28

Práce s firemními objekty

Jedním z nejlepších postupů v programování je rozdělit aplikaci na zpracovatelné a samostatné komponenty označované také za *firemní objekty* (business objects). Aplikace se pak mnohem snáze spravují a navíc lze dosáhnout opakovaného využívání již vytvořeného kódu, jelikož takové komponenty lze sdílet v různých částech jedné aplikace i v naprosto oddělených aplikacích.

Práce s firemními čili vlastními objekty vám umožňuje sestavovat aplikace ASP.NET s využitím skutečného 3vrstvého modelu, kde se firemní vrstva nachází mezi vrstvami prezentace a dat. Práce s firemními objekty vám dále dovoluje používat v aplikacích ASP.NET více programovacích jazyků. Firemní objekty lze vyvíjet v jednom programovacím jazyku, zatímco kód využívaný v prezentační logice je zadán v jiném.

Pokud přesunujete do prostředí ASP.NET nějaké starší aplikace či jejich aspekty, pak budete zřejmě muset využívat různé komponenty modelu COM. Tato kapitola ukazuje, jak na stránkách ASP.NET a v kódu pracovat s komponentami .NET i COM.

Kapitola také vysvětluje, jak lze kombinovat staré knihovny DLL prvků ActiveX (COM) s novými komponentami .NET. Až si všechno popíšeme, mělo by se vám ulevit. Uvidíte totiž, že vaše práce při sestavování komponentových aplikací pomocí nejnovějších technologií ActiveX nepřišla vniveč.

Práce s firemními objekty v ASP.NET 3.5

Kapitola 1 nabízí úvod do práce s firemními a existujícími objekty .NET v aplikacích ASP.NET 3.5. Technologie ASP.NET nyní zahrnuje složku `\App_Code`, kterou můžete v aplikacích vytvořit a vložit do ní všechny své firemní objekty .NET. Na složce `App_Code` je hezké to, že do ní můžete prostě umístit své nezkompilované objekty .NET (jako `Calculator.vb` nebo `Calculator.cs`) a že se ASP.NET postará o jejich kompilaci na použitelné firemní objekty .NET.

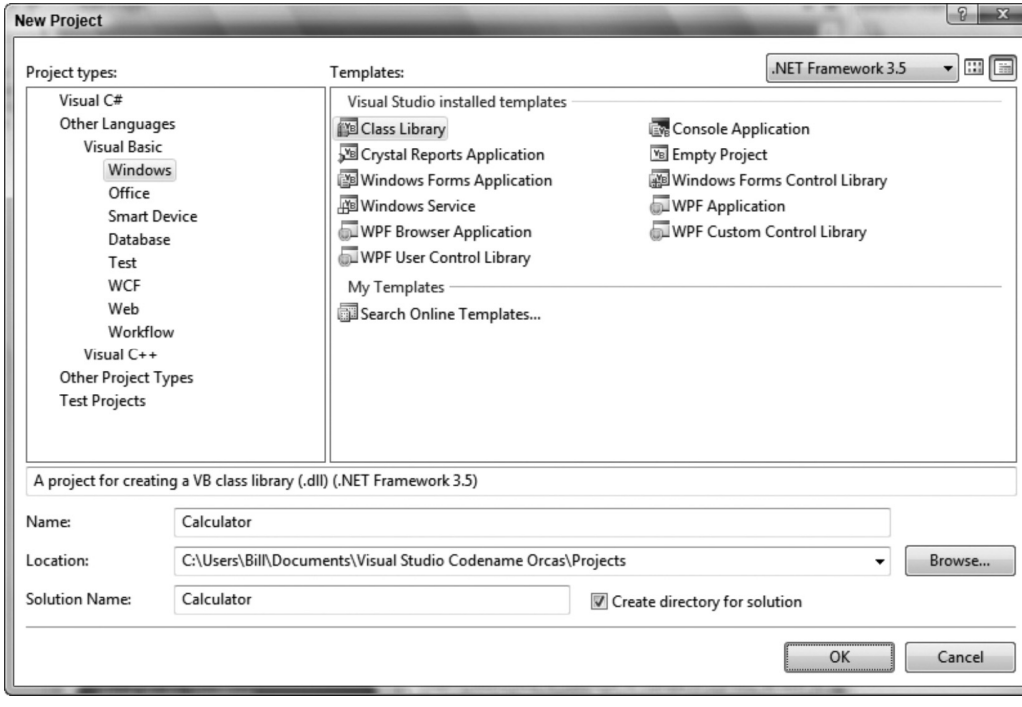
Kapitola 1 také ukazuje, jak lze umístit do složky `App_Code` více dalších složek, jež umožňují využívat firemní objekty napsané v různých programovacích jazycích. S využitím této metody mů-

že ASP.NET zkompilovat jednotlivé firemní objekty do příslušných knihoven DLL, jež budou vaše aplikace ASP.NET dále využívat.

Tvorba předkompilovaných firemních objektů .NET

I když máte k dispozici složku `App_Code`, můžete si také předem zkompilovat své firemní objekty do knihoven DLL a pak je používat v aplikacích ASP.NET. Právě tato metoda se používala před uvedením ASP.NET 2.0 a i dnes je stále k dispozici. Také ani nemusíte mít jinou možnost, když přijímáte firemní objekty .NET přímo ve formě knihoven DLL.

