B

Windows Vista и Windows Server 2008

В настоящем приложении мы представим информацию, которую вам необходимо знать, чтобы разрабатывать приложения для Windows Vista и Windows Server 2008, а также использовать новые средства Windows из приложений .NET. Мы не будем здесь описывать все средства, полезные для пользователей Windows Vista или администраторов Windows Server 2008, а сосредоточимся на тех из них, что важны для разработчиков.

Если ваше приложение не ориентировано только на Windows Vista, вам следует помнить, что хотя WPF, WCF, WF и LINQ также доступны в Windows XP, этот случай не будет предметом рассмотрения настоящего приложения. Если вы все еще ориентируетесь на Windows XP, вам следует знать о проблемах, ожидающих ваши программы при их запуске под Windows Vista, и чему следует уделить внимание. В этом случае вы должны особо сосредоточиться изменениях в управлении пользовательскими учетными записями и системе каталогов.

В приложении будут рассмотрены следующие вопросы:

- Vista Bridge;
- управление пользовательскими учетными записями;
- структура каталогов;
- новые элементы управления и диалоги;
- поиск.

Vista Bridge

В версии .NET 3.5 новые вызовы Windows API, доступные в Windows Vista и Windows Server 2008, не доступны из .NET Framework. Однако Windows SDK содержит библиотеку примеров под названием Vista Bridge, которая помещает вызовы "родного" API в оболочки, делая их доступными из библиотеки .NET. Вы можете использовать эту библиотеку в своих приложениях Windows Forms или WPF.

После инсталляции Windows SDK вы можете найти пример Vista Bridge в zip-файле `<program files>\Microsoft SDKs\Windows\v6.0\Samples\CrossTechnologySamples.zip`. Распаковав этот файл, вы получите три проекта: `VistaBridgeLibrary`, `VistaBridgeControls` и `VistaBridgeDemoApp`. Проект `VistaBridgeLibrary` содержит несколько классов и элементов управления.

Управление пользовательскими учетными записями

Управление пользовательскими учетными записями (User Account Control — UAC) — это одно из средств, которые вы первым делом увидите в Windows Vista и Windows Server 2008, будучи разработчиком. Хотя в руководствах по Windows всегда упоминается об этой проблеме, многие приложения по-прежнему должны запускаться от имени учетной записи администратора. Например, нормальный пользователь не имеет права писать данные в каталог программных файлов — это требует административных привилегий. Поскольку многие приложения вообще не работают, не имея таких привилегий (хотя их функциональность этого и не требует), многие пользователи входят в систему от имени учетной записи Administrator. Поступая так, вы подвергаете свою систему риску установки нежелательных программ — "троянских коней".

Рис. B.1. Вкладка Compatibility окна свойств приложения

Windows Vista избегает этой проблемы, потому что Administrator по умолчанию не имеет административных привилегий. С этим процессом ассоциированы два маркера защиты — один с нормальными пользовательскими привилегиями, а другой — с административными привилегиями (в этом случае вход выполняется от имени Administrator). С приложениями, требующими административных привилегий, пользователь может повысить уровень приложения для работы с правами Administrator. Это делается либо через контекстное меню Run as Administrator (Запуск от имени администратора), либо путем конфигурирования приложения на постоянное требование административных привилегий на вкладке Compatibility (Совместимость) окна свойств приложения, как показано на рис. B.1. Эта установка добавляет в реестр флаг совместимости `HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers` со значением `RUNASADMIN`.

Приложения, требующие административных привилегий

Для приложений, требующих административных привилегий, вы можете также добавлять манифест приложения.

Visual Studio 2008 имеет новый шаблон для добавления манифеста к приложению. Такой манифест может быть либо добавлен к существующему приложению, либо посредством встраивания ресурсного файла Win32 в сборку. После добавления файла манифеста к проекту Visual Studio, он добавляется в ресурсы проекта, как вы можете увидеть в свойствах проекта, выбрав закладку Application (Приложение) в категории Resources (Ресурсы). Наличие вхождения здесь встраивает манифест в ресурс Win32 сборки.

Манифест приложения — это XML-файл, подобный конфигурационному файлу приложения. Но в отличие от конфигурационного файла приложения, имеющего расширение `.config`, файл манифеста оканчивается на `.manifest`. Имя файла должно совпадать с именем приложения, включая расширение файла `.exe`, за которым следует `.manifest`. Имя файла манифеста должно быть установлено по имени приложения, включая расширение файла `exe`, за которым следует `.manifest`. Visual Studio

переименовывает и копирует файл app.manifest так же, как делает это с конфигурационным файлом приложения. Файл манифеста содержит XML-данные, как показано ниже. Корневым элементом является <assembly>, содержащий в себе дочерний элемент <trusInfo>. Административные требования определены атрибутом level элемента <requestedExecutionLevel>.

```
<?xml version="1.0" encoding="UTF-8"?>
  <asmv1:assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1"
     xmlns:asmv1="urn:schemas-microsoft-com:asm.v1
     xmlns:asmv2="urn:schemas-microsoft-com:asmv2
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app" />
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
      <requestedExecutionLevel level="requireAdministrator"
        uiAccess="false"/>
      </requestedPrivileges>
    </security>
  </trustInfo>
</asmv1:assembly>
```

При таком запуске приложения пользователь получит запрос на разрешение запускать приложение с административными привилегиями.

При установке requestedExecutionLevel вы можете специфицировать значения requireAdministrator, highestAvailable и asInvoker. Значение highestAvailable означает, что приложение получает те привилегии, которыми обладает пользователь. requireAdministrator означает требование административных привилегий. Если пользователь не зарегистрирован в системе как Administrator, появляется диалоговое окно входа, через которое пользователь может для этого приложения войти в систему как Administrator. asInvoker означает, что приложение работает с маркером доступа текущего пользователя.

