and user rights available from the user is sufficient for the program's operation. For more information about program launch behavior, see the Application Manifest Marking and Application Launch Behavior section of this document.

## Key Decisions for Designing Administrator-Only Applications

### Back-End Business Objects

This section provides an overview of the three models a developer can choose when developing an administrative application that provides the best user experience.

- The Admin Broker model
- The Back-End Service model

- The Admin COM Object model

**Admin Broker Model**

In the Admin Broker model, the application is broken into two independent executables—a standard user executable and an administrative executable. The developer, using an application manifest, marks the standard user program with a requestedExecutionLevel of asInvoker and marks the administrative program with a requestedExecutionLevel of requireAdministrator. A user will launch the standard user program first. When the user attempts to perform an operation that the standard user program knows requires a full administrator access token, the standard user program performs a ShellExecute() and launches the administrative program. The Windows ShellExecute() API looks at the application manifest and requests approval from the user before running the application with the user's full administrator access token. The administrative program can then perform the administrative tasks.

**Note**

The administrative executable program may enable inter-process communication with a standard user executable using shared memory, local Remote Procedure Call (RPC), or named pipes. If the administrative program does enable communication with the standard user executable, the developer needs to use good security practice to validate all inputs from the lower privilege program.

**Note**

There is no communication channel between the two programs once the second program launches

The following list details uses for the admin broker model:

- Wizards – When the Hardware Wizard realizes that the required driver is not installed on the computer or located in the enterprise's approved location, it needs an elevated application with the ability to move a driver into the computer store.
- Autorun.exe calling Setup.exe – The first time you put in a game CD, the required operation from Autorun.exe is to set up the application. The second time you insert the CD, the default operation is to play the game.

A benefit to using the admin broker model is that it is probably the easiest mechanism for the developer to implement.

The following list details some drawbacks to using the Admin Broker Model:

- The transitions from application to application can be confusing to the user. It can be hard to keep the user apprised of why a new application is “popping up” on the monitor.

- In addition, state is harder to pass between these two applications. For example, you would not use this to pass state between a standard user control panel (CPL) and its administrator counterpart simply to allow the same CPL to have administrative and standard user functionality. The standard user CPL would have to store its state somewhere.
- Often, there is a lot of replicated code when splitting the functionality between two programs.

To implement the admin broker model, create two programs (one standard user and one administrative), mark them with the appropriate application manifest requestedExecutionLevel, and then launch the administrative program from the standard user program using ShellExecute().

### The Back-End Service Model

In the back-end service model, the application is again broken into two independent executables—a standard user executable that provides the user interface to the user and a back-end service running on the system. The front-end application is marked with a requestedExecutionLevel of asInvoker and the back-end service is running as SYSTEM. Communication between the application and the back-end service is accomplished with RPC.

One use for the back-end service model is to control programs that could impact the system, such as antivirus programs or anti-spyware). The front-end application provides the means by which the logged on user can control aspects of the service.

A major benefit of using the back-end service model is that no elevation prompting is required.

The following list details some drawbacks to using the back-end service model:

- The service needs to limit the types of activities the front-end application can tell it to do. For example, an antivirus service may allow a standard user to initiate a scan of the system but not to disable real-time virus checking.
- Adding an unnecessary service to the system can impact the entire system. Ensure that your service is truly necessary for your Windows Vista implementation and that the service is properly architected.

To implement the back-end service model, create a standard user front-end application and a back-end service. Install the service in the system during product installation time.

### The Admin COM Object Model

This model is included here, but was discussed in detail previously in this document. The admin COM object model allows dynamic administrative elevation to perform specific operations from within an application or control panel.

A major benefit for using the admin COM object model is that it presents the best user experience for the user.

The following list details some drawbacks to using the admin COM object model:

- Requires the most work for the developer as each application feature has to be evaluated and tested for administrator functionality and that function has to be provided by a back-end COM object.
- User needs to provide elevation approval.
- The resulting "unit" of standard user application and admin backend COM object is now "drivable" and is not protected by UIPI and other isolation mechanisms.

To implement the admin COM object model, create a standard user front-end application and launch elevated back-end COM objects to perform administrative tasks.

## Step Five: Redesign Your Application's Installer

The following best practices are for well-behaved application installations in a Windows Vista or UAC environment. This list is not comprehensive. For a more detailed explanation of the Logo Requirements for Windows Vista, including the UAC requirements, please see the Windows Vista Logo documentation and the in-depth version of the latest draft of the Windows Vista Logo guidelines document (http://go.microsoft.com/fwlink/?LinkId=71497).

Use these requirements while redesigning your application.

1. **Use the Windows Installer 4.0 for your setup package.** Many of the following requirements are already integrated into the Windows Installer engine. Using Windows Installer for your setup package will assist you with following Windows Vista installation requirements.
2. **Use versioned files and do not downgrade files during installation.** File versioning ensures that the final installation state is correct when setup is complete. Without file versions, some special handing will be needed to ensure that your installation works properly for many different installation scenarios. Also, when installing versioned files, do not downgrade versions, especially shared files. Downgrading versions may be good for your application, but it frequently causes

issues with other applications. By declaring the correct versions of your files in your Windows Installer package, Windows Installer natively supports this feature.

3. **Install applications and store per-user data in different locations.** Applications should be installed in a folder under the Programs Files directory. To configure this, you can use the ProgramFilesFolder property in the Directory table of your Windows Installer package. Per-user configuration data should be stored in files either under the \Users\Username\AppData directory or in registry keys under the HKEY_CURRENT_USER root. User data, templates, and application-created files all have proper locations in the \Users\Username subdirectory. Although this was not enforced in the past, since many users would run programs with a full administrator access token, applications that do not place information in the correct location are likely to fail. This is especially true when virtualization is disabled.
4. **Use a consistent folder location when installing shared components.** Shared components should be installed to the Common Files directory by using the CommonFilesFolder property in the Directory table of your Windows Installer package. Managing shared components can be problematic and should be avoided, if possible. A developer who does not install shared components consistently can end up with Component Object Model (COM) registration information pointing to older components. The Windows Installer Merge Modules (MSM) feature is specifically designed to enable shared components to consistently install in the context of all packages that install the shared component. Other problems arise when modifications of shared components cause existing applications to fail. One way to address this issue is for applications to be built using Microsoft .NET– or Win32–versioned assemblies.
5. **Perform setup rollback if an installation fails.** Partially installed software can fail in strange and unexpected ways providing for a poor user experience. Windows Installer supports this rollback feature.
6. **Do not install application shortcuts all over the user's profile**. While it may be tempting to add your application icon to every known exposure point in Windows, it often results in users feeling that they have lost control of their computer. Users are then forced to manually remove these shortcuts to return the computer to a desired look and feel. If the developer wants to add icons to the desktop, ask the user for permission during the installation. Windows Vista addresses discoverability of applications post install and includes the most recently used application list to avoid excessive Start menu traversing.
7. **Avoid automatically launching background applications at user logon.** Although it is possible to add programs to the startup group or Run key during installation, it adds overhead to the system. Over time, the performance of the user's

