Autor: Thomas Reinwart

Applikationsentwicklung für Windows 7

2009-11-24

office@reinwart.com

Inhalt

Applikationsentwicklung für Windows 7

Mit der aktuellen Windows Version 7 stellt sich für Entwickler und Administratoren die Frage, welche der bisher eingesetzten Applikationen kompatibel sind. Den Entwickler interessiert, was am bestehenden Code zu ändern ist und was nicht (mehr) erlaubt ist. Den Admin interessiert, warum die Applikation unbedingt unter Admin Rechten laufen muss. Aber nicht nur das, warum und wann muss die Anwendung bei Windows 7 im XP Mode laufen.
Ein weiterer Aspekt sind die neuen Features des Betriebssystems, die für Windows 7 Applikationen genutzt werden können.

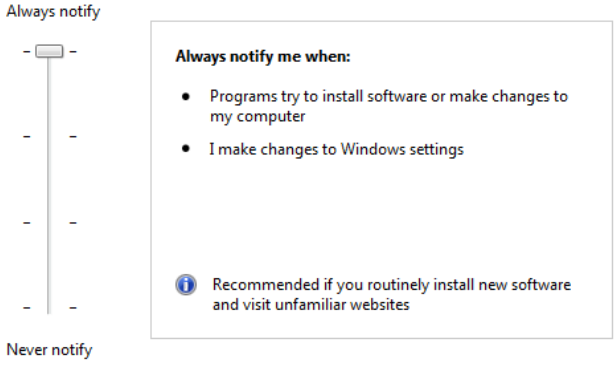
Windows 7 Features für Entwickler

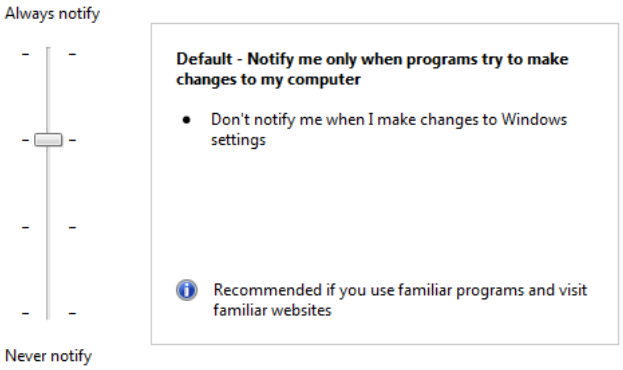
- Jumplists (Sprunglisten)
- Anwendungssymbole, Taskbar Button, Fortschrittsanzeigen
- Ribbon – Multifunktionsleisten
- Multitouch
- Windows Shell (API Code Pack)
- API für Sensor und Location (Licht und Bewegungssensoren)
- Windows 7 SDK (Code Samples)