Атрибут uiAccess специфицирует, что приложение требует ввода в более высокопривилегированное окно на рабочем столе. Например, экранная клавиатура требуется для ввода в других окнах на рабочем столе, поэтому эта настройка должна быть установлена в true для приложений, отображающих экранную клавиатуру. Приложения, не требующие доступа к пользовательскому интерфейсу, должны устанавливать этот атрибут в false.

Другой способ получения административных привилегий заключается в написании службы Windows (Windows Service). Поскольку UAC применяется только к интерактивным процессам, Windows Service может получать административные привилегии. Вы можете также написать непривилегированное Windows-приложение для взаимодействия с Windows Service, используя WCF или другую технологию коммуникаций.

Службы Windows рассматриваются в главе 23, а WCF — в главе 42.

Защитная пиктограмма

Если приложение или некоторая задача, выполняемая приложением, требует административных привилегий, пользователь информируется об этом легко узнаваемой защитной пиктограммой (с изображением щита). Такая пиктограмма прикрепляется к элементам управления, требующим повышения уровня привилегий. Пользователь ожидает увидеть соответствующее приглашение, щелкая на элементе с такой пиктограммой. На рис. B.2 и B.3 можно видеть эту пиктограмму. Диспетчер задач (Task Manager) требует повышения уровня привилегий для просмотра процессов пользова-

телей. В окне User Accounts (Учетные записи пользователей) панели управления изменение типа учетной записи и предоставление другим пользователям доступа к компьютеру требуется повышение уровня привилегий.

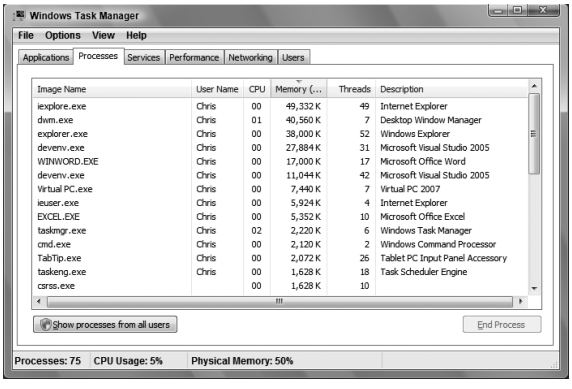

Рис. B.2. Окно диспетчера задач

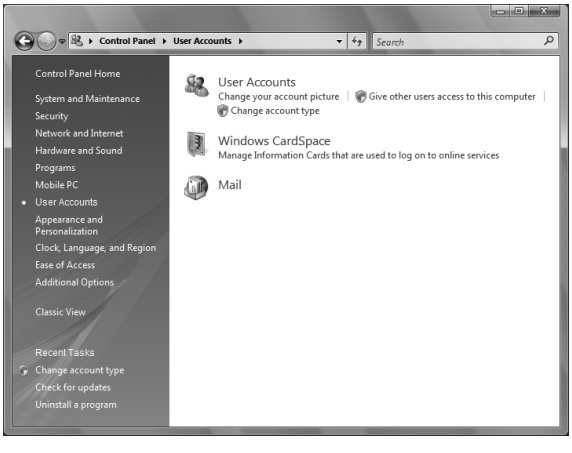

Рис. B.3. Окно User Accounts *панели управления*

Создать защитную пиктограмму в своем приложении можно с использованием нового элемента управления типа командной ссылки, который рассматривается далее в настоящем приложении.

Когда пользователь щелкает на элементе управления с защитной пиктограммой, появляется приглашение для повышения уровня привилегий. Эти приглашения бывают разными, в зависимости от типа приложения, которое их инициирует.

- Для продолжения работы Windows требуются ваши права. Это предупреждение показывается для приложений, поставляемых с Windows.
- Для продолжения работы программе требуются ваши привилегии. Это предупреждение показывается для приложений, содержащих сертификат для предоставления информации об издателе.
- Неизвестной программе требуется доступ к вашему компьютеру. Это предупреждение показывается для приложений, не имеющих сертификатов.

Структура каталогов

Структура каталогов Windows в версии Windows Vista претерпела изменения. Теперь уже нет каталога `c:\Documents and Settings\<username>`. Он заменен новой папкой `c:\Users\<username>`. Windows XP определяет подкаталог `My Documents` (Мои документы) для хранения пользовательских данных. Windows Vista имеет для этого `c:\Users\<username>\Documents`.

Если вы следуете простому правилу — не использовать жестко закодированные значения путей в программе, то не важно, где хранятся реальные папки. Эти папки все равно отличаются при использовании разных языков в Windows. Для обращения к специальным папкам используйте класс `Environment` и перечисление `SpecialFolders`:

```
string folder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
```

Некоторые папки, определенные в перечислении `SpecialFolders`, описаны в табл. В.1.

Таблица В.1. Некоторые папки, определенные в перечислении `SpecialFolders`

Содержимое	Перечисление `SpecialFolder`	Каталог Windows Vista по умолчанию
Специфичные для пользователя документы	`Personal`	`c:\Users\<User>\Documents`
Специфичные для "блуждающих" пользователей данные	`ApplicationData`	`c:\Users\<User>\AppData\Roaming`
Специфичные для пользователя данные, локальные по отношению к системе	`LocalApplicationData`	`c:\Users\<User>\AppData\Local`
Программные файлы	`ProgramFiles`	`c:\Program Files`
Программные файлы, разделяемые разными программами	`CommonProgramFiles`	`c:\Program Files\Common Files`
Прикладные данные, общие для всех пользователей	`CommonApplicationData`	`c:\ProgramData`

При выходе из системы содержимое "блуждающих" каталогов (независимых от конкретного компьютера) копируется на сервер, так что если пользователь войдет в сеть с другой системы, то же самое содержимое будет скопировано на его новый компьютер, т.е. все, что нужно, будет доступно с любой системы сети.