computer can significantly degrade. If your application can benefit from a background task, allow it to be user-configurable. Also, adding a startup task with the HLKM run key may prevent a standard user account from modifying the behavior in the future. If the user wants an application to launch at logon, store the information in the run key of HKEY_CURRENT_USER.

8. **Follow clean removal logic.** A user might remove an application not only to free up disk space, but also to return the computer to its state prior to the application being installed. The application's uninstall process should correctly and fully remove the application. Windows Installer defaults to the following removal rules:

    - All non-shared application files and folders.
    - Shared application files whose reference count (refcount) reaches zero.
    - Registry entries, except for keys that might be shared by other programs.
    - All shortcuts from the Start menu that the application created at the time of installation.
    - User preferences may be considered user data and left behind, but an option to do a completely clean removal should be included.
    - The uninstaller itself (if not using Windows Installer).

## Step Six: Create and Embed an Application Manifest with Your Application

In Windows Vista, the correct way to mark your applications is to embed an application manifest within your program that tells the operating system what the application needs. In the Windows Vista release, there are provisions to allow non-manifested or unsigned code to run with a full administrative access token.

**Note**

In future releases, the ONLY way to run an application elevated will be to have a signed application manifest that identifies the privilege level that the application needs.

### Application Manifest Schema

Application manifests are not new to the Windows Vista release. Manifests were used in Windows XP to help application developers identify such things as which versions of DLLs the application was tested with. Providing the execution level is an extension to that existing application manifest schema.

The Windows Vista application manifest has been enhanced with attributes that permit developers to mark their applications with a requested execution level. The following is the format for this:

```
<requestedExecutionLevel
level="asInvoker|highestAvailable|requireAdministrator"
uiAccess="true|false"/>
```

### Requested Execution Levels

**Possible Requested Execution Level Values**

| Value | Description | Comment |
|---|---|---|
| asInvoker | The application runs with the same access token as the parent process. | Recommended for standard user applications. Do refractoring with internal elevation points, as per the guidance provided earlier in this document. |
| highestAvailable | The application runs with the highest privileges the current user can obtain. | Recommended for mixed-mode applications. Plan to refractor the application in a future release. |
| requireAdministrator | The application runs only for administrators and requires that the application be launched with the full access token of an administrator. | Recommended for administrator only applications. Internal elevation points are not needed. The application is already running elevated. |

**Note**

Hosting applications can become standard user or administrator-only applications only if they support that certain type of hosted application. For example, the MMC now only hosts administrative snap-ins, and Explorer.exe only hosts standard user code.

**System behavior**

| Application Marking | Virtualize? |
|---|---|
| Unmarked | Yes |
| asInvoker | No |
| requireAdministrator | No |
| highestAvailable | No |

### Application Manifest Marking and Application Launch Behavior

This section details the behavior of the elevation prompt depending on the parent process access token, the setting for the **User Account Control: Behavior of the elevation prompt for administrators in Admin Approval Mode** policy and the **User Account Control: Behavior of the elevation prompt for standard users** policy, and the requested execution level marking for the application.

Whether an application can run and which user rights and administrative Windows privileges it can obtain are dependent upon the combination of the application's requested execution level in the application compatibility database and the administrative privileges available to the user account that launched the application. The following tables identify the possible run-time behavior based on such possible combinations.

**Application launch behavior for a member of the local Administrators group**

| Parent Process Access Token | Consent Policy for Members of the Local Administrators Group | None or asInvoker | highestAvailable | requireAdministrator |
|---|---|---|---|---|
| Standard user | No prompt | Application launches as a standard user | Application launches with a full administrative access token; no prompt | Application launches with a full administrative access token; no prompt |

| Parent Process Access Token | Consent Policy for Members of the Local Administrators Group | None or asInvoker | highestAvailable | requireAdministrator |
|---|---|---|---|---|
| Standard user | Prompt for consent | Application launches as a standard user | Application launches with a full administrative access token; prompt for consent | Application launches with a full administrative access token; prompt for consent |
| Standard user | Prompt for credentials | Application launches as a standard user | Application launches with a full administrative access token; prompt for credentials | Application launches with a full administrative access token; prompt for credentials |
| Administrator (UAC is disabled) | NA | Application launches with a full administrative access token; no prompt | Application launches with a full administrative access token; no prompt | Application launches with a full administrative access token; no prompt |

**Application launch behavior for a standard user account**

| Parent Process Access Token | Consent Policy for Standard Users | asInvoker | highestAvailable | requireAdministrator |
|---|---|---|---|---|
| Standard user | No prompt | Application launches as a standard user | Application launches as a standard user | Application fails to launch |

| **Parent Process Access Token** | **Consent Policy for Standard Users** | **asInvoker** | **highestAvailable** | **requireAdministrator** |
|---|---|---|---|---|
| Standard user | Prompt for credentials | Application launches as a standard user | Application launches as a standard user | Prompt for administrator credentials before running application |
| Standard user (UAC is disabled) | NA | Application launches as a standard user | Application launches as a standard user | Application might launch but will fail later |

**Application launch behavior for a standard user with additional privileges (E.G. Backup Operator)**

| **Parent Process Access Token** | **Consent Policy for Standard Users** | **asInvoker** | **highestAvailable** | **requireAdministrator** |
|---|---|---|---|---|
| Standard user | No Prompt | Application launches as a standard user | Application launches as a standard user with additional privileges | Application fails to launch |
| Standard user | Prompt for credentials | Application launches as a standard user | Prompt for credentials before running the application | Prompt for administrator credentials before running  application |
| Standard user (UAC is disabled) | NA | Application launches as a standard user | Application launches as a standard user with additional privileges | Application might launch but will fail later |

**uiAccess Values**

**Possible uiAccess values**

| Value | Description |
|---|---|
| False | The application does not need to drive input to the user interface of another window on the desktop. Applications that are not providing accessibility should set this flag to false. Applications that are required to drive input to other windows on the desktop (on-screen keyboard, for example) should set this value to true. |
| True | The application is allowed to bypass user interface control levels to drive input to higher privilege windows on the desktop. This setting should only be used for user interface Assistive Technology applications. |

**Important**

Applications with the **uiAccess** flag set to **true** must be **Authenticode signed** to start properly. In addition, the application must reside in a protected location in the file system. \Program Files\ and \Windows\System32\ are currently the two allowable protected locations.