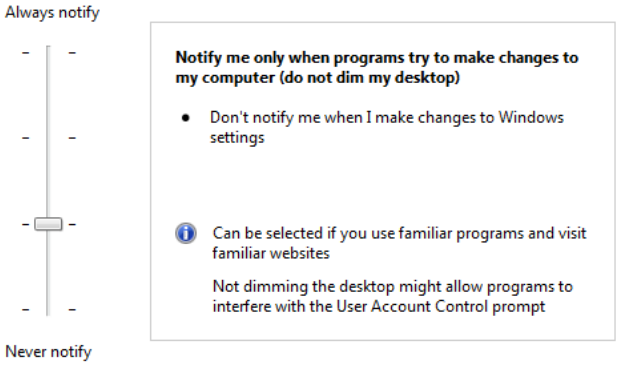
UAC

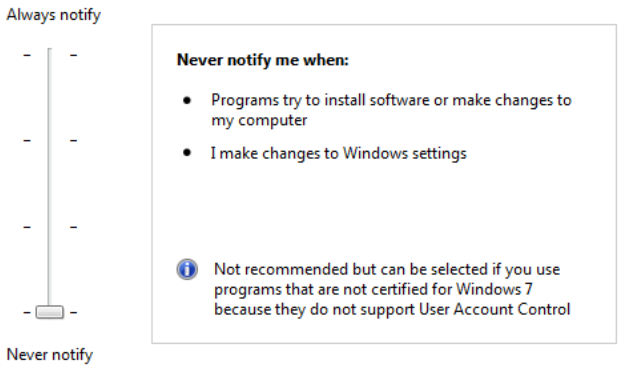
UAC steht für *User Account Control* (Benutzerkontensteuerung) und wurde mit Windows Vista eingeführt. Es handelt sich um einen Berechtigungsmechanismus, der bei Vista als häufiges Popup Fenster (elevation) dargestellt wurde. Im Popup Fenster wurde man Informiert, dass die gestartete Applikation Admin Rechte benötigt, um gestartet oder weiterarbeiten zu können. Beispiel bei Installation, Update Prüfung oder beim Speichern von Daten. Gleiches Verhalten ergab sich beim Starten der Systemsteuerung oder Änderungen an der Konfiguration des Rechners. Viele Applikation haben sich lange nicht an die Security Design Vorschriften für Vista Applikationen gehalten, dadurch haben viele Anwendungen Admin Rechte zum Starten benötigt. Die dadurch entstandene Meldungshäufigkeit der Popups war ein Kritikpunkt bei Vista. Bei Windows 7 kann die Meldungshäufigkeit der Popups definiert werden. Die „Always notify“ Einstellung in der Systemsteuerung von Windows 7 entspricht dem Verhalten von Windows Vista. Nichts desto Trotz sollte es nun an der Zeit sein, seine Anwendung Windows 7 (bzw. eigentlich schon Windows Vista) tauglich zu machen.

Windows 7 UAC Einstellungen:









Sinn von UAC

Ist es, die Angriffsmöglichkeiten auf das Betriebssystem zu reduzieren. Die Anwendung wird per default unter dem Standard Benutzer gestartet.
Applikationen die nicht den UAC Security Vorschriften entsprechen, müssen immer unter Admin Rechten laufen. Das hat den Nachteil, wenn es Sicherheitslücken in der Applikation gibt oder schadhafter Code eingeschleust wird, wird dieser unter den Admin Rechten ausgeführt und kann mehr Schaden anrichten als unter den Rechten eines Standard Benutzers. Eine Einladung für Angreifer.

Rechte eines Standard Users (ab Windows Vista):

- Ändern der Zeitzone
- WLAN Verbindung herstellen (WEP - Wired Equivalent Privacy installieren)
- VPN Verbindung herstellen
- Ändern der Energiesparoption
- Ändern der Anzeigeeinstellungen
- Schriftarten installieren

- Drucker installieren
- Installieren von Updates (Installer muss UAC kompatibel sein)

Man erkennt, dass ein Arbeiten auch als Nicht Admin selbst bei Reisetätigkeit möglich ist.

Least privileges

Dieser Ausdruck definiert den minimalen Rechtebedarf einer Applikation. Je weniger Rechte, umso weniger Möglichkeiten in das Betriebssystem einzugreifen. Beispiel sind angehängte Viren an der ausgeführten Anwendung. D.h. der Entwickler muss etwa dafür sorgen, seine Files an den erlaubten Stellen abzulegen und nicht etwa ein Temp File mit Windows\System32 Verzeichnis einzulegen, was Admin Rechte beim Ausführen der Anwendung benötigt. Ein weiteres Beispiel ist das Starten eines weiteren Prozesses. Unter dem Prinzip von *least privileges* wird eine Anwendung im vornhinein mit den Rechten des Standarduser gestartet und nicht mit Admin Rechten.

Welche Bereiche der Applikation müssen UAC fit gemacht werden?

Die meisten Anwendungen benötigen noch immer Zugriff auf Bereiche in Verzeichnisse und die Registry, die ab Vista für den Standardbenutzer gesperrt wurden. Also z.B. die Daten direkt in *program files* anstatt diese in *CommonApplicationData* zu speichern.

An der Applikation sind also jene Bereiche anzupassen, die nur mit Admin Rechten zu erreichen sind. Man muss sich als Entwickler die Frage stellen, ob Files nicht an anderer Stelle gespeichert werden können.

Unterstützung mit Microsoft Tools

Mittels *Application Verifierer* erhält man eine XML Log Datei der issues.
Mit dem *standard user analyzer* (SUA) lassen sich die UAC relevanten Zugriffe der Applikation erkennen. Diese sind:

- File Access
- Registry Access
- Config Files (ini)
- Token issues (Schlüssel)
- Security privileges (Sicherheitseinstellungen)
- Name space issues (Namensraum-Probleme)

File Access

Irgendwie wild ins Dateiverzeichnis schreiben ist gar nicht gut. (Jeder Admin wird hellhörig).

