Алексей Федоров

Обеспечение совместимости приложений в Microsoft® Windows Vista® и Windows® 7

Руководство для разработчиков

Алексей Федоров

Обеспечение совместимости приложений в Microsoft Windows Vista и Windows 7

Руководство для разработчиков

Алексей Федоров — руководитель партнерской группы
Департамента стратегических технологий
российского представительства компании Microsoft.
(alexeif@microsoft.com)

© Федоров А. Г., 2009

Содержание

Введение ... 5
Общие рекомендации по созданию приложений, корректно работающих в Windows ... 6
Установка приложения ... 6
Корректное выполнение приложений ... 7
Корректное использование ресурсов операционной системы ... 8
Корректная работа с сервисами и другими компонентами операционной системы ... 10
Файлы, защищенные Windows File Protection ... 10
Использование драйверов ... 11
Новые функции в Windows Vista и подготовка к сертификации ... 12
Безопасность и совместимость ... 13
Установка ... 16
Надежность работы ... 20
Практические вопросы ... 22
Проверка версии операционной системы ... 23
Работа в привилегированном режиме ... 25
Запись в Program Files ... 26
Виртуализация ... 28
Использование манифеста ... 36
Использование токена Admin ... 39
Mandatory Integrity Control ... 41
Установка приложений ... 43
Изоляция Session 0 ... 44
Windows Resource Protection ... 47
Утилита System File Check ... 50
Создание объектов в области Global ... 52
Механизмы обеспечения надежности приложений ... 54
Windows Feedback Platform ... 54
Restart Manager ... 64
Дополнительные механизмы обеспечения надежности ... 71
Windows 7: рекомендации по обеспечению совместимости приложений ... 72
Версия операционной системы ... 72
Новая версия Internet Explorer ... 74
Отсутствие компонента Windows Mail ... 75
Использование библиотек (Library) ... 77
Отсутствие сервиса Windows Registry Reflection ... 77
Отсутствие драйвера WPDUSB.SYS для Windows Portable Devices ... 78
Новые бинарные компоненты операционной системы ... 78

Работа Internet Explorer в режиме DEP ... 80
Изменения в MSMQ ... 81
Технология Switchback ... 81
Полезные утилиты ... 82
Платформа Windows Troubleshooting ... 82
Компонент Reliability Analysis ... 83
Windows Error Reporting Problem Steps Recorder ... 84
Подготовка к сертификации приложений ... 84
Обзор программы сертификации ... 85
Шаги по прохождению сертификации ... 86
Требования к приложениям ... 86
Материалы по сертификации ... 88
Приложение. Технологии защиты приложений в Windows Vista, Windows 7 и Windows Server 2008/R2 ... 89
Проверка переполнения стека ... 89
Защита при обработке исключений ... 90
Поддержка No Execute (NX), Data Execution Prevention (DEP) и Execute Disable (XD) ... 90
Случайное распределение адресного пространства — Address Space Layout Randomization (ASLR) ... 95
Случайное распределение «кучи» ... 95
Случайное распределение стека ... 95
Определение повреждения «кучи» ... 95
Заключение ... 96

Введение

Совместимость существующих приложений с операционной системой Microsoft Windows Vista (и выходящей в этом году операционной системой Windows 7, построенной на ядре Windows Vista) является одной из основных проблем, с которой могут столкнуться пользователи, переходящие на новую версию операционной системы. Несмотря на усилия, прилагаемые компанией Microsoft, некоторые производители программного обеспечения продолжают использовать устаревшие функции операционной системы, некорректно выполняют операции по проверке версий ОС, не следуют рекомендациям по работе с файловой системой и зачастую не руководствуются советами по обеспечению корректной работы приложений в новых версиях системы. Все это приводит к тому, что в операционной системе Microsoft Windows Vista есть более 5600 «системных заплаток» (shims) для обеспечения корректной работы приложений различных производителей — от утилит китайских производителей до крупных продуктов известных фирм. В Windows 7 число «системных заплаток» увеличилось — в бета-версии новой операционной системы их насчитывается более 5700!

Можно выделить три основных подхода к обеспечению совместимости приложений: использование упомянутых выше «системных заплаток», запуск приложения в виртуальной среде (терминальные сервисы или использование Microsoft Application Virtualization) и, изменение кода приложения таким образом, чтобы оно соответствовало требованиям по корректной работе в операционной системе, — для этого служат руководства по сертификации приложений для получения логотипов Works With Windows Vista и Certified for Windows Vista, а также соответствующие тесткейсы, которые можно найти на сайте **http://www.innovateon.com** в разделе, посвященном Windows Vista.

В данном обзоре мы рассмотрим ключевые подходы к созданию приложений, совместимых с Windows Vista и Windows 7, а также приведем рекомендации по подготовке приложений к сертификации для получения статуса Works with Windows Vista, Certified for Windows Vista и Compatible with Windows 7.

Общие рекомендации по созданию приложений, корректно работающих в Windows

Написание корректных приложений, которые работают под управлением текущего поколения операционных систем компании Microsoft — Windows XP и Windows Server 2003, а также будут продолжать корректно работать под Microsoft Windows Vista и Windows 7, не является тривиальной задачей и требует отдельной дисциплины. Приводимые ниже рекомендации помогут вам не только повысить качество создаваемых вами приложений, но и предоставят шанс успешно пройти тесты — будь то тесты группы Platform Test для получения партнерской компетенции или тесты для получения сертификации “Certified For” для ваших продуктов.

Корректно написанное приложение должно удовлетворять следующим требованиям.

- Устанавливаться с помощью стандартных средств и не требовать перезагрузки после установки. Программа установки должна поддерживать режим «**All Users**».
- Выполняться под управлением операционной системы без сбоев. Приложение должно работать в режиме Fast User Switching, а также поддерживать режим **Limited User**.
- Корректно использовать ресурсы операционной системы.
- Приложение не должно влиять на работу других сервисов — ни при установке, ни в режиме эксплуатации, ни при его удалении.
- Приложение не должно пытаться изменить файлы, находящиеся под защитой **Windows File Protection**.
- Использовать только подписанные электронной подписью драйверы. Драйверы, выполняющиеся на уровне **kernel**, должный пройти специальный тест — **Windows Driver Verification**.

Ниже мы рассмотрим эти требования более подробно, объясним их назначение, приведем рекомендации по их соблюдению и перечислим дополнительные ресурсы, доступные на сайте Microsoft.

Установка приложения

При установке под Windows XP и последующих версиях операционной системы Microsoft только в ряде исключительных случаев требуется перезагрузка компьютера. Приложения не должны предполагать перезагрузку во время или после установки. В качестве исключения допускается перезагрузка после установки пакета обновления Windows (Windows Service Pack), вызванного при установке приложения. Не требуется перезагрузка

после установки таких расширений операционной системы, как Graphical Identification and Authentication (**GINA.DLL**) или драйверов-фильтров.

Для поддержки возможности использования приложения более чем одним пользователем (на одном компьютере), приложение должно поддерживать режим «**All Users**» либо по умолчанию, либо в качестве одной из опций — в случае, когда приложение по умолчанию устанавливается для текущего пользователя.

Частью процесса установки может быть проверка версии операционной системы, на которую устанавливается приложение и, при необходимости, запрос на установку (автоматическая установка) требуемого пакета обновления Windows. Для проверки версии операционной системы рекомендуется использование функции Windows API **GetVersionEx()**, либо **VerifyVersionInfo()** для получения более подробной информации о компонентах операционной системы — данная функция доступна в версиях Windows, начиная с Windows 2000. Для приложений, написанных на управляемом коде, рекомендуется использовать класс **System.OperatingSystem**. Более подробно об этом классе см. **http://msdn2.microsoft.com/en-us/library/system.operatingsystem.aspx**.

По умолчанию приложение должно устанавливаться в папку Program Files. Для определения местоположения данной папки следует использовать либо функцию **SHGetFolderPath()** Windows API, либо свойство **ProgramFilesFolder** Windows Installer, либо, в случае приложений на управляемом коде, рекомендуется использовать метод **GetFolderPath()** класса **System.Environment**.

Корректное выполнение приложений

Под корректной работой приложения понимается выполнение им основных функций без сбоев, приводящих к нарушению стабильности работы самого приложения, операционной системы или к потере данных. В случае возникновения каких-либо проблем в работе приложения, оно должно отобразить сообщение, содержимое которого должно быть понятно конечному пользователю и позволить ему либо продолжить работу, либо завершить использование приложения.

Типичными проблемами, приводящими к нарушению функционирования приложений, могут быть: попытка обращения к несуществующим файловым ресурсам (приложение не должно предполагать установку Windows только на диске **C:**, должны использоваться только системные функции для обращения к файловым ресурсам), обращение к файлам или принтерам с длинными именами (при тестировании приложения необходимо проверить его функционирование с различными комбинациями имен файлов и принтеров), обращение к устройствам, для которых не установлен драйвер (в этом случае приложение должно корректно завер-

шить работу и затребовать установку необходимого драйвера), попытка работы в режиме 256 цветов (приложение должно автоматически переключаться в этот режим) и ряд других, включая, например, корректную работу на мультипроцессорных компьютерах.

Приложение должно использовать системные функции для определения местоположения «стандартных» папок (типа **My Documents**). Для этих целей используется функция **SHGetFolderPath()**. Для приложений на управляемом коде рекомендуется использовать метод **GetFolderPath()** класса **System.Environment**.

Поддержка режима **Limited User** включает корректную работу приложения без наличия привилегий администратора со стороны пользователя. Режим **Limited User** позволяет пользователям сохранять данные только в разделах реестра **HKEY_CURRENT_USER**, пользовательских каталогах (параметр **CSIDL_PROFILE** для перечисленных в предыдущем параграфе функций) или в папке, созданной пользователем в корневом каталоге системного диска. Остальные файловые ресурсы и ветви реестра доступны пользователям в режиме «только чтение».

Режим **Fast User Switching** доступен в Windows XP и последующих версиях операционных систем семейства Windows и позволяет быстро переключать компьютер на другого пользователя без остановки работы приложений. Данный режим включается из контрольной панели (**Control Panel | User Accounts | Change the way users log on or off**). Приложение, поддерживающее режим работы **Fast User Switching**, не должно терять данные или завершаться с ошибкой при переключении пользователей.

Для определения режима работы приложений (есть ли еще запущенные копии приложения) следует использовать функции Windows API **FindWindow()** или **FindWindowEx()** — они позволяют найти главное окно приложения. Другим способом является использование семафоров или мьютексов (**mutex**) — они создаются при запуске приложения и закрываются при завершении приложения. Так как приложения, запущенные от лица различных пользователей, используют различные области хранения локальных объектов, рекомендуется использовать глобальные семафоры или мьютексы. Для того чтобы пользователь в другой сессии не смог запустить приложение, уже запущенное в какой-то другой сессии, семафор или мьютекс должен иметь имя с префиксом ***Global***. Это позволяет обнаруживать экземпляры приложений, запущенных в других пользовательских сессиях.

Корректное использование ресурсов операционной системы

Операционная система предоставляет в распоряжение приложений определенные ресурсы, которые должны корректно использоваться всеми приложениями. К таким ресурсам, в частности, относятся: глобальная «куча»

(**heap**), критические секции (**critical section**), ссылки (хэндлеры, **handles**). Требования по корректному использованию таких системных ресурсов, в первую очередь относятся к приложениям, создаваемым на неуправляемом коде — в случае использования .NET задачу корректного использования ресурсов берет на себя **Common Language Runtime** и, в частности, ее компонент **Garbage Collector**. Тем не менее, рекомендации по использованию системных ресурсов будут полезны как тем, кто создает приложения на неуправляемом коде, так и тем, кто использует .NET.

- **Использование глобальной «кучи»**. Глобальная «куча» используется для динамического выделения памяти приложениям. Неверное использование «кучи» может приводить к ошибкам безопасности и сбоям в работе приложений. Неверные способы использования глобальной «кучи» могут включать:
  - выделение слишком малого объема памяти и запись за пределами региона памяти (**buffer overrun**);
  - попытку использования памяти после ее освобождения;
  - повторное освобождение уже освобожденной памяти;
  - попытку освобождения невыделенной памяти;
  - использование неверных указателей (хэндлеров) на регионы памяти.
- **Использование критических секций**. Критические секции — это механизм синхронизации, гарантирующий эксклюзивный доступ к данным приложения в многопоточных средах. Неверные способы использования критических секций могут включать:
  - попытку освобождения критической секции, не принадлежащей текущему потоку;
  - завершение потока, которому еще принадлежат неосвобожденные критические секции;
  - попытку использования критической секции до ее инициализации;
  - «утечку» критических секции — например, не использование функции **DeleteCriticalSection()** по завершении использования критической секции;
  - попытку повторной инициализации секции.
- **Использование хэндлеров**. Хэндлеры — это «указатели» на запрошенные с помощью вызова соответствующих функций Windows API ресурсы типа файлов, событий, памяти и т. п. Неверные способы использования хэндлеров могут включать:
  - попытку использования хэндлера после его закрытия;
  - использование неверного типа хэндлера;
  - использование «нулевого» или псевдо-хэнджера, например значения, возвращаемого функцией **GetCurrentProcess()**;

Обычно большинство приведенных выше ошибок блокируется на уровне компилятора или системной библиотеки (при использовании соответствующих опций) или практически недостижимо при использовании .NET, но тем не менее, понимание и правильное использование ресурсов операционной системы никогда не перестанет быть актуальной задачей.

Корректная работа с сервисами и другими компонентами операционной системы

Работа приложения или программы его установки не должна влиять на работу других сервисов и компонентов операционной системы. Под сервисами понимаются компоненты операционной системы, управляемые **Service Control Manager** (**SCM**). В целом, приложение имеет право приостанавливать работу или перезапускать только те сервисы, которые являются частью данного приложения. Сервисы, которые являются частью операционной системы или принадлежат другим приложениям, не должны затрагиваться (перезапускаться, приостанавливаться) текущим приложением. Возможны два способа хостинга сервисов — внутри процесса (например, использование **W3svc** внутри worker-процесса) или вне процесса (SQL Server, Exchange и т. п.). Сервис считается недоступным, если он: остановлен средствами **SCM**, не реагирует на три последовательных клиентских запроса или обращение к сервису вызывает системную ошибку.

Файлы, защищенные Windows File Protection

Приложения должны устанавливаться без попытки изменить файлы, защищенные **Windows File Protection (WFP)**. Для того чтобы избежать конфликтов с WFP, при установке любых файлов, не создаваемых самим приложением, следует вызывать системную функцию **SfcIsFileProtected()** для того, чтобы убедиться, что файл не защищен WFP, — программа Windows Installer делает это автоматически. Файлы, защищенные WFP — это компоненты операционной системы, находящиеся на установочном диске:

- большинство файлов с расширениями .SYS, .DLL, .EXE и .OCX;
- шрифты Micross.ttf, Tahoma.ttf, Tahomabd.ttf, Dosapp.fon, Fixedsys.fon, Modern.fon, Script.fon и Vgaoem.fon;
- также следует отметить, что в состав операционной системы могут входить версии библиотеки **Microsoft Foundation Classes** (**MFC**) — набор DLL, защищенных WFP и необходимых для функционирования ряда компонентов операционной системы.

Механизм защиты файлов Windows (**Windows File Protection**, **WFP**) служит для предотвращения перезаписи программами важных файлов операционной системы, которые используются как самой операционной системой, так и другими программами. Защита таких файлов необходима

для предупреждения возможных неполадок в работе операционной системы и установленного программного обеспечения. Замена защищенных файлов операционной системы возможна только в следующих случаях.

- Установка пакетов обновления для Windows с помощью программы **Update.exe**.
- Установка исправлений с помощью программ **Hotfix.exe** и **Update.exe**
- Обновление операционной системы с помощью программы **Winnt32.exe**.
- Использование службы **Windows Update**.

Защита файлов операционной системы обеспечивается двумя способами. Во-первых, механизм WFP работает в фоновом режиме и активируется в случае получения уведомления об изменении файла в защищенной папке. После получения уведомления WFP идентифицирует измененный файл. Если был изменен защищенный файл, по подписи из файла каталога проверяется новая версия. Если версия является неправильной, новый файл заменяется исходным из папки кэша или источника установки. Поиск файла допустимой версии производится в следующем порядке.

1. Папка кэша (по умолчанию **%systemroot%\system32\dllcache**).
2. Путь к сетевому источнику установки, если он был использован для установки операционной системы.
3. Компакт-диск Windows, если он был использован для установки операционной системы.

Если файл удается найти в папке кэша или автоматически на источнике установки, он восстанавливается без уведомления пользователя. В противном случае появляется одно из сообщений, связанных с нарушением защиты файлов Windows.

Использование драйверов

Некорректно написанные драйверы, особенно те, которые выполняются в режиме kernel, могут повлиять на стабильность работы как использующего их приложения, так и самой операционной системы. Поэтому драйверы, поставляемые в составе ваших приложений, должны пройти специальную проверку. Для этого используется специальная утилита — Windows Driver Verifier Manager (**Verifier.exe**), которую можно найти в каталоге **\system32** на диске, где установлена операционная система.

Помимо этого, драйверы должны пройти специальное тестирование с помощью утилиты Windows Hardware Compatibility Test (HCT) версии 10.0 или более поздней. Дополнительная информация доступна по адресу **http://www.microsoft.com/hwtest**.

Упомянутый выше ресурс (**WHDC**) содержит информацию по изменениям и расширениям в ядре операционной системы Windows Vista. Эта

информация может быть крайне полезной как компаниям, занимающимся разработкой драйверов для различных устройств, так и тем, кто интересуется вопросами функционирования операционных систем. На момент написания данной статьи были доступны следующие документы:

- Architecture of the Windows Driver Foundation
- Kernel Enhancements for Windows Vista and Windows Server Longhorn
- Direct Application Launch from System Startup on Windows Vista
- _OSC Method and PCI Express in Windows Vista
- Impact of Session 0 Isolation on Services and Drivers in Windows Vista
- Device Finish-Install Actions in Windows Vista

Также следует обратить внимание на новую библиотеку для создания драйверов: для режима kernel — Kernel-Mode Driver Framework, для режима user: User-Mode Driver Framework, а также утилиты для установки драйверов — Driver Install Frameworks Tools.

Новые функции в Windows Vista и подготовка к сертификации

Выше мы обсудили некоторые рекомендации по написанию корректных приложений, которые работают под управлением текущего поколения операционных систем компании Microsoft — Windows XP и Windows Server 2003, а также будут продолжать корректно работать под управлением Microsoft Windows Vista и Microsoft Windows 7.

Здесь мы продолжим эту тему и остановимся на основных требованиях, выдвигаемых к приложениям, работающим под управлением новой клиентской операционной системы Microsoft Windows Vista, и претендующим на получение логотипа Windows Vista Logo — как Works with Windows Vista, так и Certified for Windows Vista. Соблюдение этих требований поможет улучшению качества приложений, работающих на платформе Windows, обеспечению их совместимости, надежности и безопасности, а также позволит компаниям-разработчикам программного обеспечения при необходимости успешно пройти тесты на получение логотипа Windows Vista Logo.

Требования, выдвигаемые к приложениям, претендующим на получение логотипа Windows Vista Logo, а также ко всем приложениям, которые должны работать под управлением операционной системы Microsoft Windows Vista, можно разделить на три категории — безопасность и совместимость, установка и надежность работы.

Ниже мы более подробно рассмотрим требования, включенные в каждую из этих категорий. Как уже отмечалось выше, настоятельно рекомен-

дуется изучить каждое из этих требований и, по возможности, включить их в соответствующие разделы проектной документации к разрабатываемому приложению — таким образом они будут учтены и при написании кода приложения, и при его тестировании. Учет этих требований позволит вам в дальнейшем гарантировать, что ваши приложения надежны, стабильны, совместимы с операционной системой и предоставляют пользователям именно тот базовый набор сервисов, которые они ожидают от программных продуктов, рассчитанных на работу под управлением новых версий клиентских операционных систем Windows Vista и Windows 7.

Безопасность и совместимость

В этой категории собраны требования, удовлетворение которых позволит обеспечить работоспособность приложений на Windows Vista на различных архитектурах, под различными конфигурациями, в различных режимах. Помимо этого, приложения, отвечающие данным требованиям, продолжают корректно работать после обновлений операционной системы (установка пакетов обновлений, выходящих после выпуска основной версии операционной системы) и гарантируется улучшение общей безопасности приложений.

Давайте сначала кратко перечислим требования из категории «Безопасность и совместимость», а затем рассмотрим каждое из них более подробно. К требованиям, включенным в данную категорию, относятся:

- следование рекомендациям User Account Protection;
- поддержка 64-битных версий Windows Vista/Windows 7;
- наличие подписанных файлов;
- наличие подписанных драйверов;
- использование только документированных интерфейсов;
- корректная проверка версий операционной системы;
- поддержка переключения между задачами;
- поддержка работы в режиме нескольких пользователей;
- поддержка длинных имен файлов и устройств;
- корректная работа в режиме «Safe Mode»;
- следование политикам борьбы с вредоносным ПО.

Следование рекомендациям User Account Protection

Как известно, наиболее безопасный режим выполнения приложений — использование того уровня привилегий, который реально требуется для обеспечения их полноценной функциональности. За исключением тех случаев, когда приложение предназначено для системных администраторов, оно должно выполняться с минимальными привилегиями.

Каждое приложение должно содержать встроенный манифест, в котором задается уровень выполнения приложения:

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

Для большинства приложений уровень должен быть установлен как **asInvoker**. Также подавляющее большинство приложений не должно управлять интерфейсом других приложений, поэтому атрибут **uiAccess** должен иметь значение **false**.

Поддержка 64-битных версий Windows Vista/Windows 7

Приложения, предполагаемые к использованию под управлением операционной системы Windows Vista и Windows 7, должны поддерживать 64-битную версию этой операционной системы. Это означает, что приложения и их программы установки не должны содержать 16-битного кода или зависеть от 16-битных компонентов. Если приложение использует какие-либо драйверы, должны поставляться 64-битные версии соответствующих драйверов — программа установки приложения должна определять корректный набор компонентов для 64-битной версии операционной системы.

Наличие подписанных файлов

Процедура подписания позволяет пользователям убедиться в том, что файлы находятся в целостности, а приложениям — работать в средах типа locked-down. Все файлы с расширениями .EXE и .DLL должны быть подписаны сертификатом Authenticode. Доступны сертификаты от различных поставщиков — см. список на сайте по адресу **http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/rootcertprog.asp**. После получения сертификата файлы подписываются утилитой **Sign Tool**, которая входит в состав Windows SDK (http://windowssdk.msdn.microsoft.com/library/).

Наличие подписанных драйверов

Так как некорректно написанные драйверы могут серьезно повлиять на стабильность и безопасность системы, все драйверы уровня kernel-, требуемые для выполнения того или иного приложения, должны быть подписаны сертификатом Microsoft, который доступен через программы WHQL или DRS. Дополнительная информация о подписании драйверов доступна на сайте **http://www.microsoft.com/whdc/winlogo/hwrequirements.mspx**.

Использование только документированных интерфейсов

Неподдерживаемые и недокументированные программные интерфейсы с высокой долей вероятности могут измениться или быть исключены в сле-

дующих версиях операционной системы. Поэтому приложения не должны использовать устаревшие программные интерфейсы и недокументированные внутренние функции операционной системы. Помимо этого не должны использоваться интерфейсы, помеченные в Visual Studio 2005 как **Obsolete**.

Корректная проверка версий операционной системы

Приложения должны корректно работать при изменении номеров версий операционной системы. Исключение составляют те случаи, когда в лицензионном соглашении в явном виде сказано, что приложение не поддерживает будущие версии операционных систем. Для проверки версии операционной системы следует использовать только рекомендованные документированные функции Windows API, проверка версии операционной системы через записи реестра не допускается.

Поддержка переключения между задачами

Приложения должны корректно отрабатывать переключение между задачами и не блокировать функциональность **ALT+TAB** или других сценариев переключения между задачами.

Поддержка работы в режиме нескольких пользователей

Приложения должны корректно работать в режиме нескольких пользователей — работать в параллельных сессиях без конфликтов и сбоев. К основным требованиям относятся: воспроизведение приложением звука только в данной сессии, отсутствие зависимости от глобально названных объектов, а также отказ от эксклюзивного доступа к системным и пользовательским ресурсам без крайней необходимости (например, не использовать блокировки чтения/записи для операций «только чтение).

Поддержка длинных имен файлов и устройств

Приложения должны корректно работать с длинными именами файлов, папок, пользователей, устройств и ключей реестра. Приложения должны поддерживать максимальную длину имен, определенную в **Windows Platform SDK**.

Корректная работа в режиме «Safe Mode»

Режим **Safe Mode** позволяет пользователям диагностировать и корректировать конфигурации Windows. Поддержка работы в этом режиме гарантирует, что пользователям будет доступна вся функциональность приложения без зависаний и сбоев. Любые некритичные сервисы по умолчанию не должны запускаться в режиме **Safe Mode**. К некритичным сервисам относится сервисы, от которых не зависит широкий класс приложе-

ний и сервисов — например, сервис **RemoteRegistry**. Приложения, которым требуются сервисы, недоступные в режиме **Safe Mode**, должны элегантно завершать свою работу с соответствующим сообщением и записью необходимой информации в системный протокол.

Следование политикам борьбы с вредоносным ПО

Приложения должны следовать рекомендациям **Anti-Spyware Coalition** (**http://www.antispywarecoalition.org/documents/index.htm**). Приложения, претендующие на получение логотипа, проходят дополнительную проверку в Microsoft и заносятся в специальный каталог, содержащий список всех корректных приложений.

Установка

В этой категории собраны требования, удовлетворение которых позволит пользователям быть уверенными, что приложения установятся под Windows без необходимости в изменениях настроек операционной системы и конфликтов с другими приложениями. Эти требования таковы.