Nejprve se podívejme na tvorbu jednoduchého firemního objektu .NET pomocí Visual Studia 2008. Prvním krokem není vytvoření projektu ASP.NET, ale zadání File → New Project z nabídky Visual Studia. Tím se otevře dialog New Project, v němž jako typ projektu zvolíte Class Library (knihovna tříd) a projekt nazvete Calculator (viz obrázek 28.1).

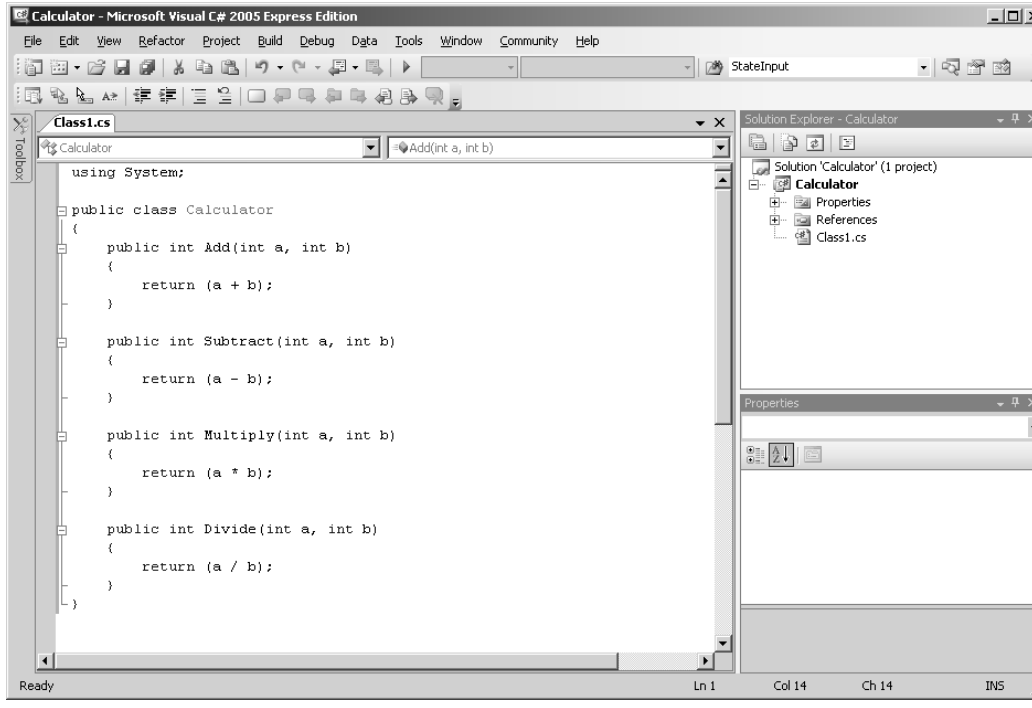


Obrázek 28.1

Třídu v souboru `Class1.vb` nebo `Class1.cs`, který se pro vás v projektu vytvořil, upravte tak, aby zahrnovala funkce `Add`, `Subtract`, `Multiply` a `Divide`. Odpovídající kód v C# najdete na obrázku 28.2.

Při sestavování komponent .NET je zapotřebí věnovat pozornost metadatům sestavy, která se s ní ukládají. Když se podíváte na vlastnosti projektu, klepněte na kartu Application (ta je první nabízenou). K vlastnostem projektu se rovněž dostanete, když klepnete pravým tlačítkem myši na název projektu v prohlížeči řešení. Na kartě najdete tlačítko nazvané Assembly Information

(údaje o sestavě). Po jeho stisku se zobrazí dialog umožňující zadání metadat firemního objektu, a to včetně údajů o verzi (viz obrázek 28.3).



Obrázek 28.2

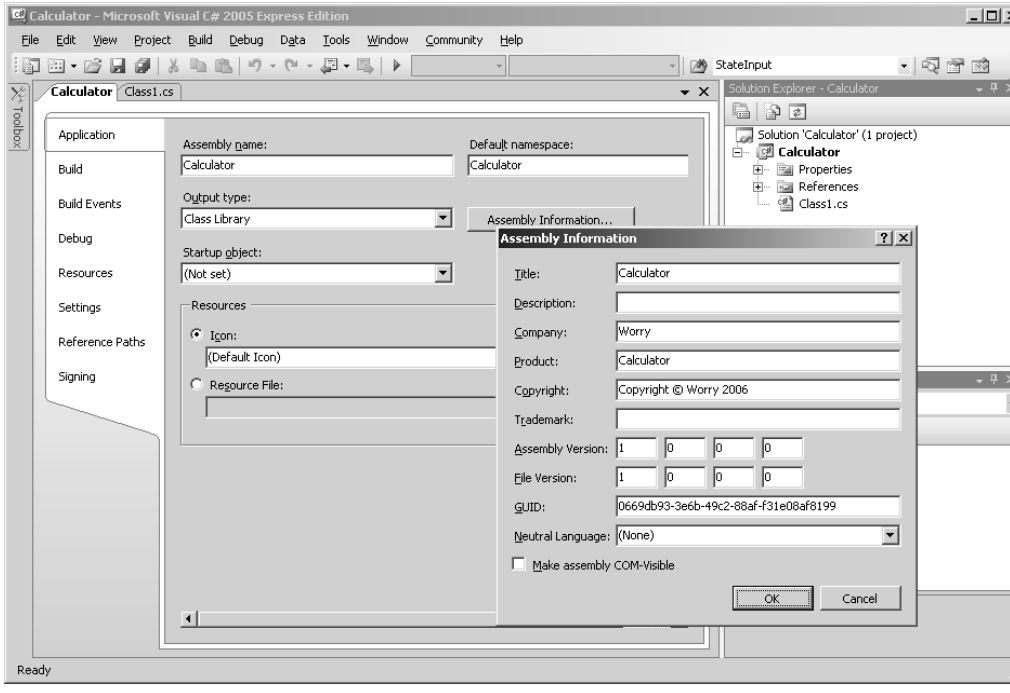
Nyní jste připraveni na kompilaci firemního objektu na použitelný objekt. Toho dosáhnete zadáním Build → Build Calculator z nabídky Visual Studia. Celý proces zkompiluje vše obsažené v řešení až do formy souboru `Calculator.dll`. Ten najdete v adresáři `bin\debug` svého projektu. Standardně se bude jednat o `C:\Users\[uživatel]\Dokumenty\Visual Studio 2008\Projects\Calculator\Calculator\bin\Debug\Calculator.dll` (pouze v systému Windows Vista).

