Aplikace

Pokud jde o konfiguraci a ovládání aplikací a práci s prostředky, mnohé se změnilo. Proto byla tato kapitola od základu přepsána. Ale žádný strach – postupy pro konfigurační soubory, popsané ve staré knize kódů, stále ještě fungují, stávající kód se měnit nemusí.

Změny, týkající se správy konfiguračních dat, se týkají jak prostředí .NET, tak i Visual Studia. Některé nové třídy dovolují cílené přístupy ke konfiguračním souborům a v současnosti také umožňují uložení uživatelsky specifických dat pomocí aplikace. Vývojové prostředí bylo rozšířeno pomocí průvodců a automatismů, které pokrývají a podstatně zjednodušují mnohé obvyklé případy aplikací.

48 Vytvoření konfigurace aplikace pomocí Visual Studia

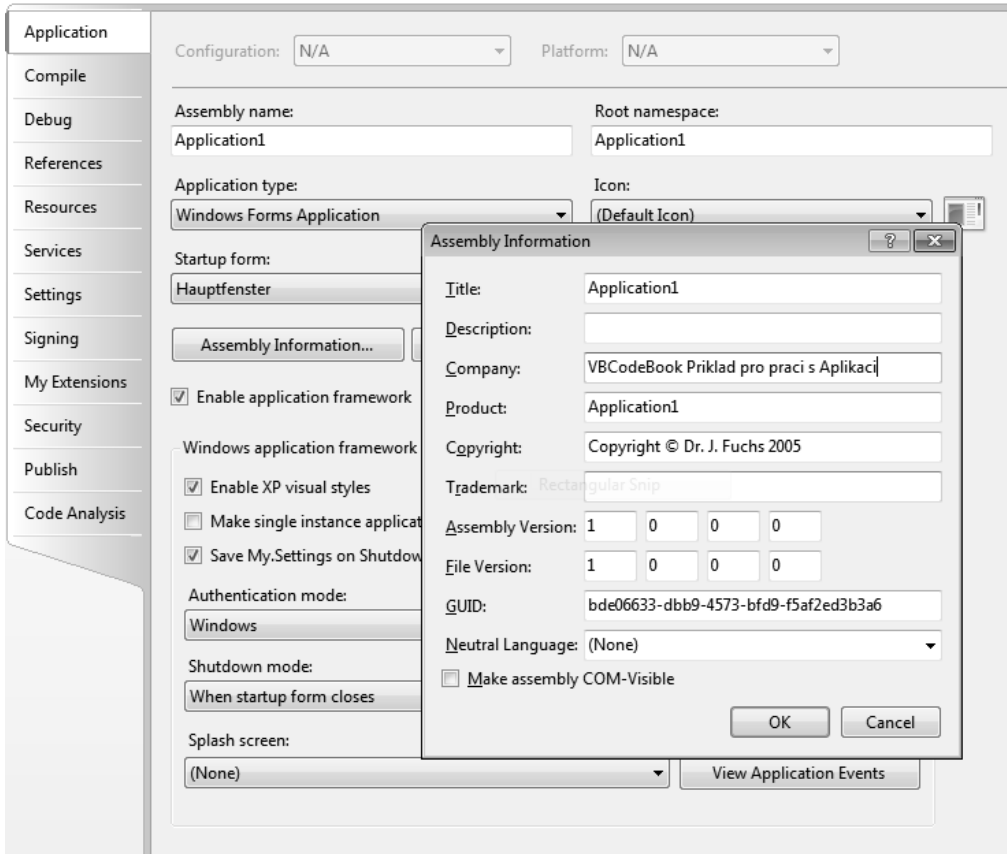
Pomocí návrháře lze vytvořit dva druhy konfiguračních dat: data specifická pro aplikaci a data specifická pro uživatele. Data specifická pro aplikaci jsou součástí aplikace a ukládají se jako soubor XML v domovském adresáři aplikace (tam, kde je uložen soubor .exe). Tato data by se pokud možno neměla při běhu programu měnit, v žádném případě neslouží pro ukládání uživatelských dat. Spuštěná aplikace také často nemá dostatečná oprávnění, aby mohla provádět změny v konfiguračních souborech aplikace. Většinou se jedná o statické informace, které se musí měnit jen zřídkakdy.

Naproti tomu uživatelská konfigurační data se ukládají jako soubor XML v domovském adresáři přihlášeného uživatele. Nacházejí se většinou v adresáři jako `C:\Documents and Settings\Jméno uživatele\Local Settings\Data aplikací\Jméno firmy\Jméno aplikace\1.0.0.0`. Skutečnost, že jméno firmy je standardně součástí cesty, naznačuje, že toto jméno musíme v aplikaci také správně zadat. Jak ukazuje obrázek 16, může se to provést pomocí vlastností projektu na kartě Aplikace, tlačítkem Assembly Information.