## How to Create an Embedded an Application Manifest with Microsoft Visual Studio®

Visual Studio® provides the capability to automatically embed an XML application manifest file within the resource section of the Portable Executable (PE) image. This section addresses how to use Visual Studio to create a signed PE image containing an application manifest. This application manifest can therefore include the necessary requestedExecutionLevel attributes, allowing the application to run with the desired privilege level on Windows Vista. When the program is launched, the application manifest information will be extracted from the resource section of the PE and used by the operating system. It is not necessary to use the Visual Studio graphical user interface (GUI) to include a manifest. Once the necessary changes are in the source code, compiling and linking using command-line tools will also include the application manifest in the resulting PE image.

## Manifest File

To mark your application with a requestedExecutionLevel, first create an application manifest file to use with the target application. This file can be created by using any text editor. The application manifest file should have the same name as the target executable file with a **.manifest** extension. For example: IsUserAdmin.exe.manifest.

### Example

```
Executable: IsUserAdmin.exe
Manifest:IsUserAdmin.exe.manifest
Sample application manifest file:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity version="1.0.0.0"
     processorArchitecture="X86"
     name="IsUserAdmin"
     type="win32"/>
  <description>Description of your application</description>
  <!-- Identify the application security requirements. -->
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel
          level="requireAdministrator"
          uiAccess="false"/>
        </requestedPrivileges>
       </security>
  </trustInfo>
</assembly>
```

The parts of the application manifest that need to be adjusted for your application are marked in the previous example in bold. They include the following:

- The assembly identity
- The name
- The type
- The description
- The attributes in the requestedExecutionLevel

## Building Application Manifests with Visual Studio® 2005 for Windows Vista Only Applications

**Important**

If your application is intended to run on both Windows Vista and Windows XP, you must follow the procedures detailed in the next section: Building and Embedding an Application Manifest with Microsoft Visual Studio 2005 for Windows XP and Windows Vista Applications.

Next, you have to attach the application manifest to the executable by adding a line in the resource file of the application (the .rc file) to have Microsoft Visual Studio embed your manifest within the resource section of the PE file. To accomplish this, place the application manifest in the same directory as the source code for the project you are building and edit the resource file to include the following lines:

```
#define MANIFEST_RESOURCE_ID 1
MANIFEST_RESOURCE_ID RT_MANIFEST "IsUserAdmin.exe.manifest"
```

Replace IsUserAdmin.exe.manifest with the name of your application's manifest. After rebuilding the application, the application manifest should be embedded in the resource section of the executable.

## Building and Embedding an Application Manifest with Microsoft Visual Studio® 2005 for Windows XP and Windows Vista Applications

In Visual Studio 2005, the C/C++ integrated development environment (IDE) interface that permits the inclusion of additional manifest files in a target executable file does some processing on the XML, which inserts a duplicate **xmlns** tag.  Because of this, the previously documented method on how to include an application manifest in a Visual Studio 2005 C++ project cannot be used if the application should run on both Windows Vista and Windows XP. The following procedures are modified to include explicit version tags in the trustInfo section.

A fix is planned for the mt.exe tool to address the problem where it generates the duplicate namespace declaration in the XML. Until a new version of mt.exe is available, you can avoid the problem of merging application manifests by explicitly adding in version tags into the trustinfo section of the manifest. A sample application manifest is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
   <ms_asmv2:trustInfo xmlns:ms_asmv2="urn:schemas-microsoft-com:asm.v2">
      <ms_asmv2:security>
         <ms_asmv2:requestedPrivileges>
```

```
            <ms_asmv2:requestedExecutionLevel level="asInvoker">
            </ms_asmv2:requestedExecutionLevel>
         </ms_asmv2:requestedPrivileges>
      </ms_asmv2:security>
   </ms_asmv2:trustInfo>
</assembly>
```

### C or C++ Project

The following procedure details how to create an application manifest for a C or C++ project type in Visual Studio 2005.

**To create a manifest for a C or C++ project in Microsoft Visual Studio 2005**

1. Open your project in Microsoft Visual Studio 2005
2. Under Project, select **Properties**.
3. In **Properties**, select **Manifest Tool**, and then select **Input and Output**.
4. Add in the name of your application manifest file under **Additional manifest files**.
5. Rebuild your application.

**Note**

The updated manifests that include explicit version tags will permit the application to run correctly on both Windows Vista and Windows XP.

### Managed Code (C#, J# and Visual Basic)

Visual Studio does not currently embed a default application manifest into managed code. For managed code, the developer should insert a default application manifest into the target executable using mt.exe. The following procedure details this process.

**To insert a default manifest file into the target executable with mt.exe**

1. Use a text editor, such as Windows Notepad, to create a default manifest file, temp.manifest.
2. Use mt.exe to insert the manifest. The command would be: `mt.exe –manifest temp.manifest –outputresource:YourApp.exe;#1`

### Adding the Application Manifest as a Step in Visual Studio Post-Build

Adding the application manifest can be automated as a post-build step as well. This option is available for C/C++ and for the two managed code languages of C# and J#.

**Note**

The IDE does not currently include a post-build option for a Visual Basic application.

To automate the addition of the application manifest as a post-build step, place the following line as a post build task in your Visual Studio project's **Project Properties**:

```
mt.exe -manifest "$(ProjectDir)$(TargetName).exe.manifest" -
updateresource:"$(TargetDir)$(TargetName).exe;#1"
```

## Step Seven: Test Your Application

Next, you will need to test your redesigned or new application for application compatibility with the Standard User Analyzer. A procedure detailing this process was described earlier in this document in the Test Your Application for UAC Compatibility section.

Use the following workflow to test your application.

**To test your application for final UAC compatibility**

1. Test the application with the Standard User Analyzer tool.
2. Log on to a Windows Vista computer as an administrator in Admin Approval Mode and run your program. Ensure that you test all functionality and note the user experience. File any elevation or user interface bugs accordingly.
3. Log on to a Windows Vista computer as a standard user and run your program. Ensure that you test all functionality and note any differences or failures in the standard user experience in comparison to the administrator in Admin Approval Mode user experience. Record any elevation and user experience problems accordingly.

## Step Eight: Authenticode Sign Your Application

The application now contains an application manifest, which will be detected when the application launches. The executable can, however, be tampered with. To prevent this, you should sign the application with an Authenticode signature.

**Note**

Windows Vista will have the ability to prevent any unsigned application from launching with a full administrator access token. If you want your application to operate correctly in locked-down environments, while displaying a more user friendly user interface, it should be signed with an Authenticode signature.

To sign the application, you can either generate a certificate from makecert.exe or obtain a code-signing key from one of the commercial certification authorities (CAs), such as VeriSign, Thawte, or a Microsoft CA.

**Note**

You will need a commercial certificate if you wish your application to be trusted on the target computer of a customer installing your application.

If you use the makecert.exe file to generate your signing key pair, be aware that it only generates a 1024-bit key. Authenticode signatures should have at least a 2048-bit key. The makecert.exe file should only be used for testing purposes.

The following procedure details the high level requirements for using makecert.exe to generate your signing key pair. An example and makecert.exe parameters follow this procedure.

**To use makecert.exe to generate your signing key pair**

1. Generate the certificate.
2. Sign the code.
3. Install the test certificate.

## Example Signing Procedure

The following procedures are provided as examples and are not intended to be strictly followed. For example, replace the test certificate's name with your certificate's name and ensure that you tailor the procedures to map to your specific CA and development environment.

### Step 1: Generate the certificate

```
makecert -r -pe -ss PrivateCertStore -n "CN=Contoso.com(Test)" ContosoTest.cer
```

**makecert.exe parameters**

| Parameter | Description |
|---|---|
| /r | Create self-signed certificate |
| /pe | Makes the certificate's private key exportable to the signing machine. |
| /ss StoreName | The certificate store name that will store the test certificate. Example: PrivateCertStore |
| /n X500Name | The certificate subject's X500 name. Example: Contoso.com(Test) |
| CertificateName.cer | Certificate name. Example: ContosoTest.cer |

### Step 2: Sign the Code

**Important**

Applying a timestamp while signing your application will ensure that the application will continue to run after the validity period of the original certificate.