K sestavení a zkompilování svých firemních objektů do formy knihoven DLL nemusíte používat jen Visual Studio 2008, ale stejné věci můžete dosáhnout také manuálně. Stačí v Poznámkovém bloku vytvořit stejný soubor třídy, jak byl zachycen na obrázku 28.2, a uložit jej jako `Calculator.vb` nebo `Calculator.cs` podle používaného programovacího jazyka. Jakmile je soubor uložen, musíte tuto třídu zkompilovat do sestavy (knihovny DLL).

Rámec .NET Framework nabízí kompilátor pro každý z cílových jazyků. Naše kniha se zaměřuje na kompilátory Visual Basic 2008 a C# 2008, jež jsou součástí tohoto rámce.

Třídu zkompilujete tak, že si otevřete příkazový řádek Visual Studia 2008 zadáním Všechny programy → Microsoft Visual Studio 2008 → Visual Studio Tools → Visual Studio 2008 Command Prompt. Po zobrazení výzvy systému DOS přejděte do adresáře, v němž se nachází vaše

třída `Calculator` (příkladem přechodového příkazu je `cd c:\Moje soubory`). Používáte-li kompilátor Visual Basicu, pak zadejte následující příkaz:



Obrázek 28.3

```
vbc /t:library Calculator.vb
```

Máte-li třídu napsanou v jazyce C#, zadejte tento příkaz:

```
csc /t:library Calculator.cs
```

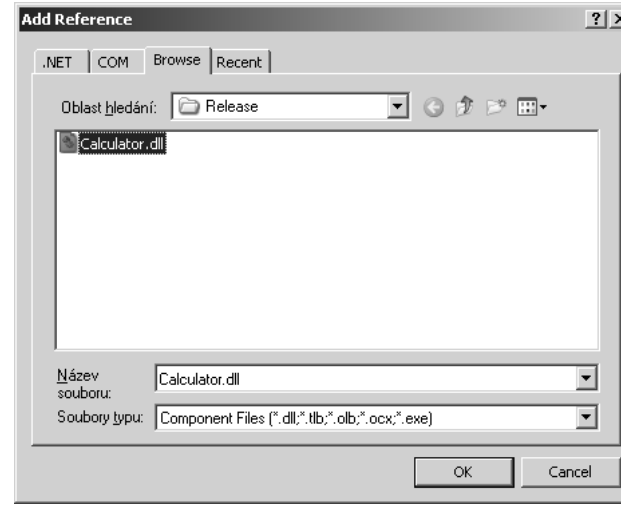
Jak bylo řečeno, každý jazyk má svůj vlastní kompilátor. Visual Basic využívá kompilátor `vbc.exe` v adresáři `C:\Windows\Microsoft.NET\Framework\v3.5\`. Kompilátor C# nazvaný `csc.exe` najdete v téže složce. V předchozích příkladech přepínač `/t:library` říká, že chcete zkompilovat soubor třídy `Calculator.vb` (nebo `.cs`) do knihovny DLL, a nikoli spustitelného souboru (`.exe`), což je výchozí formát. Za přepínačem `t:/library` je název kompilovaného souboru.

> *Existuje mnoho různých přepínačů, které můžete kompilátoru předat – dokonce víc, než jich nabízí Visual Studio 2008. Chcete-li se ve své sestavě kupříkladu odkazovat na určité knihovny DLL, pak budete muset zadat přepínače jako `/r:system.data.dll`. Celý seznam přepínačů kompilátoru najdete v dokumentaci MSDN.*

Jakmile doběhne kompilátor, budete mít vytvořenou knihovnu DLL a budete připraveni na věci následující.

Práce s předkompilovanými firemními objekty v aplikacích ASP.NET

Chcete-li ve svém projektu ASP.NET 3.5 používat nějaké knihovny DLL, musíte v kořenovém adresáři aplikace vytvořit složku `Bin`. Klepněte tedy pravým tlačítkem myši na projekt v okně Solution Explorer a zadejte příkaz Add ASP.NET Folder → Bin. Ve Visual Studiu 2008 se ikona adresáře `Bin` zobrazí šedou barvou a s ozubeným kolečkem. Do této složky doplňte svou novou knihovnu DLL – na složku klepněte pravým tlačítkem myši a z místní nabídky zadejte příkaz Add Reference. Tím se spustí dialog Add Reference (přidat odkaz). V něm vyberte kartu Browse a vyhledejte soubor `Calculator.dll`. Pak jej zvýrazněte a stiskem tlačítka OK přidejte do složky `Bin` svého projektu. Používaný dialog najdete na obrázku 28.4.