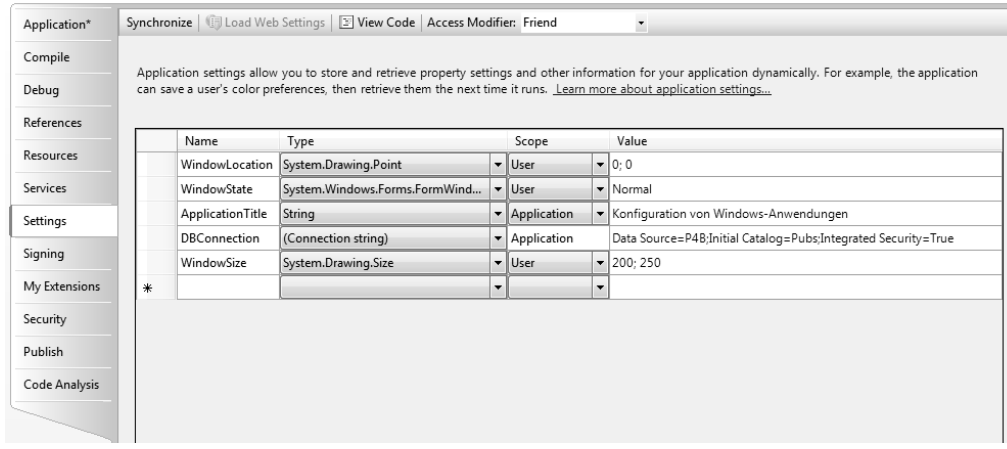
Jméno firmy se ovšem bere v úvahu jen do určité délky. Na obrázku uvedené jméno se po zhruba 25 znacích zalomí. Na to byste měli ve svých aplikacích dávat pozor.

Jednodušší je také definice konfiguračních souborů. Už nemusíte ručně zakládat soubor `App.config`; konfiguraci aplikace využijete i k zadávání dat (karta Settings) (viz obrázek 17). Nastavení aplikace můžete také provést pomocí položky Settings.Settings v průzkumníku projektu. V prvním sloupci zadáte jméno vlastnosti konfigurace, ve druhém sloupci zvolíte typ. Přitom můžete použít různé datové typy a nejste omezeni pouze na znakové řetězce. Jak ukazuje obrázek, můžete použít také výčty jako Point a Size.

Ve sloupci Scope zvolíte, zda vlastnost má být specifická pro aplikaci nebo specifická pro uživatele. Ve čtvrtém sloupci předvolíte implicitní hodnotu vlastnosti. Hodnoty zadáváte v textové podobě a musíte je proto sestavit tak, aby určený datový typ uměl tento text také interpretovat. Syntaxe je závislá na typu a odpovídá zobrazování příslušných hodnot v okně vlastností (`PropertyGrid`).

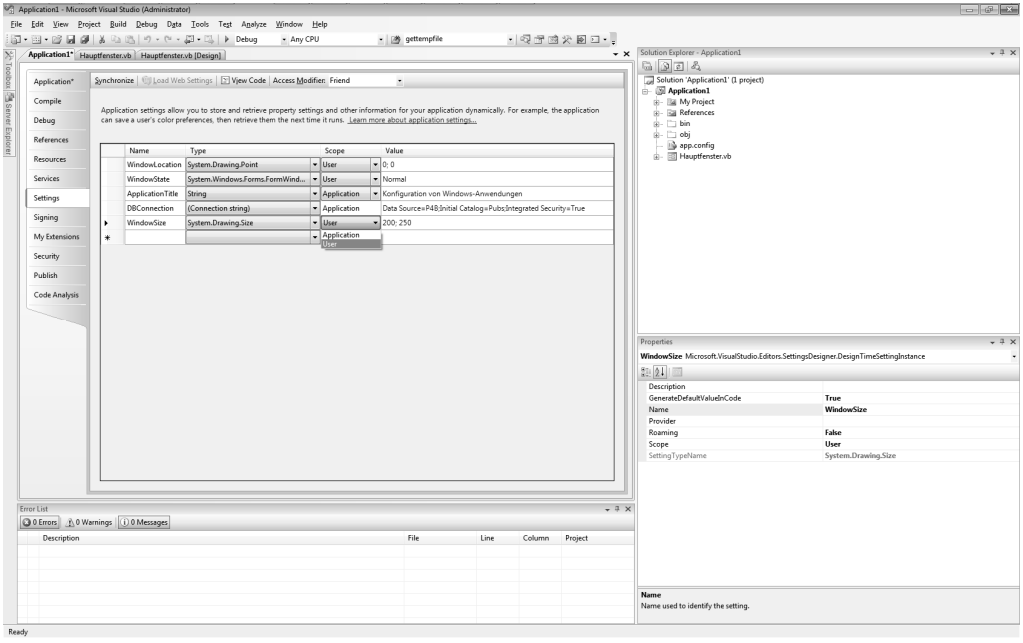


Obrázek 16: *Jméno firmy hraje při uložení uživatelských konfiguračních dat důležitou roli*



Obrázek 17: *Zadání uživatelských dat a výběr datového typu*

Pro vlastnosti specifické pro aplikaci zvolte odpovídající položku ve sloupci SCOPE (viz obrázek 18).



Obrázek 18: *Lze definovat také statické vlastnosti, které jsou specifické pro aplikaci a jsou chráněné proti zápisu*

Jak ukazuje výpis 48, z hodnot v tabulce návrhář automaticky vygeneruje konfigurační soubor. Ten se jako `App.config` přidá do projektu a při spuštění aplikace se zkopíruje do binárního adresáře pod jménem `Jméno aplikace.exe.config`. V prvku `configSections` se deklarují všechny konfigurační skupiny. K nim v našem případě patří `userSettings` a `ApplicationSetting`. Pro obě skupiny je níže uvedena datová část, v níž zase najdete zadané vlastnosti a jejich hodnoty.