- Для установки используйте Windows Components for Installation.
- Установка не должна требовать прав Local Administrator.
- Установка в корректные папки.
- Подписание манифестов ClickOnce.
- Изоляция установки ClickOnce.
- Выполнение «чистой» деинсталяции.
- Корректная конфигурация идентификаторов инсталяционных пакетов.
- Поддержка обслуживания приложений.
- Корректная установка ресурсов Windows.
- Следование рекомендациям по созданию расширенных событий в инсталляторах.
- Поддержка User Account Protection при установке.
- Исключение перезагрузки в процессе установки.
- Поддержка пакетной установки.
- Следование правилам создания компонентов.

Рассмотрим требования по установке приложений более подробно.

Для установки используйте Windows Components for Installation

Для реализации программ установки программного обеспечения рекомендуется использовать соответствующие компоненты операционной системы — **Windows Components for Installation**. С их помощью легко создавать удобные, транзакционные инсталляторы с возможностью отмены произве-

денных операций. Приложения должны пользоваться сервисами **Windows Installer (MSI)** или **ClickOnce**. Созданные инсталляционные пакеты не должны получать ошибок от **Internal Consistence Evaluators (ICE)**.

Установка не должна требовать прав Local Administrator

Программа установки программного обеспечения не должна подразумевать что пользователь обладает правами локального администратора. Более того, программа установки должна работать так, чтобы она могла быть завершена другим по отношению к запустившему ее пользователем. Если процесс установки запущен пользователем с правами **Standard User** или **Protected Administrator**, процесс установки должен корректно обработать переход на полные администраторские права. Все действия, происходившие до этого перехода, должны выполняться с правами **Standard User**.

Установка в корректные папки

Пользователи должны иметь возможность гибкого выбора места установки программного обеспечения и выбора места хранения файлов по умолчанию. По умолчанию приложения должны устанавливаться в папку **Program Files** или в пользовательскую папку **AppData**. Данные должны записываться при первом запуске приложения, а не при его установке — это связано с тем, что при установке возможно переключение пользователей и нет возможности корректно определить местоположение пользовательских данных.

Подписание манифестов ClickOnce

Так как приложения, устанавливаемые сервисами **ClickOnce**, могут разворачиваться из внешних по отношению к локальному компьютеру источников, пользователи должны иметь возможность убедиться в том, что такие приложения работают корректно. Для этого необходимо, чтобы все приложения были подписаны сертификатом **Authenticode**.

Изоляция установки ClickOnce

Приложения, устанавливаемые сервисами **ClickOnce**, должны минимально воздействовать на систему. Такие приложения должны хранить данные только в пользовательских папках и, по возможности, не изменять ключи реестра.

Выполнение «чистой» деинсталляции

«Чистая» деинсталляция приложений существенно облегчает поддержку операционной системы и в ряде случаев может улучшить производительность всей системы. Приложения должны корректно и полностью удаляться с компьютера. Процесс деинсталляции включает удаление файлов, ключей реестра, сборок из GAC, таблиц баз данных, настроек метабазы, записей в Active Directory и т. п.

Корректная конфигурация идентификаторов инсталляционных пакетов

Корректная идентификация инсталляционных пакетов помогает пользователям в более простом управлении приложениями при их обслуживании, обновлении или обновлении всей операционной системы. Помимо этого, идентификация инсталляционных пакетов существенно облегчает процесс удаления приложений средствами **Software Explorer** (в предыдущих версиях операционной системы — **Add/Remove Programs**). Пакет должен содержать заполненные соответствующими данными свойства **ProductName**, **Manufacturer** и **ProductVersion**.

Поддержка обслуживания приложений

Для поддержки обновлений приложений пакеты **Windows Installer** должны включать сертификат обновления в таблице **MsiPathCertificate**. Если обновления подписаны таким сертификатом, приложения могут обновляться автоматически, без привлечения администратора.

Корректная установка ресурсов Windows

Механизмы **Windows Resource Protection (WRP)** призваны гарантировать, что защищенные системные ресурсы обновляются только одобренными Microsoft средствами и соответствующими механизмами. Приложения не должны устанавливать файлы, защищенные WRP. Если приложению требуется новая версия системного компонента, эти компоненты должны устанавливаться на компьютер средствами **Microsoft Service Pack** или через предоставляемый Microsoft инсталляционный пакет, содержащий требуемый программный компонент.

Следование рекомендациям по созданию расширенных событий в инсталляторах

Расширенные события в пакетах инсталляции предоставляют разработчикам широкие возможности по расширению функциональности программ установки. При использовании механизмов расширений следует придерживаться следующих правил.

- Все расширения должны быть документированы.
- Разработчики не должны добавлять новые колонки в стандартные таблицы — формат таблиц может измениться в новых версиях Windows Installer.
- Из расширений не должна вызываться утилита **gacutil** — эта утилита не рассчитана на использование внутри инсталляционных пакетов.
- Успех или ошибка при выполнении расширенного события должны записываться в протокол с использованием **MsiProcessessage**, а не стандартных кодов завершения операций.

- Новые колонки или свойства не должны иметь префикс msi.
- Расширенные события, изменяющие состояние системы должны восстанавливать его по завершении работы инсталляционного пакета.
- Расширенные события, изменяющие состояние системы, должны быть парой **deferred/rollback**.

Поддержка User Account Protection при установке

В операционной системе Windows Vista механизмы **User Account Protection** по умолчанию включены. Программы установки программного обеспечения должны соответствующим образом управлять требованиями к уровням привилегий. Приложения должны содержать манифест, в котором указан требуемый уровень привилегий для их установки. Если приложение запускается сразу же после установки, оно должно корректно выполняться с тем уровнем привилегий, который использовался для запуска программы установки.

Исключение перезагрузки в процессе установки

Для практически всех установок программного обеспечения не должна использоваться опция **ForceReboot Windows Installer**. В диалоговых панелях **Files In Use** должна быть опция «**Automatically Close Applications and Attempt to Restart After Setup is Complete**». Эта дополнительная функциональность позволит пользователям избежать перезагрузок операционной системы и по возможности задействовать инфраструктуру Restart Manager, позволяющую перезапускать приложения, блокирующие ресурсы, а не всю ОС.

Поддержка пакетной установки

Наличие обязательного пользовательского интерфейса может помешать реализации некоторых сценариев установки. Все приложения должны поддерживать пакетную установку из командной строки. Приложения должны устанавливаться в режиме **quiet mode** (опция командной строки **/qn**). Также приложения должны устанавливаться из командной строки с опцией **FASTOEM=1**.

Следование правилам создания компонентов

Для того, чтобы установка или удаление одного приложения не влияли на другие приложения, а также чтобы гарантировать, что сервисы **Windows Installer** корректно удалят все ресурсы, связанные с приложением, необходимо выполнять следующие правила.

- Поля **ComponentId** в таблице **Components** должны быть ненулевыми — в противном случае компоненты не будут удалены средствами **Windows Installer**.

- Не используйте более одного COM-сервера на компонент — если компонент содержит COM-сервер, он должен выполнять роль **KeyPath** для компонента.
- Не указывайте более одного файла в качестве элемента меню Start или ссылки на рабочем столе на компонент.

Надежность работы

При условии следования этим требованиям приложения становятся более надежными, снижается число сбоев, зависаний, перезагрузок, приложения становятся более «предсказуемыми» и управляемыми. К требованиям, входящим в категорию «Надежность работы», относятся:

- устранение лишних перезагрузок;
- перезагрузка после принудительного завершения или сбоев;
- устранение сбоев приложений;
- следование рекомендациям по обработке исключений;
- стабильная работа приложений.

Рассмотрим эти требования более подробно.

Устранение лишних перезагрузок

Не секрет, что необходимость в лишней перезагрузке компьютера вызывает у пользователей раздражение, приводит к снижению продуктивности и потере времени. Приложения, выполняющиеся под управлением Windows Vista и Windows 7, могут использовать функциональность компонента операционной системы **Restart Manager** для того, чтобы зарегистрировать необходимость перезагрузки и «слушать» соответствующие сообщения ядра операционной системы (**WM_QUERYENDSESSION** и **WM_ENDSESSION**) — более подробно об этом механизме см. ниже. Для установки приложений должна использоваться утилита **Windows Installer** (см. выше), а для доступа к **Restart Manager API** — механизм расширенных событий Windows Installer.

Перезагрузка после принудительного завершения или сбоев

Возможность повторного запуска приложений после их принудительного завершения или сбоя — отличный способ улучшить пользовательские ощущения, связанные с тем или иным программным продуктом. За счет сохранения данных и состояния приложение может свести к минимуму потери, связанные со сбоями. Все приложения должны указывать, как они могут быть повторно загружены после автоматического принудительного завершения — для этого используются программные интерфейсы **RegisterApplicationRestart()**, реализованные в ядре операционной системы. Предполагается, что приложения сохраняют данные и состояние

перед принудительным завершением. Соответствующие опции командной строки должны позволять восстановить состояние приложения до его принудительного завершения и соответствующие данные. После повторного запуска приложение должно отобразить этот факт в своем интерфейсе. На уровне операционной системы поддержка перезапуска приложений реализована в компоненте **Service Control Manager (SCM)**, управление настройками которого позволит разработчикам указать число попыток при перезапуске приложений, задержку между повторными перезапусками и т. п.

Устранение сбоев приложений

Тщательное тестирование приложений, как собственными средствами, так и с помощью сценариев, предоставляемых в рамках пакета **Windows Application Verifier** (доступно на Microsoft Download Center — **http://www.microsoft.com/downloads**), позволяет разработчикам свести к минимуму ситуации, приводящие к сбоям в работе приложений. Тесты, входящие в состав **Windows Application Verifier**, используются для проверки совместимости приложений с требованиями Windows Vista Logo и включают в себя проверку заполнения «кучи», нарушений доступа (включая переполнение буфера), проблемы с обработкой критических секций, попытки использования недопустимых ссылок, ошибки TLS-индексов, проблемы использования виртуальной памяти и ошибки, связанные с обработкой исключений.

Большинство сбоев приложений можно предотвратить, если приложение может разрешить пользователям останавливать операции ввода/вывода и другие операции, которые могут заблокировать приложение, без необходимости аварийного завершения процесса. Улучшенные механизмы остановки операций ввода/вывода, поддерживаемые в Windows Vista, включают программные интерфейсы **CancelSynchronousIo()**, которые позволяют отменять вызовы синхронных операций ввода/вывода (**OpenFile()**, **GetFileAttributes()** и т. п.). Так как эти программные интерфейсы должны вызываться из другого потока, приложения, поддерживающие механизмы отмены операций ввода/вывода, должны быть мультипоточными. Еще одно добавление к программным интерфейсам, появившееся в Windows Vista, — это интерфейс **CancelIoEx()**, позволяющий отменить все стоящие в очереди операции ввода/вывода для указанного файла.

Следование рекомендациям по обработке исключений

После возникновения некоторых критических исключений (например, нарушение прав доступа — **access violation**) приложение может перейти в состояние, в котором его дальнейшее выполнение не имеет смысла — например, могут быть разрушены некоторые важные структуры, включая «кучу» процесса или его структуры данных. В этом случае дальнейшее выполнение приложения может привести к искажению или потере данных.

Рекомендации по обработке исключений могут быть следующими.

- Обрабатывайте только известные и ожидаемые исключения.
  - Приложение не должно пытаться обработать глобальные исключения, установить соответствующие фильтры, например, приложение не должно использовать функцию **SetUnhandledExceptionFilter()** для фатальных исключений.
- Не допускайте завершения приложения операционной системой без соответствующих уведомлений пользователей.
  - Приложение не должно вызывать функцию **SetErrorMode()** с параметром **SEM_NOGPFAULTERRORBOX** для завершения средствами операционной системы за исключением тех случаев, когда приложение содержит фильтр для корректной обработки соответствующего исключения.

Общая рекомендация сводится к проверке использования в приложении функций **SetUnhandledExceptionFilter()**, **SetErrorMode()**, блоков **__try** и **__except** для того, чтобы убедиться, что обрабатываются только известные и ожидаемые исключения.

Стабильная работа приложений

В состав ядра Windows Vista входят средства проверки надежности операционной системы, приложений и процессов — **Reliability Analysis Component Services (RAC)**. Сервисы RAC содержат исполняемые файлы агентов, запускаемые через **WMI.Jobs**. Агенты RAC собирают метрики надежности, базирующиеся на наборе предопределенных задач. Одной из специфичных для процесса метрик является **Mean Time To Failure (MTTF)** — среднее время до сбоя, которая высчитывается как общее время работы, деленное на число сбоев. К сбоям относятся зависания приложений и фатальные сбои в работе приложений. Данные собираются на пользовательских компьютерах (там, где запущены RAC-агенты) и агрегируются в Microsoft. Данные, собранные для конкретного приложения в течение 180 дней, будут использованы для генерации соответствующей MTTF-метрики.

Практические вопросы

В этом параграфе мы затронем различные практические вопросы создания приложений, отвечающих требованиям Windows Vista Logo. Предыдущие параграфы были посвящены общим вопросам написания корректно работающих приложений и обсуждению требований, выдвигаемых к приложениям, претендующим на получение логотипов “Certified for Windows Vista” и “Works with Windows Vista”.

Ниже мы рассмотрим основные проблемы, связанные с несовместимостью с Windows Vista, возникающие из-за несоблюдения основных рекомендаций по созданию приложений для операционной системы Windows. К наиболее

часто возникающим проблемам несовместимости можно отнести: некорректную проверку версии операционной системы, работу в привилегированном режиме, использование ресурсов Windows, включая рабочие каталоги операционной системы и реестр и т. п. Мы приведем примеры некорректно работающих приложений и покажем, как устранить проблемы несовместимости — либо средствами, реализованными в операционной системе, либо путем внесения соответствующих изменений в код приложения.

Знакомство с материалами данного параграфа поможет вам не только обеспечить работоспособность приложений, но и сделать их более стабильными и отвечающими требованиям совместимости с операционными системами Windows Vista и Windows 7.

Проверка версии операционной системы

Внутренний номер версии в операционных системах Windows Vista и Windows Server 2008 равен 6. При вызове функции **GetVersion**() возвращается именно этот номер. В следующей таблице показаны номера версий ключевых операционных систем семейства Windows.

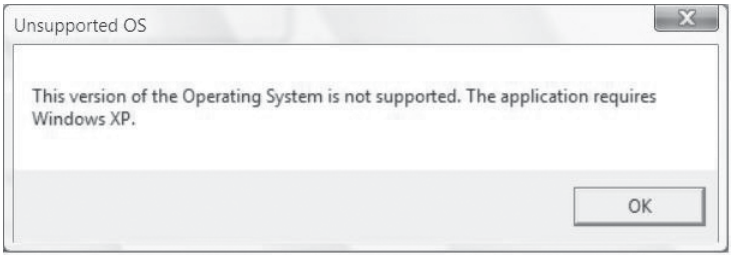
ОС	Windows 2000	Windows XP	Windows Server 2003	Windows Vista	Windows Server 2008
Версия	5.0	5.1	5.2	6.0	6.0

Для корректной проверки номера версии следует изменить код, чтобы выполнялась проверка **>=6**, и проверять не только версию операционной системы, но и ее издание, так как может оказаться, что требуемая для работы приложения функциональность, отсутствует в данном издании операционной системы.

Рассмотрим следующий пример. Пусть мы выполняем проверку версии операционной системы, используя следующий код:

```
static private bool IsOSSupported()
{
    OperatingSystem os = Environment.OSVersion;
    Version vs = os.Version;

    if ((vs.Major == 5) && (vs.Minor == 1))
    {
        return true;
    }
    else
    {
         MessageBox.Show("This version of the Operating System
         is not supported. The application requires Windows
         XP.", "Unsupported OS");
     return false;
    }
}
```

В данном примере проверяется, запускается ли программа под управлением операционной системы Microsoft Windows XP, и естественно, при запуске программы под Windows Vista мы получим следующее сообщение об ошибке (см. рисунок ниже):



Сообщение об ошибке — несовместимая версия операционной системы

Для исправления ситуации существует два способа. Все зависит от того, насколько для конкретного приложения критично выполнение именно под управлением данной версии операционной системы — возможно, разработчики использовали какие-то особенности той или иной версии (обычно — недокументированные и нерекомендуемые к использованию!) операционной системы или просто не совсем корректно выполнили проверку.

В первом случае мы можем указать режим совместимости для выполнения данного приложения. Нажатие правой кнопки на исполняемом файле приложения и выбор команды **Properties** приводит к появлению одноименной диалоговой панели, в которой нас интересует вкладка **Compatibility**. В разделе **Compatibility Mode** укажем требуемую для данного приложения версию операционной системы и включим опцию **Run this program in compatibility mode for**, указав **Windows XP (Service Pack 2)** для нашего примера. Теперь при запуске приложения Windows Vista будет идентифицироваться как Windows XP Service Pack 2 — приложение запустится без сообщений об ошибках.

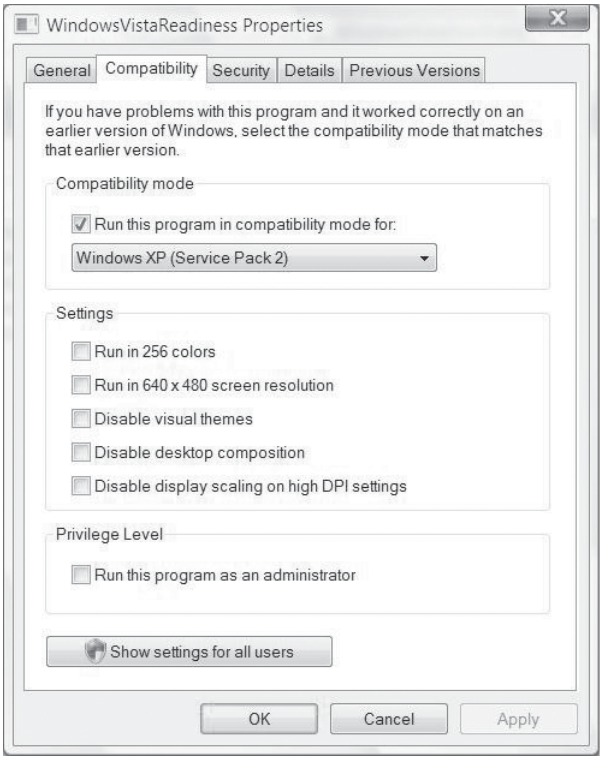
Если же мы не хотим использовать режим совместимости (либо не предполагается использование каких-либо уникальных свойств конкретной версии операционной системы), мы можем исправить код проверки версии. В нашем случае, мы должны заменить строку

```
if ((vs.Major == 5) && (vs.Minor == 1))
```

на

```
if ((vs.Major == 5) && (vs.Minor >= 1) || vs.Major >= 6 )
```

и перекомпилировать приложение. Дальнейших проблем с определением версии операционной системы не будет.



Задание режима совместимости с предыдущими версиями операционной системы

Работа в привилегированном режиме

Одна из новинок в Windows Vista, наиболее часто приводящая к несовместимости с существующими приложениями, — это повышенные требования к безопасности, благодаря которым пользователи работают под учетной записью **standard user** с пониженным набором разрешений (permissions) и привилегий. По умолчанию в Windows Vista каждый пользователь работает под учетной записью **standard user**, даже если он зарегистрировался как член группы администраторов. Соответственно, когда пользователь пытается запустить приложение, которое требует привилегий администратора, операционная система запрашивает подтверждение. Только приложения, выполняемые с привилегиями администратора, могут вносить изменения в глобальные настройки операционной системы. Этот механизм, впервые реализованный в Windows Vista, называется **User Account Control** (**UAC**).

Ниже мы рассмотрим несколько примеров работы механизма **User Account Control** и приведем рекомендации по обеспечению совместимости существующих и создаваемых приложений с этим механизмом.

Запись в Program Files

В первом примере мы убедимся в том, что когда пользователь работает под учетной записью **standard user**, задействуется механизм виртуализации некоторых системных ресурсов, включая области файловой системы и реестра.

Предположим, в нашем приложении есть код, который создает файл в папке **Program Files** и записывает в него какую-то информацию. Код выглядит следующим образом:

```
private void OnWriteToProgramFiles(object sender, EventArgs e)
{
 try
  {
   string folderPath =
    Path.Combine(Environment.GetFolderPath(
      Environment.SpecialFolder.ProgramFiles),
      "WindowsVistaReadiness");
     if (!Directory.Exists(folderPath))
      {
        Directory.CreateDirectory(folderPath);
      }
      string filePath = Path.Combine(folderPath, "log.txt");
      using (TextWriter tw = new StreamWriter(filePath, true))
      {
        tw.WriteLine("entry added " + DateTime.Now);
      }
      MessageBox.Show("Entry has successfully been added to Log.txt",
       "Write To Program Files", MessageBoxButtons.OK,
       MessageBoxIcon.Information);
  }
  catch (System.Security.SecurityException secEx)
  {
   MessageBox.Show(string.Format("SecurityException: {0}",
    secEx.Message), "Write To Program Files", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
  }
  catch (UnauthorizedAccessException authEx)
  {
   MessageBox.Show(string.Format("UnauthorizedAccessException: {0}",
    authEx.Message), "Write To Program Files", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
  }
  catch (Exception ex)
  {
   MessageBox.Show(ex.Message, "Write To Program Files",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
  }
}
```

Вторая процедура, используемая в нашем примере, проверяет принадлежность пользователя к группе администраторов. Вот код этой процедуры:

```
private void OnIsAdministrator(object sender, EventArgs e)
{
 WindowsIdentity wi = WindowsIdentity.GetCurrent();
 WindowsPrincipal wp = new WindowsPrincipal(wi);
 if (wp.IsInRole("Administrators"))
 {
  MessageBox.Show("Yes, you are an Administrator.",
  "Is Administrator?", MessageBoxButtons.OK,
  MessageBoxIcon.Information);
 }
 else
 {
  MessageBox.Show("No, you are not an Administrator!",
  "Is Administrator?", MessageBoxButtons.OK, MessageBoxIcon.Error);
 }
}
```

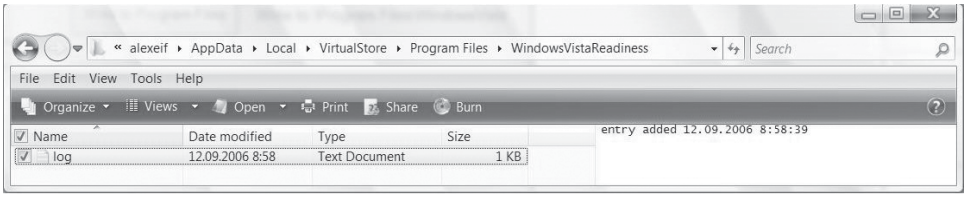
В процедуре инициализации формы — **OnFormLoad** — добавим следующий код:

```
AppDomain.CurrentDomain.SetPrincipalPolicy
 (PrincipalPolicy.WindowsPrincipal);
```

Запустим наше приложение и активизируем функцию **OnWriteToProgramFiles**. В результате мы получим сообщение, подтверждающее успешную запись файла в папку **Program Files**. Если же мы попытаемся обнаружить файл в этой папке, то его там не будет — сработал упомянутый выше механизм виртуализации ресурсов. На самом деле, файл был записан в папку users; чтобы увидеть ее саму и ее содержимое, в Windows Explorer необходимо включить режим отображения файлов с атрибутом «**hidden**» — для этого, как и в предыдущих версиях Windows используется команда **Folder Options**, вкладка **View** и опция **Show Hidden Files and Folders**. Папка, используемая для виртуализации, находится по следующему адресу:

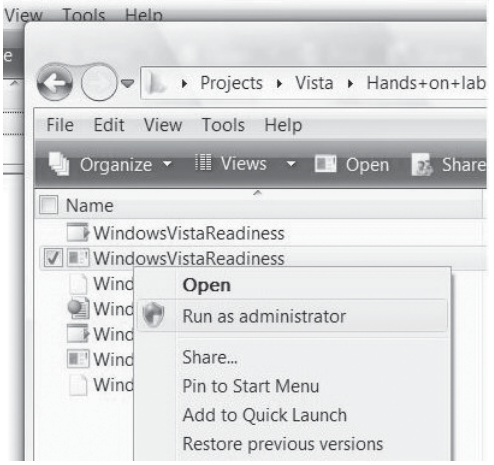
```
c:\users\{user name}\AppData\Local\VirtualStore\Program Files\
```

и действительно, в ней мы и обнаружим файл, который мы пытались записать в папку **Program Files**.



Виртуалиация системных ресурсов

Если же запустить наше приложение от лица администратора (опция **Run As Administrator**) при нажатии правой кнопки мыши на исполняемом файле, то вместо виртуального каталога файл будет записан непосредственно в папку **Program Files**.