Obrázek 28.4

V projektu máte soubor `Calculator.dll`, který je nyní přístupný. To znamená, že nyní můžete využívat všechny funkce vystavované jeho rozhraním. Obrázek 28.5 je příkladem funkce IntelliSense, která výrazně zjednodušuje práci s komponentou .NET.

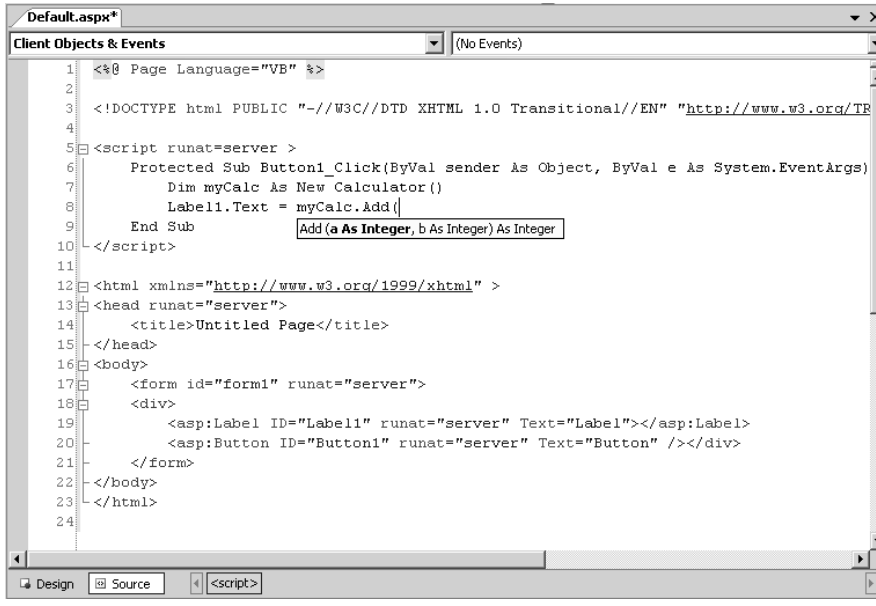
Jak vidíte, komponenty .NET lze vytvářet a používat v aplikacích ASP.NET docela jednoduše. Nyní se podívejme na práci s komponentami modelu COM.

Spolupráce s COM: Využití COM v rámci .NET

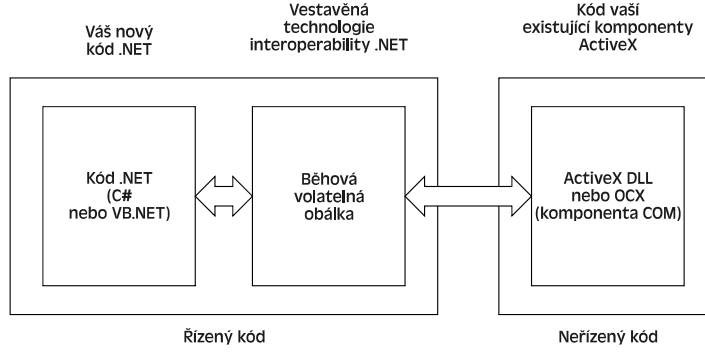
Společnost Microsoft dobře ví, že všichni vývojáři by byli velmi zklamaní, kdyby nemohli využívat tisíce svých prvků COM, které během řady let sestavili, spravovali a vylepšovali. Je jasné, že nikdo takové prvky ze dne na den neopustí a nepřejde čistě do světa .NET.

Proto společnost Microsoft zajistila možnost spolupráce neboli interoperability s modelem COM. Jedná se o technologii (označovanou zkráceně COM Interop), jež dovoluje rámci .NET obalit funkčnost objektu COM rozhraním komponenty .NET, takže váš kód rámce .NET může komunikovat s objektem COM, aniž by musel využívat jeho techniky a rozhraní.

Obrázek 28.6 ilustruje Runtime Callable Wrapper, čili obálku volatelnou za běhu, což je prostřednická komponenta směrující provoz mezi kódem .NET a komponentou modelu COM.



Obrázek 28.5



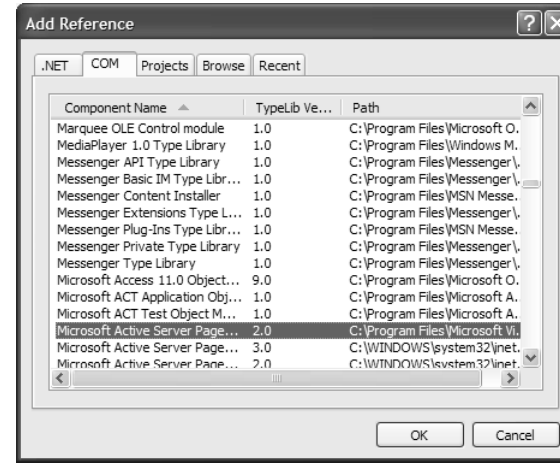
Obrázek 28.6

Běhová volatelná obálka

Runtime Callable Wrapper neboli RCW je onen magický útržek kódu, který umožňuje interakci mezi .NET a COM. Pro každou komponentu COM ve vašem projektu se vytvoří jedna obálka RCW. K tvorbě obálky RCW komponenty modelu COM můžete použít Visual Studio 2008.