Výpis 48: *Soubor XML app.config, generovaný návrhářem*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="userSettings"
      type="System.Configuration.UserSettingsGroup,
      System, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" >
        <section name="Application1.My.MySettings"
          type="System.Configuration.ClientSettingsSection,
          System, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b77a5c561934e089"
          allowExeDefinition="MachineToLocalUser"
          requirePermission="false" />
    </sectionGroup>
    <sectionGroup name="applicationSettings"
      type="System.Configuration.ApplicationSettingsGroup,
      System, Version=2.0.0.0, Culture=neutral,
```

```
      PublicKeyToken=b77a5c561934e089" >
        <section name="Application1.My.MySettings"
          type="System.Configuration.ClientSettingsSection,
          System, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b77a5c561934e089"
          requirePermission="false" />
    </sectionGroup>
  </configSections>

  <system.diagnostics>
      ...
  </system.diagnostics>
      <userSettings>
    <Application1.My.MySettings>
      <setting name="WindowLocation" serializeAs="String">
        <value>0, 0</value>
      </setting>
      <setting name="WindowSize" serializeAs="String">
        <value>200, 150</value>
      </setting>
      <setting name="WindowState" serializeAs="String">
        <value>Normal</value>
      </setting>
    </Application1.My.MySettings>
  </userSettings>
  <applicationSettings>
    <Application1.My.MySettings>
      <setting name="ApplicationTitle" serializeAs="String">
        <value>Konfigurace aplikací ve Windows</value>
      </setting>
    </Application1.My.MySettings>
  </applicationSettings>
</configuration>
```

Visual Studio 2008 ale nezůstane jen při vytvoření souboru XML – jde ještě o jeden krok dále. Do projektu se pod SETTINGS.SETTINGS přidá kódový soubor `Settings.Designer.vb` (výpis 49). Tato třída obsahuje definice tříd pro zadané vlastnosti. Pro každou vlastnost, která byla pomocí tabulky pro nastavení aplikace definována, se deklaruje vlastnost (`Property`). Uživatelsky definované vlastnosti je možno číst a zapisovat, naproti tomu vlastnosti specifické pro aplikaci jsou chráněny proti zápisu.

Pomocí atributů se definice vlastnosti přiřadí různým oblastem (`UserScopedSettingAttribute` resp. `ApplicationScopedSettingAttribute`). Počáteční hodnoty se stanoví atributem `DefaultSettingValueAttribute`.

Výpis 49: *Kód pro typově bezpečné začlenění konfiguračních dat, automaticky generovaný návrhářem*

```
Namespace My

  <Global.System.Runtime.CompilerServices...> _
  Partial Friend NotInheritable Class MySettings
    Inherits Global.System.Configuration.ApplicationSettingsBase
```

```
    Private Shared defaultInstance As MySettings = _
      CType(Global.System.Configuration.ApplicationSettingsBase. _
      Synchronized(New MySettings), MySettings)

#Region " My.Settings Auto-Save Functionality "
...
#End Region

    Public Shared ReadOnly Property [Default]() As MySettings
      Get

#If _MyType = "WindowsForms" Then
        If Not addedHandler Then
          SyncLock addedHandlerLockObject
            If Not addedHandler Then
              AddHandler My.Application.Shutdown, _
                AddressOf AutoSaveSettings
              addedHandler = True
            End If
          End SyncLock
        End If
#End If
        Return defaultInstance
      End Get
    End Property

    <Global.System.Configuration.UserScopedSettingAttribute(), _
     Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
     Global.System.Configuration. _
       DefaultSettingValueAttribute("0, 0")> _
    Public Property WindowLocation() As _
      Global.System.Drawing.Point
      Get
        Return CType(Me("WindowLocation"), _
          Global.System.Drawing.Point)
      End Get
      Set(ByVal value As Global.System.Drawing.Point)
        Me("WindowLocation") = Value
      End Set
    End Property

    <Global.System.Configuration.UserScopedSettingAttribute(), _
     Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
     Global.System.Configuration. _
       DefaultSettingValueAttribute("200, 150")> _
    Public Property WindowSize() As Global.System.Drawing.Size
      Get
        Return CType(Me("WindowSize"), Global.System.Drawing.Size)
      End Get
      Set(ByVal value As Global.System.Drawing.Size)
        Me("WindowSize") = Value
      End Set
```

```
    End Property

    <Global.System.Configuration.UserScopedSettingAttribute(), _
     Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
     Global.System.Configuration. _
       DefaultSettingValueAttribute("Normal")> _
    Public Property WindowState() As _
      Global.System.Windows.Forms.FormWindowState
      Get
        Return CType(Me("WindowState"), _
          Global.System.Windows.Forms.FormWindowState)
      End Get
      Set(ByVal value As _
        Global.System.Windows.Forms.FormWindowState)
        Me("WindowState") = Value
      End Set
    End Property

    <Global.System.Configuration. _
      ApplicationScopedSettingAttribute(), _
     Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
     Global.System.Configuration.DefaultSettingValueAttribute _
       ("Konfigurace aplikací ve Windows")> _
    Public ReadOnly Property ApplicationTitle() As String
      Get
        Return CType(Me("ApplicationTitle"), String)
      End Get
    End Property
  End Class
End Namespace