Запуск приложения от лица администратора

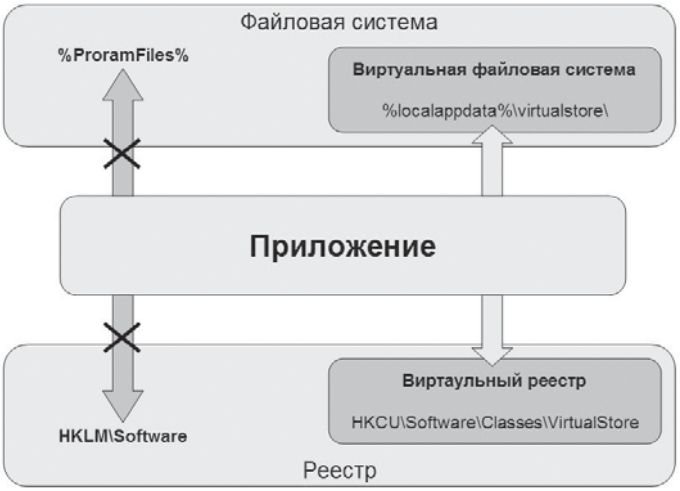
Виртуализация

Выше, обсуждая доступ к файлам и папкам при различных режимах работы, мы упомянули виртуализацию. **Виртуализация** — *это механизм обеспечения совместимости приложений, работающих под учетной записью Standard User. Разработчики не должны предполагать, что механизмы виртуализации останутся неизмененными в последующих версиях операционной системы Microsoft Windows.*

В предыдущих версиях операционной системы Microsoft Windows многие приложения обычно выполнялись под учетной записью администратора. В результате, такие приложения могли свободно читать и записывать системные файлы и ключи реестра. При выполнении таких приложений под учетной записью Standard User возникали ошибки, связанные с недостаточным набором прав для выполнения некоторых операций. В Windows Vista улучшена поддержка таких приложений за счет того, что запись (и последующие операции с файлами и ключами реестра) перенаправляются в некоторую область на диске, определенную в профиле пользователя. Такая область поддерживается для каждого пользователя компьютера. Например, если приложение пытается записать файл **C:\Program Files\Contoso\Settings.ini** и пользователь не имеет прав доступа к этому каталогу, запись будет перенаправлена в каталог **C:\Users\<ИмяПользователя>\AppData\Local\VirtualStore\Program Files\Contoso\Settings.ini**. Для операций с реестром, если приложение

пытается записать в ветвь **HKEY_LOCAL_MACHINE\Software\Contoso**, запись будет автоматически перенаправлена в **HKEY_CURRENT_USER\Software\Classes\VirtualStore\MACHINE\Software\Contoso** или в **HKEY_USERS\<User SID>_Classes\VirtualStore\Machine\Software\Contoso**.

На следующем рисунке показан процесс виртуализации в Windows Vista.



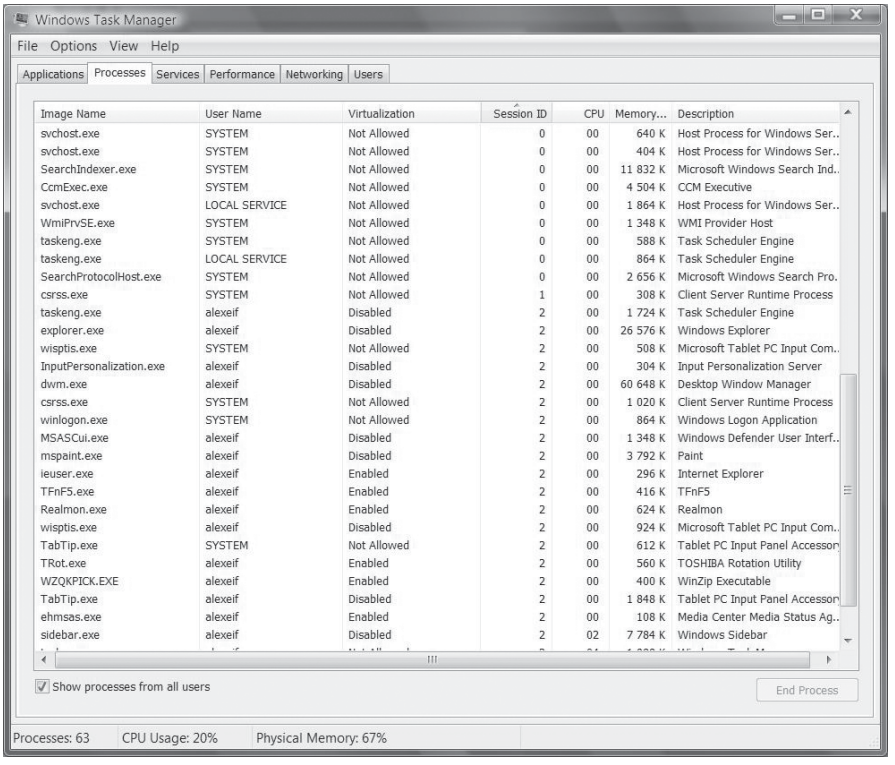
Процесс виртуализации в Windows Vista

Для того чтобы определить, какие приложения будут использовать механизмы виртуализации, а какие смогут записывать данные напрямую, можно воспользоваться утилитой **Task Manager**.

Запустим утилиту **Task Manager**, в меню **View** выберем команду **Select Columns...** и включим опцию **Virtualization**. Результат работы утилиты **Task Manager** показан на следующем рисунке.

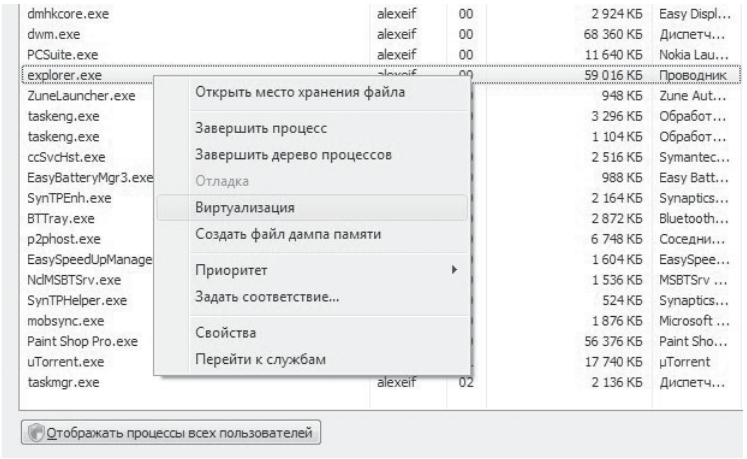
В следующей таблице показаны возможные значения, отображаемые в колонке **Virtualization** утилиты **Task Manager**.

Значение	Описание
Пусто	Защищенный процесс — невозможно получить значение параметра virtualization. Обычно такие процессы работают с отключенной виртуализацией
Enabled	У процесса разрешена виртуализация реестра и файловой системы
Disabled	У процесса запрещена виртуализация реестра и файловой системы
Not Allowed	У процесса не может быть включена виртуализация реестра и файловой системы



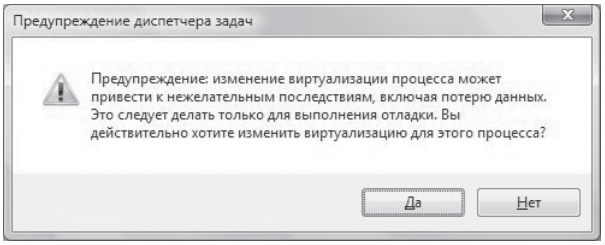
Использование утилиты Task Manager

Для динамического изменения режима виртуализации необходимо щелкнуть правой кнопкой мыши по колонке **Virtualization**. Если рядом со значением присутствует отметка — виртуализация включена, в противном случае виртуализация отключена.



Динамическое изменение режима виртуализации

Прежде чем у вас появится возможность изменить режим виртуализации, вам необходимо подтвердить эту операцию в диалоговой панели, показанной на рисунке ниже.



Диалоговая панель подтверждения изменения режима виртуализации

Как мы отметили выше, виртуализация распространяется на два компонента системы — файловую систему (файловая виртуализация) и реестр (виртуализация реестра).

Важно! При разработке приложений для Windows Vista для того, чтобы отключить механизмы файловой виртуализации и виртуализации реестра, используйте встроенный в приложение манифест с указанием необходимого значения параметра **requestedExecutionLevel**.

Основные правила виртуализации

Виртуализация поддерживается для:

- 32-битных интерактивных процессов;
- файлов/папок и ключей реестра, доступных для записи администраторам.

Виртуализация запрещена для:

- 64-битных процессов;
- неинтерактивных процессов;
- имперсонированных процессов;
- вызовов в режиме Kernel;
- исполняемых файлов со встроенным манифестом, в котором указано необходимое значение параметра requestedExecutionLevel.

Виртуализация и роуминг:

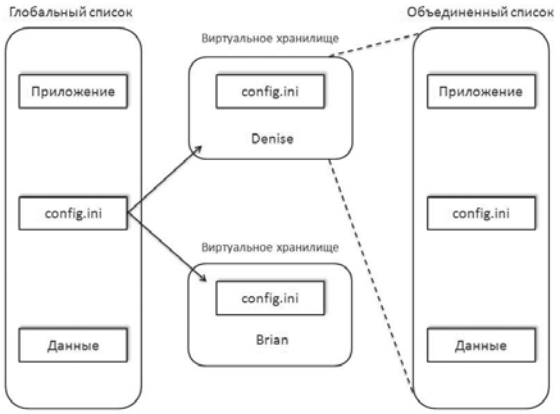
- не поддерживается роуминг для виртуализации файлов/папок и ключей реестра;
- не поддерживается роуминг для процессов, ассоциированных с глобальными объектами.

Файловая виртуализация

Файловая виртуализация используется в тех случаях, когда приложению необходимо сохранить файл, например файл конфигурации, в системном каталоге, доступном только для администраторов. Выполнение данной операции под учетной записью Standard User может привести к возникновению ошибок, связанных с недостаточным уровнем привилегий. Когда приложение записывает данные в системный каталог, доступный только для администраторов, Windows Vista перенаправляет операцию записи, а также последующие файловые операции в специальную область на диске, выделяемую каждому пользователю. Этот каталог располагается по адресу **%LOCALAPPDATA%\VirtualStore**. Затем, когда приложение пытается считать ранее записанный файл, Windows Vista предоставляет доступ к файлу из виртуального каталога. Так как инфраструктура безопасности в Windows Vista обрабатывает виртуализацию без непосредственного участия приложения, оно работает также, как если бы ему был предоставлен доступ к каталогу **Program Files**. Прозрачность механизмов файловой виртуализации обеспечивает функционирование большого числа приложений, которые продолжают «верить», что работают с защищенными ресурсами, тогда как на самом деле они обращаются к их виртуальной версии.

Отметим, что при программном переборе ресурсов — папок и реестра — Windows Vista объединяет глобальные файлы/папки и ключи реестра в единый список. В этом списке глобальные (защищенные) ресурсы отображаются вместе с виртуальными ресурсами.

Важно понимать, что виртуальная копия всегда представляется приложению в первую очередь. Например, если файл config.ini присутствует в каталоге **\Program Files\<AppName>\config.ini** и в **%LOCALAPPDATA%\VirtualStore\config.ini**, файл **config.ini** из виртуального каталога будет всех считываться первым, даже если содержимое файла **\Program Files\<AppName>\config.ini** было обновлено.



Пример виртуализации

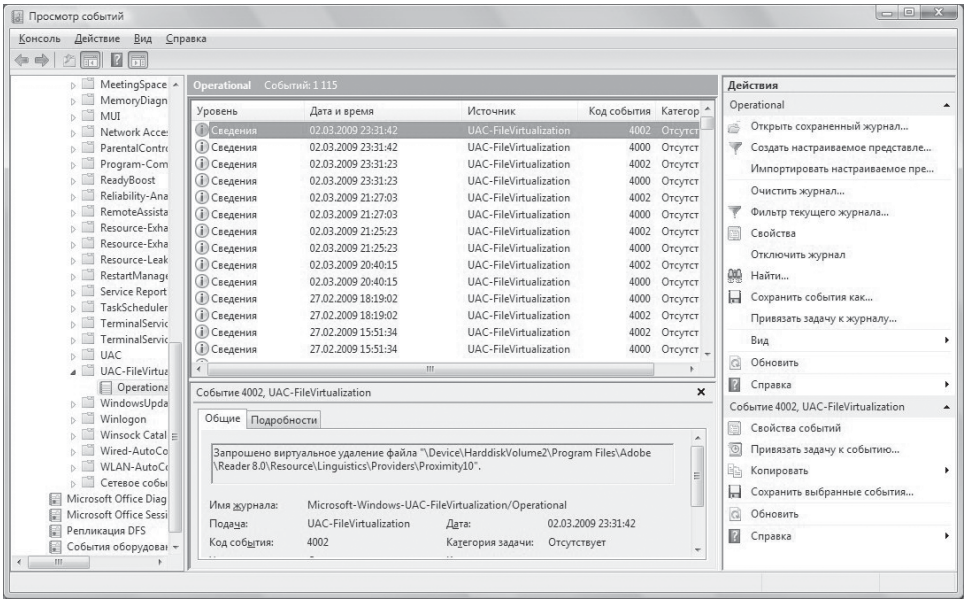
На рисунке показано, как глобальные и объединенные виртуальные ресурсы отображаются для пользователей с разными учетными записями. В данном примере пользователь Denise работает под учетной записью администратора, пользователь Brian — под учетной записью Standard User.

Файловые операции и системный протокол

Виртуализация файловых операций отражается в системном протоколе (утилита **Event Viewer**) в специальной ветви с именем **Applications and Services Logs/Microsoft/Windows/UAC-FileVirtualization**.

Протокол содержит список событий, которые сообщают об успешности или неуспешности операций файловой виртуализации, а также содержат информацию о файлах, которые были виртуализированы, и процессах, использовавших эти файлы.

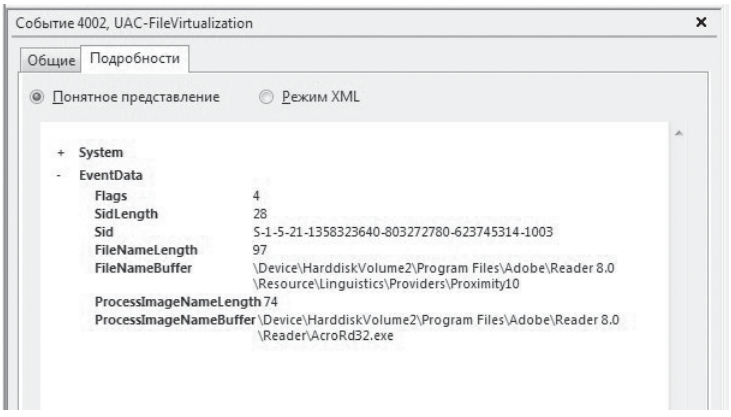
На следующем рисунке показана общая информация о событии, отображаемая в системном протоколе.



Общая информация о событии, отображаемая в системном протоколе

Детальная информация о событии, доступная при выборе вкладки **Details** и опции **Friendly View**, содержит имя процесса, а также другие данные о виртуализации.

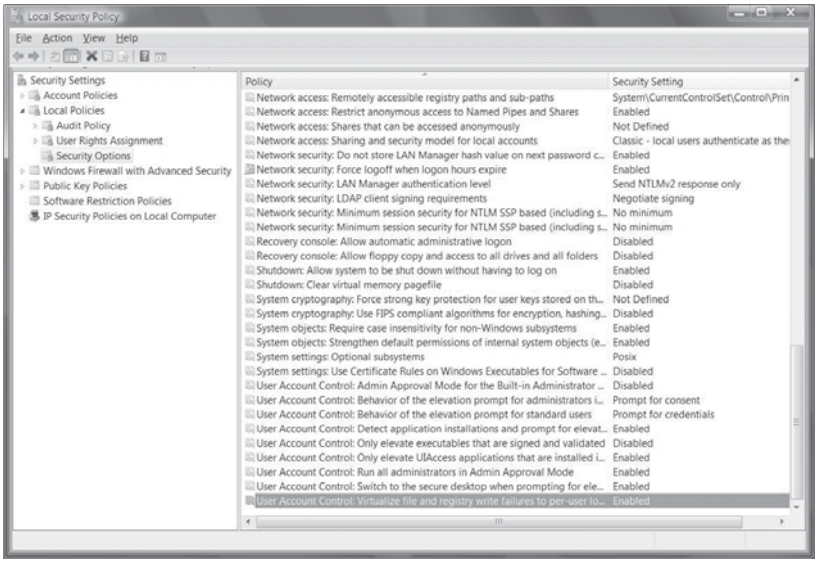
При необходимости процесс виртуализации можно сконфигурировать через механизм **Group Policy**. По умолчанию виртуализация включена. В следующем разделе мы рассмотрим использование механизма **Group Policy** для управления процессом виртуализации.



Детальная информация о событии

Конфигурация виртуализации через Group Policy

Для доступа к механизму **Group Policy** необходимо воспользоваться утилитой **SecPol.msc**, вызов которой приводит к появлению Microsoft Management Console с утилитой **Local Security Policy**. Для доступа к настройкам виртуализации необходимо раскрыть ветвь **Local Policies | Security Options** и выбрать элемент **User Account Control: Virtualize file and registry write failures to per-user locations**. Как мы отметили выше, по умолчанию значение этой опции включено (**Enabled**). Для отключения виртуализации следует присвоить этой опции значение **Disabled**. На следующем рисунке показан интерфейс утилиты **Local Security Policy**.

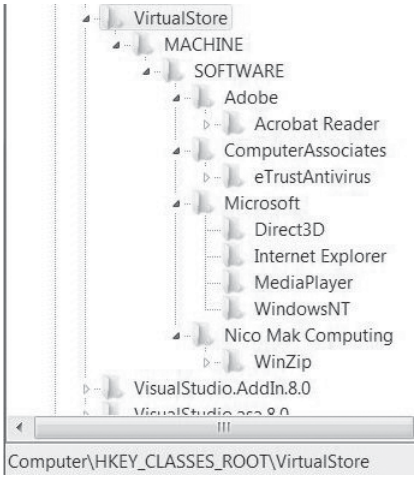


Утилита Local Security Policy

Виртуализация реестра

Механизм виртуализации реестра схож с механизмом файловой виртуализации и применим к ключам в ветви **HKEY_LOCAL_MACHINE\SOFTWARE** реестра. Виртуализация реестра позволяет приложениям, которые хранят информацию в данной ветви продолжать работать без ошибок, даже если они запущены под учетной записью **Standard User**. Ключи и данные перенаправляются в ветвь с именем **HKEY_CLASSES_ROOT\VirtualStore\SOFTWARE**. Как и в случае с файловой виртуализацией, каждый пользователь имеет собственную копию всех данных, хранимых приложениями в ветви **HKEY_LOCAL_MACHINE\SOFTWARE**.

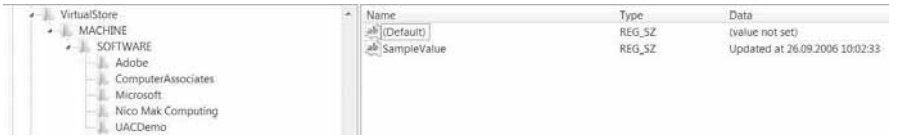
На следующем рисунке показана ветвь реестра **HKEY_CLASSES_ROOT\VirtualStore\SOFTWARE**.



Ветвь HKEY_CLASSES_ROOT\VirtualStore\SOFTWARE

Отметим, что виртуализация реестра, как и файловая виртуализация, предназначена для обеспечения совместимости с приложениями, написанными для предыдущих версий операционной системы Microsoft Windows. Приложения, написанные для работы под управлением Microsoft Windows Vista и Windows 7, не должны записывать данные в системные области.

Обратите внимание на то, что Microsoft планирует удалить механизмы виртуализации в последующих версиях операционной системы, по мере того как большинство приложений будет смигрировано на Windows Vista. Например, механизмы виртуализации запрещены для 64-битных приложений.



Виртуализация реестра

Использование манифеста

Одно из требований, выдвигаемых к приложениям, претендующим на сертификацию под Windows Vista и Windows 7, — это наличие специального ресурса — манифеста, в котором, в частности, описываются требования по привилегиям, необходимым для запуска данного приложения. Манифест — это XML-файл, который выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8" ?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
   manifestVersion="1.0">
<assemblyIdentity version="1.0.0.0"
    processorArchitecture="X86"
    name="WindowsVistaReadiness"
    type="win32" />
  <description>WindowsVistaReadiness Application</description>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
          <requestedExecutionLevel level="asInvoker" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

В ветви **trustInfo** описывается требуемый для приложения уровень привилегий — он задается в виде атрибута **level** параметра **requestedExecutionLevel**. Этот атрибут может иметь три значения:

Значение	Описание
asInvoker	Наследовать привилегии от вызывающего процесса
highestAvailable	Предоставить максимально возможные привилегии, доступные пользователю
requireAdministrator	Пользователь должен быть администратором

Включение манифеста в приложение требует выполнения нескольких шагов.

1. Добавить к проекту XML-файл (нажать правую кнопку мыши на проекте, выбрать команду **Add | New Item...**, выбрать **XML File**).
2. Сохранить файл с именем проекта и расширением **exe.manifest**.
3. Добавить к проекту файл ресурса (нажать правую кнопку мыши на проекте, выбрать команду **Add | New Item...**, выбрать **Text File**).
4. Сохранить файл с именем проекта и расширением **.rc**.
5. Открыть файл ресурса (**Open with... Source Code Editor**) и добавить в него следующие строки:

```
#define RT_MANIFEST 24
#define APP_MANIFEST 1
APP_MANIFEST RT_MANIFEST App.exe.manifest
```

6. Заменить **App.exe.manifest** на имя файла манифеста из п. 2.
7. В свойствах проекта (нажать правую кнопку мыши на проекте, выбрать команду **Properties**) выбрать вкладку **Build Events** и в строке **Pre-build event command line** указать:

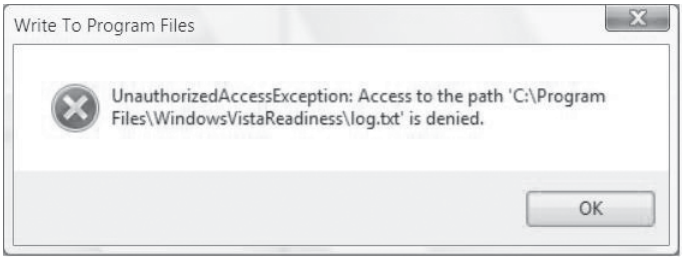
```
"path\rc.exe" $(ProjectDir)$(ProjectName).rc
```

8. Заменить path на местонахождение компилятора ресурсов — rc.exe, который может находиться в одном из следующих каталогов:
   - **Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin**
   - **Program Files\Microsoft SDKs\Windows\v1.0\Bin**
9. Добавить элемент **MSBuild** для включения ресурсов в исполняемый файл. Для этого необходимо открыть скрипт проекта в редакторе (нажать правую кнопку мыши на проекте, выбрать команду **Unload Project**, затем, нажать правую кнопку мыши на проекте, выбрать команду **Edit <Project name>**.
10. Добавить в скрипт следующий элемент

```
<PropertyGroup>
  <Win32Resource>App.res</Win32Resource>
</PropertyGroup>
```

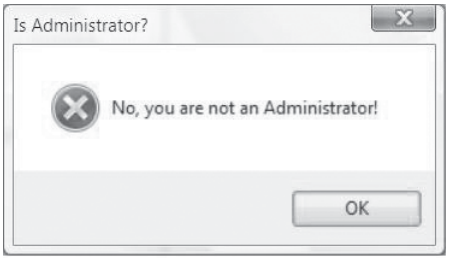
11. После этого выполнить команду **Rebuild Solution**.

После того как мы добавили к нашему приложению манифест, запустим приложение еще раз и убедимся в том, что указание параметра параметра requestedExecutionLevel приводит к появлению ошибки — механизм виртуализации не работает.



Ошибка доступа к системным ресурсам

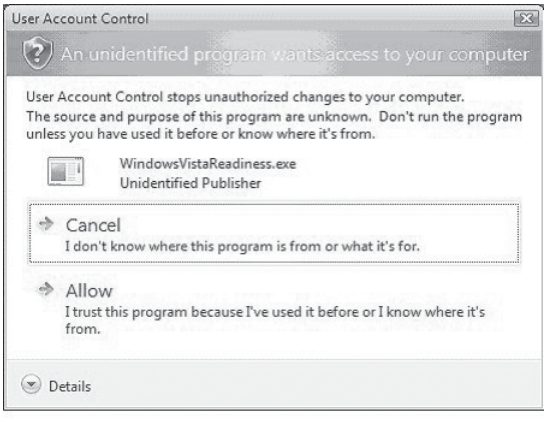
Выполнение процедуры **OnIsAdministrator** покажет, что мы не являемся администратором.



Проверка полномочий

Вернемся к манифесту приложения и заменим значение атрибута **level** параметра **requestedExecutionLevel** на **highestAvailable**. После этого выполним команду **Rebuild Solution** и запустим приложение. В этом случае мы получим запрос на подтверждение повышения полномочий для выполнения приложения.

Нажатие кнопки **Allow** приведет к повышению полномочий и выполнению нашего приложения — это можно проверить, вызвав процедуру **OnIsAdministrator**. Процедура записи файла в папку **Program Files** также будет работать (без использования механизма виртуализации), т. к. наше приложение теперь работает с привилегиями администратора.