Chcete-li do sekce References neboli odkazů svého projektu doplnit knihovny DLL prvků ActiveX, zadejte příkaz Website → Add Reference nebo vyberte příkaz Add Reference z nabídky, jež se objeví poté, co klepnete pravým tlačítkem myši na kořenový uzel svého projektu v okně Solution Explorer.

Zobrazí se dialog přidání odkazu s pěti kartami (záložkami): .NET, COM, Projects (není ve verzi Express), Browse a Recent (viz obrázek 28.7). V našem příkladu vybereme kartu COM a vyhledáme komponentu, kterou chceme do projektu .NET vložit. Pak stačí danou položku vybrat a stiskem tlačítka OK přidat do projektu odpovídající odkaz. Nově doplněná komponenta se bude nacházet uvnitř složky `Bin` projektu.



Obrázek 28.7

Z knihovny ActiveX, jejíž použití jste ve Visual Studiu 2008 zadali, se automaticky vytvoří knihovna Interop. Právě ta představuje komponentu RCW přizpůsobenou ovládacímu prvku ActiveX, jak bylo uvedeno na obrázku 28.6. Názvem tohoto souboru interoperability je prostě `Interop.PůvodníNázev.DLL`.

Soubory RCW můžete vytvářet také manuálně, aniž byste pracovali ve Visual Studiu 2008. V rámci .NET Framework najdete manuální metodu tvorby souborů Interop obálky RCW ovládacích prvků – jedná se o nástroj příkazového řádku nazvaný Type Library Importer. Tohoto importéra knihoven typů vyvoláte spustitelným souborem `tlbimp.exe`.

Chcete-li tak kupříkladu vytvořit knihovnu Interop pro dříve použitý objekt SQLDMO, spusťte si příkazový řádek Visual Studia 2008 ze skupiny Microsoft Visual Studio 2008 → Visual Studio Tools v nabídce tlačítka Start. Na příkazový řádek pak zadejte:

```
tblimp sqldmo.dll /out:sqldmoex.dll
```

V tomto případě specifikuje přepínač `/out`: název vytvořené knihovny Interop obálky RCW. Když jej vynecháte, získáte stejný název, jaký pro vás běžně generuje Visual Studio.

Type Library Importer je užitečný, když jako vývojové prostředí nepoužíváte Visual Studio 2008, požadujete-li vyšší kontrolu nad sestavami, jež se pro vás vytvářejí, nebo automatizujete-li proces napojení na komponenty modelu COM.

Type Library Importer je obálková aplikace třídy `TypeLibConverter` oboru názvů `System.Runtime.InteropServices`.

Práce s objekty COM v kódu ASP.NET

Nyní budeme pokračovat dalšími ukázkami a podíváme se na jednoduchý příklad použití objektu COM napsaného ve Visual Basicu 6 na stránce ASP.NET.

V prvním kroku vytvoříme knihovnu DLL ActiveX, s níž budeme dále pracovat. Do třídy nazvané `NameFunctionsClass` doplňte kód Visual Basicu 6 podle výpisu 28.1 a zkompilujte ji jako knihovnu DLL ovládacího prvku ActiveX s názvem `NameComponent.dll`.

Výpis 28.1: Kód VB6 knihovny DLL ActiveX s názvem NameComponent.DLL

```
Option Explicit

Private m_sFirstName As String
Private m_sLastName As String

Public Property Let FirstName(Value As String)
  m_sFirstName = Value
End Property

Public Property Get FirstName() As String
  FirstName = m_sFirstName
End Property

Public Property Let LastName(Value As String)
  m_sLastName = Value
End Property

Public Property Get LastName() As String
  LastName = m_sLastName
End Property

Public Property Let FullName(Value As String)
  m_sFirstName = Split(Value, " ")(0)
  If (InStr(Value, " ") > 0) Then
    m_sLastName = Split(Value, " ")(1)
  Else
    m_sLastName = ""
  End If
End Property

Public Property Get FullName() As String
  FullName = m_sFirstName + " " + m_sLastName
End Property

Public Property Get FullNameLength() As Long
  FullNameLength = Len(Me.FullName)
End Property
```

Jakmile máte ovládací prvek ActiveX ve formě knihovny DLL, kterou můžete využívat na stránkách ASP.NET, vytvořte ve Visual Studiu 2008 nový projekt ASP.NET. Nahraďte kód HTML v souboru `Default.aspx` kódem uvedeným ve výpisu 28.2. Tím se na stránku HTML přidá řada textových políček a popisků a funkčnost zajišťovaná kódem Visual Basicu nebo C#.

Výpis 28.2: Použití NameComponent.dll.

VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub AnalyzeName_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)
```

```
    Dim Name As New NameComponent.NameFunctionsClass()

    If (FirstName.Text.Length > 0) Then
      Name.FirstName = FirstName.Text
    End If

    If (LastName.Text.Length > 0) Then
      Name.LastName = LastName.Text
    End If

    If (FullName.Text.Length > 0) Then
      Name.FullName = FullName.Text
    End If

    FirstName.Text = Name.FirstName
    LastName.Text = Name.LastName
    FullName.Text = Name.FullName
    FullNameLength.Text = Name.FullNameLength.ToString
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Práce s komponentami COM</title>
</head>
<body>
  <form id="form1" runat="server">
    <P>
      <asp:Label ID="Label1" runat="server">Jméno:</asp:Label>
       
      <asp:TextBox ID="FirstName" runat="server"></asp:TextBox>
    </P>
    <P>
      <asp:Label ID="Label2" runat="server">Příjmení:</asp:Label>
       
      <asp:TextBox ID="LastName" runat="server"></asp:TextBox>
    </P>
    <P>
      <asp:Label ID="Label3" runat="server">Celé jméno:</asp:Label>
       
      <asp:TextBox ID="FullName" runat="server"></asp:TextBox>
    </P>
    <P>
      <asp:Label ID="Label4" runat="server">Délka celého jména:</asp:Label>
       
      <asp:Label ID="FullNameLength" runat="server"
        Font-Bold="True">0</asp:Label>
    </P>
    <P>
      <asp:Button ID="AnalyzeName" runat="server"
        OnClick="AnalyzeName_Click" Text="Analyzovat jméno"></asp:Button>
```

```
    </P>
  </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<script runat="server">

  protected void AnalyzeName_Click(object sender, System.EventArgs e)
  {
    NameComponent.NameFunctionsClass Name =
      new NameComponent.NameFunctionsClass();

    if (FirstName.Text.Length > 0)
    {
      string firstName = FirstName.Text.ToString();
      Name.set_FirstName(ref firstName);
    }

    if (LastName.Text.Length > 0)
    {
      string lastName = LastName.Text.ToString();
      Name.set_LastName(ref lastName);
    }

    if (FullName.Text.Length > 0)
    {
      string fullName = FullName.Text.ToString();
      Name.set_FullName(ref fullName);
    }

    FirstName.Text = Name.get_FirstName();
    LastName.Text = Name.get_LastName();
    FullName.Text = Name.get_FullName();
    FullNameLength.Text = Name.FullNameLength.ToString();
  }
</script>
```

Nyní je zapotřebí doplnit odkaz na knihovnu DLL prvku ActiveX, který jste vytvořili v předchozím kroku. Postupujte následovně:

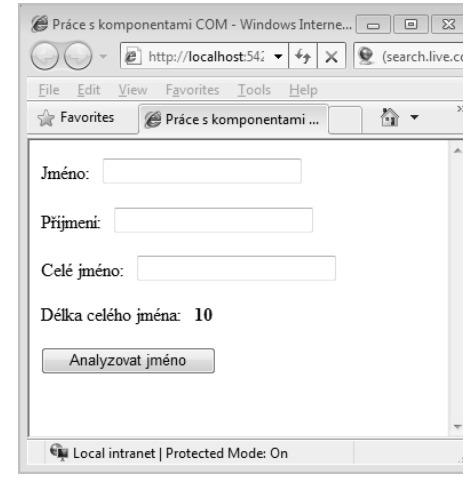
1. Klepněte pravým tlačítkem myši na svůj projekt v okně Solution Explorer.
2. Zadejte příkaz Add Reference.
3. V dialogu přidání odkazu vyberte kartu Browse.
4. Vyhledejte soubor `NameComponent.dll`.
5. Stiskněte tlačítko OK, čímž přidáte knihovnu `NameComponent.dll` na seznam vybraných komponent a dialog zavřete.

Nepoužíváte-li Visual Studio 2008 nebo stránky s kódem v pozadí, můžete přidat odkaz na svou komponentu COM manuálním vytvořením obálky RCW pomocí nástroje Type Library Converter. Na stránku pak umístíte příkaz `Imports` (VB) nebo `using` (C#).

Jakmile máte komponentu vybranou, vytvoří se její obálkový soubor RCW a doplní se do vaší aplikace.

A to je vlastně vše! Aplikaci jen spusťte a podívejte se na vrstvu interoperability COM v akci.

Obrázek 28.8 ukazuje právě vytvořenou stránku ASP.NET. Po stisku tlačítka Analyzovat jméno se odešlou hodnoty textových políček jména, příjmení a celého jména obálce RCW, která je postoupí komponentě ActiveX s názvem `NameComponent.DLL`. Data se převezmou stejným způsobem a použijí se k naplnění textových políček a uvedené délky celého jména.



Obrázek 28.8

Přístup k záludným členům COM v jazyce C#

Občas se některé členy objektů COM nevystavují správně v jazyce C#. V předchozích příkladech se takto nevystavovaly vlastnosti typu `String`, ovšem vlastnost typu `Long` (`FullNameLength`) ano.

Problém odhalíte poměrně snadno – odpovídající vlastnost sice vidíte, ovšem aplikaci nelze zkompilovat. Místo kódu jazyka C# podle výpisu 28.2 použijete k nastavení vlastnosti `FirstName` komponenty ActiveX v knihovně `NameComponent.dll` kupříkladu toto zadání:

```
if (FirstName.Text.Length > 0)
  Name.FirstName = FirstName.Text.ToString();
```

Když se kód pokusíte přeložit, objeví se následující chyba:

```
Vlastnost, indexovací člen nebo událost FirstName nejsou tímto jazykem podporovány. Zkuste
přímo volat metody přístupového objektu NameComponent._NameFunctionsClass.get_FirstName()
nebo NameComponent._NameFunctionsClass.set_FirstName(ref string).
```

V zásadě říká, že vlastnost `FirstName` není jazykem podporována a že máte zkusit přímo volat metody aksesoru. Přitom se `FirstName` jeví jako v pořádku – ukazuje se v IntelliSense, ovšem nemůžete s ní pracovat. Musíte místo toho používat `set_FirstName` (a `get_FirstName` ke čtení). Tyto metody se v IntelliSense neobjevují, ovšem můžete být v klidu, skutečně existují.

Navíc zmíněné metody vyžadují parametr typu `ref string`, a nikoli `String`. Aby vše fungovalo správně, používáme ve výpisu 28.2 dva kroky. Nejprve se `String` přiřadí lokální proměnné a ta se pak předá odpovídající metodě pomocí `ref`.

Manuální uvolnění objektů COM

Jednou z vynikajících věcí na rámci .NET je jeho vlastní uvolňování paměti (neboli sběr odpadků – garbage collection). Rámec po sobě umí opravdu dobře uklidit. To však neplatí vždy při

využívání spolupráce s COM. Rámec .NET nemůže vědět, kdy lze objekt modelu COM uvolnit z paměti, protože ten nezahrnuje mechanismus uvolňování paměti využívaný rámcem .NET.

Kvůli tomuto omezení byste měli uvolňovat objekty COM z paměti co nejdříve pomocí metody `ReleaseComObject` třídy `System.Runtime.InteropServices.Marshal`:

C#

```
System.Runtime.InteropServices.Marshal.ReleaseComObject(Object);
```

Je vhodné říci, že pokud se pokusíte tento objekt použít znovu, než dojde k jeho zrušení, dojde k výjimce.

Zpracování chyb

Zpracování chyb v rámci .NET používá výjimky, a nikoli hodnoty `HRESULT` aplikací Visual Basicu 6. Naštěstí se však obálka RCW postará o většinu práce s převodem těchto dvou věcí.

Vezměme si kupříkladu kód podle výpisu 28.3. Zde dochází k vyvolání uživatelem definované chyby, je-li dělenec nebo dělitel větší než 1 000. Všimněte si také, že nezachytáváme chybu dělení nulou. Podívejme se, k čemu dojde, když komponenta ActiveX sama vyvolá chybu.

Příklad začneme kompilací kódu podle výpisu 28.3 do třídy nazvané `DivideClass` v komponentě ActiveX označené `DivideComponent.dll`. (Soubor opět najdete mezi zdrojovými kódy, po zkopírování jej ovšem nezapomeňte zaregistrovat v systému.)

Výpis 28.3: Vyvolávání chyb ve VB6

```
Public Function DivideNumber(Numerator As Double, _
  Denominator As Double) As Double

  If ((Numerator > 1000) Or (Denominator > 1000)) Then
    Err.Raise vbObjectError + 1, _
      "DivideComponent:Divide.DivideNumber", _
      "Dělenec i dělitel musejí být " + _
      "menší než nebo rovny hodnotě 1000."
  End If

  DivideNumber = Numerator / Denominator
End Function
```

Vytvořme nový projekt ASP.NET a přidejme do něj odkaz na knihovnu `DivideComponent.dll` (nechme Visual Studio 2008, aby si vytvořilo svou kopii obálky RCW). Nezapomínejme, že téhož lze dosáhnout manuálně nástrojem `tlbimp.exe`.

Nyní na stránku ASP.NET vložme kód podle výpisu 28.4.

Výpis 28.4: Zpracování chyb v rámci .NET

VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub Calculate_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)
```

```
    Dim Divide As New DivideComponent.DivideClass()

    Try
      Answer.Text = Divide.DivideNumber(Numerator.Text, Denominator.Text)
    Catch ex As Exception
      Answer.Text = ex.Message.ToString()
    End Try

    System.Runtime.InteropServices.Marshal.ReleaseComObject(Divide)
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Práce s komponentami COM</title>
</head>
<body>
  <form id="form1" runat="server">
    <P>
      <asp:Label ID="Label1" runat="server">Dělenec:</asp:Label>
       
      <asp:TextBox ID="Numerator" runat="server"></asp:TextBox>
    </P>
    <P>
      <asp:Label ID="Label2" runat="server">Dělitel:</asp:Label>
       
      <asp:TextBox ID="Denominator" runat="server"></asp:TextBox>
    </P>
    <P>
      <asp:Label ID="Label3" runat="server">
        Dělenec vydělený dělitelem:</asp:Label>
       
      <asp:Label ID="Answer" runat="server" Font-Bold="True">0</asp:Label>
    </P>
    <P>
      <asp:Button ID="Calculate"
        runat="server"
        OnClick="Calculate_Click"
        Text="Vypočítat">
      </asp:Button>
    </P>
  </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
  protected void Calculate_Click(object sender, System.EventArgs e)
  {
```