Namespace My

  <Global.Microsoft.VisualBasic.HideModuleNameAttribute(), _
   Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
   Global.System.Runtime.CompilerServices. _
     CompilerGeneratedAttribute()> _
  Friend Module MySettingsProperty

    <Global.System.ComponentModel.Design.HelpKeywordAttribute _
      ("My.Settings")> _
    Friend ReadOnly Property Settings() As _
      Global.Application1.My.MySettings
      Get
        Return Global.Application1.My.MySettings.Default
      End Get
    End Property
  End Module
End Namespace
```

Visual Basic 2008 přiřazuje jmennému prostoru `My` třídu. Pomocí `My.Settings` tak máte přístup ke všem konfiguračním datům. Ve výpisu 50 vidíte malý příklad použití, kdy se konfigurační data načtou a využijí v události `Load` okna, a při zavření tohoto okna se opět uloží.

Výpis 50: *Načtení a uložení konfiguračních dat*

```
Private Sub HlavníOkno_Load(...) Handles MyBase.Load
  Me.Location = My.Settings.WindowLocation
  Me.Size = My.Settings.WindowSize
  Me.WindowState = My.Settings.WindowState
  Me.Text = My.Settings.ApplicationTitle


  ToolStripStatusLabel1.Text = "Počáteční velikost okna: " & _
    My.Settings.WindowSize.ToString() & ", Pozice: " & _
    My.Settings.WindowLocation.ToString() & ", Stav: " & _
    My.Settings.WindowState

End Sub

Protected Overrides Sub OnClosing(...)
  If Me.WindowState = FormWindowState.Normal Then
    My.Settings.WindowLocation = Me.Location
    My.Settings.WindowSize = Me.Size
  End If
  My.Settings.WindowState = Me.WindowState

  My.Settings.Save()

  MyBase.OnClosing(e)
End Sub
```

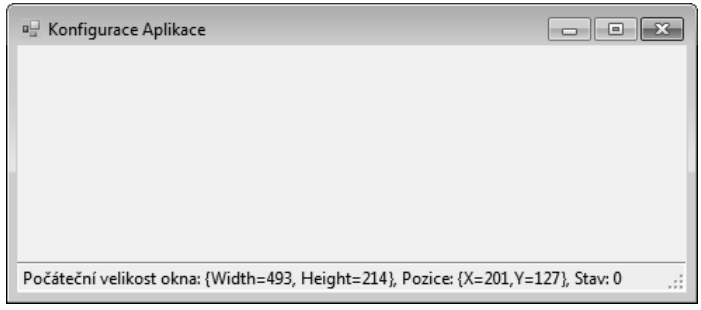
Volání `My.Settings.Save()` způsobí uložení uživatelsky specifických vlastností v domovském adresáři přihlášeného uživatele. Skutečné místo uložení záleží na tom, zda byla aplikace spuštěna prostřednictvím vývojového prostředí nebo jinak. V jednotlivých případech si sami prověřte, jaké adresáře byly pod `C:\Documents and Settings\Jméno uživatele\Local Settings\Data aplikací\Jméno firmy\` založeny.

Soubor se ovšem vytvoří jen tehdy, jestliže se vlastnosti odlišují od počátečních hodnot konfiguračního souboru aplikace. Možnou konstrukci uživatelského souboru `user.config` ukazuje výpis 51, okno příkladu aplikace ukazuje obrázek 19.

Výpis 51: *Uživatelská data se ukládají v souboru user.config.*

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <userSettings>
    <Application1.My.MySettings>
      <setting name="WindowLocation" serializeAs="String">
        <value>117, 322</value>
      </setting>
      <setting name="WindowSize" serializeAs="String">
        <value>511, 154</value>
      </setting>
```

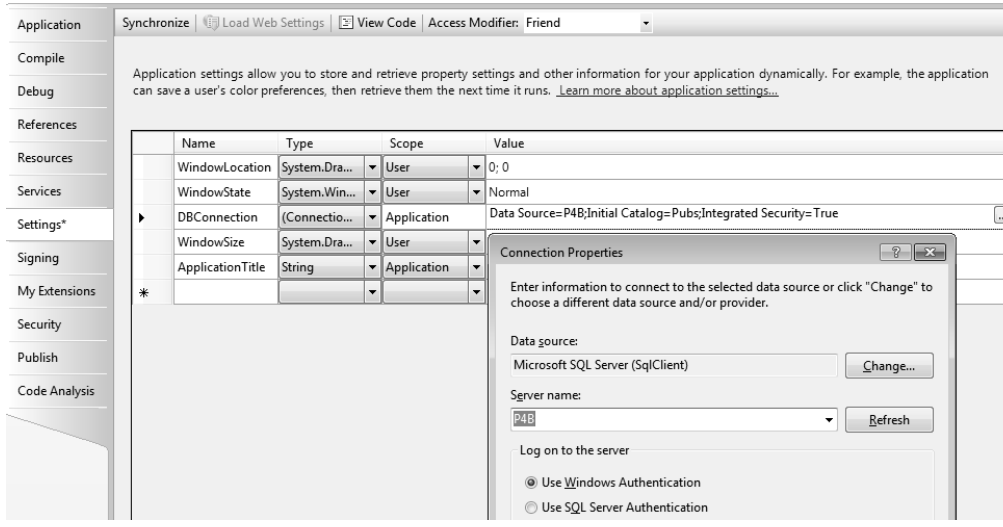
```
      <setting name="WindowState" serializeAs="String">
        <value>Normal</value>
      </setting>
    </Application1.My.MySettings>
  </userSettings>
</configuration>
```