```
Signtool sign /v /s PrivateCertStore /n Contoso.com(Test) /t
http://timestamp.verisign.com/scripts/timestamp.dll file.exe
```

### Step 3: Install the Test Certificate

**To install the test certificate**

1. Launch an elevated command window by right-clicking **Command Prompt** and selecting **Run as administrator**.
2. In **Command Prompt**, type mmc.exe and press **Enter**.
3. In the mmc, select **File** and then select **Add/Remove Snap-in…**
4. In **Add or Remove Snap-ins**, select **Certificates**, click **Add**, and then click **OK**.
5. In the **Certificates snap-in** dialog box, select **Computer account** and click **Next**.
6. In **Select Computer**, select **Local Computer**, and then click **OK**.
7. In Add or Remove Snap-ins, click **OK**.

8. In the **Certificates** snap-in, and navigate to **Trusted Root Certificate Authorities**, right-click **Certificates**, select **All Tasks**, and then select **Import…**
9. In the Certificate Import Wizard, import the test certificate, ContosoTest.cer.

For more information about Authenticode signatures, see MSDN:

Frequently Asked Questions about Authenticode (http://go.microsoft.com/fwlink/?LinkId=71496).

Microsoft Authenticode Technology (http://go.microsoft.com/fwlink/?LinkId=71361).

### Step Nine: Participate in the Windows Vista Logo Program

Microsoft offers the Windows Vista Logo program to help customers identify systems and peripherals that meet a comprehensive baseline definition of platform features and quality goals to ensure a great computing experience for users.

Preliminary guidelines for the UAC requirements for obtaining a Windows Vista Logo are available at the Windows Vista Logo page (http://go.microsoft.com/fwlink/?LinkId=71497).

## Deploying and Patching Applications for Standard Users

Generally, enterprises will have to consider how they will install applications on their users' workstations in an automated manner, thereby reducing administrative costs. There are fundamentally two parts to this problem--first, how these applications should be packaged for deployment and, second, what technology should be used to deploy them. In the case of smaller enterprise environments, a robust, automated deployment mechanism may not be necessary.

Assuming that the enterprise has already taken an inventory of the software that is run in its environment, the next step is to repackage these applications for deployment. Microsoft recommends the Windows Installer format because it has the unique ability to separate managing per-user settings from per-machine settings. This type of management generally is not possible with other packaging formats, especially deployment executables that are simply run by an account with more privileges, such as SYSTEM. The MSDN library (http://go.microsoft.com/fwlink/?LinkId=71498) contains many articles on Windows Installer; one suggestion is the Roadmap to Windows Installer documentation (http://go.microsoft.com/fwlink/?LinkId=71499).

The Windows Installer format includes the ability to user control the installation of these applications through Group Policy (Microsoft® IntelliMirror) and also through SMS.  To enable Install on Demand with file extension or shortcuts, the following tables in the Windows Installer–based package must be populated with advertising data: Shortcut, Extension, Icon, and Verb. It is recommended that you also populate class, MIME, ProgID, and TypeLib. More information about IntelliMirror and Install on Demand is available at MSDN (http://go.microsoft.com/fwlink/?LinkId=71492).

There are other installer technologies that allow applications to install per-user and support auto-update, such as ClickOnce. This means that the installer will not require administrator or higher privileges to install and that the user will always run the latest version as long as the computer is connected to the network. It also places some limits on an IT professional's ability to control the installation of these applications.

ClickOnce (http://go.microsoft.com/fwlink/?LinkId=71500) deployment is a Microsoft .NET installation technology that automatically installs and configures a client-side application when a user clicks an application manifest link, such as an application manifest in a Web site, on a CD, or on a universal naming convention (UNC) path. By default, the application will copy itself to the Temporary Internet Files folder and run within a restricted environment.

**Note**

Even if your application has been signed with the IT strong name that gives it Full Trust, you still cannot do anything that requires administrator permissions, such as access certain parts of the file system and registry. ClickOnce applications however, are targeted as per-user applications, so this should not be a problem.

**Important** ClickOnce should not be used for deploying applications that perform administrative operations.

## Deploying to a Single Computer

To deploy an application for a single computer, the administrator must “publish” the application on that computer.

## Deploying to all users in a Domain

To advertise for all users in a domain, the administrator must “publish” the application through Group Policy deployment. Currently, only the Group Policy–based software deployment component of the Windows Server® 2003 operating systems and Windows® 2000 Server operating system takes advantage of this functionality.

## Patching Applications as a Standard User with Windows Installer 4.0

Standard user account patching enables Windows Installer package authors to identify signed patches that can be applied by a future standard user. The following conditions must be met to enable standard user patching with Windows Installer 4.0:

- The application was installed on using Windows Installer 4.0.
- The application was originally installed per-machine.
- The MsiPatchCertificate table is present and populated in the original Window Installer package (.msi file).
- The patches are digitally signed by a certificate listed in the MsiPatchCertificate table.
- The patches can be validated against the digital signature.
- Standard user account patching has not been disabled by setting the MSIDISABLELUAPATCHING property or the DisableLUAPatching policy.

### Windows Installer 4.0 Standard User Uninstall Behavior

The expected behavior for a Windows Installer 4.0 patch applied by a standard user is that it can also be removed by the standard user.

### Launching an Un-Elevated Application from an Elevated Process

A frequently asked question is how to launch an un-elevated application from an elevated process, or more fundamentally, how to I launch a process using my un-elevated token once I'm running elevated.  Since there is no direct way to do this, the situation can usually be avoided by launching the original application as standard user and only elevating those portions of the application that require administrative rights.  This way there is always a non-elevated process that can be used to launch additional applications as the currently logged on desktop user.  Sometimes, however, an elevated process needs to get another application running un-elevated.  This can be accomplished by using the task scheduler within Windows Vista.  The elevated process can register a task to run as the currently logged on desktop user.

A C ++ code sample illustrating how to use the Task Scheduler to perform this operation is available in the references section of this document.

# Troubleshooting Common Issues

The following sections detail common issues encountered with applications in Windows Vista.

Common issues include:

- ActiveX installation issues
- ActiveX documents do not install
- Application, framework, or add-in required
- Administrative permission is required for installation/patching
- Per-user application settings locations
- Application defaults to saving in a protected directory

## ActiveX Installation Issues

ActiveX controls must be installed by an administrator. ActiveX controls are typically used in LOB applications to extend Web browser capabilities to create more flexible user interfaces or to elevate access to computer resources normally denied to applications running within the Web browser. ActiveX controls are typically installed by embedding a reference to the ActiveX control in a Web page. This will cause Microsoft Internet Explorer® to download and install the control if it does not exist on the local computer. Typically, ActiveX controls downloaded in this way reside in the %HOMEPATH%\Local Settings\Temporary Internet Files directory, which is writable by standard users. However, to function within Internet Explorer, the controls must have multiple-registry entries, which standard users cannot access.

### Resolution

Removing the ActiveX control from the application almost always results in a loss of functionality. Therefore, this is not recommended for remediation unless the ActiveX control is providing some visual or functional enhancement that is not part of the site's core functionality. An example is a stock ticker on a non-stock–related portal.

In most cases, packaging the ActiveX control for installation by SMS or Group Policy is the correct solution. However, most of the controls will not be included in the base image, so Web sites must modify their pages to fail gracefully. This should comprise detecting the missing ActiveX control and redirecting to the Managed Desktop software request page.

## ActiveX Documents Do Not Install

ActiveX documents are a deprecated technology from Microsoft Visual Basic® 4 and Microsoft Visual Basic® 5. They can be downloaded in a similar way as ActiveX controls.

### Resolution

Since Visual Basic 4 and Visual Basic 5 are deprecated, Microsoft recommends that you replace the application. It should be possible to install the ActiveX document as part of a client installation; however, updates to the document will be restricted without redeployment through SMS or Group Policy.

## Application, Framework, or Add-in Required

Many applications have dependencies on other software, which may not be installed by default, either because they are already available on the computer or because the other application does not provide distributable binaries for use by third parties. Under normal circumstances, the user would be directed to acquire and install the additional software. Under a managed desktop, installation is not possible. Examples include Adobe Acrobat, Microsoft Office, Office Web components, and WinZip.

### Resolution

Once the dependencies are identified, they can either be packaged with the base image or made available through on-demand SMS installation. The application might have to change how it notifies the end user of the missing software, directing the user to the SMS installation site instead of to the manufacturer.

## Administrative Permission is Required for Installation/Patching

Since installation of a program requires adding files to the Program Files directory, it will always require administrative permissions and, therefore, must be run as a user with elevated permissions.

**Note**

You can also "push" the patch with SMS or Group Policy in conjunction with the Add or Remove Programs (ARP) control panel. In this method, the user selects the software to install and the system installer completes the installation—the user does not have to be an administrator. For initial installations, this can be dealt with by packaging the software for an installation agent to push out.

However, some applications rely on frequent automatic updates that may not align well with a centrally managed application model.

Applications that detect updates and attempt to apply patches will be unable to do so, as they will not have permission to modify files in the system directories.

### Resolution

- Package your application/patch for deployment with SMS. Applications can still detect that an upgrade is available (as long as they do it without requiring administrative permissions) and can redirect to the provisioning site.
- Question whether your application needs elevated computer permissions, such as file system, registry access, or COM interoperability. If not, then it might be possible to rewrite the application as a ClickOnce deployment package, which will run in the Microsoft .NET sandbox.
- Convert to a Web application without any client-side dependencies.

## Per-User Application Settings Locations

For Windows Vista, the application settings that need to be changed at run time should be stored in one of the following locations:

- CSIDL_APPDATA
- CSIDL_LOCAL_APPDATA
- CSIDL_COMMON_APPDATA

Documents saved by the user should be stored in CSIDL_MYDOCUMENTS.

**Note**

A user's **Documents** folder is no longer stored under **Documents and Settings**. In Windows Vista, a new root directory on the file system called **Users** now contains the profiles for users of the computer.