Запрос на повышение полномочий

Отметим, что в диалоговой панели **User Account Control** указано, что приложение имеет атрибут **Unidentified Publisher** — это вызвано тем, что приложение не имеет цифровой подписи. Наличие цифровой подписи у всех исполняемых файлов и их бинарных компонентов также является одним из обязательных требований для получения сертификации под Windows Vista и Windows 7. Это требование относится к файлам со сле-

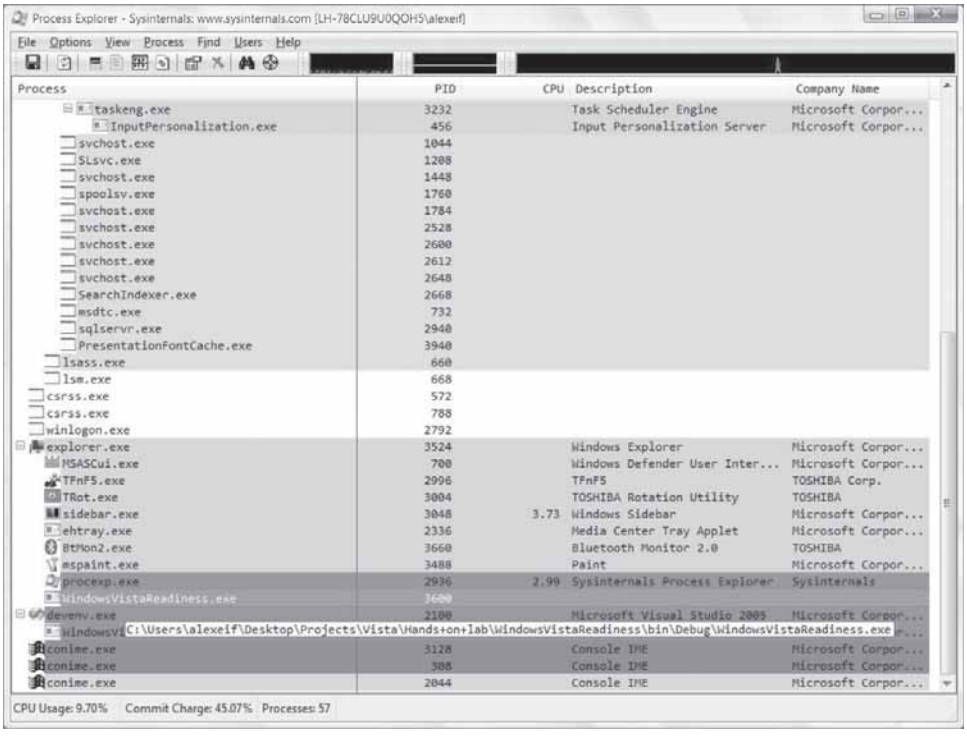
дующими расширениями — .exe, .dll, .ocx, .sys, .cpl, .drv и .scr. Для того чтобы проверить наличие цифровой подписи, можно воспользоваться следующей командой:

```
signtool verify /pa /v <AppName>
```

Утилита **SignTool** поставляется в составе Windows SDK — наборе программных средств для создания Windows-приложений, который доступен для скачивания по адресу **http://msdn.microsoft.com/platformsdk/**.

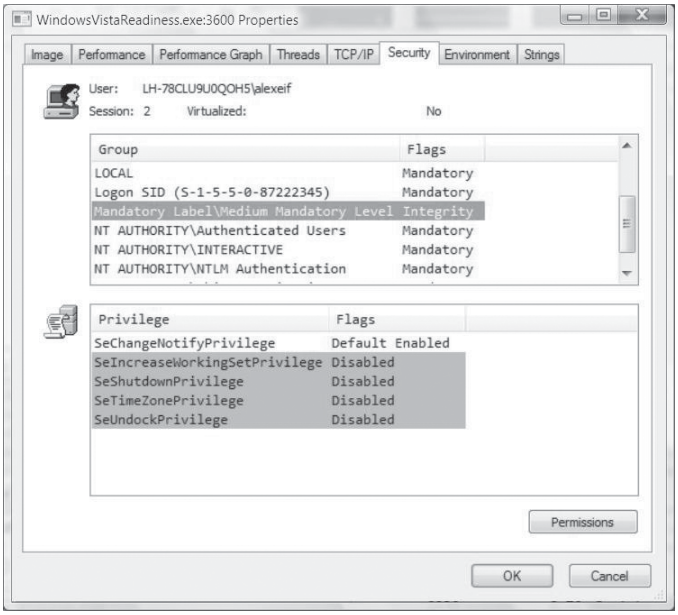
Использование токена Admin

Для определения режима работы приложения — под полным токеном **Admin** или под токеном с ограниченными полномочиями, воспользуемся утилитой **Process Explorer** с сайта **http://www.sysitnernals.com**. Сначала изменим значение атрибута **level** параметра **requestedExecutionLevel** обратно на **asInvoker**, выполним команду **Rebuild Solution** и запустим приложение. Теперь запустим утилиту **Process Explorer**. В списке приложений найдем наш демо-пример (**WindowsVistaReadiness**).



Утилита Process Explorer

Выберем приложение **WindowsVistaReadiness** и перейдем на вкладку **Security**. Обратим внимание на то, что опция **Mandatory** указывает на **Medium Mandatory Level** и у приложения имеется минимальный набор привилегий.

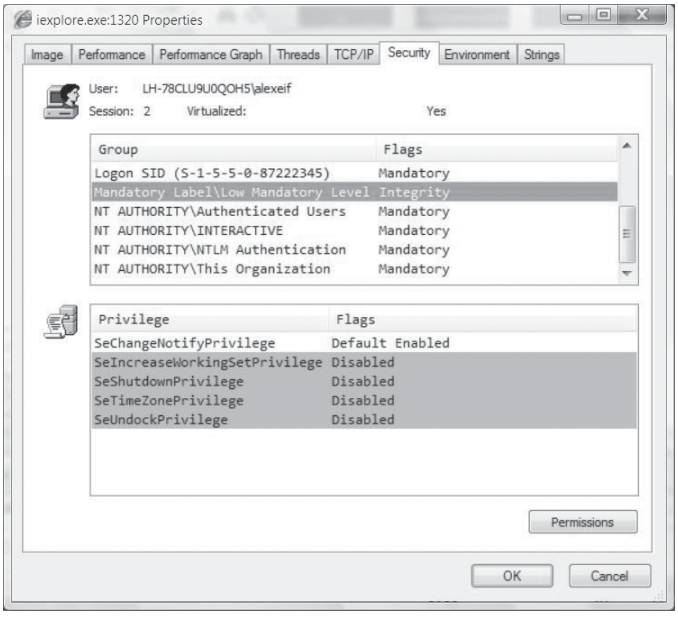


Набор привилегий приложения

Завершим работу нашего приложения и запустим его еще раз, использовав опцию **Run as Administrator**. Убедимся в том, что теперь уровень **Mandatory** приложения стал **High Mandatory Level** и у приложения появился набор дополнительных привилегий. В этом заключается отличие режимов работы приложения под полным токеном **Admin** от работы под токеном с ограниченными полномочиями.

Отметим еще одну интересную особенность. Если мы запустим Internet Explorer и изучим его уровень **Mandatory**, то обнаружим, что он имеет значение **Low Mandatory Level**.

Это отражает тот факт, что Internet Explorer работает с минимальными привилегиями и, в частности, эта программа не может посылать Windows-сообщения другим приложениям.



Набор привилегий Internet Explorer

Mandatory Integrity Control

Mandatory Integrity Control (MIC) — это механизм Windows, позволяющий управлять коммуникациями между процессами за счет задания различных уровней целостности выполнения процессов. Как мы увидели выше, стандартное приложение, выполняемое под учетной записью **standard user**, имеет **Medium Mandatory Level**, тогда как приложение, выполняемое под учетной записью **administrator**, имеет **High Mandatory Level** и, соответственно, набор дополнительных привилегий. Изоляция процессов позволяет расширить модель авторизации на меж-процессные коммуникации.

Как мы отметили выше, процессы с ограниченными возможностями, например код внутри Internet Explorer, выполняются с минимальными привилегиями.

Таким образом, приложения, выполняющиеся с низким уровнем привилегий, не могут посылать сообщения приложениям, выполняющимся с более высоким уровнем привилегий, за исключением тех случаев, когда последние непосредственно не разрешили получение сообщений, используя функцию **ChangeWindowMessageFilter()**. Точно также, приложения с низкими привилегиями, не могут модифицировать ссылки типа **HWND**, владельцами которых являются приложения с более высокими привилегиями. Для обеспечения совместимости функция **SendMessage()** и ряд

других функций для отсылки сообщений возвращают код успешного завершения, даже если выполнение таких функций заблокировано из-за описанных выше проблем с уровнями привилегий.

Рассмотрим пример, в котором одно приложение посылает Windows-сообщение другому приложению. Вот код, используемый для посылки сообщения, — обратите внимание на использование механизмов **P/Invoke**:

```
[DllImport("user32", SetLastError = true, CharSet = CharSet.Auto)]
static extern System.IntPtr FindWindow(string lpClassName, string lpWindowName);

[DllImport("user32.dll", SetLastError = true)]
static extern bool PostMessage(IntPtr hWnd, uint Msg, IntPtr wParam, IntPtr lParam);

public const int WM_USER = 0x400;
public const int WM_IPC = WM_USER + 5001;

private void sendWindowsMessage_Click(object sender, EventArgs e)
{
 PostMessage(FindWindow(null, "WM Listener"),
 WM_IPC, IntPtr.Zero, IntPtr.Zero);
}
```

Второе приложение «слушает» очередь сообщений через MessageFilter и реагирует на событие WM_IPC. Вот код этого приложения:

```
public class WindowsMessageHelper
{
[DllImport("user32", SetLastError = true, CharSet = CharSet.Auto)]
static extern System.IntPtr FindWindow(string lpClassName, string lpWindowName);

[DllImport("user32.dll", SetLastError = true)]
static extern bool PostMessage(IntPtr hWnd, uint Msg, IntPtr wParam, IntPtr lParam);

public const int WM_USER = 0x400;
public const int WM_IPC = WM_USER + 5001;

public static void SendMessage(string title, IntPtr wParam, IntPtr lParam)
{
 PostMessage(FindWindow(null, title), WM_IPC, wParam, lParam);
 }
}
public class MessageFilter : IMessageFilter
{
 private WMForm mainForm;

 public MessageFilter(WMForm mainForm)
 {
  this.mainForm = mainForm;
 }
```

```
 public bool PreFilterMessage(ref Message m)
 {
  bool returnValue = false;
  if (m.Msg == WindowsMessageHelper.WM_IPC)
  {
   MessageBox.Show(mainForm, "Got the message.");
   returnValue = true;
  }
  return returnValue;
 }
}
```

Запустим оба приложения и убедимся в том, что сообщение корректно передается от одного приложения к другому. Если же теперь запустить приложение-приемник с опцией **Run as Administrator**, то сообщение до него не дойдет — сработает механизм изоляции процессов, имеющих разные привилегии, для успешной отсылки сообщения необходимо также запустить и посылающее приложение с опцией **Run as Administrator**.

Установка приложений

Как показывает практика, большинство инсталляционных пакетов подразумевает, что они запускаются под учетной записью **administrator**. В Windows Vista и Windows 7 это может привести к тому, что установка приложения завершится неудачно.

Предположим, что у нас есть инсталляционный пакет, рассчитанный на запуск под учетной записью **administrator**. Выполнение такого пакета под Windows Vista и Windows 7 завершится с ошибкой — нам потребуется внесение изменений в MSI-файл.

По умолчанию MSI-файл выполняется с уровнем привилегий asInvoker, если в манифесте не указан другой уровень. Предположим, что наш пакет не имеет манифеста и, соответственно, выполняется под Local User Account (LUA). Внутри пакета происходит попытка установки сервиса, которая требует привилегий администратора — таким образом установка завершается с ошибкой.

Используя утилиту **Orca**, которая входит в состав Platform SDK, убедимся в том, что MSI-файл содержит вложенные **custom actions** (см. таблицу **Custom Action**) — это не соответствует одному из требований к сертификации приложений, где указано, что инсталляционные пакеты не должны содержать **custom actions** с кодами **7** (установка вложенного пакета), **23** (установка внешнего пакета) и **39** (установка уже установленного продукта). Для того чтобы наш инсталляционный пакет мог работать нормально, необходимо удалить указанные custom actions и перенести их в отдельные пакеты.

Изоляция Session 0

В Windows XP, Windows Server 2003 и более ранних версиях операционной системы Microsoft Windows все системные сервисы выполнялись в той же сессии, что и сессия, в которую загрузился первый пользователь системы. Такая сессия называется Сессия 0 (Session 0). Выполнение сервисов и пользовательских приложений в Сессии 0 могло приводить к определенным рискам, связанным с безопасностью, т. к. сервисы могут выполняться с повышенными привилегиями (например, под учетной записью **Local System**), вредоносные приложения могли пытаться использовать эти сервисы для повышения собственного уровня привилегий.

На следующих рисунках показан результат выполнения утилиты Task Manager в операционной системе Windows XP. Обратите внимание на то, что системные процессы (**SYSTEM**), сетевые сервисы (**NETWORK SERVICE**), локальные сервисы (**LOCAL SERVICE**) и пользовательские приложения (**AlexeiF**) работают в одной сессии — Сессии 0, а также потенциальные возможности для атаки в Windows XP.

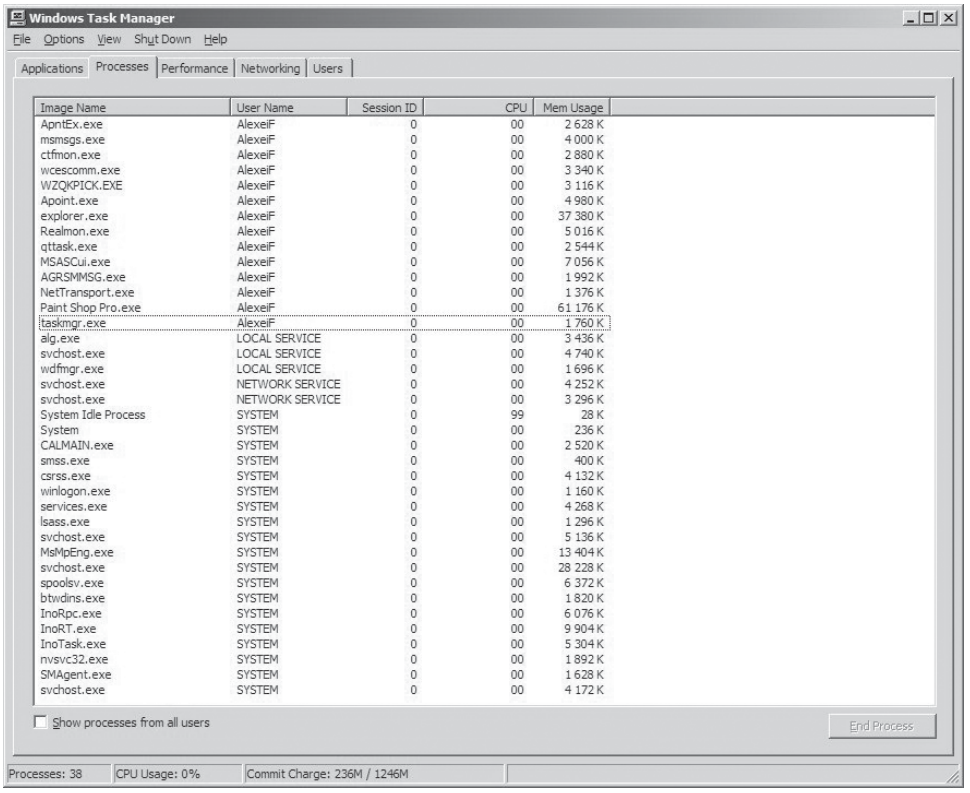
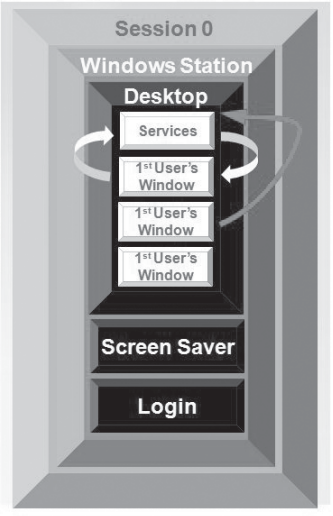


Image Name	User Name	Session ID	CPU	Mem Usage
ApntEx.exe	AlexeiF	0	00	2 628 K
msmsgs.exe	AlexeiF	0	00	4 000 K
ctfmon.exe	AlexeiF	0	00	2 880 K
wcescomm.exe	AlexeiF	0	00	3 340 K
WZQKPICK.EXE	AlexeiF	0	00	3 116 K
Apoint.exe	AlexeiF	0	00	4 980 K
explorer.exe	AlexeiF	0	00	37 380 K
Realmon.exe	AlexeiF	0	00	5 016 K
qttask.exe	AlexeiF	0	00	2 544 K
MSASCui.exe	AlexeiF	0	00	7 056 K
AGRSMMSG.exe	AlexeiF	0	00	1 992 K
NetTransport.exe	AlexeiF	0	00	1 376 K
Paint Shop Pro.exe	AlexeiF	0	00	61 176 K
taskmgr.exe	AlexeiF	0	00	1 760 K
alg.exe	LOCAL SERVICE	0	00	3 436 K
svchost.exe	LOCAL SERVICE	0	00	4 740 K
wdfmgr.exe	LOCAL SERVICE	0	00	1 696 K
svchost.exe	NETWORK SERVICE	0	00	4 252 K
svchost.exe	NETWORK SERVICE	0	00	3 296 K
System Idle Process	SYSTEM	0	99	28 K
System	SYSTEM	0	00	236 K
CALMAIN.exe	SYSTEM	0	00	2 520 K
smss.exe	SYSTEM	0	00	400 K
csrss.exe	SYSTEM	0	00	4 132 K
winlogon.exe	SYSTEM	0	00	1 160 K
services.exe	SYSTEM	0	00	4 268 K
lsass.exe	SYSTEM	0	00	1 296 K
svchost.exe	SYSTEM	0	00	5 136 K
MsMpEng.exe	SYSTEM	0	00	13 404 K
svchost.exe	SYSTEM	0	00	28 228 K
spoolsv.exe	SYSTEM	0	00	6 372 K
btwdins.exe	SYSTEM	0	00	1 820 K
InoRpc.exe	SYSTEM	0	00	6 076 K
InoRT.exe	SYSTEM	0	00	9 904 K
InoTask.exe	SYSTEM	0	00	5 304 K
nvsvc32.exe	SYSTEM	0	00	1 892 K
SMAgent.exe	SYSTEM	0	00	1 628 K
svchost.exe	SYSTEM	0	00	4 172 K

Выполнение процессов в Windows XP

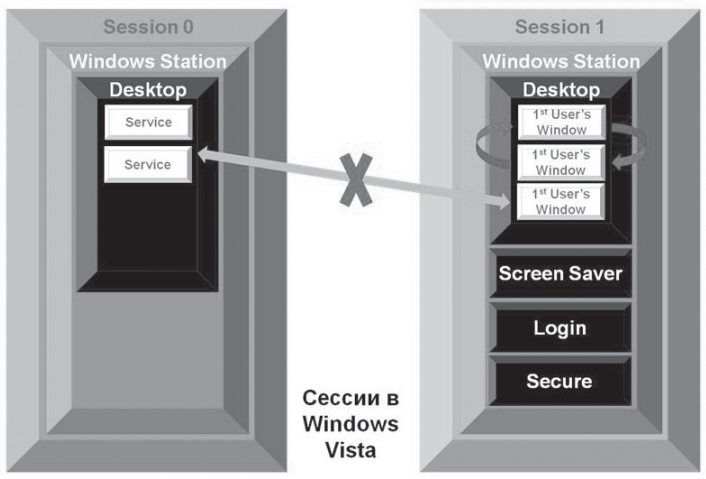


Потенциальные возможности для атаки в Windows XP

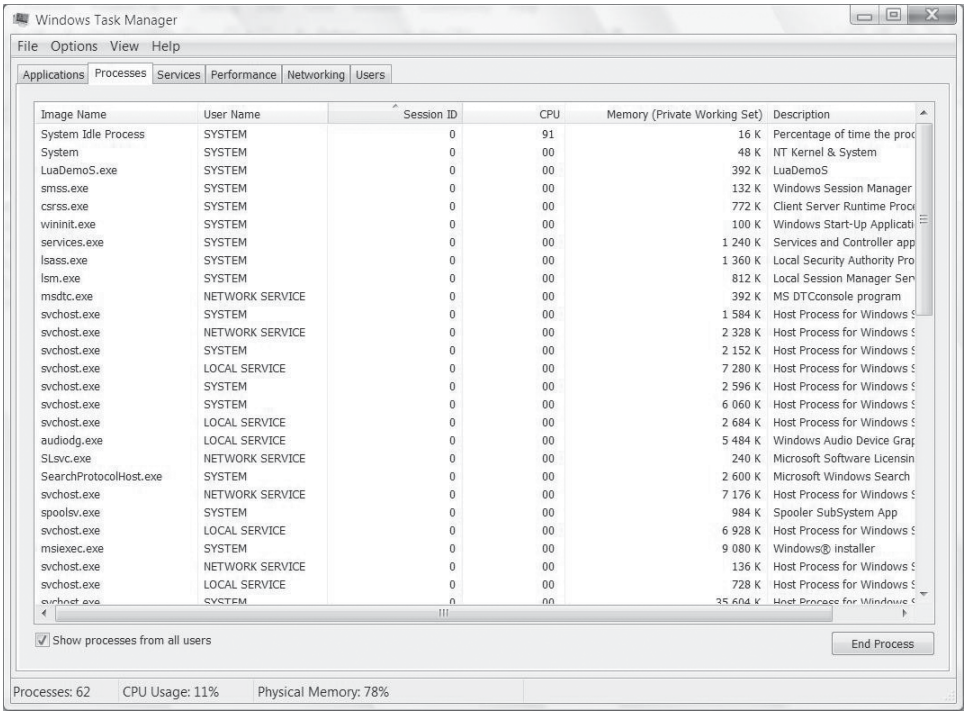
В операционной системе Windows Vista описанные выше риски устраняются за счет того, что сервисы, выполняемые в Сессии 0, изолированы и сама сессия не является интерактивной. В Windows Vista в Сессии 0 выполняются только системные процессы и сервисы. Первый пользователь, подключающийся к системе, загружается в Сессию 1, все последующие пользователи — в сессии с увеличивающими номерами. Это означает, что сервисы никогда не выполняются в той же сессии, что и пользовательские приложения и, таким образом, сервисы защищены от атак, которые могут происходить от вредоносных приложений.

Используя утилиту **Task Manager**, убедимся в том, что все сервисы Windows отделены от приложений и выполняются в Сессии 0.

Запустим утилиту **Task Manager**, выберем вкладку **Process** и включим отображение колонки **Session ID** (команда **View | Select Columns**). Активизируем опцию **Show processes from all users** для получения списка всех сервисов и убедимся в том, что они выполняются в отличной от приложений сессии — Сессии 0.



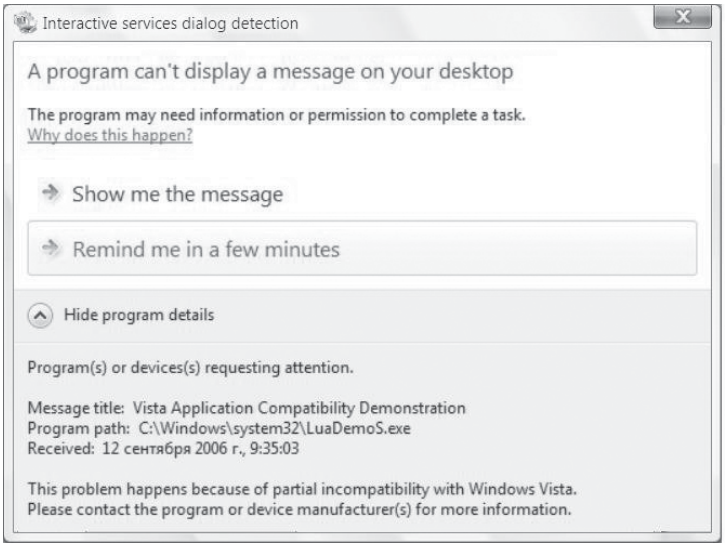
Сессии в Windows Vista и Windows Server 2008



Системные процессы выполняются в Сессии 0

Изоляция Сессии 0 может привести к проблеме с теми сервисами, которым необходимо отображение пользовательского интерфейса. Так как

сервис теперь выполняется в другой, по сравнению с **Desktop**, сессии, пользовательский интерфейс не будет виден конечным пользователям и, таким образом, интерактивный сервис может оказаться в зависшем состоянии.



Сообщение об обнаружении интерактивного сервиса

В Windows Vista решение этой проблемы состоит в том, что пользователям предоставляется возможность временного переключения в Сессию 0 для взаимодействия с интерактивным сервисом. Нажатие кнопки **Show me the message** на показанной выше диалоговой панели **Interactive service dialog detection** приведет к переключению в Сессию 0.

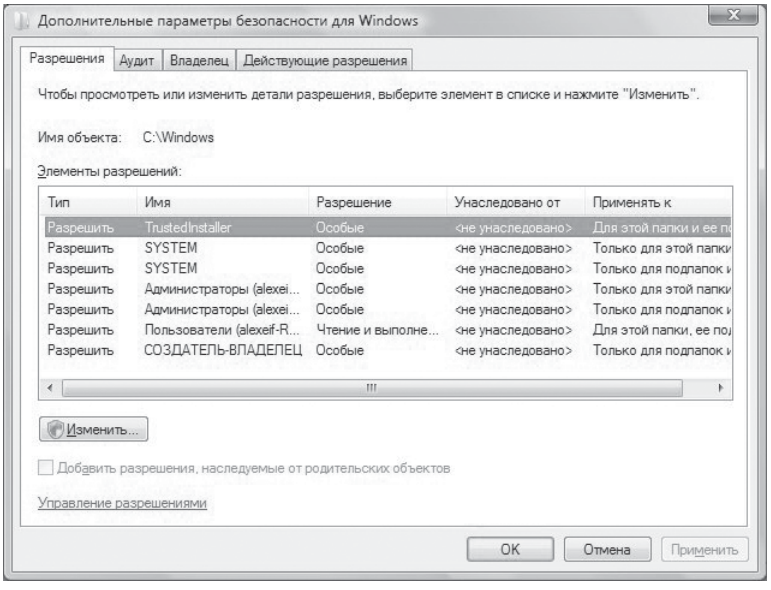
Windows Resource Protection

Механизм **Windows Resource Protection (WRP)** — в Windows XP он назывался **Windows File Protection** — предназначен для сохранения системы Windows в состоянии «только чтение» и, таким образом, позволяет повысить стабильность, надежность и предсказуемость системы. Этот механизм предохраняет системные файлы, папки и ключи реестра. Обновления защищенных этим механизмом ресурсов возможны только при использовании так называемых **OS Trusted Installers**, к которым относится, например, механизм **Windows Servicing**. Это позволяет защитить ключевые компоненты операционной системы от воздействия со стороны приложений и администраторов.