Obrázek 19: *Uložená data okna se použijí při spuštění aplikace a změny se při zavření okna uloží*

49 Uložení konfigurace pro připojení k databázi (se šifrováním a bez šifrování)

Do konfiguračního souboru se dají prostřednictvím návrháře zanést také připojovací řetězce (connection strings) pro přístup k databázím. Jako datový typ se k tomuto účelu volí `Connection string` (viz obrázek 20). Pomocí tlačítka ve sloupci VALUE lze spustit známého Průvodce pro spojení s databází. Vygenerovaný připojovací řetězec je pak možné pomocí `My.Settings` v programu načíst.



Obrázek 20: *Definice připojovacího řetězce pro přístup k databázím*

V konfiguračním souboru se založí vlastní sekce pro připojovací řetězce (výpis 52), pro každé spojení v něm bude jedna položka. Řetězce se přirozeně ukládají jako otevřený text (nešifrované).

Výpis 52: *Připojovací řetězec pro připojení k databázi v konfiguračním souboru*

```
<connectionStrings>
  <add name="Application1.My.MySettings.DBConnection"
       connectionString="Data Source=P4B;Initial Catalog=Pubs;
         Integrated Security=True"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Šifrování dat

Zatímco u souboru `web.config` nejsou obsažená data normálnímu uživateli tak jako tak přístupná, protože server prohlížeči tento soubor neposílá, může připojovací řetězec v nešifrovaném stavu znamenat pro aplikaci ve Windows bezpečnostní riziko. Potom má za určitých okolností smysl tuto sekci zašifrovat.

Nový .NET poskytuje metody i k tomuto účelu. Výpis 53 ukazuje, jak lze zašifrovat všechna data příslušné sekce konfiguračního souboru. Změněný úsek konfiguračního souboru je vidět na výpisu 54. V příkladu aplikace se připojovací řetězec načítá při spuštění a zobrazuje se v komponentě typu `Label`. Pomocí dvou tlačítek se dá oblast konfiguračního souboru zašifrovat a dešifrovat. Kód pro dešifrování ukazuje výpis 55, konečný výsledek je na obrázku 21.

Výpis 53: *Zašifrování připojovacího řetězce v konfiguračním souboru*

```
' Otevření konfiguračního souboru pomocí třídy ConfigurationManager
Dim config As Configuration = _
  ConfigurationManager.OpenExeConfiguration( _
    ConfigurationUserLevel.None)

' Přístup k sekci s připojovacím řetězcem
Dim cs As ConnectionStringsSection = config.ConnectionStrings

' Zašifrování
cs.SectionInformation.ProtectSection( _
  "DataProtectionConfigurationProvider")

' Uložení konfigurace
config.Save()
```

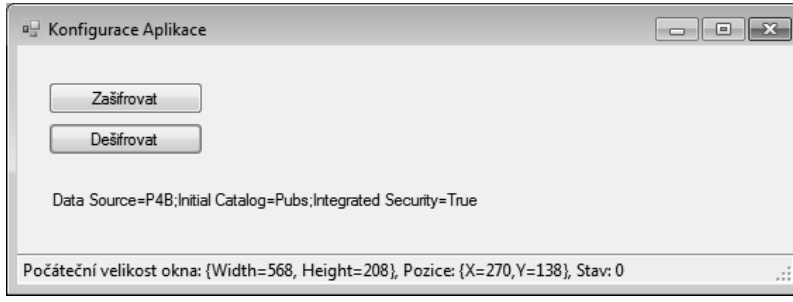
Výpis 54: *Data pro připojení jsou nyní chráněna před dotěrnými pohledy*

```
<connectionStrings configProtectionProvider="DataProtectionConfigurationProvider">
  <EncryptedData>
    <CipherData>
      <CipherValue>AQAAANCMnd8BFd4jvKQi ... </CipherValue>
    </CipherData>
  </EncryptedData>
</connectionStrings>
```

Výpis 55: *Dešifrování zašifrované sekce*

```
' Otevření konfiguračního souboru pomocí třídy ConfigurationManager
Dim config As Configuration = _
  ConfigurationManager.OpenExeConfiguration( _
    ConfigurationUserLevel.None)
```

```
' Přístup k sekci s připojovacím řetězcem
Dim cs As ConnectionStringsSection = config.ConnectionStrings

' Jen pokud existuje šifrování
If cs.SectionInformation.IsProtected Then

  ' Dešifrování
  cs.SectionInformation.UnprotectSection()