Because these directories have changed, developers are encouraged to use CSIDLs to locate the path to specific well-known directories in a system-independent way. For more information, see the MSDN article on CSIDLs (http://go.microsoft.com/fwlink/?LinkId=71501).

An application needs write access to the file system. When running under a managed desktop, an application only has write permission to the following folders and their children.

- CSIDL_PROFILE
- CSIDL_COMMON_APPDATA

**Note**

Standard users cannot write to Users\Common.

- C:\Users\Common>cd "Application Data"

--C:\Users\Common\Application Data>echo File > File.txt

--C:\Users\Common\Application Data>

Applications should not attempt to write to other locations, such as the following:

- C:\Windows
- C:\Windows\System32
- Program Files\{application}
- C:\{application}

**Note**

This will work if the user created the folder, which members of the Users group can do by default.

An application is trying to specifically create C:\Users\Profiles\Username is not allowed since the user can only create folders under C:\Users\Username. The location chosen appears to be confused based on where Microsoft has stored the **Documents** folder on previous versions of the operating system.

Application settings that need to be changed at run time should be stored in one of the following locations:

- CSIDL_APPDATA
- CSIDL_LOCAL_APPDATA
- CSIDL_COMMON_APPDATA

Documents saved by the user should be stored in the CSIDL_MYDOCUMENTS folder.

All paths should not be hard-coded but should use the Environment.GetFolderPath() function.

### Application Defaults to Saving in a Protected Directory

Some applications allow users to save or export data to their local computer. Often, the dialog box defaults to places like C:, to which standard users do not have write permissions. In addition, some applications do not respond well when the code to write the file fails because as a result of an access denied from the operating system.

#### Resolution

Assume that users can only write to their own profiles. For documents intentionally saved by users, initialize the dialog boxes to start at **Documents** (Environment.GetFolderPath(Environment.SpecialFolder.Personal). Remember that the **Save** dialog box will allow a user to browse to other locations than the user's profile, so the application should include logic to ensure that it fails gracefully if a user choose a different directory than those located in his/her profile.

## References

This section includes a virtualization reference and a security settings reference.

### Virtualization Reference

#### File virtualization

- Virtualize (%SYSTEMROOT%, %PROGRAMDATA%, %PROGRAMFILES%\(Subdirectories)
- Redirect to: %LOCALAPPDATA%\VirtualStore
- Excluded binary executables: .exe, .dll, .sys

#### Registry Virtualization:

- Virtualize (HKEY_LOCAL_MACHINE\SOFTWARE)
- Redirect to: HKEY_CURRENT_USER\Software\Classes\VirtualStore\MACHINE\SOFTWARE\<Application Registry Keys>
- Keys excluded from virtualization
- HKEY_LOCAL_MACHINE\Software\Classes

- HKEY_LOCAL_MACHINE \Software\Microsoft\Windows
- HKEY_LOCAL_MACHINE \Software\Microsoft\Windows NT

### Applicability

- Virtual stores do not roam
- Corresponding global objects would not roam
- Enabled only for interactive standard users
- Disabled for non-interactive processes
- Disabled for 64-bit executables
- Disabled for executables that request an execution level (requestedExecutionLevel) in their application manifest, the model for separation
- Disabled for kernel mode and impersonated callers
- Only administrator writeable registry keys and files are virtualized

## UAC Security Settings Reference

This reference details the security settings available to administer UAC with Group Policy or the computer's local security policy.

**Note**

The procedures presented in this section are intended for administering unmanaged computers. To use Group Policy to administer the settings centrally in a managed environment, use Active Directory Users and Computers (dsa.msc) instead of local Security Policy Manager snap-in (secpol.msc).

### Configuring UAC Security Settings

The following procedure details how to configure the UAC security settings with the Security Policy Manager. The procedure details the default user experience for an administrator in Admin Approval Mode.

**To view/set the UAC security settings with Security Policy Manager**

1. Click the **Start** button, type **secpol.msc** into the search box, and then press **Enter**.
2. At the **User Account Control** consent prompt, click **Continue**.

3. In **Local Security Settings**, expand **Local Policies**, and then click **Security Options**.
4. Right-click the security setting that you would like to change and select **Properties**.

The following procedure details how to configure the UAC security settings with the Group Policy. The procedure details the default user experience for an administrator in Admin Approval Mode.

**To view/set the UAC security settings with the Group Policy Object Editor**

1. Click the **Start** button, type **gpedit.msc** into the search box, and then press **Enter**.
2. At the **User Account Control** consent prompt, click **Continue**.
3. In **Group Policy**, expand **User Configuration**, and then expand **Security Options**.
4. Right-click the security setting that you would like to change and select **Properties**.

## UAC Security Settings

The following table lists the configurable UAC security settings. These settings can be configured with the Security Policy Manager (secpol.msc) or managed centrally with Group Policy (gpedit.msc).

**UAC settings**

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Admin Approval Mode for the Built-in Administrator account. | There are two possible settings:<br>• Enabled - The built-in Administrator will be run as an administrator in Admin Approval Mode.<br>• Disabled - The administrator runs with a full administrator access token. | • Disabled for new installations and for upgrades where the built-in Administrator is NOT the only local active administrator on the computer. The built-in Administrator account is disabled by default for installations and upgrades on domain-joined computers.<br>• Enabled for upgrades when Windows Vista determines that the built-in Administrator account is the only active local administrator on the computer. If Windows Vista determines this, the built-in Administrator account is also kept enabled following the upgrade. The built-in Administrator account is disabled by default for installations and upgrades on domain-joined computers. |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Behavior of the elevation prompt for administrators in Admin Approval Mode | There are three possible values:<br>• No prompt – The elevation occurs automatically and silently. This option allows an administrator in Admin Approval Mode to perform an operation that requires elevation without consent or credentials. Note: this scenario should only be used in the most constrained environments and is NOT recommended.<br>• Prompt for consent – An operation that requires a full administrator access token will prompt the administrator in Admin Approval Mode to select either **Continue** or **Cancel**. If the administrator clicks **Continue**, the operation will continue with their highest available privilege.<br>• Prompt for credentials – An operation that requires a full administrator access token will prompt an administrator in Admin Approval Mode to enter an administrator user name and password. If the user enters valid credentials, the operation will continue with the applicable privilege. | Prompt for consent |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Behavior of the elevation prompt for standard users | There are two possible values:<br>• No prompt – No elevation prompt is presented and the user cannot perform administrative tasks without using **Run as administrator** or by logging on with an administrator account. Most enterprises running desktops as standard user will configure the “No prompt” policy to reduce help desk calls.<br>• Prompt for credentials – An operation that requires a full administrator access token will prompt the user to enter an administrative user name and password.  If the user enters valid credentials the operation will continue with the applicable privilege. | • Home: Prompt for credentials<br>• Enterprise: No prompt |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Detect application installations and prompt for elevation | There are two possible values:<br>• Enabled - The user is prompted for consent or credentials when Windows Vista detects an installer.<br>• Disabled - Application installations will silently fail or fail in a non-deterministic manner. Enterprises running standard users desktops that leverage delegated installation technologies like GPSI or SMS will disable this feature. In this case, installer detection is unnecessary and therefore not required. | Enabled |
| User Account Control: Only elevate executables that are signed and validated | There are two possible values:<br>• Enabled - Only signed executable files will run. This policy will enforce PKI signature checks on any interactive application that requests elevation. Enterprise administrators can control the administrative application allowed list through the population of certificates in the local computers Trusted Publisher Store.<br>• Disabled - Both signed and unsigned code will be run. | Disabled |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Only elevate UIAccess applications that are installed in secure locations | There are two possible values:<br>• The system will only give UIAccess privileges and user rights to executables that are launched from under %ProgramFiles% or %windir%. The ACLs on these directories ensure that the executable is not user-modifiable (which would otherwise allow elevation of privilege). UIAccess executables launched from other locations will launch without additional privileges (i.e. they will run "asInvoker").<br>• Disabled - The location checks are not done, so all UIAccess applications will be launched with the user's full access token upon user approval. | Enabled |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Run all administrators in Admin Approval Mode | There are two possible values:<br>• Enabled - Both administrators and standard users will be prompted when attempting to perform administrative operations. The prompt style is dependent on policy.<br>• Disabled - UAC is essentially "turned off" and the AIS service is disabled from automatically starting. The Windows Security Center will also notify the logged on user that the overall security of the operating system has been reduced and will give the user the ability to self-enable UAC.<br>Note: Changing this setting will require a system reboot. | Enabled |
| User Account Control: Switch to the secure desktop when prompting for elevation | There are two possible values:<br>• Enabled - Displays the UAC elevation prompt on the secure desktop. The secure desktop can only receive messages from Windows processes, which eliminates messages from malicious software.<br>• Disabled - The UAC elevation prompt is displayed on the interactive (user) desktop. | Enabled |

| Setting | Description | Default Value |
|---|---|---|
| User Account Control: Virtualize file and registry write failures to per-user locations | There are two possible values:<br>• Enabled - This policy enables the redirection of pre-Windows Vista application write failures to defined locations in both the registry and file system. This feature mitigates those applications that historically ran as administrator and wrote runtime application data back to %ProgramFiles%; %Windir%; %Windir%\system32; or HKLM\Software\.... This setting should be kept enabled in environments that utilize non-UAC compliant software.  Applications that lack an application compatibility database entry or a requested execution level marking in the application manifest are not UAC compliant.<br>• Disabled - Virtualization facilitates the running of pre-Windows Vista (legacy) applications that historically failed to run as a standard user. An administrator running only Windows Vista compliant applications may choose to disable this feature as it is unnecessary. Non-UAC compliant applications that attempt to write %ProgramFiles%; %Windir%; %Windir%\system32; or HKLM\Software\.... will silently fail if this setting is disabled. | Enabled |

**Note**

Modifying the **User Account control: Run all administrators in Admin Approval Mode** setting will require a computer restart before the setting becomes effective. All other UAC Group Policy settings are dynamic and do not require a reboot.

## Task Scheduler Code Sample

The following C++ code sample illustrates how to use Task Scheduler to run an un-elevated application as the currently logged on desktop user from an elevated process. This method works for both the consent prompt and the credential prompt.