Механизм **Windows Resource Protection** распространяется на следующие компоненты операционной системы.

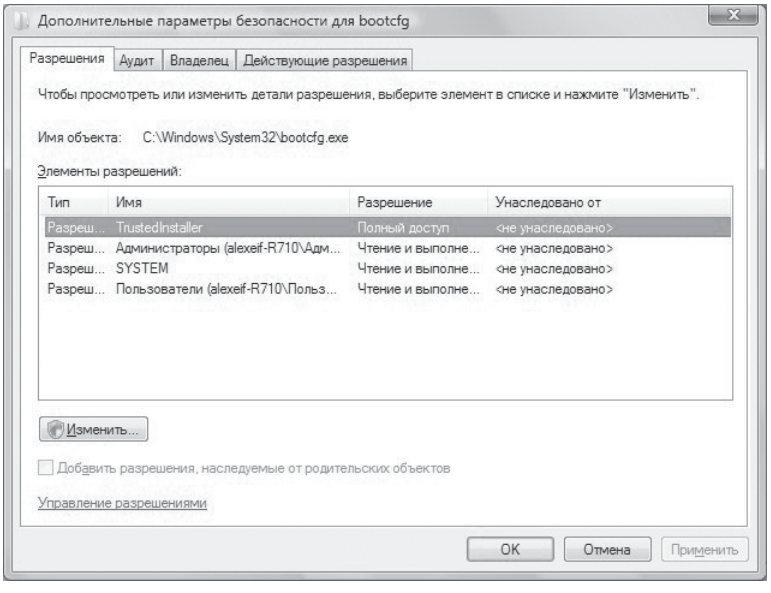
- Системные файлы с расширениями:
  - .acm, .ade, .adp, .app, .asa, .asp, .aspx, .ax
  - .bas, .bat, .bin
  - .cer, .chm, .clb, .cmd, .cnt, .cnv, .com, .cpl, .cpx, .crt, .csh
  - .dll, .drv, .dtd
  - .exe, .fxp, .grp
  - .h1s, .hlp, .hta
  - .ime, .inf, .ins, .isp, .its
  - .js, .jse, .ksh, .lnk
  - .mad, .maf, .mag, .mam, .man, .maq, .mar, .mas, .mat, .mau, .mav, .maw, .mda, .mdb, .mde, .mdt, .mdw, .mdz, .msc, .msi, .msp, .mst, .mui
  - .nls, .ocx, .ops
  - .pal, .pcd, .pif, .prf, .prg, .pst
  - .reg, .scf, .scr, .sct, .shb, .shs, .sys
  - .tlb, .tsp, .url
  - .vb, .vbe, .vbs, .vsmacros, .vss, .vst, .vsw
  - .ws, .wsc, .wsf, .wsh
  - .xsd, .xsl
- Ключи реестра, например:
  - HKEY_CLASSES_ROOT\Interface\{**GUID**}
  - HKEY_CLASSES_ROOT\Interface\{**GUID**}\NumMethods
  - HKEY_CLASSES_ROOT\Interface\{**GUID**}\ProxyStubClsid
  - HKEY_CLASSES_ROOT\Interface\{**GUID**}\ProxyStubClsid32
- Некоторые системные папки, например:
  - $(runtime.bootDrive)\inetpub\uddi\webroot\details\
  - $(runtime.bootDrive)\inetpub\uddi\webroot\edit\
  - $(runtime.bootDrive)\inetpub\uddi\webroot\controls\
  - $(runtime.bootDrive)\inetpub\uddi\bootstrap\

Механизм защиты ресурсов Windows работает на основе **Windows Discretionary Access Control List (DACL)** и **Access Control Lists (ACL)**. Возможность замены ресурсов операционной системы разрешена только для учетной записи типа **Windows Trusted Installer**; приложения и программы установки, пытающиеся заменить ресурсы, защищенные через WRP, получают сообщение об ошибке, — “Access Denied”. Ниже показаны права доступа для каталога Windows/System.



Права доступа для каталога Windows/System

Права доступа к отдельным компонентам операционной системы более жесткие — как видно из следующего рисунка, компонент под названием **bootcfg.exe** может модифицировать только учетная запись **Trusted Installer**; остальные учетные записи могут лишь читать данный файл и вызывать его на выполнение.



Права доступа для компонента операционной системы

В Windows Vista поддерживаются следующие механизмы замены системных ресурсов:

- **Windows Service Pack** (пакет обновления), устанавливаемый через механизм **Windows Trusted Installer**;
- **Hotfixes**, устанавливаемые через механизм **Windows Trusted Installer**;
- обновления операционной системы, устанавливаемые через механизм **Windows Trusted Installer**;
- отдельные обновления операционной системы, устанавливаемые через механизм **Windows Update**.

Операционная система поддерживает резервную копию файлов ядра системы, необходимых для перезапуска Windows Vista в специальном каталоге, называемом **Cache Directory** по адресу **%Windir%\winsxs\Backup**. Файлы, которые не требуются для перезапуска Windows Vista, в этот каталог не попадают. Размер каталога **Cache Directory** и число файлов в нем определяются ядром операционной системы и изменить эти значения нельзя.

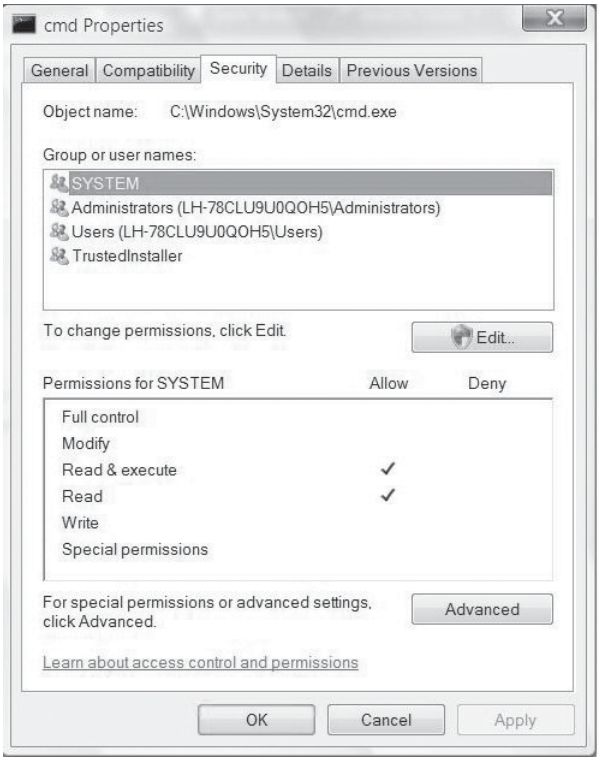
Утилита System File Check

Для проверки защищенных ресурсов операционной системы и их версий можно использовать утилиту **System File Checker (%windir%\system32\sfc.exe**), которая также известна под названием **Windows Resource Checker**. Ниже показаны параметры командной строки для данной утилиты.

```
Microsoft Windows [Версия 6.0.6001]
(C) Корпорация Майкрософт, 2006. Все права защищены.

C:\Windows\system32>sfc

Проверка ресурсов, версия 6.0, Microsoft(R) Windows(R).
(C) Корпорация Майкрософт, 2006. Все права защищены.

Проверка целостности всех защищенных системных файлов и замена неправильных
версий правильными.

SFC [/SCANNOW] [/VERIFYONLY] [/SCANFILE=<Файл>] [/VERIFYFILE=<Файл>]
    [/OFFWINDIR=<автономная папка Windows> /OFFBOOTDIR=<автономная папка загрузки>]

/SCANNOW        Проверка целостности всех защищенных системных файлов и
                восстановление, по возможности, проблемных файлов.
/VERIFYONLY     Проверка целостности всех защищенных системных файлов.
                Восстановление файлов не выполняется.
/SCANFILE       Проверка целостности указанного файла и его восстановление при
                обнаружении проблем. В параметре <Файл> следует указать полный путь
/VERIFYFILE     Проверка целостности файла, полный путь к которому указан
                в параметре <Файл>.  Восстановление файлов не выполняется.
/OFFBOOTDIR     Расположение автономного каталога загрузки для автономного восстановления
/OFFWINDIR      Расположение автономного каталога Windows для автономного восстановления

например:

        sfc /SCANNOW
        sfc /VERIFYFILE=c:\windows\system32\kernel32.dll
        sfc /SCANFILE=d:\windows\system32\kernel32.dll /OFFBOOTDIR=d:\ /OFFWINDIR=d:\windows
        sfc /VERIFYONLY
```

Параметры командной строки для утилиты System File Checker

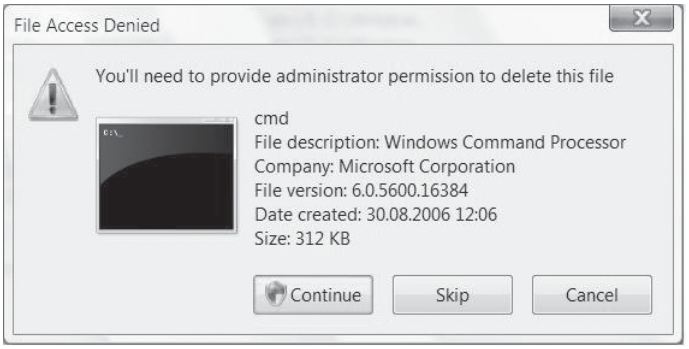
Далее мы рассмотрим, как файлы операционной системы защищены от модификаций. Перейдем в папку **\Windows\System32**, выберем утилиту **cmd.exe** и при нажатии правой кнопки мыши выберем команду **Properties**. На вкладке **Security** убедимся в том, что полный доступ к утилите не имеют ни учетная запись **System**, ни запись **Administrator**.



Контроль доступа к системному ресурсу

Теперь попробуем удалить утилиту **cmd.exe** — после подтверждения (кнопка **Yes**), диалоговой панели **File Access Denied** (кнопка **Continue**) и диалоговой панели **User Account Control** (кнопка **Continue**) мы получим сообщение о том, что данный файл удалить невозможно.

Как мы указали выше, только **Trusted Installers** могут модифицировать защищенные компоненты операционной системы.



Windows Resource Protection в действии

Определить, защищен ли файл механизмом Windows Resource Protection, можно следующими способами:

1. Из кода, используя функцию **SfcIsFileProtected**().
2. В Explorer проверить свойства файла:
   - открыть папку, в которой находится файл, свойства которого нас интересуют;
   - нажать правую кнопку мыши на интересующем нас файле и выбрать **Properties**.

Файл, который защищен механизмом WRP, покажет, что режим **Full Control** доступен только для учетной записи **Trusted Installer**. Для учетных записей **SYSTEM**, **Administrators** и **Users** доступен только режим **Read.**

Определить, защищен ли элемент реестра механизмом Windows Resource Protection, можно следующими способами.

3. Из кода, используя функцию **SfcIsKeyProtected**().
4. В Registry Editor проверить свойства файла:
   - нажать правую кнопку мыши на интересующем нас элементе реестра и выбрать **Properties**;
   - элемент, который защищен механизмом WRP, покажет, что режим **Full Control** доступен только для учетной записи Trusted Installer. Для учетных записей **SYSTEM**, **Administrators** и **Users** доступен только режим **Read**.

Создание объектов в области Global

Для создания объектов, видимых в глобальном пространстве имен (**Global Namespace**) — пространстве имен, которое доступно из всех сессий **Terminal Service Sessions**, пользователи должны иметь привилегию **SeCreateGlobalPrivilege**, которой не обладают учетные записи **Standard**

User. В предыдущих версиях операционной системы префиксы пространства имен типа **global** и **local** игнорировались, если не были активизированы терминальные сервисы. В Windows Vista режим **Fast User Switching** является одной из опций терминальных сервисов, сами сервисы всегда активизированы и упомянутые выше префиксы никогда не игнорируются. Приложения, которые могли создавать объекты в глобальном пространстве имен, теперь требуют выполнения только под учетной записью администратора. Ошибка возникает только когда приложения пытаются создавать такие объекты — доступ к глобальным объектам, созданным, например, системными сервисами, неограничен.

Рассмотрим пример, в котором создается проецируемый на память файл (**memory mapped file**) в глобальном пространстве имен. В результате выполнения приведенного ниже кода мы получим ошибку, т. к. стандартный пользователь не может создать такой файл. Вот код создания проецируемого на память файла:

```
[DllImport("kernel32", SetLastError = true, CharSet = CharSet.Auto)]
public static extern UIntPtr CreateFileMapping(
 UIntPtr hFile, UIntPtr lpAttributes, uint flProtect, uint
 dwMaximumSizeLow, uint dwMaximumSizeHigh, String lpName);

[DllImport("kernel32", SetLastError = true, CharSet = CharSet.Auto)]
public static extern bool CloseHandle( UIntPtr hHandle );

private void createSharedObject_Click(object sender, EventArgs e)
{
 string objName = "Global\" + "luademo_object";
 UIntPtr file = UIntPtr.Zero;

 UIntPtr handle = CreateFileMapping(
  file,                 //INVALID_HANDLE_VALUE
  UIntPtr.Zero,
  0x4,              // PAGE_READWRITE
  0,                // dwMaximumSizeHigh
  4,                // dwMaximumSizeLow
  objName);

 if (handle != UIntPtr.Zero)
 {
  CloseHandle(handle);
 }
 Win32Exception exc = new Win32Exception(Marshal.GetLastWin32Error());

 MessageBox.Show(
  "CreateFileMapping (" + "Global\" + " )\n\r\n\r" +
  "Result: "+ exc.Message,
  "Success: " + (handle != UIntPtr.Zero));
}
```

При выполнении такого кода мы получим сообщение об ошибке — Access Denied. Если заменить строку

```
string objName = "Global\" + "luademo_object";
```

на

```
string objName = "Local\" + "luademo_object";
```

то код выполнится без проблем. Для того чтобы приведенный выше код мог корректно выполняться, приложение следует запускать с опцией **Run as Administrator...**

В следующем разделе мы познакомимся с двумя включенными в состав операционной системы Microsoft Windows Vista механизмами обеспечения надежности приложений — Windows Feedback Platform и Restart Manager, а также кратко обсудим ряд других механизмов, появившихся в Windows Vista.

Механизмы обеспечения надежности приложений

Компонент операционной системы под названием **Windows Feedback Platform** служит для сбора информации о сбоях, произошедших в приложениях, и отсылки этой информации на специальный сайт. Ее проанализируют разработчики приложения и в случае обнаружения ошибок выпустят пакет обновлений, который станет доступным пользователю. Своевременное оповещение разработчиков об ошибках в приложениях позволяет существенно сократить время создания обновлений и, таким образом, повышает качество приложений.

Механизм **Restart Manager** служит для сохранения информации при внезапных сбоях, перезапусках приложений и в ряде других ситуаций и предоставляет разработчикам набор программных интерфейсов, использование которых в приложениях поможет сделать их более надежными.

Windows Feedback Platform

Windows Feedback Platform (WFP) — это дальнейшее развитие механизмов **Windows Error Reporting** (WER). В него включены все возможности **Windows Error Reporting**, а также ряд новинок, впервые появивших-

ся в Windows Vista. На высоком уровне процесс использования WFP выглядит следующим образом.

1. Пользователь работает с приложением.
2. В приложении внезапно происходит сбой, зависание или утечка памяти — все эти три состояния отслеживаются Windows Vista.
3. Данные, описывающие возникшие проблемы, отсылаются на специальный портал **WinQual**.
4. Разработчик приложения анализирует отосланные данные.
5. В приложение вносятся исправления, которые публикуются на портале.
6. Пользователь узнает о появлении исправленной версии через **Problem Center** в Windows Vista.

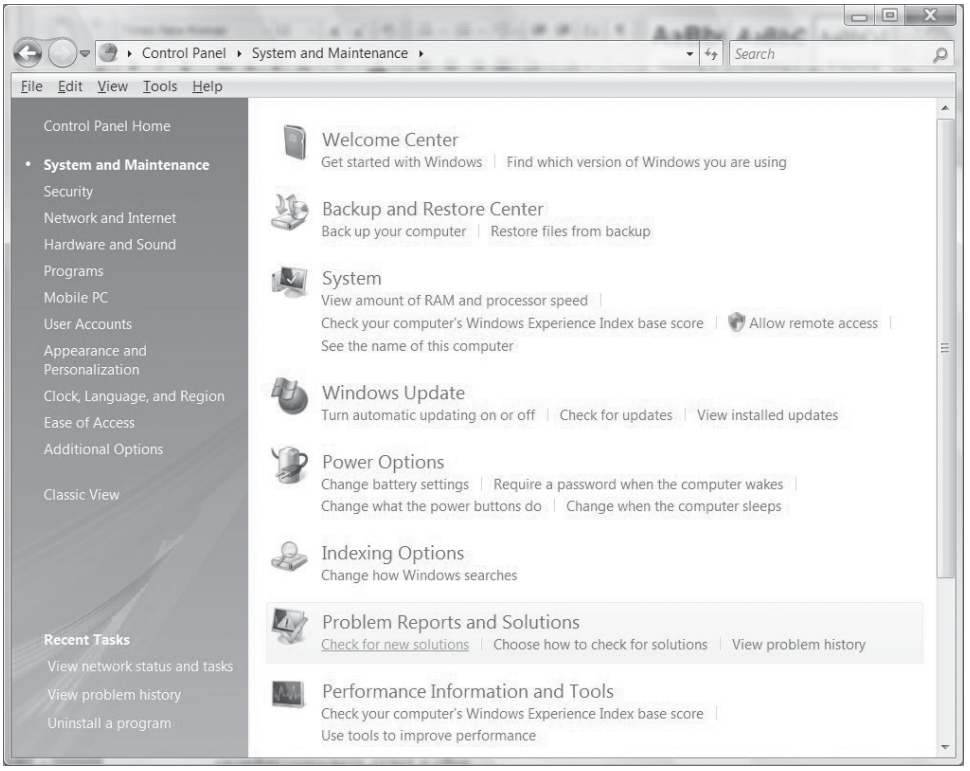
Как следует из описания процесса, он состоит из двух частей: в одной из них участвуют пользователи, а другая рассчитана на разработчиков. Полноценная отдача от механизма **Windows Feedback Platform** возможна только при использовании обеих частей.

Windows Feedback Platform на уровне пользователей

Пользователи не должны игнорировать предложения отослать отчет о сбое (кнопка **Send the Report**) — его необходимо отсылать при первом соединении с Интернетом, а затем периодически проверять наличие решения проблемы в специальном разделе **Control Panel**, который называется **System and Maintenance/Problem Center** (**Problem Reports and Solutions**). При поступлении решения проблемы следует активизировать соответствующую ссылку и выполнить указанные действия.

Пользователям предоставляется возможность сконфигурировать сервисы WFP и указать, нужно отсылать данные о сбоях или нет. Это возможно как при установке Windows Vista, так и через настройки **Group Policy**.

Новый компонент **Control Panel** в Windows Vista — **Problem Center** (**Problem Reports and Solutions**), доступный в разделе **System and Maintenance**, — отображает список проблем в хронологическом порядке, отосланные отчеты об ошибках и предоставляет доступ к решениям проблем по мере их поступления.

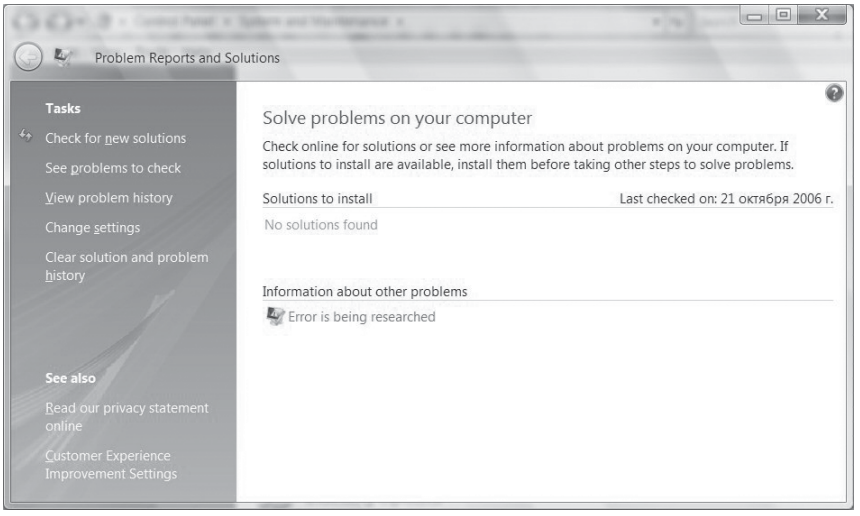


Раздел System and Maintenance в Control Panel

Problem Center можно вызывать из панели управления или как отдельную утилиту — в этом случае командная строка вызова будет выглядеть так: **%SystemRoot%\system32\wercon.exe**.

Как видно из следующего рисунка, информация в Problem Center делится на три группы: **Check for new solutions** (Искать новые решения), **Choose how to check for solutions** (Выбор способа проверки наличия решений) и **View problem history** (Просмотр истории сбоев).

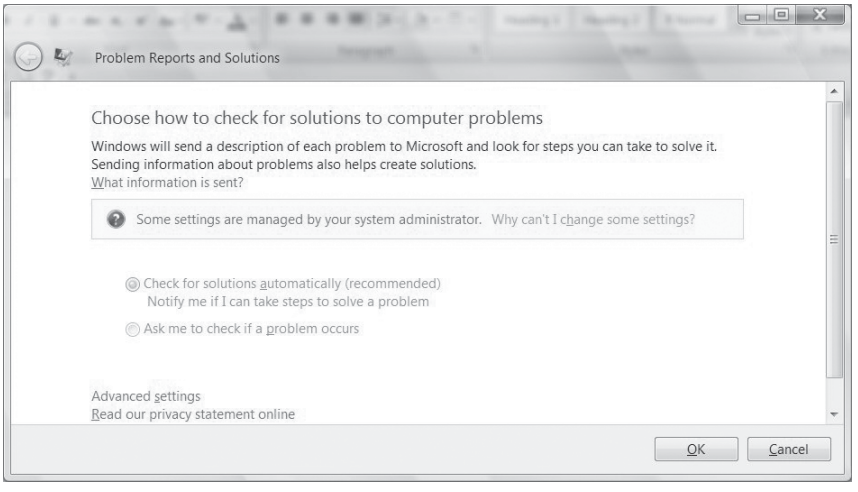
При открытии экрана поиска новых решений (см. рис. ниже) можно узнать, появились ли решения проблем, информация о которых была отослана ранее.



Раздел Check for new solutions

При наличии решений у пользователей появляется возможность обратиться к онлайновой службе (требуется подключение к Интернету) для получения дальнейших указаний.

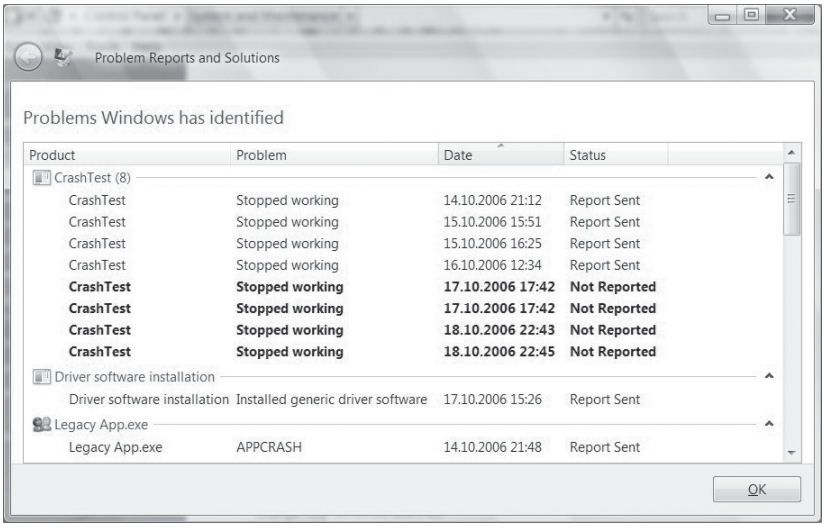
Раздел **Check for new solutions** позволяет выбрать между автоматическим поиском (рекомендуется включить эту опцию) и поиском после получения подтверждения от пользователя.



Раздел Choose how to check for solutions

В разделе расширенных настроек можно включить или вы ключить использование механизмов WFP, а также выбрать приложения, информацию о сбоях в которых отсылать не нужно.