Имея дело со специальными папками, вы должны соблюдать осторожность, чтобы рядовой пользователь не имел доступа на запись в каталоге программных файлов. Специфичные для пользователя данные вы можете записывать в `LocalApplicationData`, либо — для блуждающих пользователей — в `CommonApplicationData`.

Поскольку многие приложения пишут содержимое в каталог программных файлов, они не могут выполняться в среде Windows Vista без административных привилегий. Windows Vista предусматривает решение для работы с такими программами — перенаправление папки в виртуальное хранилище, которое может быть прочитано и записано приложением без генерации ошибок. Эта техника называется файловой виртуализацией.

Проверим это, написав простую программу, записывающую файл в подкаталог `WroxSampleApp` папки `Program Files`. Вызов `Environment.GetFolderPath()` со значением перечисления `ProgramFiles` возвращает папку `Program Files`; точное наименование этой папки зависит от языка текущей инсталляции Windows.

Папка Program Files комбинируется с каталогом WroxSampleApp, и в этот каталог пишется файл samplefile.txt.

```
string programFiles = Environment.GetFolderPath(
  Environment.SpecialFolder.ProgramFiles);
string appDir = Path.Combine(programFiles, "WroxSampleApp");
if (!Directory.Exists(appDir))
{
   Directory.CreateDirectory(appDir);
}
string demoFile = Path.Combine(appDir, "samplefile.txt");
File.WriteAllText(demoFile, "test content");
```

При запуске этого приложения без повышения уровня привилегий файл не записывается в каталог c:\Program Files\WroxSampleApp. Вместо этого в системе Windows Vista вы найдете его в каталоге c:\Users\<username>\AppData\Local\Virtual Store\Program Files\WroxSampleApp.

Как видите, данные сохраняются в специфичный для пользователя каталог и не разделяются между разными пользователями одной и той же системы. Если же ставится такое требование, вы должны запустить приложение в режиме с повышенным уровнем привилегий. При запуске вашего приложения из привилегированного процесса Visual Studio файл пишется в папку Program Files вместо виртуального хранилища, поскольку приложение, запущенное из привилегированного процесса, также является привилегированным.

Для чтения файлов необходим другой механизм. Поскольку инсталляционной программе разрешено писать содержимое в папку Program Files, вполне корректным является разрешение программе читать из папки Program Files. Пока программа записывает в эту папку без дополнительных привилегий, происходит перенаправление. Когда же она читает записанное снова, то перенаправление осуществляется и для чтения.

Виртуализация выполняется не только для каталогов, но также и для элементов реестра. Если приложение пишет в ключ реестра Software в ветви HKEY_LOCAL_MACHINE, то запись перенаправляется в ветвь HKEY_CURRENT_USER.

Вместо записи в HKLM_Software\{Manufacturer} она пишет в HKCU\Software\Classes\VirtualStore\MACHINE\SOFTWARE\{Manufacturer}.

Файловая и реестровая виртуализация доступна только для 32-разрядных приложений, но не доступна для 64-разрядных приложений для Windows Vista.

Не применяйте файловую и реестровую виртуализацию в качестве средства вашего приложения. Лучше исправить приложение, чем писать в папку Program Files и ветвь HKLM реестра, не имея расширенных привилегий пользователя. Перенаправление — лишь временный способ исправить ошибочные приложения.

Новые элементы управления и диалоговые окна

Windows Vista предлагает несколько новых элементов управления. Элемент типа командной ссылки (command link) является расширением элемента управления типа Button и применяется с несколькими другими элементами управления. Диалог задач (task dialog) — это элемент MessageBox нового поколения, а для открытия и сохранения файлов также предусмотрены новые диалоги.

Командная ссылка

Элементы управления типа командных ссылок — это Windows-расширение элемента управления Button. Командные ссылки содержат необязательную пиктограмму и

текст примечания. Этот элемент управления часто используется в диалогах задач и мастерах. На рис. B.4 показано две командных ссылки, которые предоставляют намного больше информации, чем кнопки с надписями OK и Cancel (Отмена).

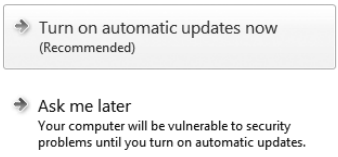

Рис. B.4. Командные ссылки

В приложениях .NET вы можете создавать командные ссылки, используя библиотеку примеров Vista Bridge. Если вы добавите проект `VistaBridgeLibrary` в решение, то сможете вставлять элементы `CommandLinkWinForms` из панели инструментов в свое приложение Windows Forms. Класс `CommandLinkWinForms` унаследован от класса `System.Windows.Forms.Button`. Командная ссылка — расширение "родного" Windows-элемента `Button`, определяющая дополнительные сообщения Windows и новый стиль для конфигурирования `Button`. Класс-оболочка `CommandLinkWinForms` посылает сообщения Windows `BCM_SETNOTE` и `BCM_SETSHIELD` и устанавливает стиль `BS_COMMANDLINK`. Общедоступные методы и свойства, добавленные к списку членов класса `Button` — это `NoteText` и `ShieldIcon`.