  ' Uložení konfigurace
  config.Save()
End If
```



Obrázek 21: *Ukázková aplikace, umožňující šifrování a dešifrování připojovacích řetězců*

Upozornění

Při testování tuto aplikaci spusťte raději pomocí Průzkumníku a nikoli v režimu ladění. Jinak totiž vývojové prostředí po ukončení programu konfigurační soubory opět překopíruje, takže provedené změny se ztratí. Dále mějte na paměti, že vývojové prostředí založí v binárním adresáři dva soubory typu `.config`, z nichž jeden se využívá pro účely ladění.

Ke změně konfiguračního souboru musí aplikace disponovat dostatečnými oprávněními.

Shora popsané akce jsou možné díky třídě `ConfigurationManager`, která v .NET 2.0 poskytuje centrální přístup ke konfiguračním souborům. Návrhářem generované třídy zde už nepomůžou. Navíc návrhářem generovaný kód nemůže v sekci dat aplikace provést žádnou změnu. Pomocí třídy `ConfigurationManager` to možné je, disponuje-li ovšem aplikace dostatečnými oprávněními.

Upozornění

Chcete-li využít třídu `ConfigurationManager`, musíte do projektu přidat odkaz na knihovnu `System.configuration.dll`.

50 Dodatečné sekce v konfiguračním souboru

Třída `ConfigurationManager` je klíčem pro všechny druhy přístupu ke konfiguračním souborům. Poskytuje různé metody pro otevírání a ukládání souborů, stejně jako pro zacházení s jednotli-

vými sekcemi. Například metoda `GetSection` otevře sekci a zpřístupní objekt, jehož prostřednictvím se lze dostat k datům. Typ tohoto objektu závisí na deklaracích v konfiguračním souboru. Je možné používat i vlastní typy.

Každá sekce musí být v konfiguračním souboru deklarována v prvku `configSections`. Pro sekci musí být určeno jméno a typ obsluhy. V příkladu ve výpisu 56 deklarujeme sekci `BookSection` a jako typ obsluhy `Application2.BookInfo`.

Výpis 56: *Výpis pro vlastní definovanou sekci*

```
<configuration>
    <configSections>
      ...
      <section name="BookSection"
               type="Application2.BookInfo, Application2"/>
    </configSections>

  ...

  <BookSection Title=" VB-Kniha příkladů " Publisher=" Computer press "/>

</configuration>
```

Volání metody `ConfigurationManager.GetSection` pak vede k vytvoření instancí zadaného typu. V našem příkladu je k dispozici dodatečná třída, která dokáže vyhodnotit informace této sekce (třída `BookInfo`, výpis 57). Pro obě vlastnosti jsou implementovány dvě veřejné vlastnosti, mající atribut `StringValidator` pro rozpoznání možných chyb konfigurace. Interně pracují s indexerem základní třídy `ConfigurationSection`. Prostřednictvím jména vlastnosti jako indexu se lze při čtení a zapisování dostat k hodnotě.

Aby objekt věděl, jaké vlastnosti odpovídají kterému typu, deklarují se tyto vlastnosti v konstruktoru. Do seznamu `Properties` se přidají instance třídy `ConfigurationProperty`, které obsahují potřebné informace.

Výpis 57: *Obsluha pro vlastní sekci konfiguračního souboru*

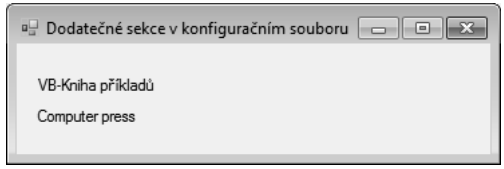
```
Imports System.Configuration
Public Class BookInfo
  Inherits System.Configuration.ConfigurationSection

  Public Sub New()

    ' Přidání vlastnosti Title
    Dim propTitle As New _
      ConfigurationProperty("Title", GetType(String))
    Me.Properties.Add(propTitle)

    ' Přidání vlastnosti Publisher
    Dim propPublisher As New _
      ConfigurationProperty("Publisher", GetType(String))
    Me.Properties.Add(propPublisher)

  End Sub
```

```
  ' Vlastnost Title
  <StringValidator( _
  MinLength:=1, MaxLength:=50)> _
  Public Property Title() As String
    Get
      Return CStr(Me("Title"))
    End Get
    Set(ByVal value As String)
      Me("Title") = value
    End Set
  End Property

  ' Vlastnost Publisher
  <StringValidator( _
  InvalidCharacters:=" !@$&;", _
  MinLength:=1, MaxLength:=100)> _
  Public Property Publisher() As String
    Get
      Return CStr(Me("Publisher"))
    End Get
    Set(ByVal value As String)
      Me("Publisher") = value
    End Set
  End Property

End Class
```

V události `Load` aplikace z příkladu se prostřednictvím `GetSection` vytvoří instance třídy `BookInfo` a sekce se načte. Uložené hodnoty se pak dají získat pomocí poskytnutých vlastností. Obrázek 22 ukazuje okno programu z tohoto příkladu, výpis 58 pak implementaci.

Výpis 58: *Načtení dodatečné sekce*

```
' Čtení sekce
Dim bi As BookInfo = CType( _
  ConfigurationManager.GetSection("BookSection"), BookInfo)