Nicht erlaubt: root, windows, windows32, program files

Erlaubte Ablagemöglichkeiten (.net C# Code)

```
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
```

Name	Beschreibung
ApplicationData	Verzeichnis, das als Repository für anwendungsspezifische Daten des aktuellen roaming Users genutzt wird.
CommonApplicationData	Gibt den Pfad zum Ordner für Anwendungsdaten aller Benutzer zurück
LocalApplicationData	Verzeichnis, das als Repository für anwendungsspezifische Daten des aktuellen (nicht roaming) Users genutzt wird.
CommonProgramFiles	Gemeinsames Verzeichnis für Komponenten die in unterschiedlichen Anwendungen genutzt werden

Registry Access

Auf HKCR (classes root) und HKLM (local machine) Verzeichnisse in der Registry kann ab Windows Vista nur mit Admin Rechten zugegriffen werden. Wenn es Anforderung der Anwendung ist, dass eine Funktion in sensiblen Bereiche agiert, muss die Anwendung entweder mit Admin Rechten gestartet werden (runas Admin), oder im Manifest File ist definiert, dass Admin Rechte notwendig sind (Prompt erscheint).

In die Registry gehört mit .net Anwendungen ohnehin nichts mehr rein, dafür gibt es die config files (xml files).

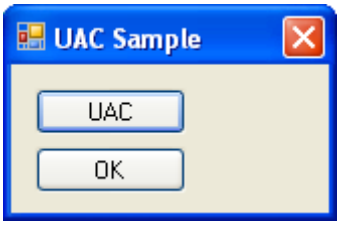
Code Samples

Angenommen meine Anwendung muss auf einen Bereich zugreifen, der Admin Rechte benötigt. Wie kann ich das bewerkstelligen? Innerhalb des mitgelieferten Manifest Files wird festgelegt, mit welchen Rechten die Applikation gestartet wird. Nun will ich das aber nicht immer, sondern nur bei einer bestimmten Funktion, die hinter einem Button liegt. Das geht so:

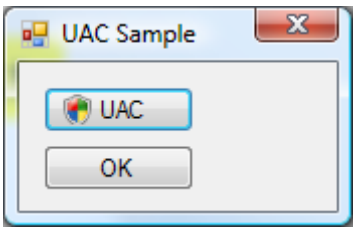
Wurde die Anwendung mit Admin Rechten gestartet?

```
public static bool IsAdmin()
{
    WindowsIdentity id = WindowsIdentity.GetCurrent();
    WindowsPrincipal p = new WindowsPrincipal(id);
    return p.IsInRole(WindowsBuiltInRole.Administrator);
}
```

Admin Logo auf Button



Unter Windows XP (kein UAC vorhanden) ausgeführt – es ist kein UAC Schild vorhanden



Unter Windows Vista / 7 ausgeführt – das UAC Schild ist erkennbar
Nach dem Drücken des UAC Buttons erscheint der Elevation Dialog.

Verwendung der Klasse:

```
UACshield.SetShieldOnButton(this.buttonUAC, true);
```

Klasse:

```
    public class UACshield
    {
        [DllImport("shell32.dll", EntryPoint = "ExtractIconExW", CallingConvention = CallingConvention.StdCall)]

        public static extern int ExtractIconEx([In][MarshalAsAttribute(UnmanagedType.LPWStr)]string lpszFile, int nIconIndex,
ref IntPtr phiconLarge, ref IntPtr phiconSmall, int nIcons);

        private static Image GetVistaShield(bool smallIcon)
        {
            int icoIdx = 6;
            Image img = null;
            Icon icon = null;
            IntPtr iconLargePtr = IntPtr.Zero;
            IntPtr iconSmallPtr = IntPtr.Zero;

            string path = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.System),"user32.dll");

            try
            {
                int result = ExtractIconEx(path, icoIdx, ref iconLargePtr, ref iconSmallPtr, 1);

                if (result < 1)
                    return null;

                if (smallIcon && iconSmallPtr != IntPtr.Zero)
                {
                    icon = Icon.FromHandle(iconSmallPtr);
                }
                else if (iconLargePtr != IntPtr.Zero)
                {
                    icon = Icon.FromHandle(iconLargePtr);
                }
                else
                    return null;

                img = (Image)icon.ToBitmap();
            }
            catch (Win32Exception)
            {
                return null;
            }
            finally
            {
                iconLargePtr = IntPtr.Zero;
                iconSmallPtr = IntPtr.Zero;
            }

            return img;
        }

        public static Image SmallVistaShield
        {
            get { return GetVistaShield(true); }
        }

        public static Image LargeVistaShield
        {
            get { return GetVistaShield(false); }
        }

        public static void SetShieldOnButton(Control ctrl, bool smallIcon)
        {
            Button btn = (Button)ctrl;

            if (smallIcon)
            {
                btn.Image = SmallVistaShield;
            }
            else
            {
                btn.Image = LargeVistaShield;
            }

            btn.TextImageRelation = TextImageRelation.ImageBeforeText;
        }
    }
```