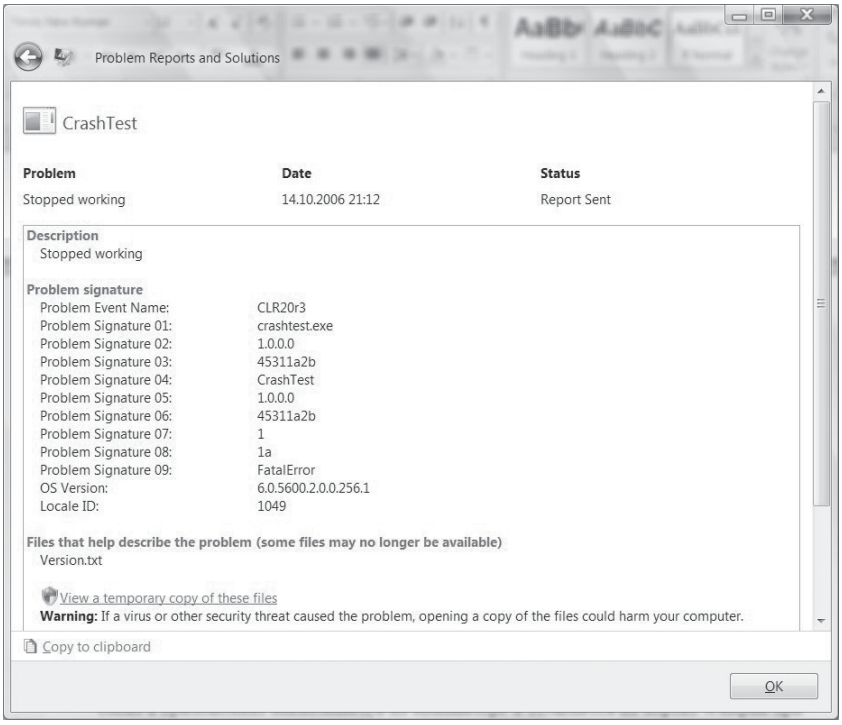
Раздел истории сбоев (см. рис. ниже) позволяет получить краткую информацию о сбоях, происходивших в системе. Отображаются приложение, в котором произошел сбой, тип сбоя (включая ошибки на уровне операционной системы), дата и время сбоя, а также статус — была отослана информация о сбое или нет.



Раздел View problem history

Каждая запись в протоколе ошибок может быть раскрыта для получения более подробной информации. Например, можно узнать, что именно было отослано в составе отчета.

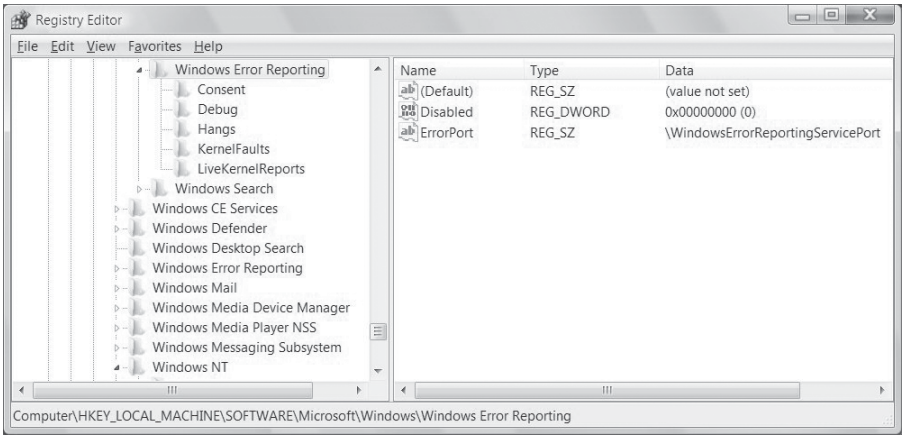
Как видно из следующего рисунка, никакой конфиденциальной информации в Microsoft не отправляется. В ряде случаев для решения конкретной проблемы с тем или иным приложением или с работоспособностью компонентов Windows может потребоваться дополнительная информация. В этом случае запрос на дополнительные данные будет отображен в соответствующей строке протокола.



Детальная информация о сбое

Как мы отметили выше, процесс обработки и исправления ошибок в приложениях состоит из двух частей. Задача пользователей — отсылать информацию о возникновении ошибок в приложениях и обращаться к Problem Center за возможными решениями. Задача разработчиков приложений — своевременно обрабатывать и анализировать данные о сбоях в приложениях и создавать обновления или «заплатки» к своим приложениям.

Для настройки механизмов **Windows Feedback Platform** можно использовать либо **Group Policy**, либо соответствующие ключи реестра (см. рис. ниже):

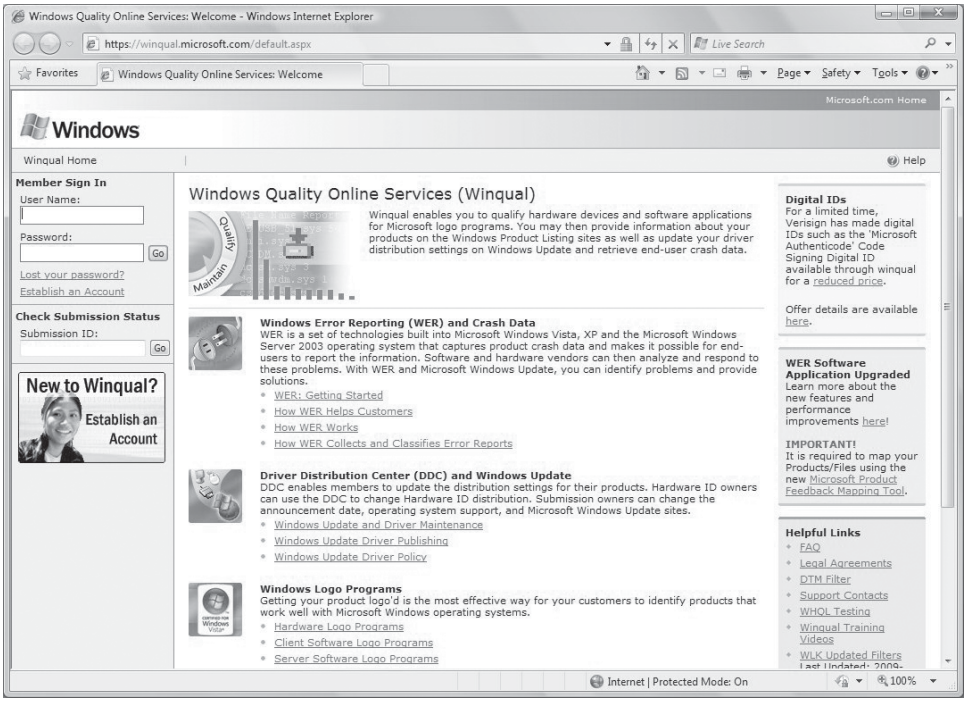


Ключи реестра, соответствующие Windows Feedback Platform

- **HKEY_CURRENT_USER\ Software\Microsoft\Windows\Windows Error Reporting**;
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Windows Error Reporting**.

Windows Feedback Platform на уровне разработчиков

Данные, собранные о приложении, в котором произошел сбой, отсылаются на специальный портал Windows Quality Online Services (см. рис. ниже), расположенный по адресу: **https://winqual.microsoft.com/**. Любая компания, производящая программное обеспечение, может бесплатно зарегистрироваться на этом портале и получать информацию о сбоях в своих приложениях. Единственная затрата — приобретение сертификата у компании Verisign (на момент написания данного обзора обсуждалась возможность снижения цены сертификата для компаний, использующих сервисы WinQal, с 399 до 99 долл.).



Страница Windows Quality Online Services

Программные интерфейсы Windows Feedback Platform

По умолчанию в процессе сообщения об ошибках и сбоях участвуют все приложения — при возникновении любой исключительной ситуации на уровне приложения операционная система генерирует соответствующий отчет. Обширный набор программных интерфейсов позволяет управлять различными аспектами создания и отсылки отчетов на сайт **WinQual**.

При необходимости приложение может быть отключено от механизмов WFP — для этого следует использовать функцию **WerAddExcludedApplication()**, указав ей в качестве параметра полное имя исполняемого файла приложения. Для отмены операции отключения приложения от механизмов WFP используется функция **WerRemoveExcludedApplication()**. Для выполнения этих функций приложение должно быть запущено под учетной записью администратора или получить соответствующие привилегии при запуске под другими учетными записями.

Помимо исключения приложения из процесса WFP, программные интерфейсы этого механизма позволяют управлять содержимым отчетов. Приложения могут включать в состав отчета новые объекты данных, а также изменять настройки отчетов по умолчанию. Такие изменения могут вно-

ситься динамически, в процессе работы приложения, что делает управление отчетами об ошибках максимально гибким.

Данные, отсылаемые на сайт WinQual, делятся на две группы. К пер вой группе относятся данные, используемые для идентификации и категоризации проблемы. Данные содержатся в наборе из десяти строчных параметров, которые описывают возникшую проблему (см. раздел Problem Signature на рис. «Детальная информация о сбое»). В режиме отчетов Standard или Simple значения этих параметров заполняются автоматически. При создании расширенных отчетов данные заносятся с помощью функции **WerReportSetParameter()**.

Обычный набор данных включает название приложения (например, **winword.exe**), версию приложения (например, 10.0.2627.0), имя модуля (например, **mso.dll**), версию модуля (например, 10.0.2613.1) и смещение внутри модуля (например, 0003cbb). Во все отчеты автоматически включается информация о языке и идентификаторе страны (**LocaleID**) и номере версии операционной системы.

Во вторую группу включаются данные, которые могут использоваться для отладки приложения и нахождения причины, вызвавшей сбой. Сюда могут входить файлы, мини-дамп памяти на момент сбоя, содержимое программной «кучи», содержимое реестра, запросы (к базе данных, WMI и т. п.) и пр., в зависимости от конкретного приложения и характера сбоя.

Используя программные интерфейсы WFP, можно включать в состав отчета два типа объектов — файлы и блоки памяти.

Для включения в состав отчета файла используется функция WerRegisterFile(), которой в качестве параметров передаются: полное имя файла, его тип (одно из значений **WER_REGISTER_FILE_ TYPE**) и два флага — **WER_ DELETE_FILE_WHEN_DONE,** указывающий на то, что файл должен быть удален после отсылки отчета, и **WER_ANONYMOUS_ DATA**, указывающий на то, что в файле не содержатся приватные данные. Возможные значения параметра **WER_REGISTER_FILE_ TYPE** приведены в следующей таблице.

Возможные значения параметра WER_REGISTER_FILE_TYPE

Значение	Описание
WerFileTypeHeapdump	Расширенный дамп памяти, содержащий дополнительные данные, например список процессов
WerFileTypeMicrodump	Дамп памяти, содержащий только трассировку стека
WerFileTypeMinidump	Дамп памяти
WerFileTypeOther	Любой другой тип файла
WerFileTypeUserDocument	Документ, использовавшийся в приложении в момент сбоя

Отметим, что задача генерации дампа памяти возлагается на разработчика приложения — для ее решения можно применять, например, отладочные механизмы, описанные в Windows SDK (см. функцию **MiniDumpWriteDump**).

Для исключения файла из отчета следует использовать функцию **WerUnRegisterFile()**, указав ей в качестве параметра имя исключаемого файла.

В большинстве сценариев отсылка дополнительных файлов происходит только при получении соответствующего запроса от сервера. В случае отсылки дополнительных файлов необходимо применять флаг **WER_ADD_ REGISTERED_DATA** при вызове функции **WerReportSubmit()** — о ней мы расскажем ниже.

Для включения в состав отчета копии области памяти используется функция **WerRegisterMemoryBlock()**, в качестве параметров которой передаются адрес начала включаемого блока памяти и размер этого блока в байтах (максимальный размер блока памяти — **WER_MAX_MEM_BLOCK_SIZE**). Для отмены включения копии области памяти в отчет следует применять функцию **WerUnRegisterMemoryBlock()**. В случае отсылки данных из памяти необходимо использовать флаг **WER_ADD_ REGISTERED_DATA** при вызове функции **WerReportSubmit()**.

Генерация и отсылка отчетов

Процесс генерации и отсылки отчета состоит из нескольких шагов. Инициализация отчета выполняется вызовом функции **WerReportCreate()**, с помощью которой указывается тип события, для которого создается отчет, тип отчета (**WerReportNonCritical** — для сбоев с возможностью восстановления и **WerReportCritical** — для сбоев, повлекших аварийное завершение приложения), ссылка на информацию, включаемую в отчет (см. структуру **WER_REPORT_INFORMATION**), и переменная, которая будет содержать ссылку на созданный отчет, — **ReportHandle**.

После того как отчет успешно инициализирован, необходимо добавить в него параметры первой и второй групп. Параметры первой группы задаются с помощью функции **WerReportSetParameter()**, которой передается ссылка на созданный отчет (результат успешного выполнения функции **WerReportCreate**), набор флагов, имя параметра и его значение (16-битная строка в Unicode, заканчивающаяся нулем).

Для включения в состав отчета дополнительных параметров применяется функция **WerReportAddSecondaryParameter()**, которой передается ссылка на отчет, имя параметра и его значение.

Помимо возможности включения в состав отчетов файлов и снимков областей памяти, предусмотрена передача в составе отчета и дампов памяти — для этого можно использовать функцию **WerReportAddDump()**,

в качестве параметров которой указываются ссылка на отчет, ссылки на процесс и поток, для которых был создан дамп, тип дампа (одно из значений **WER_DUMP_TYPE**), информация об исключении (указатель на структуру типа **WER_EXCEPTION_INFORMATION**), дополнительные опции (тип данных **WER_DUMP_CUSTOM_OPTIONS**) и флаги. Отметим, что процесс, для которого создается дамп, должен иметь права доступа **STANDARD_RIGHTS_READ** и **PROCESS_QUERY_INFORMATION**.

Для включения в состав отчета файлов мы применяем функцию **WerReportAddFile()**, которой передаем ссылку на отчет, полное имя файла, тип файла (**WER_FILE_ TYPE**) и дополнительные флаги.

Помимо этого разработчикам предоставляется возможность настройки пользовательского интерфейса — выбора информации, отображаемой в системной диалоговой панели. Для этих целей служит функция **WerReportSetUIOption()**, которой передается ссылка на отчет, тип интерфейса отчета (**WER_REPORT_UI**) и значение отображаемой строки. Приложение может модифицировать любое из полей интерфейсного элемента, заданного параметром **WER_REPORT_UI**; каждый вызов функции позволяет модифицировать только одно поле. Функция **WerReportSetUIOption()** может вызываться в любой момент работы приложения до непосредственной отсылки отчета.

После того как отчет сформирован и настроен, мы используем функцию **WerReportSubmit()** для отсылки отчета. В качестве параметров этой функции передаются ссылка на отчет, тип пользовательского интерфейса (наличие прав администратора, подтверждение отсылки и т. п.) и набор флагов. После того как отчет отослан, следует закрыть ссылку на него, используя функцию **WerReportCloseHandle()**.

Среди дополнительных функций, доступных для разработчиков, можно отметить **WerGetFlags()**, позволяющую узнать текущие настройки механизма отсылки отчетов об ошибках для указанного процесса.

Все вышеописанные функции, реализующие механизмы настройки Windows Feedback Platform, находятся в системных модулях **Kernel32.dll** и **Wer.dll**, а их прототипы описаны в заголовочном файле **werapi.h**, который входит в состав Windows SDK.

Дополнительная информация по механизмам Windows Feedback Platform доступна на сайте Microsoft по адресу: **http://msdn.microsoft.com/isv/resources/wer/default.aspx**.

Restart Manager

Выше мы начали рассмотрение ряда механизмов обеспечения надежности приложений, реализованных в новой версии операционной системы Microsoft — Windows Vista. Мы обсудили механизм **Windows Feedback**

Platform, позволяющий компаниям-разработчикам централизованно получать данные о сбоях в приложениях и на основе анализа этих данных выпускать обновления, которые будут распространяться среди пользователей с помощью средств, реализованных в Windows Vista. Еще один механизм обеспечения надежности приложений — это набор программных интерфейсов и компонентов ядра операционной системы, известный под названием Restart Manager. Данный механизм может использоваться как программами установки (инсталляторами) для снижения необходимости перезагрузок операционной системы за счет отслеживания занятых приложениями ресурсов и перезапуска приложений, так и непосредственно прикладными программами для обеспечения возможности восстановления после сбоев и восстановления данных.

Механизм Restart Manager

В основе механизма Restart Manager лежат две функции. Вызов функции **RegisterApplicationRestart()** позволяет вашему приложению перезапуститься после сбоя и отослать отчет о произошедшем сбое (используя рассмотренные ранее механизмы Windows Feedback Platfrom), таким образом обеспечивая пользователю возможность продолжить работу. Вызов еще одной функции из состава Restart Manager — **RegisterApplicationRecoveryCallback()** — позволит вам указать ядру Windows Vista, какую функцию вашего приложения нужно вызывать перед непосредственным перезапуском приложения, следовательно, у вас появляется возможность сохранения данных с их последующим восстановлением после перезапуска приложения.

При вызове функции **RegisterApplicationRestart()** указывается командная строка, применяемая для повторного запуска приложения, таким образом, используя опции командной строки, параметры или другие способы, вы можете указать приложению на то, что оно запускается после сбоя, и инициировать процесс восстановления данных.

Функция **RegisterApplicationRecoveryCallback()** задает точку входа в приложение, которая вызывается ядром операционной системы после сбора данных, необходимых для генерации отчета о произошедшем сбое. При получении управления приложение должно попытаться сохранить данные на диске. В процессе сохранения данных необходимо вызывать функцию **RecoveryInProgress()** приблизительно каждые 5 с для того, чтобы операционная система помнила о том, что приложение находится в процессе сохранения данных, в противном случае ядро операционной системы сочтет приложение зависшим и принудительно завершит его выполнение. Это необходимо в тех случаях, когда попытка сохранения данных приводит к дополнительному сбою в приложении и появляется возможность «зацикливания» обработки сбоев. По завершении сохранения данных вызывается функция **RecoveryFinished()**.

Сигналом для завершения работы Windows-приложения является получение сообщений **WM_QUERYENDSESSION** и **WM_ENDSESSION** со значением параметра **LPARAM**, равным **ENDSESSION_CLOSEAPP (0x1)**.

Консольные приложения должны проверять нажатие комбинации клавиш **Ctrl+C**. Ниже показан пример обработчиков событий для Windows-приложения и консольного приложения.

```
//----------------------
// Windows-приложение
//----------------------
...
hr = RegisterApplicationRestart(CommandLineParameter, NULL);
...
switch(message)
{
 case WM_QUERYENDSESSION:
 {
 //
 // Принудительное завершение приложения
 //
  if(lParam & ENDSESSION_CLOSEAPP)
  {
    //Сохранение данных, состояния приложения
    hr = SaveData();
  }
  // Выход из обработчика событий
  return 1;
 }
case WM_ENDSESSION:
 ...
//----------------------
// Консольное приложение
//----------------------
BOOL ControlHandlerRoutine(DWORD ControlEvent)
{
switch(ControlEvent)
{
//
// Принудительное завершение приложения
//
case CTRL_C_EVENT:
{
TerminateRequest = TRUE;
return FALSE;
}
...
Соответственно код, сохраняющий данные, может выглядеть так:
HRESULT SaveData()
{
// Создать временный файл
uReturnValue = GetTempFileName(PathBuffer,"~rm",0,RecoveryFile);
```

```
// Создать файл для сохранения данных
FileHandle = CreateFile((LPTSTR)RecoveryFile,...);
// Получить размер буфера редактора
TextBufferLength = GetWindowTextLength(EditControlHwnd);
// Скопировать текст из редактора во временную строку
GetWindowText(EditControlHwnd, TextBuffer, TextBufferLength+1);
// Сохранить содержимое строки в файле
Result = WriteFile(FileHandle, TextBuffer,
TextBufferLength+1, &NumberOfBytesWritten, NULL);
}
```

Restart Manager и программы установки приложений

Как мы уже отмечали, механизм Restart Manager должен использоваться программами установки приложений для предотвращения лишних перезагрузок операционной системы. Указанная функциональность базируется на нескольких функциях, реализованных в рамках механизма Restart Manager; (см. табл.).

Функции, реализованные в рамках механизма Restart Manager

Функция	Описание
RmStartSession	Начинает новую сессию
RmRegisterResources	Регистрирует ресурсы в Restart Manager
RmGetList	Возвращает список приложений и сервисов, использующих зарегистрированные ресурсы
RmShutdown	Завершает работу приложений и сервисов для освобождения занятых ими ресурсов
RmRestart	Перезапускает завершенные приложения или сервисы
RmEndSesion	Завершает сессию

Работа с Restart Manager начинается с создания новой сессии через вызов функции **RmStartSession()** — все последующие операции выполняются в рамках этой сессии. Для одной учетной записи поддерживается до 64 одновременно открытых сессий Restart Manager.

При необходимости расширенные скрипты инсталляционного пакета (**custom actions**) могут подключиться к уже существующей сессии — для этого применяется функция **RmJoinSession()**. После того как сессия создана, необходимо зарегистрировать ресурсы в рамках данной сессии с помощью функции **RmRegisterResources()**. К ресурсам относятся файлы, описываемые их полными именами, а также процессы, идентифицируемые через PID и время создания процесса, и сервисы, описываемые их именами. Кроме того, имеется возможность регистрации ресурсов, описанных структурой **RM_UNIQUE_PROCESS**.

Функция **RmGetList()** возвращает список приложений и сервисов, применяющих зарегистрированные ресурсы в виде массива структур **RM_PROCESS_INFO**. Она базируется на расширенной функциональности Windows Vista/Longhorn Server, позволяющей определить процессы, использующие те или иные файлы. Эта функциональность позволяет определить DLL, файлы данных и файлы, отображаемые в области памяти (**memory mapped files**). Также имеется возможность идентифицировать сервисы внутри службы svchost, применяющие необходимые нам файлы. Процессы подразделяются на «видимые» GUI-приложения, «невидимые» GUI-приложения, сервисы, консольные приложения, Windows Explorer, критичные процессы и процессы неизвестного типа. Типы процессов описаны структурой **RM_APP_TYPE** (см. табл.).

Типы процессов, описанные структурой RM_APP_TYPE

Тип	Код	Описание
RmUnknownApp	0	Приложение не может быть классифицировано
RmMainWindow	1	Windows-приложение в отдельном процессе с главным окном
RmOtherWindow	2	Windows-приложение без отдельного процесса и главного окна
RmService	3	Сервис Windows
RmExplorer	4	Windows Explorer
RmConsole	5	Консольное приложение
RmCritical	1000	Процесс, критичный для Windows

Кроме того, функция **RmGetList()** позволяет автоматически определить необходимость в перезагрузке процесса.

Функция **RmShutdown()** использует те же нотификационные механизмы и протоколы, что и системная функция завершения процессов:

- Windows-приложения получают сообщение **WM_QUERYENDSSION** и сообщение **WM_ENDSESSION** с параметром **LPARAM** со значением **ENDSESSION_CLOSEAPP**;
- Windows-приложения, написанные для предыдущих версий операционной системы, получают сообщение **WM_CLOSE**;
- сервисы завершаются через команды **Service Control Manager**. При этом учитываются зависимости, которые могут существовать и между сервисами;
- консольные приложения получают сообщение **CRTL_C_EVENT**.

Приложения, идентифицированные как критичные для обеспечения работоспособности операционной системы, не могут быть завершены

принудительно. Среди возможных опций отметим возможность принудительного завершения приложения и сервисов, которые завершились с ошибками, а также возможность завершения только приложений, зарегистрированных в Restart Manager.

Перезапуск приложений и сервисов выполняется с помощью функции **RmRestart()**, повторно запускающей приложения и сервисы, работа которых была завершена с помощью функции **RmShutdown()**. Windows-приложения и консольные приложения перезапускаются посредством командной строки, указанной при их регистрации с помощью функции **RegisterApplicationRestart()**. Сервисы перезапускаются с помощью **Service Control Manager**; все сервисы, которые зависели от перезапускаемого сервиса и были принудительно завершены вместе с ним, также перезапускаются. Приложения, которые поддерживают автоматическое сохранение своего состояния и данных (например, приложения, входящие в состав Microsoft Office 2007), автоматически восстанавливают свое состояние. Для перезапуска приложений после рестарта операционной системы используется функция **InitiateShutdown()** с флагом **SHUTDOWN_RESTARTAPPS** — ее действие распространяется только на приложения, зарегистрированные в **Restart Manager**.

Открытая сессия Restart Manager завершается вызовом функции **RmEndSesion()**.

Программа установки приложений **Microsoft Windows Installer** (**MSI**) версии 4.0 поддерживает механизмы **Restart Manager** автоматически. Для программ установки, создаваемых собственными средствами, потребуется использование описанных ранее функций. Приведем пример такого применения функций:

```
// Начать новую сессию
RmStartSession(&dwSessionHandle, sessKey);
// Зарегистрировать элементы, которые должны быть
// установлены, заменены, обработаны и т.п.
RmRegisterResources(dwSessionHandle,
nFiles, rgsFiles,      // Файлы
nProcs, NULL,          // Процессы
nServices, rgsServices // Сервисы
);
// Получить список приложений и сервисов, которые используют
// ранее зарегистрированные файлы
RmGetList(dwSessionHandle, &nProcInfoNeeded,
&nAffectedApps, rgAffectedApps, &bRebootNeeded);
// Завершить работу приложений и сервисов, которые используют
// нужные нам файлы
RmShutdown(dwSessionHandle, 0, NULL);
// Выполнение операций над устанавливаемыми файлами
// Перезапуск приложений, которые мы принудительно завершили
RmRestart(dwSessionHandle, NULL);
```

Выше мы рассмотрели основные функции, относящиеся к механизмам **Restart Manager**. Помимо этого есть ряд дополнительных функций, которые приведены в следующей таблице.

Дополнительные функции, относящиеся к механизмам Restart Manager

Функция	Описание
RmAddFilter	Добавляет фильтр для компонентов, которые должны быть завершены и перезапущены
RmRemoveFilter	Удаляет ранее установленный фильтр
RmGetFilterList	Возвращает список установленных фильтров
RmCancelCurrentTask	Прерывает текущую операцию Restart Manager
RM_WRITE_STATUS_CALLBACK	Косвенно вызываемая функция для обновления статуса выполняемой операции

Примеры

В состав Windows Vista SDK включен пример использования механизмов Windows Error Reporting, в котором показано, как зарегистрировать приложения для его последующего восстановления, приложение для его перезапуска, а также как зарегистрировать файл и блок памяти. Данный пример можно найти по адресу: **c:\Program Files\Microsoft SDKs\Windows\v6.0\ Samples\winbase\WindowsErrorReporting\Registration**.