В следующем фрагменте кода создается новая командная ссылка с установкой `NoteText` и `ShieldIcon`. На рис. B.5 показано, как выглядит сконфигурированная командная ссылка во время выполнения.

```
this.commandLinkDemo =
   new Microsoft.SDK.Samples.VistaBridge.Library.CommandLinkWinForms();
this.commandLinkDemo.NoteText =
   "The application deletes important files on your system";
this.commandLinkDemo.ShieldIcon = true;
this.commandLinkDemo.Size = new System.Drawing.Size(275, 68);
this.commandLinkDemo.Text = "Give access to this computer";
this.commandLinkDemo.UseVisualStyleBackColor = true;
this.Controls.Add(commandLinkDemo);
```

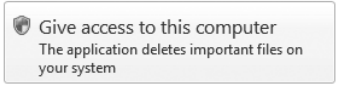

Рис. B.5. Сконфигурированная командная ссылка во время выполнения

Диалог задач

Диалог задач (task dialog) — это диалог следующего поколения, пришедший на смену старому окну сообщений (message box). Диалог задач — также один из новых элементов управления общего назначения. В Windows API определены функции `TaskDialog` и `TaskDialogIndirect` для создания диалогов задач. `TaskDialog` позволяет создавать простые диалоги, а `TaskDialogIndirect` служит для создания более сложных диалогов, содержащих командные ссылки и расширенное содержимое.

С помощью библиотеки Vista Bridge "родной" API-вызов `TaskDialogIndirect()` помещается в оболочку `PInvoke`:

```
[DllImport(ExternDll.ComCtl32, CharSet = CharSet.Auto, SetLastError = true)]
internal static extern HRESULT TaskDialogIndirect(
[In] NativeMethods.TASKDIALOGCONFIG pTaskConfig,
[Out] out int pnButton,
[Out] out int pnRadioButton,
[Out] out bool pVerificationFlagChecked);
```

Первый параметр `TaskDialogIndirect()` определен как класс `TASKDIALOGCONFIG`, отображаемый на ту же структуру, что и вызов "родного" API:

```
[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto, Pack = 4)]
internal class TASKDIALOGCONFIG
{
  internal uint cbSize;
  internal IntPtr hwndParent;
  internal IntPtr hInstance;
  internal TASKDIALOG_FLAGS dwFlags;
  internal TASKDIALOG_COMMON_BUTTON_FLAGS dwCommonButtons;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszWindowTitle;
  internal TASKDIALOGCONFIG_ICON_UNION MainIcon;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszMainInstruction;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszContent;
  internal uint cButtons;
  internal IntPtr pButtons;          // Указатель на структуры TASKDIALOG_BUTTON
  internal int nDefaultButton;
  internal uint cRadioButtons;
  internal IntPtr pRadioButtons;     // Указатель на структуры TASKDIALOG_BUTTON
  internal int nDefaultRadioButton;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszVerificationText;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszExpandedInformation;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszExpandedControlText;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszCollapsedControlText;
  internal TASKDIALOGCONFIG_ICON_UNION FooterIcon;
  [MarshalAs(UnmanagedType.LPWStr)]
  internal string pszFooter;
  internal PFTASKDIALOGCALLBACK pfCallback;
  internal IntPtr lpCallbackData;
  internal uint cxWidth;
}
```

Общедоступный класс из Vista Bridge, используемый для отображения диалогов задач — это `TaskDialog`. Чтобы отобразить пример диалога, должен быть вызван только статический метод `Show()`. Пример такого диалога показан на рис. B.6.

```
TaskDialog.Show("Simple Task Dialog");
```

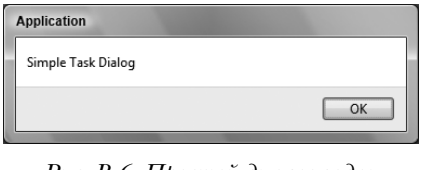

Рис. B.6. Простой диалог задач

Дополнительные свойства класса `TaskDialog` — `Caption`, `Content`, `StandardButtons` и `MainIcon` — устанавливаются отдельно. Результат можно увидеть на рис. В.7.

```
TaskDialog dlg1 = new TaskDialog();
dlg1.Caption = "Title";
dlg1.Content = "Some Information";
dlg1.StandardButtons = TaskDialogStandardButtons.OkCancel;
dlg1.MainIcon = TaskDialogStandardIcon.Information;
dlg1.Show();
```

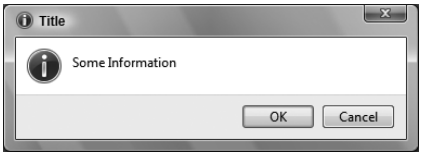

Рис. В.7. Диалог задач с дополнительными возможностями

Для диалога задач вы можете установить пиктограмму, которая сначала была показана в командных ссылках. Также вы можете расширить ее, установив свойство `ExpansionMode`. Перечислением `TaskDialogExpandedInformationLocation` вы можете специфицировать, что либо содержимое, либо нижний колонтитул должен быть расширен. На рис. В.8 показан диалог задач в свернутом состоянии. На рис. В.9 — он же в развернутом режиме.

```
TaskDialog dlg2 = new TaskDialog();
dlg2.Caption = "Title";
dlg2.Content = "Some Information";
dlg2.StandardButtons = TaskDialogStandardButtons.YesNo;
dlg2.MainIcon = TaskDialogStandardIcon.Shield;
dlg2.ExpandedText = "Additional Text";
dlg2.ExpandedControlText = "More information";
dlg2.CollapsedControlText = "Less information";
dlg2.ExpansionMode = TaskDialogExpandedInformationLocation.ExpandContent;
dlg2.FooterText = "Footer Information";
dlg2.FooterIcon = TaskDialogStandardIcon.Information;
dlg2.Show();
```