Elevation PopUp

Eine Anwendung meldet sich dann (Elevation PopUp), wenn diese beim ersten Start höhere Privilegien benötigt oder wenn die Anwendung explizit mit *runas administrator* gestartet wird. Die Anwendung meldet sich einmalig um diese höheren Rechte zu erhalten. Im Hintergrund wird der alte Prozess (der mit den unzureichenden Rechten) beendet und mit den gewünschten Rechten neu zu starten. Die einzige Ausnahme des Elevation PopUps ist dann, wenn eine Anwendung von einem bestehenden Prozess gestartet wird. Hier erhält der zweite Prozess die Rechte des ersten vererbt und

es erfolgt dabei kein Elevation PopUp. Per Default hat der Elevation Dialog eine orange Farbe, wenn der Code signiert ist erhält er eine blaue Farbe.

Prozess

Beim Drücken des Knopfes soll ein neuer Prozess gestartet werden:

```
Process p =  new Process();
p.StartInfo.UseShellExecute = true;
p.StartInfo.Verb = "runas";
p.StartInfo.FileName = @"c:\Programme\Paint.NET\PaintDotNet.exe";
p.Start();
```

Visual Studio Einstellungen

Manifest File

Das Manifest file einer .net Anwendung legt den UAC access level beim Starten fest.
Es ist ein XML ressource file und kann embedded sein. Die Privilegien werden im Abschnitt *requestedExecutionLevel* festgelegt.

Name	Beschreibung
asInvoker	Benötigt keine „elevation" (es kommt kein Dialog), hat die Rechte des Mutter Prozesses.
highestAvailable	Dialog für Standard Benutzer das Admin Rechte zum benötigt werden. Verlangt nach den höchst möglichen Rechten des des Mutter Prozesses. Von einem Administrator wird der volle Admin Level verlangt.
requireAdministrator	Dialog für Standard Benutzer das volle Admin Rechte notwendig sind.

Bsp. Manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1" xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
xmlns:asmv2="urn:schemas-microsoft-com:asm.v2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- UAC Manifest Options
            If you want to change the Windows User Account Control level replace the
            requestedExecutionLevel node with one of the following.

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

            If you want to utilize File and Registry Virtualization for backward
            compatibility then delete the requestedExecutionLevel node.
        -->
        <requestedExecutionLevel level="asInvoker" uiAccess="false" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</asmv1:assembly>
```

Für die meisten Anwendungen reicht der Level *asInvoker*, der nicht jedes Mal den Elevation Dialog bringt wenn die Anwendung gestartet wird und die Standard Privilegien verwendet für den Zugriff auf die Ressourcen. Die anderen Applikationen, die diesen Zugriff beabsichtigen, benötigen einen höheren Zugriffslevel konfiguriert.

uiAccess: legt fest, dass die Anwendung Zugriff auf das User Interface (UI) benötigt, Bsp eine System Dialog Box. Funktioniert nur mit signierten Assemblies. Der Default Wert von *uiAccess* ist false.

Manifest Tool (mt.exe)

Ist ein Command Line Tool zum Erstellen von signierten Files, als Übergabeparameter gibt man das Manifest Files und die Assembly an. Das Manifest Tool erzeugt Hashes (mittels der Crypto API einen SHA-1)

Bsp. Aufruf:

mt.exe -manifest "MyApp.exe.manifest" -outputresource:" MyApp.exe"

mt.exe -manifest " MyApp.dll.manifest" -outputresource:" MyApp.dll"

Deaktivieren von UAC

UAC ist per default in Windows Vista und Windows 7 aktiviert. Deaktivieren lässt es sich weiterhin, sinnvoll ist es nicht. Sinnvoll ist es, die Anwendungen anzupassen.

Windows Vista

http://www.administrator.de/Windows_Vista_-_UAC_deaktivieren.html

Windows 7

http://www.tobbis-blog.de/microsoft/anleitungen/2009-02-11-uac-unter-windows-7-deaktivieren/