```
//---------------------------------------------------------------------
//  This file is part of the Microsoft .NET Framework SDK Code Samples.
//
//  Copyright (C) Microsoft Corporation.  All rights reserved.
//
//This source code is intended only as a supplement to Microsoft
//Development Tools and/or on-line documentation.  See these other
//materials for detailed information regarding Microsoft code samples.
//
//THIS CODE AND INFORMATION ARE PROVIDED AS IS WITHOUT WARRANTY OF ANY
//KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
//IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
//PARTICULAR PURPOSE.
//---------------------------------------------------------------------

/*****************************************************************************
* Main.cpp - Sample application for Task Scheduler V2 COMAPI                *
Component: Task Scheduler
* Copyright (c) 2002 - 2003, Microsoft Corporation
* This sample creates a task to that launches as the currently logged on deskup
user. The task launches as soon as it is registered.
*
*****************************************************************************/
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <comdef.h>
#include <comutil.h>
//Include Task header files - Included in Windows Vista Beta-2 SDK from MSDN
#include <taskschd.h>
#include <conio.h>
#include <iostream>
#include <time.h>
```

```
using namespace std;

#define CLEANUP \
pRootFolder->Release();\
        pTask->Release();\
        CoUninitialize();

HRESULT CreateMyTask(LPCWSTR, wstring);

void __cdecl wmain(int argc, wchar_t** argv)
{
wstring wstrExecutablePath;
WCHAR taskName[20];
HRESULT result;

if( argc < 2 )
{
printf("\nUsage: LaunchApp yourapp.exe" );
return;
}

// Pick random number for task name
srand((unsigned int) time(NULL));
wsprintf((LPWSTR)taskName, L"Launch %d", rand());

wstrExecutablePath = argv[1];

result = CreateMyTask(taskName, wstrExecutablePath);
printf("\nReturn status:%d\n", result);