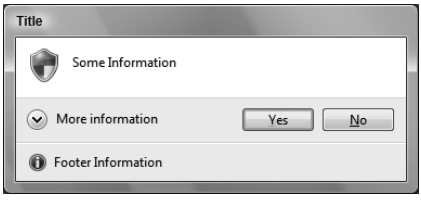

Рис. В.8. Диалог задач в свернутом состоянии

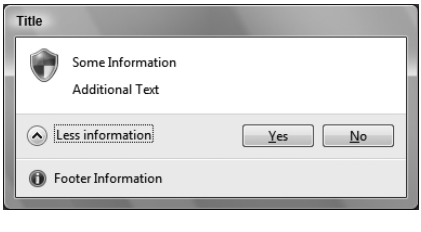

Рис. В.9. Диалог задач в развернутом состоянии

Диалог задач также содержит и другие элементы управления. В следующем фрагменте кода создается диалог задач, который содержит два переключателя, командную ссылку и элемент типа прямоугольника. Вы уже видели командные ссылки в предыдущем разделе; на самом деле, командные ссылки очень часто используются внутри диалогов задач. На рис. B.10 показан диалог задач с элементами управления в области содержимого. Конечно, вы также можете скомбинировать расширенную область с элементами управления.

```
TaskDialogRadioButton radio1 = new TaskDialogRadioButton();
radio1.Name = "radio1";
radio1.Text = "One";
TaskDialogRadioButton radio2 = new TaskDialogRadioButton();
radio2.Name = "radio2";
radio2.Text = "Two";
TaskDialogCommandLink commandLink = new TaskDialogCommandLink();
commandLink.Name = "link1";
commandLink.ShowElevationIcon = true;
commandLink.Text = "Information";
commandLink.Instruction = "Sample Command Link";
TaskDialogMarquee marquee = new TaskDialogMarquee();
marquee.Name = "marquee";
marquee.State = TaskDialogProgressBarState.Normal;
TaskDialog dlg3 = new TaskDialog();
dlg3.Caption = "Title";
dlg3.Instruction = "Sample Task Dialog";
dlg3.Controls.Add(radio1);
dlg3.Controls.Add(radio2);
dlg3.Controls.Add(commandLink);
dlg3.Controls.Add(marquee);
dlg3.Show();
```

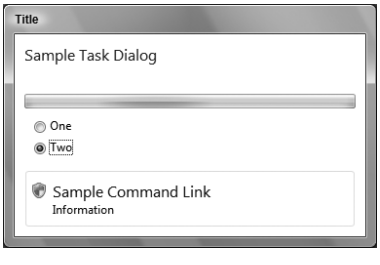

Рис. B.10. Диалог задач с элементами управления в области содержимого

Файловые диалоги

Диалоги открытия и сохранения файлов также изменились. На рис. B.11 показан традиционный диалог открытия файла, который имеет класс-оболочку Windows Forms `System.Windows.Forms.OpenFileDialog` и класс-оболочку WPF из сборки `PresentationFramework`: `Microsoft.Win32.OpenFileDialog`.

Новый диалог Windows Vista представлен на рис. B.12. Он содержит в себе панели Navigation (Навигация), Details (Подробности) и Preview (Предварительный просмотр), которые могут быть сконфигурированы через пункт меню Organize⇨Layout (Организовать⇨Компоновка). Этот диалог также содержит функциональность поиска и является полностью настраиваемым. В библиотеке Vista Bridge этот диалог имеет класс-оболочку `CommonOpenFileDialog`.

```
CommonOpenFileDialog dlg = new CommonOpenFileDialog();
dlg.ShowDialog();
```

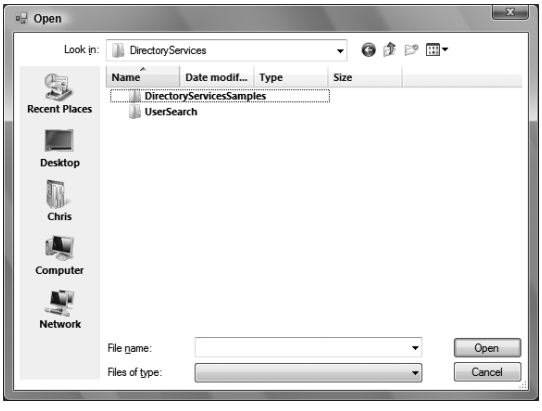

Рис. B.11. Традиционный диалог открытия файла

Рис. B.12. Диалог открытия файла в Windows Vista

Новый диалог Windows Vista, предназначенный для сохранения файлов, также является настраиваемым. По умолчанию он определяет свернутый (рис. B.13) и развернутый режимы (рис. B.14). Этому диалогу соответствует класс-оболочка `CommonSaveDialog`.

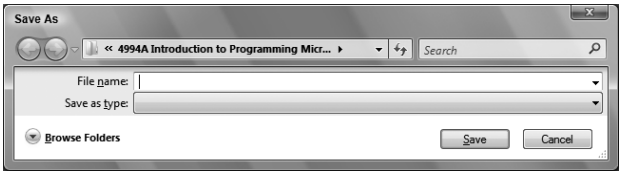

Рис. B.13. Диалог сохранения файлов Windows Vista в свернутом режиме

Поиск

Поиск – это важнейшее средство, с которым вы можете столкнуться во многих приложениях, инструментах и утилитах Windows Vista. Меню Start (Пуск) предостав-

ляет возможность поиска. Здесь вы можете искать программы, которые можно запустить. Иногда после использования поиска возникает потребность сохранить его результаты. В Windows XP довольно трудно найти программы через кнопку Start, когда на компьютере инсталлировано много систем. Теперь, благодаря новой функции поиска, эта задача значительно упростилась.