Второй пример, относящийся к теме данного обзора, иллюстрирует использование механизмов Restart Manager. Он находится по адресу: **c:\Program Files\Microsoft SDKs\Windows\v6.0\Samples\winbase\Restart Manager**. В этом каталоге собрано пять демонстрационных приложений, которые иллюстрируют минимальную поддержку механизмов Restart Manager для консольных приложений (**RmCuiApp**), применение Restart Manager Filter API (**RMFilterApp**), сериализацию данных и их восстановление после перезагрузки в классическом Windows-приложении (**RmGuiApp**), аналогичную функциональность для приложений, разрабатываемых на .Net Framework с использованием Windows Forms (**RmWinFormApp**), и пример блокировки при попытке принудительного завершения приложения (**ShutdownBlockReasonTestApp**).

Дополнительные механизмы обеспечения надежности

В данном разделе мы рассмотрели два механизма обеспечения надежности приложений, реализованных в операционной системе Microsoft Windows Vista, — **Windows Feedback Platform** и **Restart Manager**. К другим механизмам, обеспечивающим надежность как самой платформы, так и выполняемых под ее управлением приложений, можно отнести следующие:

- отмена выполнения операций ввода-вывода (**Cancellable I/O Operations**) — обеспечивает возможность завершения запросов на операции ввода-вывода, которые могут привести к повышенному использованию недоступных в данный момент ресурсов. Примерами новых функций являются **CancelSynchronousIo()** и **CancelIoEx()**. Отметим, что применение механизмов отмены операций ввода-вывода позволяет решить ряд проблем с такими операциями без принудительного завершения потоков и приложений;
- защита реестра — предотвращает возможность изменения ключевых настроек системы;
- определение утечек памяти — автоматически определяет утечки памяти и реагирует на это соответствующим образом;
- определение зависаний приложений — позволяет принудительно завершить и перезапустить приложения, которые перестали реагировать на действия пользователей, сообщений **Windows Messages** и т. п.;
- инфраструктура **Windows Diagnostic Infrastructure** (**WDI**) — служит для идентификации и выдачи сообщений об обнаруженных проблемах.

Встроенные в Windows Vista механизмы автоматической диагностики позволяют идентифицировать ряд проблем, которые могут возникать в процессе эксплуатации как самой операционной системы, так и приложений. К таким механизмам относятся: защита от чрезмерного использования ресурсов (например, от потерь данных, зависаний, сбоев и т. п. при большом количестве одновременно открытых приложений), защита от аппаратных сбоев (диски, дефекты оперативной памяти), встроенная диагностика функционирования проводных и беспроводных сетей, диагностика производительности системы (замедление реакции системы и визуализации элементов интерфейса, замедление загрузки системы, аутентификации и завершения работы), автоматическое восстановление поврежденных системных файлов, автоматическое восстановление системы при проблемах с загрузкой.

Windows 7: рекомендации по обеспечению совместимости приложений

Новая версия клиентской операционной системы — Microsoft Windows 7 — продолжает развитие технологий, появившихся в Windows Vista. Для того, чтобы приложения могли корректно работать под управлением Windows 7, помимо рекомендаций, приведенных в данном руководстве, следует обратить внимание на ряд изменений, которые произошли в операционной системе. К таким изменениям относятся:

- версия операционной системы;
- новая версия Internet Explorer;
- отсутствие компонента Windows Mail;
- использование библиотек;
- отсутствие сервиса Windows Registry Reflection;
- отсутствие драйвера WPDUSB.SYS для Windows Portable Devices;
- новые бинарные компоненты операционной системы;
- работа Internet Explorer в режиме DEP;
- изменения в MSMQ.

Рассмотрим каждое из перечисленных изменений и возможные проблемы более подробно. Начнем с версии операционной системы.

Версия операционной системы

Версия операционной системы Windows 7 и Windows Server R2 — **6.1**. Функция **GetVersionEx()** возвращает именно этот номер (поле **dwMajorVersion** содержит 6, а поле **dwMinorVersion** — 1). Все рассмотренные выше рекомендации по правильной проверке номера версии применимы и в случае выполнения кода под управлением Windows 7 и Windows Server R2.

```
C:\Users\alexeif>ver
Microsoft Windows [Version 6.1.7000]
```

Работа команды Ver в Windows 7

В общем случае проверка версии операционной системы должна производиться следующим образом (приведен пример на C#):

```
static private bool IsOSSupported()
{
 OperatingSystem os = Environment.OSVersion;
 Version vs = os.Version;
```

```
 if ((vs.Major == 5) && (vs.Minor >= 1) || vs.Major >= 6 )
 {
  return true;
 }
```

Вот пример проверки версии операционной системы на неуправляемом коде:

```
#include <windows.h>
#include <stdio.h>

void main()
{
    OSVERSIONINFO osvi;
    BOOL bIsWindowsXPorLater;

    ZeroMemory(&osvi, sizeof(OSVERSIONINFO));
    osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

    GetVersionEx(&osvi);

    bIsWindowsXPorLater =
       ( (osvi.dwMajorVersion > 5) ||
       ( (osvi.dwMajorVersion == 5) && (osvi.dwMinorVersion >= 1) ));

    if(bIsWindowsXPorLater)
        printf("The system meets the requirements.\n");
    else printf("The system does not meet the requirements.\n");
}
```

Рекомендуется проверять не версию операционной системы, а ее издания (и наличие соответствующих компонентов), используя функции **GetSystemMetrics()**, **VerifyVersionInfo()** или **SystemParametersInfo()**. Для проверки наличия в текущей версии операционной системы какого-либо программного интерфейса следует использовать функцию **LoadLibrary()** для загрузки соответствующей динамической библиотеки и функцию **GetProcAddress()** для определения адреса интересующей функции. Если функция не найдена, **GetProcAddress()** возвращает код ошибки **ERROR_CALL_NOT_IMPLEMENTED**.

Дополнительно:

- **Функция GetversionEx()**
  - http://msdn.microsoft.com/en-us/library/ms724451(VS.85).aspx
- **Функция GetSystemMetrics()**
  - http://msdn.microsoft.com/en-us/library/ms724385(VS.85).aspx
- **Функция VerifyVersionInfo()**
  - http://msdn.microsoft.com/en-us/library/ms725492(VS.85).aspx
- **Функция SystemParametersInfo()**
  - http://msdn.microsoft.com/en-us/library/ms724947(VS.85).aspx

- **Функция LoadLibrary()**
  - http://msdn.microsoft.com/en-us/library/ms684175(VS.85).aspx
- **Функция GetProcAddress()**
  - http://msdn.microsoft.com/en-us/library/ms683212(VS.85).aspx

Новая версия Internet Explorer

В новой версии операционной системы появится и новая версия Internet Explorer — Internet Explorer 8. Соответственно, строка User Agent, которую веб-сервера используют для определения версии браузера, будет включать новые данные. Вот как выглядит строка User Agent для Internet Explorer 8, работающего в «родном» режиме:

```
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
```

В режиме совместимости с Internet Explorer 7 строка User Agent выглядит так:

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
```

Обратите внимание на то, что элемент «**Trident/4.0**» позволяет определить, что даже в режиме совместимости браузером является Internet Explorer 8.

Веб-приложения, которые проверяют версию браузера, должны корректно обрабатывать изменение версии браузера. Для включения режима совместимости следует использовать мета-тэги либо на уровне страниц, либо на уровне сайтов или включать соответствующие команды в заголовки HTTP-запросов.

Режим совместимости можно указать следующим образом (на уровне страницы):

```
<html>
<head>
  <!-- Mimic Internet Explorer 7 -->
  <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
  <title>My Web Page</title>
</head>
<body>
  <p>Content goes here.</p>
</body>
</html>
```

или на уровне HTTP-протокола (для ASP.NET):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.webServer>
    <httpProtocol>
```

```
      <customHeaders>
        <clear />
        <add name="X-UA-Compatible" value="IE=EmulateIE7" />
      </customHeaders>
    </httpProtocol>
  </system.webServer>
</configuration>
```

В коде режим совместимости можно определить так:

```
engine = null;
if (window.navigator.appName == "Microsoft Internet Explorer")
{
   // This is an IE browser. What mode is the engine in?
   if (document.documentMode) // IE8
      engine = document.documentMode;
   else // IE 5-7
   {
      engine = 5; // Assume quirks mode unless proven otherwise
      if (document.compatMode)
      {
         if (document.compatMode == "CSS1Compat")
            engine = 7; // standards mode
      }
   }
// the engine variable now contains the document compatibility mode.
}
```

Дополнительная информация:

- Understanding User Agent Strings
  - **http://msdn.microsoft.com/en-us/library/ms537503(VS.85).aspx**
- The Internet Explorer 8 User-Agent String
  - **http://blogs.msdn.com/ie/archive/2008/02/21/the-internet-explorer-8-user-agent-string.aspx**
- User-Agent String and Version Vector
  - **http://code.msdn.microsoft.com/ie8whitepapers/Release/ProjectReleases.aspx?ReleaseId=531**
- Hosting and Reuse
  - **http://msdn.microsoft.com/en-us/library/aa752038(VS.85).aspx**

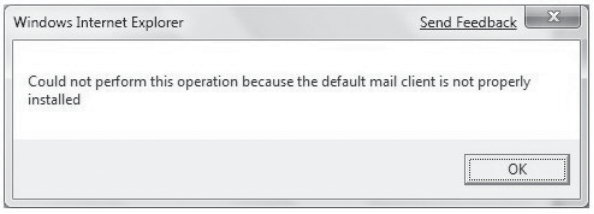
Отсутствие компонента Windows Mail

Начиная с Windows 7 компонент операционной системы **Windows Mail** (Outlook Express) заменен на **Windows Live Mail** — при необходимости можно использовать и любой другой почтовый клиент стороннего производителя. С точки зрения программных интерфейсов отсутствие влечет **Windows Mail** за собой отмену действия интерфейса **CoStartOutlookEx-**

press — все остальные программные интерфейсы продолжают функционировать.

Отсутствие этого компонента сказывается на следующем:

- все точки входа для **Windows Mail** и **Contacts** (например, меню **Start**, ссылки, созданные пользователями, команда **Start | Run** и т. п.) удалены и недоступны;
- все динамически загружаемые библиотеки (dll) по-прежнему, входят в состав операционной системы;
- программные интерфейсы, описанные в документации и не помеченные как «deprecated», продолжают работать также, как и в Windows Vista;
- программные интерфейсы, вызывающие Windows Mail, не выполняют никаких действий;
- обработчики протоколов (mailto, ldap, news, snews, nntp) не ассоциированы с Windows Mail или Contacts. При попытке запуска пользователи перенаправляются на страницу настроек, где им предлагается ассоциировать эти протоколы с установленными почтовыми клиентами (**Programs | Default Programs | Set Associations**);



Сообщение об отсутствии WinMail

- ассоциации с файлами (.eml, .nws, .contact, .group, .wab, .p7c, .vcf) либо не созданы, либо направлены в программу **Windows Contacts**. При попытке открытия файлов с указанными расширениями пользователям предлагается использовать другие приложения;
- функции предварительного просмотра файлов удалены, ссылки на рабочем столе отключены и не функционируют механизмы поиска по файлам.

Для исправления возможных ошибок следует установить **Windows Live Mail** или другую почтовую программу, которая позволит читать файлы с расширениями .eml и .nws. В приложениях не следует использовать функции **Windows Mail UI** и при создании новых версий приложений следует вообще отказаться от использования функций **Windows Mail**.

Использование библиотек (Library)

Новый интерфейсный элемент — Library представляет собой виртуальное объединение папок и с точки зрения операционной системы выглядит как файл (и как папка для пользователя). При использовании функций **IFileDialog->GetFolder()** и **IFileDialog->GetFilename()** для работы с библиотеками функция **GetFolder()** возвращает имя файла, а не папки. Для исправления этой ситуации следует использовать функцию **IFileDialog->GetResult()**.

Подробнее о библиотеках см. документацию по интерфейсу **IShellLibrary** в MSDN или Windows SDK по адресу http://msdn.microsoft.com/en-us/library/dd391719(VS.85).aspx. Данный интерфейс содержит методы для создания библиотек и управления ими. Для работы с Library из управляемого кода следует использовать библиотеку **Windows Bridge**, которая входит в состав Windows SDK или распространяется отдельно на сайте MSDN Code Gallery — http://code.msdn.microsoft.com/VistaBridge.

Отсутствие сервиса Windows Registry Reflection

Процесс, известный под названием «**Windows Registry Reflection**», копирует ключи реестра и их значения между двумя представлениями реестра и поддерживает синхронизацию между ними. В предыдущих версиях Windows 64 этот процесс создавал копии подмножества перенаправленных ключей реестра для 32- и 64-битных представлений. Начиная с версии Windows 7 механизмы синхронизации реестра выключены, т. к. единственным реальным потребителем этой функциональности была подсистема COM, которая в Windows 7 была обновлена таким образом, чтобы не зависеть от перенаправлений ключей реестра.

В ряде случаев приложения могут столкнуться с несоответствием данных в 32- и 64-битных представлениях реестра. К ключам реестра, где это может проявляться, относятся:

- **HKEY_LOCAL_MACHINE\Software\Classes**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\COM3**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\EventSystem**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Ole**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc**
- **HKEY_USERS*\Software\Classes**
- **HKEY_USERS*_Classes**

Обратите внимание на тот факт, что в Windows 7, за исключением приведенных ниже ключей реестра, все ключи объединены в единое пространство, доступное как из 32-, так и из 64-битных версий приложений.

- **HKEY_LOCAL_MACHINE\Software\Classes\CLSID**
- **HKEY_LOCAL_MACHINE\Software\Classes\Interface**

Для исправления возможных несовместимостей следует использовать один из следующих подходов:

- использовать только неперенаправленные ключи;
- использовать опцию **KEY_WOW64_64KEY** для доступа к реестру — таким образом и 32- и 64-битные приложения будут использовать только 64-битные ключи;
- использовать данные только из рекомендованных ключей реестра.

Дополнительная информация:

- Registry Reflection
  - **http://msdn.microsoft.com/en-us/library/aa384235(VS.85).aspx**
- Redirected keys list within Registry Redirector
  - **http://msdn.microsoft.com/en-us/library/aa384232(VS.85).aspx**
- Best Practices for Wow64
  - **http://www.microsoft.com/whdc/system/platform/64bit/WoW64_bestprac.mspx**

Отсутствие драйвера WPDUSB.SYS для Windows Portable Devices

В Windows 7 на смену драйверу wpdusb.sys пришел драйвер winusb.sys. Те приложения, которые использовали программные интерфейсы **Windows Portable Devices** (WPD), продолжают работать без каких-либо изменений, проблемы могут возникнуть в тех случаях, когда в коде использовались недокументированные коды IOCTL — в этом случае рекомендуется воспользоваться документированными функциями **Windows Portable Devices** (WPD).

Дополнительная информация:

- Windows Portable Devices
  - **http://msdn.microsoft.com/en-us/library/ms740786(VS.85).aspx**

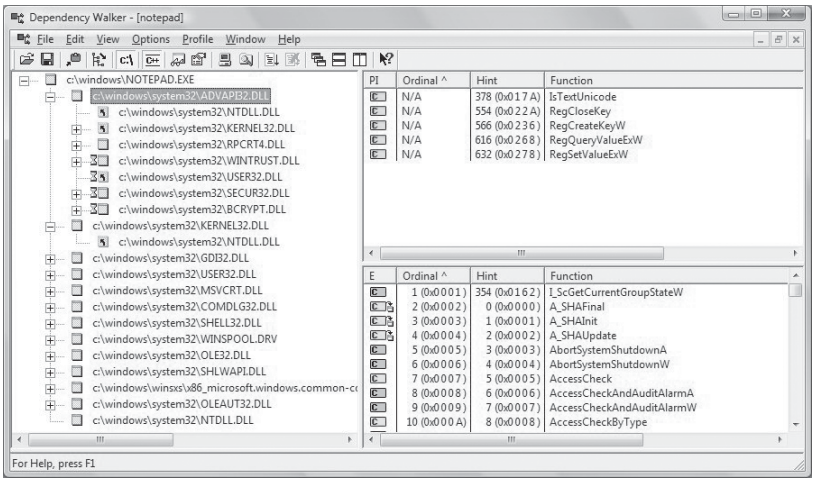
Новые бинарные компоненты операционной системы

В процессе разработки новой версии операционной системы была начата работа над модернизацией и компонентизацией ядра операционной системы (проект **MinWin**). Задача **MinWin** — обеспечить минимальный набор компонентов, требуемых для загрузки операционной системы и доступа к сети. Такая версия ядра, реализованная в библиотеке **KernelBase.dll** и ряде других компонентов, отвечает за поддержку файловой системы NT, загрузку драйверов, стек TCP/IP, базовый веб-сервер и занимает 25–30 Мбайт на диске.

В Windows 7 ядро операционной системы не полностью заменено на **MinWin**, но тем не менее ряд компонентов ОС изменился. Так, например, функциональность из **kernel32.dll** и **advapi32.dll** перенесена в библиотеку **kernelbase.dll** и экспортируемые функции перенаправляются в соответствующие библиотеки, используя механизмы виртуализации. Такие изменения могут повлиять на работоспособность приложений, использующих недокументированные структуры, программные интерфейсы и т. п. Единственный способ обеспечить совместимость — использовать только документированные функции и структуры, описанные в MSDN и документации, входящей в состав Windows SDK.

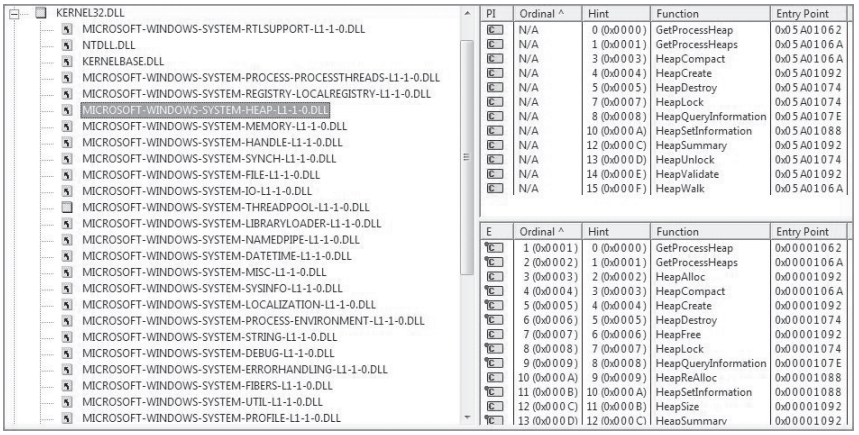
Для того чтобы увидеть изменения на уровне ядра операционной системы, воспользуемся утилитой **Dependency Walker** (**http://www.dependencywalker.com/**), которая выполняет анализ исполняемых файлов и библиотек и показывает зависимости от компонентов операционной системы и файлов сторонних производителей.

В качестве примера выберем утилиту **Notepad.exe**, которая располагается в папке **%windir%**. Сначала запустим утилиту в Windows Vista и выберем модуль **c:\windows\system32\KERNEL32.DLL** — убедимся в том, что данный модуль зависит от модуля с**:\windows\system32\NTDLL.DLL**. Взгляд на модуль **c:\windows\system32\ADVAPI32.DLL** показывает, что он зависит от **c:\windows\system32\KERNEL32.DLL**, **c:\windows\system32\NTDLL.DLL** и ряда других модулей. Результат нашего исследования показан на следующем рисунке.



Работа Dependency Walker в Windows Vista

Запуск этой же утилиты под управлением Windows 7 покажет совсем другую картину — модули **KERNEL32.DLL** и **ADVAPI32.DLL** на самом деле состоят из набора подмодулей (см. рис. ниже).



Работа Dependency Walker в Windows 7

Эти подмодули называются **ApiSet Stub DLL** и содержат подмножества функций из библиотеки **KERNEL32.DLL**, что делает ядро ОС более модульным.

Работа Internet Explorer в режиме DEP

В новой версии Internet Explorer механизм **Data Execution Prevention** (**NX**) по умолчанию включен. В Windows Vista для включения этого механизма требовался запуск IE под учетной записью администратора. Использование механизмов **Data Execution Prevention** повышает безопасность Internet Explorer, запрещая выполнение кода в области данных (более подробно об этом см. «**Технологии защиты приложений в Windows Vista, Windows 7 и Windows Server 2008/R2**» в приложении), но может повлиять на выполнение ряда модулей расширения. Для исправления данной ситуации при создании модулей расширения следует использовать версии библиотек, совместимые с DEP (например, ATL, подробности см. в статье из базы знаний «**Applications Using Older ATL Components May Experience Conflicts With DEP**», **http://support.microsoft.com/kb/948468**) и при сборке использовать опцию **/NXCOMPAT**.

Дополнительно:

- Internet Explorer 8 Security Part I: DEP/NX Memory Protection
  - **http://blogs.msdn.com/ie/archive/2008/04/08/ie8-security-part-I_3A00_-dep-nx-memory-protection.aspx**
- Data Execution Prevention
  - **http://msdn2.microsoft.com/en-us/library/aa366553.aspx**

- New NX APIs added to Windows Vista SP1, Windows XP SP3 and Windows Server 2008 R2
  - **http://blogs.msdn.com/michael_howard/archive/2008/01/29/new-nx-apis-added-to-windows-vista-sp1-windows-xp-sp3-and-windows-server-2008.aspx**

Изменения в MSMQ

На уровне Microsoft Message Queing (MSMQ) произошел ряд изменений, которые сводятся к использованию по умолчанию алгоритма хэширования SHA-2 (который не поддерживается для Windows Server 2003 и более ранних версий операционной системы) — изменение этого алгоритма возможно редактированием соответствующих значений реестра.

Технология Switchback

В манифесте приложения появилась новая секция — **CompatibilityInfo**, которая позволяет указать, для какой версии операционной системы предназначено то или иное приложение. Данная секция позволяет, например, использовать компоненты от предыдущих версий операционной системы (таких, как Common Controls), которые располагаются в папке **%windir%\winsxs**.

Правила использования секции **CompatibilityInfo** достаточно простые — если ее нет в манифесте, подразумевается совместимость с Windows Vista и задействуются ряд функций системы — **GetOverlappedResult()**, **ReadFileEx()**, обработчик исключений RPC, управление пулом потоков, режим совместимости DWM и т. п.