' Vyvolání hodnot
Label1.Text = bi.Title
Label2.Text = bi.Publisher

' Čtení hodnoty z My.Settings
Me.Text = My.Settings.Applicationname
```



Obrázek 22: *Čtení informací z vlastních sekcí*

Upozornění

Při práci s konfiguračními soubory dejte pozor na to, že v konfiguračních souborech se rozlišuje mezi velkými a malými písmeny. Má-li popisovaný postup vést k úspěchu, musí být všechna jména příznaků a atributů správně zapsána.

Možnosti pro vlastní sekce jsou velmi rozmanité a nelze je zde kompletně představit. Celá řada tříd platformy .NET je využitelná přímo nebo může sloužit jako základní třída vašich vlastních tříd, které rozšíříte o dodatečné funkce. V příkladu použitá základní třída `ConfigurationSection` nabízí četné metody, které lze přepsat, jako např. `DeserializeElement` nebo `DeserializeSection`, v nichž můžete ovlivnit převod do struktur XML. Třídy jako `SingleTagSectionHandler`, `DictionarySectionHandler`, `NameValueCollectionHandler` nebo `IgnoreSectionHandler` mohou být použity přímo.

51 Čtení konfiguračního souboru machine.config

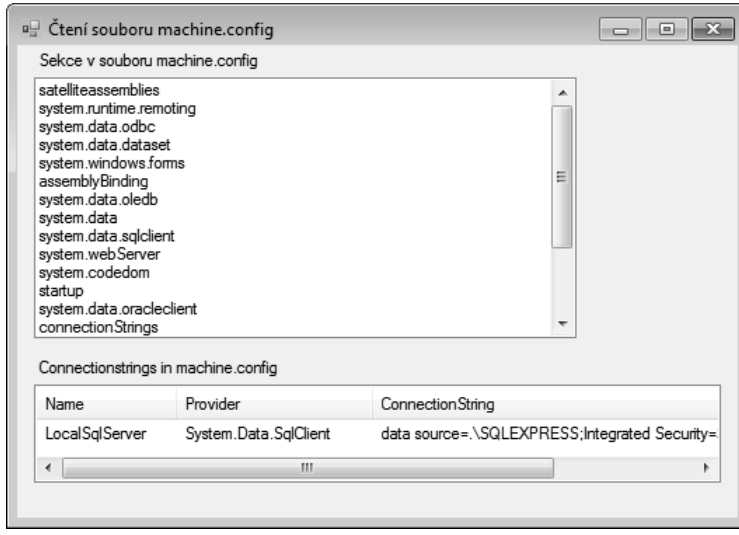
Také k jiným konfiguračním souborům lze přistupovat snadno. Tak například třída `ConfigurationManager` umožňuje otevření ústředního konfiguračního souboru `machine.config` pomocí metody `OpenMachineConfiguration`. Pomocí vlastnosti `Sections` se pak dají přečíst všechny jeho sekce, resp. pomocí `GetSection` se dá načíst určitá sekce.

Výpis 59 ukazuje příklad, v němž jsou jména všech sekcí vypsána v komponentě typu `ListBox` a všechna definovaná databázová spojení jsou načtena a zobrazena v tabulce. Výsledek je vidět na obrázku 23.

Výpis 59: *Čtení informací z konfiguračního souboru machine.config*

```
' Otevřeme soubor machine.config
Dim config As Configuration = _
  ConfigurationManager.OpenMachineConfiguration()

' Projdeme všechny sekce a vypíšeme jména
For Each section As ConfigurationSection In config.Sections
  ListBox1.Items.Add(section.SectionInformation.Name)
Next

' Zjistíme sekci ConnectionString
Dim csc As ConnectionStringsSection = CType(config.GetSection _
  ("connectionStrings"), ConnectionStringsSection)

' Projdeme seznam připojovacích řetězců
For Each connStrSetting As ConnectionStringSettings _
  In csc.ConnectionStrings

  ' Jméno připojovacího řetězce
  Dim lvi As ListViewItem = _
    ListView1.Items.Add(connStrSetting.Name)

  ' Poskytovatel databáze
  lvi.SubItems.Add(connStrSetting.ProviderName)