Выбирая пункт меню Search (Поиск), вы можете искать такие вещи, как электронные письма, документы, изображения, музыку и многое другое. В случае простого поиска вы просто вводите поисковую фразу в поле поиска, чтобы найти проиндексированные элементы. Расширенный поиск (рис. B.15) позволяет вводить имя, теги или автора, и определить места, где следует искать. На рис. B.16 показано подробное представление поисковой страницы, где можно выбрать все свойства, которые могут быть показаны в искомых элементах.

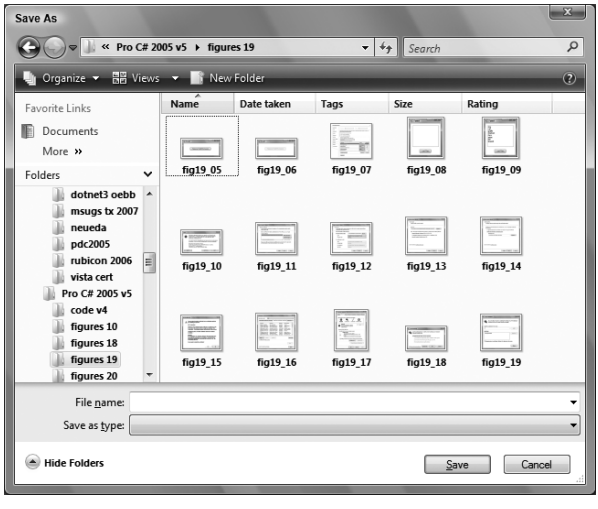

Рис. B.14. Диалог сохранения файлов Windows Vista в развернутом режиме

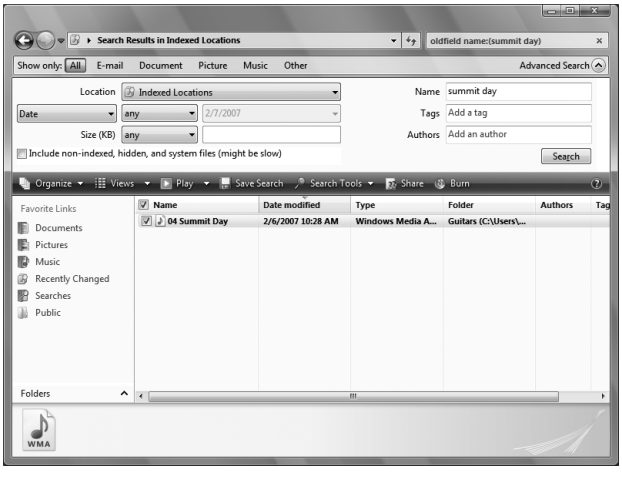

Рис. B.15. Окно расширенного поиска

Диалоги открытия и сохранения файлов Windows Vista также обладают встроенными возможностями поиска. Функция поиска может быть интегрирована в ваши приложения, и ваши приложения могут пользоваться всеми преимуществами поисковой функциональности Windows. Чтобы понять архитектуру средств поиска Windows, взгляните на рис. В.17. Сердцем поискового механизма является индексатор (indexer), который просматривает содержимое и записывает его в индекс содержимого. Для каждого хранилища (файловой системы, MAPI) дескриптор протокола отвечает за получение данных для индексатора. Протокольные дескрипторы реализуют интерфейс `IFilter`, используемый индексатором для анализа индексируемого содержимого. Система свойств описывает искомые свойства. Свойства описываются схемами свойств. Если приложение имеет собственный файловый формат, оно может реализовать дескриптор свойств для файлового формата. Если приложение имеет собственные свойства, по которым может осуществляться поиск, оно может добавлять эти свои свойства в систему свойств. Свойства определяются для основных файлов, документов Office, изображений и видеороликов. Дескрипторы свойств вызываются, когда индексируется содержимое, чтобы проанализировать свойства этого содержимого.

Рис. В.16. Установка подробностей для поисковой страницы

Рис. В.17. Архитектура средств поиска Windows

А теперь давайте воспользуемся системой запросов при построении поисковой функциональности для приложения.

Поставщик OLE DB

Вы можете интегрировать поисковую функциональность в свое приложение, используя поставщик OLE DB для элементов индекса. Создайте простое приложение Windows Forms с элементом `TextBox`, чтобы дать возможность пользователю вводить запросы, элементом `Button` для запуска запроса и элементом `ListView` для отображения результатов, как показано на рис. B.18. Измените свойство `View` элемента управления `ListView` на `Details`, чтобы отобразить всю информацию, введенную пользователем вместе с запросом.

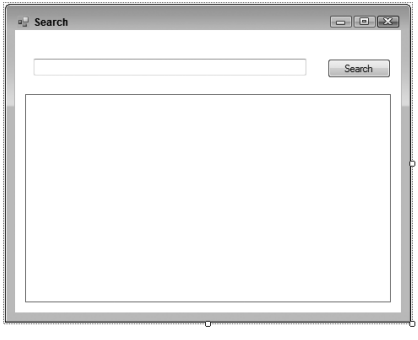

Рис. B.18. Простое приложение с функциональностью поиска

Импортируйте пространство имен `System.Data.OleDB` и добавьте следующий код в событие `Click` кнопки `Search`.

```
private void buttonSearch_Click(object sender, EventArgs e)
{
 try
 {
  listViewResult.Clear();
  string indexerConnectionString = "provider=Search.CollatorDSO.1;" +
            "EXTENDED PROPERTIES='Application=Windows'";
  OleDbConnection connection = new OleDbConnection(
            indexerConnectionString);
  connection.Open();
  OleDbCommand command = connection.CreateCommand();
  command.CommandText = textBoxQuery.Text;
  OleDbDataReader reader = command.ExecuteReader();
  DataTable schemaTable = reader.GetSchemaTable();
  foreach (DataRow row in schemaTable.Rows)
  {
    listViewResult.Columns.Add(row[0].ToString());
  }
  while (reader.Read())
  {
    ListViewItem item = new ListViewItem(reader[0].ToString());
    for (int i = 1; i < reader.FieldCount; i++)
    {
      item.SubItems.Add(reader[i].ToString());
    }
    listViewResult.Items.Add(item);
  }
```

```
    connection.Close();
   }
   catch (Exception ex)
   {
    MessageBox.Show(ex.Message);
   }
  }
```