}
HRESULT CreateMyTask(LPCWSTR wszTaskName, wstring wstrExecutablePath)
{
    //  ------------------------------------------------------
    //  Initialize COM.
TASK_STATE taskState;
int i;
    HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
    if( FAILED(hr) )
    {
        printf("\nCoInitializeEx failed: %x", hr );
        return 1;
    }

    //  Set general COM security levels.
    hr = CoInitializeSecurity(
        NULL,
        -1,
        NULL,
        NULL,
        RPC_C_AUTHN_LEVEL_PKT_PRIVACY,
        RPC_C_IMP_LEVEL_IMPERSONATE,
        NULL,
        0,
        NULL);

    if( FAILED(hr) )
    {
        printf("\nCoInitializeSecurity failed: %x", hr );
        CoUninitialize();
        return 1;
    }

    //  ------------------------------------------------------
    //  Create an instance of the Task Service.
    ITaskService *pService = NULL;
    hr = CoCreateInstance( CLSID_TaskScheduler,
                           NULL,
                           CLSCTX_INPROC_SERVER,
                           IID_ITaskService,
                           (void**)&pService );
    if (FAILED(hr))
    {
        printf("Failed to CoCreate an instance of the TaskService class: %x", hr);
        CoUninitialize();
        return 1;
    }

    //  Connect to the task service.
    hr = pService->Connect(_variant_t(), _variant_t(), _variant_t(),
_variant_t());
    if( FAILED(hr) )
    {
        printf("ITaskService::Connect failed: %x", hr );
        pService->Release();
        CoUninitialize();
        return 1;
    }

    //  ------------------------------------------------------
    //  Get the pointer to the root task folder.  This folder will hold the
    //  new task that is registered.
    ITaskFolder *pRootFolder = NULL;
    hr = pService->GetFolder( _bstr_t( L"\\") , &pRootFolder );
    if( FAILED(hr) )
    {
        printf("Cannot get Root Folder pointer: %x", hr );
        pService->Release();
        CoUninitialize();
        return 1;
    }
    //  Check if the same task already exists. If the same task exists, remove it.
    hr = pRootFolder->DeleteTask( _bstr_t( wszTaskName), 0  );

    //  Create the task builder object to create the task.
    ITaskDefinition *pTask = NULL;
    hr = pService->NewTask( 0, &pTask );

    pService->Release();  // COM clean up.  Pointer is no longer used.
    if (FAILED(hr))
    {
        printf("Failed to CoCreate an instance of the TaskService class: %x", hr);
        pRootFolder->Release();
        CoUninitialize();
        return 1;
    }


    //  ------------------------------------------------------
    //  Get the trigger collection to insert the registration trigger.
    ITriggerCollection *pTriggerCollection = NULL;
    hr = pTask->get_Triggers( &pTriggerCollection );
    if( FAILED(hr) )
    {
        printf("\nCannot get trigger collection: %x", hr );
  CLEANUP
        return 1;
    }

    //  Add the registration trigger to the task.
    ITrigger *pTrigger = NULL;

    hr = pTriggerCollection->Create( TASK_TRIGGER_REGISTRATION, &pTrigger );
    pTriggerCollection->Release();  // COM clean up.  Pointer is no longer used.
    if( FAILED(hr) )
    {
        printf("\nCannot add registration trigger to the Task %x", hr );
        CLEANUP
        return 1;
    }
    pTrigger->Release();

    //  ------------------------------------------------------
    //  Add an Action to the task.
    IExecAction *pExecAction = NULL;
    IActionCollection *pActionCollection = NULL;

    //  Get the task action collection pointer.
    hr = pTask->get_Actions( &pActionCollection );
    if( FAILED(hr) )
    {
        printf("\nCannot get Task collection pointer: %x", hr );
        CLEANUP
        return 1;
    }

    //  Create the action, specifying that it is an executable action.
    IAction *pAction = NULL;
    hr = pActionCollection->Create( TASK_ACTION_EXEC, &pAction );
    pActionCollection->Release();  // COM clean up.  Pointer is no longer used.
    if( FAILED(hr) )
    {
        printf("\npActionCollection->Create failed: %x", hr );
        CLEANUP
        return 1;
    }

    hr = pAction->QueryInterface( IID_IExecAction, (void**) &pExecAction );
    pAction->Release();
    if( FAILED(hr) )
    {
        printf("\npAction->QueryInterface failed: %x", hr );
        CLEANUP
        return 1;
    }

    //  Set the path of the executable to the user supplied executable.
   hr = pExecAction->put_Path( _bstr_t( wstrExecutablePath.c_str() ) );

    if( FAILED(hr) )
    {
        printf("\nCannot set path of executable: %x", hr );
        pExecAction->Release();
        CLEANUP
        return 1;
    }
    hr = pExecAction->put_Arguments( _bstr_t( L"" ) );

   if( FAILED(hr) )
    {
        printf("\nCannot set arguments of executable: %x", hr );
        pExecAction->Release();
        CLEANUP
        return 1;
    }

    //  ------------------------------------------------------
    //  Save the task in the root folder.
    IRegisteredTask *pRegisteredTask = NULL;
    hr = pRootFolder->RegisterTaskDefinition(
            _bstr_t( wszTaskName ),
            pTask,
      TASK_CREATE,
_variant_t(_bstr_t( L"S-1-5-32-545")),//Well Known SID for \\Builtin\Users group
_variant_t(),
TASK_LOGON_GROUP,
            _variant_t(L""),
            &pRegisteredTask);
    if( FAILED(hr) )
    {
        printf("\nError saving the Task : %x", hr );
        CLEANUP
        return 1;
    }
    printf("\n Success! Task successfully registered. " );
    for (i=0; i<100; i++)//give 10 seconds for the task to start
{
pRegisteredTask->get_State(&taskState);
if (taskState == TASK_STATE_RUNNING)
{
printf("\nTask is running\n");
break;
}
Sleep(100);
}
if (i>= 100) printf("Task didn't start\n");

    //Delete the task when done
    hr = pRootFolder->DeleteTask(
            _bstr_t( wszTaskName ),
            NULL);
    if( FAILED(hr) )
    {
        printf("\nError deleting the Task : %x", hr );
        CLEANUP
        return 1;
    }

    printf("\n Success! Task successfully deleted. " );

//  Clean up.
    CLEANUP
    CoUninitialize();
    return 0;
}
```