  ' Připojovací řetězec
  lvi.SubItems.Add(connStrSetting.ConnectionString)
```

```
Next
```



Obrázek 23: *Zobrazení dat načtených z konfiguračního souboru machine.config*

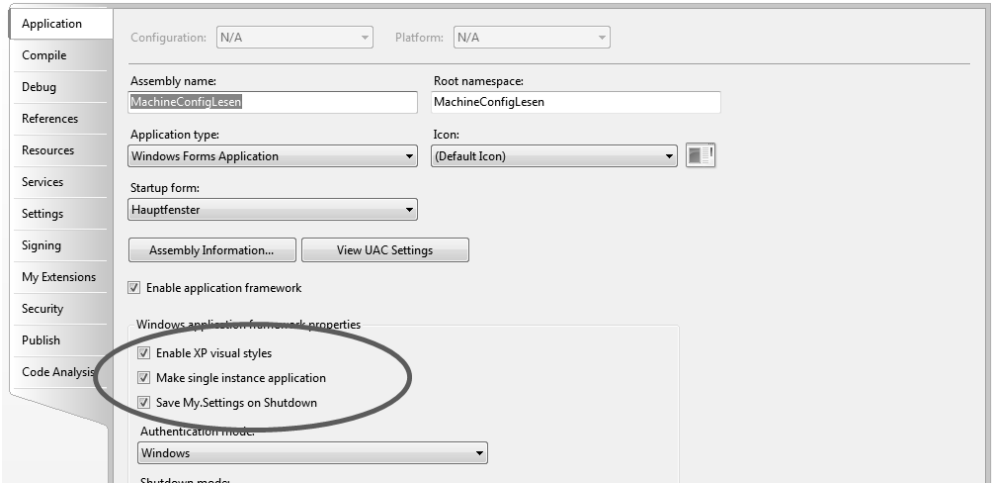
52 Nová nastavení aplikace

Pomocí konfigurační stránky aplikace je možné jednoduše provést některá nastavení, jejichž realizace vyžadovala ve starších verzích vynaložení dodatečného úsilí. Za zmínku zde stojí zejména tři vlastnosti, označené v obrázku 24.

Bude-li zatrženo `Enable XP visual styles`, budou některé ovládací prvky zobrazeny ve stylu systému Windows XP, avšak pouze za podmínky používání dané aplikace v systému Windows XP nebo Windows 2003 Server. Toto zaškrtnutí nahrazuje dříve nezbytné volání `Application.EnableVisualStyles`.

Zaškrtnete-li volbu `Make single instance application`, nebude možné aplikaci znovu spustit, poběží-li ještě nějaká jiná instance této aplikace. Již spuštěná instance pak automaticky získá fokus. Díky tomuto malému zaškrtnutí se stává zbytečným recept 53 ze staré knihy kódů, kde je na třech stránkách vysvětlováno použití systémového objektu `Mutex`.

Třetí zaškrtávací políčko říká, zda se změněná nastavení konfigurace při ukončení programu automaticky ukládají. Jestliže si to nepřejete, můžete ji uložit kdykoli v programu voláním metody `My.Settings.Save`.



Obrázek 24: *Nová nastavení konfigurace zjednodušují programování*

53 Centrální ošetření výjimek

Jsou situace, v nichž nepožadujete nebo nechcete zachytávat případné výjimky pomocí bloků `Try/Catch`. Patří sem např. výjimky, které se automaticky spustí při vázání dat a jež mohou nastat, aniž by je bylo možné ve vlastním kódu zachytit. Centrální ošetření výjimek se mnohdy používá také z organizačních nebo jiných důvodů.

Umístění volání `Application.Run` v bloku `Try` není příliš účelné. Každá neošetřená výjimka by sice způsobila volání odpovídajícího bloku `Catch`, avšak běh metody `Application.Run` – a tím i celého okna – by tím byl ukončen. Tato varianta se hodí nanejvýš pro výpis souhrnného hlášení typu „Toto zhroucení Vám předvedl ...“.

Pro každý podproces však můžete naplánovat centrální ošetření výjimek, a to tak, že k události `Application.ThreadException` připojíte nějaký obslužný program. Neošetřená výjimka pak povede k volání tohoto obslužného programu. Svázání s obslužným programem ovšem musí proběhnout před voláním metody `Application.Run`. Výpis 60 ukazuje obvyklý postup: v `Sub Main` nejdříve registrujete obslužný program a pak zobrazíte hlavní okno.

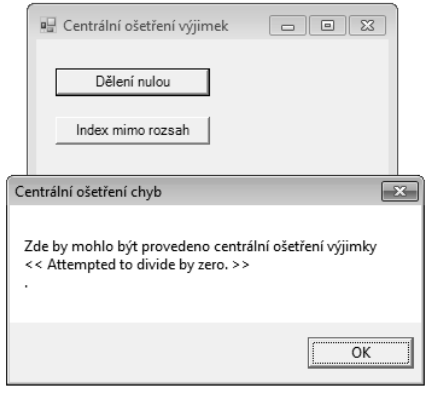
Výpis 60: *Připojení obslužného programu k centrálnímu ošetření výjimek*

```
Public Shared Sub Main()

  ' Vytvoření instance hlavního okna
  Dim mw As New MainWindow

  ' Centrální program pro zpracování chyb je členskou funkcí hlavního okna
  AddHandler Application.ThreadException, _
    AddressOf mw.CentralExceptionHandler

  ' Zobrazení hlavního okna
  Application.Run(mw)

End Sub
```

Výpis 61: *Implementace centrálního programu pro zpracování chyb podle příkladu*

```
Private Sub CentralExceptionHandler(ByVal sender As Object, _
  ByVal e As System.Threading.ThreadExceptionEventArgs)

  ' Sestavení hlášení a jeho výstup
  Dim sw As New System.io.StringWriter
  sw.Write("Zde by mohlo být provedeno centrální ošetření ")
  sw.WriteLine("výjimky")
  sw.Write("<< ")
  sw.Write(e.Exception.Message)
  sw.WriteLine(" >>")
  sw.WriteLine(".")
  sw.Close()
  MessageBox.Show(sw.ToString(), "Centrální ošetření chyb")

End Sub
```

Jak obslužný program sestavíte, je zcela na vás, provedení podle výpisu 61 slouží jen jako ukázka (viz též obrázek 25). Je samozřejmé, že chyby, které zde ohlásíme, musíme také vhodným způsobem zpracovat. Podle druhu chyby může být rovněž účelné program ukončit. Přednost centrálního obslužného programu však spočívá v tom, že program může navzdory nezachycené výjimce pokračovat, jestliže chyba není pro kód kritická nebo neodstranitelná.



Obrázek 25: *Nezachycené výjimky se zpracují v centrálním programu*

Upozornění

Chování ladicích nástrojů Visual Studia se změnilo. Výjimky, které se popsaným způsobem zachytí, jsou v ladicích nástrojích také ošetřeny. Chcete-li docílit předvedeného výsledku, spusťte program z příkladu bez ladění.