Обратимся к деталям. Индексатор предоставляет поставщика OLE DB `Search.CollatorDSO`. В строке соединения OLE DB вы можете передать информацию о поставщике и открыть подключение к индексатору.

```
string indexerConnectionString = "provider=Search.CollatorDSO.1;" +
  "EXTENDED PROPERTIES='Application=Windows'";
OleDbConnection connection = new OleDbConnection(
  indexerConnectionString);
connection.Open();
```

Запрос, используемый с индексатором, читается из элемента управления `TextBox` по имени `textBoxQuery`. Поскольку во время компиляции не известно, какие свойства будут выбраны пользователем, столбцы в элемент управления `ListView` должны добавляться динамически. Метод `GetSchemaTable()` класса `OldDbDataReader` возвращает динамически созданную информацию о схеме, относящуюся к запросу. Каждая строка описывает элемент оператора `SELECT`, и первый столбец внутри элемента задает его имя. Проходя по каждой строке возвращенной схемы, добавляется новый столбец в элемент управления `ListView`, причем заголовок столбца устанавливается в имя элемента.

```
OleDbCommand command = connection.CreateCommand();
command.CommandText = textBoxQuery.Text;
OleDbDataReader reader = command.ExecuteReader();
DataTable schemaTable = reader.GetSchemaTable();
foreach (DataRow row in schemaTable.Rows)
{
   listViewResult.Columns.Add(row[0].ToString());
}
```

Затем читаются строки из `OleDbDataReader`. Первый столбец создает новый `ListViewItem`, а каждый последующий из результирующего набора добавляет подэлемент, который отображается вместе с подробной информацией списочного представления.

```
while (reader.Read())
{
  ListViewItem item = new ListViewItem(reader[0].ToString());
  for (int i = 1; i < reader.FieldCount; i++)
  {
       item.SubItems.Add(reader[i].ToString());
  }
  listViewResult.Items.Add(item);
}
```

Теперь вы можете запустить приложение, ввести запрос и получить результат (рис. В.19).

```
SELECT System.ItemName, System.ItemTitle, System.Size FROM SYSTEMINDEX
   WHERE System.Size > 1024
```

В операторе `SELECT` запроса вы специфицируете свойства, которые должны быть возвращены. `System.ItemName`, `System.ItemTitle` и `System.Size` — все это предопределенные свойства. Вы можете найти другие предопределенные свойства в доку-

ментации MSDN и TechNet по теме “Windows Desktop Search 3.0 properties”. Вот некоторые из общих свойств файлов: `System.Author`, `System.Category`, `System.Company`, `System.DateCreated`, `System.DateModified`, `System.FileName`, `System.ItemName`, `System.ItemUrl` и `System.Keywords`. Для аудиофайлов, цифровых фотографий, графических файлов, медиафайлов, документов Office, музыкальных файлов и элементов календаря Outlook определены дополнительные свойства, например, `System.Photo.Orientation`, `System.Photo.DateTaken`, `System.Music.Artist`, `System.Music.BeatsPerMinute`, `System.Music.Mood`, `System.Calendar.Location`, `System.Calendar.Duration`, `System.Calendar.Location`.

В конструкции `WHERE` вы определяете предикаты, такие как сравнения литеральных значений <, >, = и `LIKE`, условия полнотекстового поиска вроде `CONTAINS` и `FREETEXT`, а также предикаты глубины поиска `SCOPE` и `DIRECTORY`.

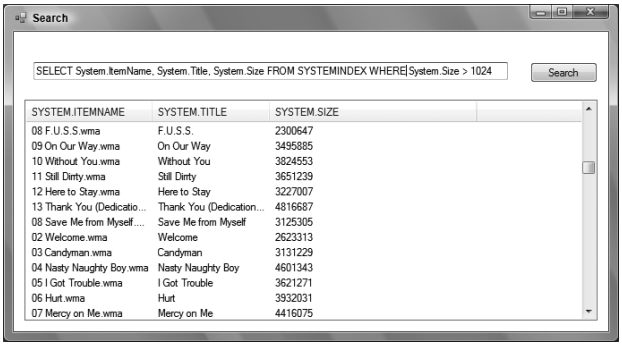

Рис. B.19. Простое поисковое приложение во время выполнения

Расширенный синтаксис запросов

Вряд ли вы пожелаете предоставлять пользователю возможность поиска путем спецификации оператора `SELECT` — вроде того, что применялось в предыдущем примере. Вы можете создать пользовательский интерфейс, чтобы запрашивать определенные элементы и затем программно строить оператор `SELECT`. Другой способ позволить пользователям выполнять их собственный поиск состоит в применении расширенного синтаксиса запросов AQS (Advanced Query Syntax).

Расширенный синтаксис запросов позволяет специфицировать поисковые элементы и ограничивать поиск на основе свойств. Например, запрос `Wrox date:past week` запустит поиск всех элементов, содержащих строку `Wrox`, которые были изменены в течение последней недели. `Wrox date:past week kind:documents` ограничит поиск только нужными документами.

Ниже дано несколько примеров того, как можно сузить поиск.