Расширенная версия манифеста приложения выглядит так:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
 manifestVersion="1.0">
  <compatibility
   xmlns="urn:schemas-microsoft-com:compatibility.v1">
    <application>
      <!-- Windows 7 supported -->
      <supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>
    </application>
  </compatibility>
</assembly>
```

GUID для Windows Vista имеет значение **{e2011457-1546-43c5-a5fe-008deee3d3f0}**, для Windows 7 — **{35138b9a-5d96-4fbd-8e2d-a2440225f93a}**.

Полезные утилиты

В процессе тестирования приложений и сбора информации об их работе можно воспользоваться рядом утилит, краткое описание которых приведено ниже.

Application Verifier

Утилита Application Verifier позволяет выполнить тестирование приложения на соответствие ключевым рекомендациям по созданию совместимых приложений и получить детальный протокол действий, выполняемых приложением — наличие отладочной информации поможет исправить обнаруженные ошибки. Статистика, собранная специалистами Microsoft по результатам анализа дампов на сайте WinQual, показывает, что более 68% всех сбоев приложений можно было избежать, если воспользоваться утилитой Application Verifier и применить на практике результаты мониторинга приложений.

Дополнительно:

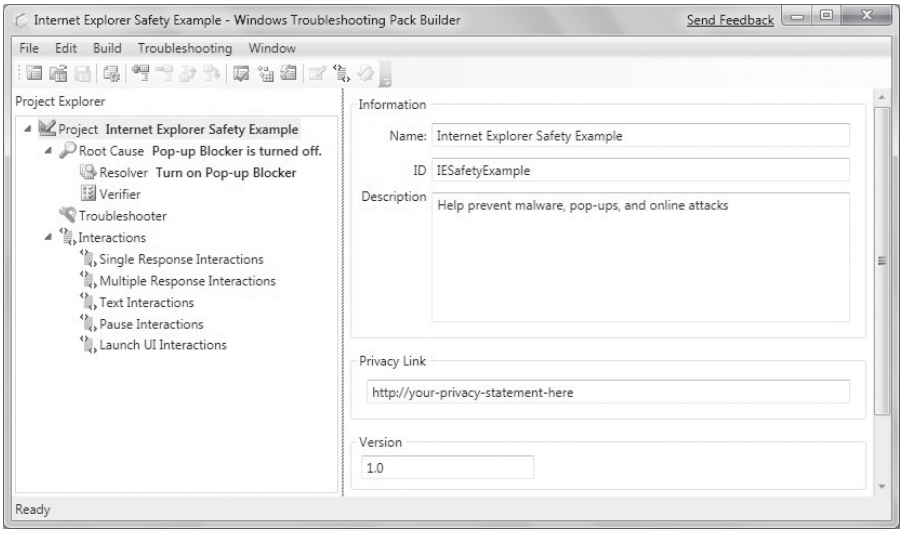
- Debugging Tools for Windows
  - **http://www.microsoft.com/whdc/DevTools/Debugging/default.mspx**
- Application Verifier
  - **http://msdn.microsoft.com/en-us/library/ms644353.aspx**
  - **http://www.microsoft.com/downloads/details.aspx?FamilyID=bd02c19c-1250-433c-8c1b-2619bd93b3a2&DisplayLang=en**
- WinQual
  - **https://winqual.microsoft.com**
- Windows Error Reporting
  - **http://www.microsoft.com/whdc/maintain/StartWER.mspx**

Платформа Windows Troubleshooting

В Windows 7 появилась возможность создания т. н. **Troubleshooter Packs** — пакетов для решения возникающих проблем при использовании программного обеспечения. Помимо встроенных в систему пакетов для решения наиболее часто встречающихся проблем — эти пакеты доступны через Action Center и, при необходимости, автоматически догружаются с сайта Microsoft, у разработчиков есть возможность создания собственных пакетов.

Для создания пакетов используется утилита **Troubleshooter Builder**, которая входит в состав **Windows SDK** и находится по адресу **%sdkdir%\bin\tspbuilder\builder.exe**. С помощью утилиты **Troubleshooter Builder** создается проект, в котором описываются условия выбора пакета исправле-

ний и действия, выполняемые пакетом. Код, вносящий исправления в систему, пишется **на PowerShell**.



Утилита Troubleshooter Builder

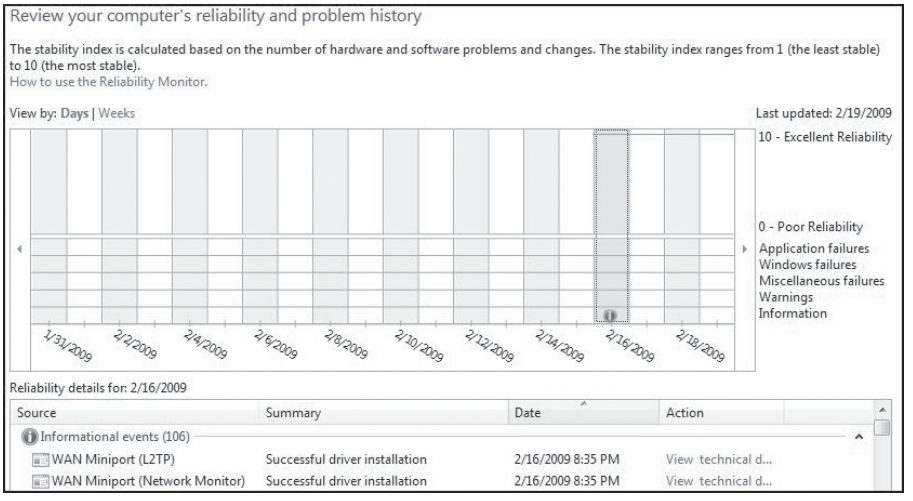
Компонент Reliability Analysis

Компонент операционной системы, известный под названием **Reliability Analysis Component** (**RAC**), предоставляет информацию для утилиты **Reliability Monitor**, с помощью которой высчитывается индекс стабильности системы (**System Stability Index**). Индекс стабильности системы — это число от 1 (наименее стабильная) до 10 (наиболее стабильная) и его «вес» высчитывается на основании числа специфических сбоев, произошедших в системе за указанный период. Соответствующие события — **Reliability Events** и отчет о стабильности системы — **System Stability Report** содержат более подробную информацию о сбоях.

Как мы отметили выше, **RAC** содержит данные о наборе событий, описывающие стабильность системы — эти данные хранятся в **Windows Management Instrumentation** (**WMI**), записях в **Win32_ReliabilityStabilityMetrics** и **Win32_ReliabilityRecords**. При необходимости, к этим данным можно обратиться из кода, например, написанного на **PowerShell**.

Для доступа к **Reliability Monitor** используйте следующую последовательность действий:

- *Start | Control Panel*
- *System and Security | Action Center*
- *Maintenance | View System History*



Reliability Monitor

Windows Error Reporting Problem Steps Recorder

При возникновении проблем с использованием системного или прикладного программного обеспечения и запуском отдельных компонентов системы можно воспользоваться специальной утилитой, которая позволит подробно записать шаги, приводящие к возникновению проблемы и соответствующие действия пользователя. Данная утилита называется **Problem Steps Recorder** и располагается по адресу **%windir%\system32\psr.exe**. Результат работы данной утилиты — MHT-файл со снимками экранов и комментариями, упакованный в ZIP-архив, этот архив можно отправить в Microsoft средствами **Windows Error Reporting**.

Подготовка к сертификации приложений

С выходом Windows 7 будет доступна программа сертификации приложений, которая представляет собой дальнейшее развитие программы сертификации приложений под Windows XP и Windows Vista. Программа сертификации направлена на обеспечение совместимости, надежности и безопасности приложений и гарантирует соответствие сертифицированных приложений стандартам Windows. Для систем, устройств и прикладных программ будет всего один логотип.

Логотип для приложений

Процесс получения сертификации существенно упрощен: больше не требуется передавать приложение на тестирование в стороннюю организацию — все действия выполняются непосредственно в компании-разработчике программного обеспечения, используя для тестирования специальный набор утилит. Дополнительная информация по сертификации приложений будет доступна на сайте **http://www.innovateonwindows.com/logo**.

Приведем несколько рекомендаций по сертификации приложений:

- сертификация под Windows Vista гарантирует совместимость с Windows 7;
- начните с учета требований к сертифицируемым приложениям;
- включите требования в процесс разработки и тестирования.

Обзор программы сертификации

Сертифицируемые приложения должны отвечать следующим критериям.

- Должны быть самостоятельными Windows-приложениями — т. е. приложениями, выполняющимися в отдельном процессе Windows 7. К таким приложениям, в частности, относятся приложения, требующие наличия на компьютере других приложений.
- Приложения должны выполняться на локальном компьютере под управлением Windows 7.
- Примеры приложений, соответствующих приведенным критериям:
  - Windows 7-приложение, требующее наличия на компьютере Outlook 2007;
  - приложение на Java или .NET;
  - Windows 7-приложение, требующее наличия привода оптических дисков;
  - Windows 7-приложение, требующее соединения с сервером.
- Примеры приложений, не соответствующих приведенным выше критериям:

  - веб-приложение;
  - расширение для веб-браузера;
  - компонент Microsoft Management Console (MMC);
  - драйвер для аппаратного компонента;
  - библиотека и среда для разработчиков, например Java, .NET, Adobe AIR и т. п.
- Компоненты приложения (отдельные файлы или приложения), требующиеся для его выполнения, также должны отвечать критериям сертификации.
- Дополнительные компоненты по возможности должны отвечать критериям сертификации.
- Не требуется тестирования компонентов Windows. К таким компонентам относятся:
  - все компоненты, входящие в комплект операционной системы;
  - все компоненты, доступные через Windows Update;
  - Windows Platform SDK;
  - **примечание:** другие компоненты производства Microsoft, включая Microsoft Visual Studio и Microsoft Office не являются Windows-компонентами.

Шаги по прохождению сертификации

1. Загрузить и установить Windows 7 Client Software Logo Toolkit.
2. Выполнить проверку приложения с помощью Windows 7 Client Software Logo Toolkit.
3. Убедиться в том, что приложение успешно прошло проверку.
4. Подписать необходимые документы.
5. Согласиться/отказаться от рассылок Microsoft, связанных с вашим продуктом.
6. Переслать в Microsoft копию вашего приложения.
7. Передать в Microsoft результаты тестирования вашего приложения.

Требования к приложениям

Список требований к сертифицируемым приложениям приведен ниже.

- Не распространяйте вредоносное ПО (malware, spyware...)
  - **Подробности**:
    - http://www.antispywarecoalition.org/
- Не изменяйте ресурсы, защищенные **Windows Resource Protection**

- **Подробности**:
  - http://msdn.microsoft.com/en-us/library/aa382503(VS.85).aspx
- Используйте механизмы **Windows Error Reporting**, тестируйте приложения с помощью **Application Verifier**, зарегистрируйтесь на портале **WinQual**
  - **Подробности**:
    - WER
      - http://msdn.microsoft.com/en-us/isv/bb190483.aspx
    - Application Verifier
      - http://msdn.microsoft.com/en-us/library/ms220948.aspx
    - WinQual
      - https://winqual.microsoft.com/
- Используйте «чистую» установку и удаление
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/aa372105.aspx
- Используйте подписанные файлы и драйверы
- Устанавливайте в корректные папки
- Поддерживайте Windows x64
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/bb427430(VS.85).aspx
- Корректно проверяйте версию ОС
- Соответствуйте требованиям UAC
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/aa905330.aspx;
      http://msdn.microsoft.com/en-us/library/bb756973.aspx
- Используйте механизмы **Restart Manager**
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/aa373524.aspx
- Не загружайте драйвера/сервисы в режиме **Safe Mode**
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/aa489541.aspx
- Не требуйте перезагрузки после установки
- Поддерживайте многопользовательские сессии
  - **Подробности**:
    - http://msdn.microsoft.com/en-us/library/aa383490(VS.85).aspx

Помимо приведенных выше рекомендаций следует корректно использовать основные возможности операционной системы, включая ассоциации с программами по умолчанию, поддержку режима High DPI, работу с Unicode и обеспечение безопасности.

Материалы по сертификации

- Windows 7 Software Logo Requirements
  - **http://go.microsoft.com/?linkid=9630189**
- Windows 7 Client Software Logo Toolkit (WCSK)
  - Доступность: Windows 7 RTM
- Информация по Windows 7 Logo
  - **http://www.innovateonwindows.com/logo**
  - **http://msdn.microsoft.com/en-us/windows/default.aspx**

Приложение. Технологии защиты приложений в Windows Vista, Windows 7 и Windows Server 2008/R2

Ряд расширений, появившихся в новой версии клиентской операционной системы Microsoft Windows Vista, а также планируемых к появлению в Windows Server 2008, предназначен для обеспечения защиты приложений и соответственно пользователей этих приложений от вредоносного кода. В этом приложении мы ознакомимся с некоторыми из этих расширений, реализованными как на уровне ядра операционной системы, так и средствами компилятора Microsoft Visual C++, а также со способами включения этих расширений в «неуправляемый» код, написанный на языках C и C++.

Проверка переполнения стека

Проверка переполнения буфера стека поддерживается на уровне компилятора языка C/C++, начиная с версии Visual Studio .NET 2002. Используя опцию компилятора **/GS** (она поддерживается по умолчанию в Visual C++, даже если ее нет в командной строке), мы указываем на необходимость вставки кода пролога и эпилога для функции. Этот код генерирует случайное число (называемое **Security Cookie**), которое помещается в стек функции. Если сгенерированное случайное число окажется «испорченным», вызывается код завершения приложения — таким образом снижается вероятность запуска «эксплойтов», базирующихся на возможности переполнения буфера.

В Visual C++ 2005 код пролога и эпилога выглядит следующим образом:

```
// Пролог
  sub    esp, 8
  mov    eax, DWORD PTR ___security_cookie
  xor    eax, esp
  mov    DWORD PTR __$ArrayPad$[esp+8], eax
  mov    eax, DWORD PTR _input$[esp+4]

// Эпилог
  mov    ecx, DWORD PTR __$ArrayPad$[esp+12]
  add    esp, 4
  xor    ecx, esp
  call   @__security_check_cookie@4
  add    esp, 8
```

Отметим, что компилятор Visual C++ 2005 также включает код, перемещающий данные по стеку, что тоже затрудняет выполнение вредоносных операций. В частности, компилятор позволяет размещать буферы в верхнюю область памяти — это позволяет защитить указатели на функ-

ции, расположенные в стеке. Перемещение указателей и аргументов буфера в нижнюю часть памяти также позволяет значительно снизить вероятность атак на буфер и использования переполнения буфера. Разработчикам рекомендуется применять самую актуальную версию компилятора Visual C++ и всегда указывать опцию /GS при компиляции приложений и библиотек.

В Visual C++ 2005 SP1 появился ряд новых прагм (**pragma**) со следующим синтаксисом:

```
#pragma strict_gs_check([push,] on )
#pragma strict_gs_check([push,] off )
#pragma strict_gs_check(pop)
```

Рекомендуется включать эти прагмы в код, содержащий функции, которые могут быть подвержены атаке, или в код, работающий с данными, получаемыми извне.

С подробным описанием опции комплилятора **/GS** можно ознакомиться на сайте MSDN по адресу: **http://msdn2.microsoft.com/en-US/library/8dbf701c.aspx**.

Защита при обработке исключений

Обработчик исключения — это код, который получает управление при возникновении какой-либо нештатной ситуации, например при делении на ноль, переполнении стека и т. п. Адрес такого обработчика хранится в стеке функции, а следовательно, может стать предметом атаки для вредоносного кода.

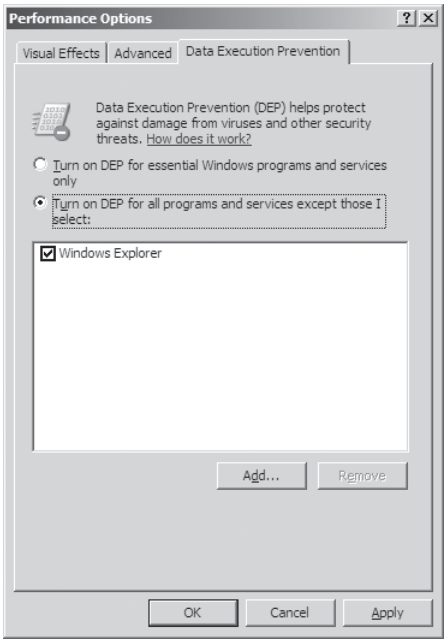
Компоновщик, входящий в состав Visual Studio 2003 и более поздних версий продукта, имеет опцию (**/SafeSEH**), позволяющую хранить ссылки на обработчики исключений в заголовке образа исполняемого файла (**PE Header**). При возникновении исключительной ситуации операционная система использует адрес обработчика, хранящийся в заголовке PE, — такая функциональность поддерживается начиная с Windows XP SP2 и является стандартной для Windows Server 2003, Windows Vista, Windows Server 2008 и будущих версий клиентской и серверной операционных систем компании Microsoft. Подробное описание опции компоновщика /SafeSEH можно найти на сайте MSDN по адресу: **http://msdn2.microsoft.com/en-US/library/9a89h429.aspx**.

Поддержка No Execute (NX), Data Execution Prevention (DEP) и Execute Disable (XD)

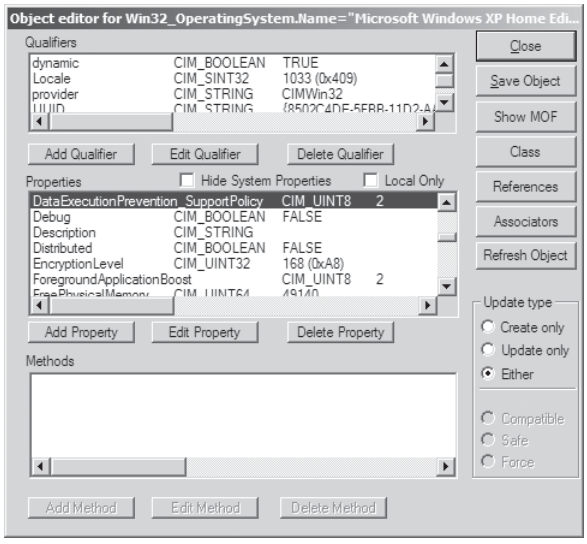
Ряд технологий — **No eXecute** (NX) компании Advanced Micro Devices (AMD), **Data Execution Prevention** (предотвращение выполнения данных, DEP) компании Microsoft и **eXecute Disable** (XD) компании Intel —

предотвращает возможность выполнения кода в сегментах, предназначенных для хранения данных. Все современные процессоры компаний Intel и AMD поддерживают технологии XD и NX соответственно. Поддержка Data Execution Prevention была впервые реализована в Windows XP SP2 и является одной из ключевых для обеспечения безопасного выполнения кода в Windows Vista, особенно когда она используется совместно с технологией Address Space Layout Randomization (ASLR), которую мы рассмотрим ниже. В силу определенной специфики Data Execution Prevention несовместима с приложениями, содержащими самомодифицируемый код, а также с приложениями, реализующими компиляцию в режиме выполнения, — при запуске таких приложений произойдет ошибка. Если создаваемые вами приложения содержат самомодифицирущийся код или выполняют компиляцию в режиме выполнения, следует использовать функцию **VirtualProtect()** с аргументом **PAGE_EXECUTE_READ** для определения наличия защиты от выполнения кода в сегментах, предназначенных для хранения данных, и для корректного завершения приложения с соответствующим сообщением об ошибке.

Разработчикам рекомендуется протестировать работу приложения под процессорами, поддерживающими технологии XD и NX, внести необходимые корректировки в код и перекомпилировать приложения, используя опцию компилятора **/NXCOMPAT**.



Настройки Data Execution Prevention в Windows Vista и Windows 7



Получение данных о Data Execution Prevention через WMI

На уровне операционной системы в Windows XP SP2 и Windows Server 2003 настройка Data Execution Prevention определяется параметрами в файле Boot.ini — эти параметры могут задаваться в панели управления: **System Properties => Advanced => Performance => Data Execution Prevention**. Windows поддерживает четыре возможных значения для Data Execution Prevention — при каждом из них может применяться как программная, так и аппаратная реализация этой технологии.

Возможные настройки Data Execution Prevention показаны в таблице ниже.

Настройка	Описание
OptIn	Используется по умолчанию. На компьютерах, оснащенных процессорами с поддержкой DEP, функция DEP включена по умолчанию для ограниченного числа системных файлов и программ. При этом по умолчанию защищаются только системные файлы Windows
OptOut	По умолчанию функция DEP включена для всех процессов. В диалоговом окне Система панели управления можно вручную создать список приложений, для которых следует отключить DEP. Для отключения функции DEP для одной или более программ можно использовать Application Compatibility Toolkit — в этом случае будут применены исправления, обеспечивающие совместимость программ с DEP

Настройка	Описание
AlwaysOn	Функция DEP включается для всей системы. Все процессы работают с выполнением проверок DEP. В этом режиме нельзя отключить функцию DEP для отдельных приложений. Исправления, обеспечивающие совместимость программ с DEP, не применяются
AlwaysOff	Функция DEP отключена для всей системы независимо от наличия аппаратной поддержки DEP. Процессор не работает в режиме PAE, если в файле Boot.ini не указан параметр /PAE

Если на уровне системы для функции DEP выбран режим **OptIn**, то основные программы и файлы Windows будут защищены как программной, так и аппаратной реализацией DEP. Если система не может использовать аппаратную реализацию DEP, то указанные программы и файлы Windows будут защищены только программной реализацией DEP. Аналогично, если на уровне системы для функции DEP выбран режим **OptOut**, программы, для которых отключена функция DEP, не будут защищены ни программной, ни аппаратной реализацией данной функции.

Параметры, указываемые в файле **Boot.ini**, выглядят так:

```
/noexecute=[AlwaysOn, AlwaysOff, OptIn или OptOut]
```

Содержимое файла Boot.ini может выглядеть следующим образом:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\
WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows
   XP Home Edition" /fastdetect /NoExecute=OptIn
multi(0)disk(0)rdisk(0)partition(2)\WINDOWS="Windows XP x64
   Edition 2003" /fastdetect
```

Для проверки поддержки Data Execution Prevention на аппаратном уровне можно воспользоваться одним из следующих методов: использовать утилиту **WMIC** или утилиту **WBEMTest**.

В первом случае мы выполняем следующую команду:

```
C:\>wmic OS Get DataExecutionPrevention_Available
```

Если возвращается значение **True**, аппаратная поддержка DEP доступна, в противном случае возвращается значение **False**. Уровень поддержки DEP можно определить командой

```
C:\>wmic OS Get DataExecutionPrevention_SupportPolicy,
```

которая может вернуть одно из следующих значений: *0* — AlwaysOff, *1* — AlwaysOn, *2* — OptIn (значение по умолчанию) или *3* — OptOut.

С помощью утилиты **WBEMTest** мы можем выполнить следующие действия.

1. В диалоговой панели Windows Management Instrumentation Tester нажмем кнопку Connect и укажем значение root\cimv2.
2. Нажмем кнопку Enum Instances и в диалоговой панели Class Info введем **Win32_OperatingSystem**.
3. В диалоговой панели Query Result выберем самый первый элемент и двойным щелчком мышью перейдем в Object Editor.
4. В списке свойств найдем **DataExecutionPrevention_Available** и обратим внимание на значение этого свойства: оно может иметь одно из значений, описанных при обсуждении использования утилиты WMIC.

С помощью Windows PowerShell также можно получить информацию о настройках Data Execution Prevention. Сначала выполним команду:

```
PS C:\> Get-WMIObject Win32_OperatingSystem | Get-Member
   DataExecutionPrevention*
```

для получения всех возможных свойств объекта **Win32_OperatingSystem**, используемых для хранения данных о DEP. Таких свойств четыре: **Data Execution Prevention_32BitApplications**, **Data Execution Prevention_Available**, **Data Execution Prevention_Drivers** и **Data Execution Prevention_SupportPolicy**. После этого мы можем получить значения каждого из перечисленных выше свойств (см. табл. ниже).

Свойство	Возможные значения
Data Execution Prevention_32BitApplications	TRUE — DEP активирована для приложений FALSE — DEP не активирована для приложений
Data Execution Prevention_Available	TRUE — DEP активирована FALSE — DEP не активирована
Data Execution Prevention_Drivers	TRUE — DEP активирована для драйверов FALSE — DEP не активирована для драйверов
Data Execution Prevention_SupportPolicy	0 — DEP запрещена для всех процессов 1 — DEP разрешена для всех процессов 2 — DEP разрешена для системных компонентов Windows 3 — DEP разрешена для всех компонентов, кроме помеченных специальным образом

Например:

```
$OS = Get-WMIObject Win32_OperatingSystem
Echo $OS.Data Execution Prevention_32BitApplications
```

Более подробно о функции **VirtualProtect()** можно узнать на сайте MSDN по адресу: **http://msdn2.microsoft.com/En-US/library/aa366898.aspx**, о

процессорах ADM — на сайте **http://www.amd.com/us-en/Processors/** (в описании процессора должна быть указана поддержка NX). Информация о процессорах Intel приведена на сайте **http://www.intel.com/products/processor/** (в описании процессора должна быть указана поддержка XD). Технология Data Execution Prevention описана на сайте Microsoft по адресу: **http://support.microsoft.com/kb/875352**. Также см. статью «Memory Protection Technologies» на сайте Microsoft TechNet по адресу: **http://technet.microsoft.com/en-us/library/bb457155.aspx**. Подробное описание опции компоновщика /NXCOMPAT приведено на сайте MSDN по адресу: **http://msdn2.microsoft.com/en-US/library/ms235442.aspx**.

Случайное распределение адресного пространства — Address Space Layout Randomization (ASLR)

Технология случайного распределения адресного пространства (ASLR) перемещает бинарный образ исполняемого кода в случайную область памяти — поддерживается до 255 различных адресов, что значительно снижает вероятность успешного проведения атаки типа return-to-libc и схожих с ней. По умолчанию в Windows Vista случайным образом загружаются системные исполняемые файлы (EXE) и динамические библиотеки (DLL). Исполняемые файлы и динамические библиотеки, создаваемые сторонними разработчиками, должны быть слинкованы (используется компоновщик Microsoft Linker версии 8.00.50727.161 или более поздней) с применением опции **/DYNAMICBASE** для того, чтобы на них распространялась ASLR.

Случайное распределение «кучи»

В Windows Vista при создании «кучи» для приложения она располагается случайным образом. В результате снижается шанс успешного выполнения атаки, основанной на переполнении буфера, располагаемого в «куче». Эта функциональность поддерживается в Windows Vista по умолчанию.

Случайное распределение стека

При создании потока для процесса, помеченного с помощью опции **/DYNAMICBASE,** Windows Vista перемещает стек потока в случайно выбранную область памяти. Тем самым снижается шанс успешного выполнения атаки, основанной на переполнении буфера, располагаемого в стеке.

Определение повреждения «кучи»

С помощью функции **HeapEnableTerminationOnCorruption()** можно включить проверку повреждения «кучи», что приведет к завершению процесса при обнаружении повреждений. Далее приведен пример кода, показывающий использование данной функции:

```
BOOL SetHeapOptions()
{
 HMODULE hLib = LoadLibrary(L"kernel32.dll");
 if (hLib == NULL) return FALSE;
 typedef BOOL (WINAPI *HSI) (HANDLE, HEAP_INFORMATION_CLASS
    ,PVOID, SIZE_T);
 HSI pHsi = (HSI)GetProcAddress(hLib,"HeapSetInformation");
 if (!pHsi)
 {
  FreeLibrary(hLib);
  return FALSE;
 }
 #ifndef HeapEnableTerminationOnCorruption
#define HeapEnableTerminationOnCorruption
(HEAP_INFORMATION_CLASS)1
 #endif
BOOL fRet = (pHsi)(NULL,HeapEnableTerminationOnCorruption,
NULL,0) ? TRUE : FALSE;
 if (hLib) FreeLibrary(hLib);
  return fRet;
}
```

Завершая наш обзор технологий защиты приложений в Windows Vista, Windows 7 и Windows Server 2008/R2, отметим, что весь код приложения и библиотек необходимо полностью протестировать на совместимость с приведенными здесь опциями, при этом рекомендуется использовать самые последние версии компилятора и компоновщика.

Заключение

В данном руководстве мы рассмотрели ключевые вопросы, связанные с написанием приложений, корректно работающих под управлением операционных систем Microsoft Windows Vista и Microsoft Windows 7. Мы обсудили основные требования, выдвигаемые к приложениям, а также технологии, реализованные в Windows Vista и Windows 7, позволяющие создавать надежные и безопасные приложения, стабильно работающие под управлением современных операционных систем. Ключевой рекомендацией, завершающей наш рассказ, будет следующее: даже создавая приложения для Microsoft Windows XP, разрабатывайте их в Microsoft Visual Studio 2008, установленной на Windows Vista, и только под учетной записью «стандартный пользователь» — в этом случае вы получите приложения, максимально совместимые с Microsoft Windows XP, Microsoft Windows Vista и Microsoft Windows 7. Использование утилиты Application Verifier позволит вам еще на этапе тестирования убедиться в корректной работе ваших приложений, а включение в процесс тестирования сценариев, рекомендованных при сертификации приложений под Windows Vista и Windows 7, гарантирует максимальную совместимость с современными операционными системами.