- Можно ограничить поиск, определив хранилище. Например, `store:outlook` выдаст только элементы из Outlook. `store:file` позволит получить элементы только из файловой системы.
- В поисковой функциональности вы можете специфицировать, какого рода элементы должны попасть в результат, например: `kind:text`, `kind:tasks`, `kind:contacts`, `kind:emails` и `kind:folders`.
- Для ограничения поиска могут быть применены булевские операции. Например, операция `OR`, использованная в запросе `Wrox OR Wiley. date:>11/25/07`, позволит найти элементы, датированные после 25 ноября 2007 г.

Для элементов, относящихся к промежутку между двумя датами, следует указывать `11/25/06..11/27/07`.

- Вы можете использовать некоторые свойства элементов, например, `webpage:www.wrox.com`, `birthday:2/14/65`, `firstname:Christian`.

Вам не нужно вручную транслировать AQS в запрос `SELECT`, поскольку имеется COM-объект, который сделает это за вас. В каталоге Windows SDK Lib можно найти файл `SearchAPI.tlb`. Это библиотека типов, описывающая COM-объект, используемый для трансляции AQS. Применяя COM Interop, вы можете использовать COM-объекты из .NET.

Создайте вызываемую оболочку .NET с помощью утилиты `tblimp` для импорта библиотеки `SearchAPI.tlb`:

```
tlbimp c:\Program Files\Microsoft SDKs\Windows\v6.0\Lib\SearchAPI.tlb
  /out:Interop.SearchAPI.dll
```

Технология COM Interop описана в главе 24.

Обращение к сгенерированной сборке COM Interop из проекта Windows Forms, созданного ранее, позволяет использовать `SearchAPI` из приложения .NET. Поскольку это сгенерированная сборка, импортируйте пространство имен `Interop.SearchAPI` из приложения и добавьте метод `GetSql()` в класс Windows Forms.

Классы `CSearchManager`, `CsearchCatalogManager` и `CSearchQueryHelper` сгенерированы утилитой `tlbimp` для вызова COM-объектов. Метод `GetCatalog()` определяет опрашиваемый каталог и возвращает `catalogManager`. С экземпляром `catalogManager` метод `GetQueryHelper()` возвращает вспомогательный объект. Передача строки AQS методу `GenerateSQLFromUserQuery()` возвращает запрос `SELECT`, который может быть использован с поставщиком OLE DB для выполнения запроса:

```
private string GetSql(string aqs)
{
  CSearchManager searchManager = new CSearchManager();
  CSearchCatalogManager catalogManager =
     searchManager.GetCatalog("SystemIndex");
  CSearchQueryHelper queryHelper = catalogManager.GetQueryHelper();
  return queryHelper.GenerateSQLFromUserQuery(aqs);
}
```

Теперь вам нужно только изменить реализацию обработчика `Click` элемента управления `Button` для вызова метода `GetSql()` с целью преобразования AQS в запрос `SELECT`, используемый приложением.

```
private void buttonSearch_Click(object sender, EventArgs e)
{
 try
 {
  listViewResult.Clear();
  string indexerConnectionString = "provider=Search.CollatorDSO.1;" +
       "EXTENDED PROPERTIES='Application=Windows'";
  OleDbConnection connection = new
       OleDbConnection(indexerConnectionString);
  connection.Open();
  OleDbCommand command = connection.CreateCommand();
  command.CommandText = GetSql(textBoxQuery.Text);
  OleDbDataReader reader = command.ExecuteReader();
  //...
```

Теперь вы сможете запустить приложение и передать запрос AQS, как показано на рис. B.20.

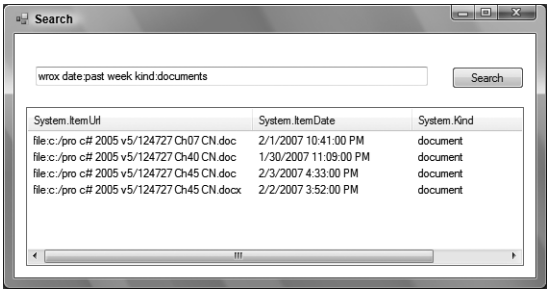

Рис. B.20. Передача запроса AQS

Резюме

В этом приложении вы ознакомились с разнообразными средствами, доступными только в Windows Vista и Windows Server 2008, и представляющими ценность для разработки приложений.

Microsoft давно разработала рекомендации о том, что неадминистративные приложения не должны требовать административных привилегий. Но поскольку многие приложения все-таки не соответствуют этому требованию, теперь операционная система стала строже в отношении UAC. Пользователь должен явно повышать уровень административных прав для приложений. Вы увидели, как это делается — посредством виртуализации папок и реестра — и как это затрагивает приложения.

В настоящем приложении мы описали несколько новых диалогов, доступных только под Windows Vista и обеспечивающих лучшее взаимодействие с пользователем. К ним относятся диалоги открытия и сохранения файлов, новый диалог задач, заменивший традиционное окно сообщений, а также командная ссылка — расширение элемента управления `Button`.

Также вы ознакомились с системой запросов Windows, использующей новый расширенный синтаксис запросов, а также расширяемой системой свойств, которые позволяют интегрировать систему поиска в разрабатываемые приложения.

Некоторые другие средства, доступные только в Windows Vista и Windows Server 2008, рассматриваются в других главах:

- глава 18 посвящена новому средству протоколирования — Event Tracing for Windows (ETW);
- глава 20 содержит информацию о Cryptography Next Generation (CNG) — новом программном интерфейсе криптографии;
- в главе 22 рассказывается о файловых и реестровых транзакциях;
- в главе 42 показано применение Windows Activation Services (WAS) для размещения службы WCF.