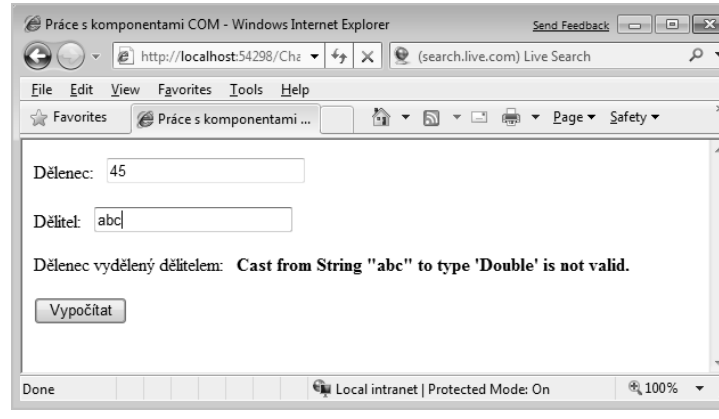
```
    DivideComponent.DivideClass myDivide = new     DivideComponent.DivideClass();

    try
    {
      double numerator = double.Parse(Numerator.Text);
      double denominator = double.Parse(Denominator.Text);
      Answer.Text = myDivide.DivideNumber(ref numerator,
        ref denominator).ToString();
    }

    catch (Exception ex)
    {
      Answer.Text = ex.Message.ToString();
    }

  System.Runtime.InteropServices.Marshal.ReleaseComObject(myDivide);
  }
</script>
```

Kód z výpisu 28.4 předá uživatelem zadané hodnoty dělence a dělitele komponentě ActiveX `DivideComponent.dll`, která zajistí vydělení. Když aplikaci spustíte s neplatnými daty, obdržíte výsledek odpovídající obrázku 28.9.



Obrázek 28.9

V závislosti na jazyku použitému k běhu aplikace ASP.NET spatříte různé údaje pro odlišná data. V případě platného zadání pochopitelně vždy obdržíte správný výsledek. Zadáte-li hodnotu převyšující 1 000, spatříte popis chyby definovaný ve Visual Basicu 6 říkající, že dělenec i dělitel musejí být menší nebo rovny 1 000.

V případě neplatných typů `String` však Visual Basic 2008 nahlásí neplatnost přetypování řetězce „abc“ na typ `Double`, zatímco C# oznámí, že vstupní řetězec nemá správný formát. V případě dělení nulou oba jazyky oznámí stejnou chybu, protože ta přichází přímo z běhového modulu Visual Basicu 6.

Nasazení komponent COM s aplikacemi .NET

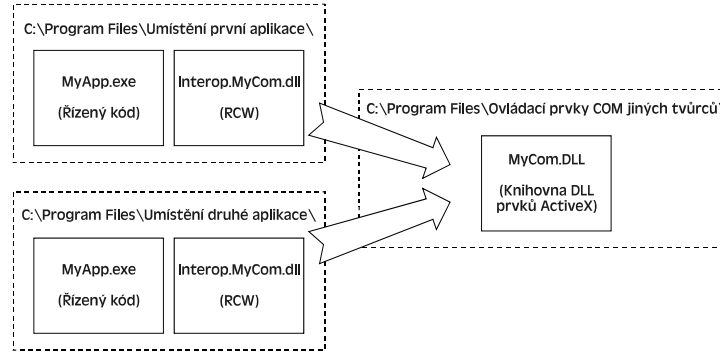
Nasazení komponent COM s aplikacemi .NET je velmi jednoduché, a to i v porovnání s prostým nasazením ovládacích prvků ActiveX. Při zavádění aplikací rámce .NET s komponentami modelu COM lze využít dva scénáře:

- používání soukromých sestav,
- využívání sdílených sestav.

Soukromé sestavy

Instalace celé komponenty ActiveX nebo jejích částí místně k aplikaci .NET se považuje za instalaci soukromých sestav. V tomto scénáři má každá instalace vaší aplikace .NET na témže počítači přinejmenším svou vlastní kopii knihovny Interop odpovídající odkazované komponenty ActiveX, jak to uvádí obrázek 28.10.

Musíte se sami rozhodnout, zda instalovat komponentu ActiveX lokálně aplikaci nebo do nějakého sdíleného adresáře, kde ji budou využívat všechny volající aplikace.



Obrázek 28.10

> *Kdysi se považovalo za vhodné umisťovat komponenty ActiveX do jejich vlastních adresářů, takže když jste se na ně opakovaně odkazovali z dalších aplikací, nemuseli jste příslušný soubor registrovat ani instalovat podruhé. Tento přístup znamenal, že upgradem jedné komponenty jste automaticky inovovali všechny ji využívající aplikace. V praxi to ovšem tak ideálně nefungovalo. Vlastně se tento postup stal význačným přispěvatelem do „pekla s knihovnami DLL" (DLL Hell) a hlavním důvodem, proč společnost Microsoft začala doporučovat instalaci soukromých komponent do sestav .NET.*

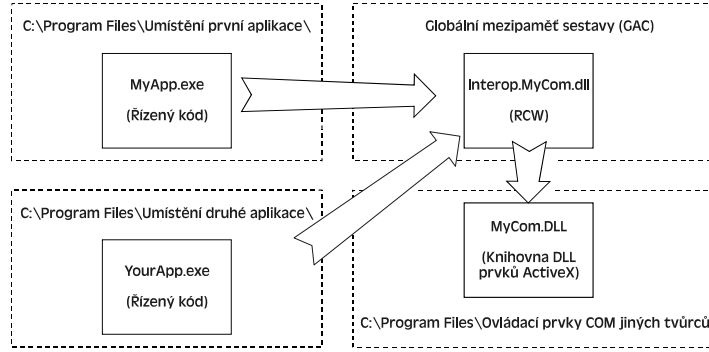
Jakmile máte komponentu fyzicky k dispozici, zbývá ji jen zaregistrovat jako ActiveX pomocí `regsvr32`, stejně jako při zavádění běžných aplikací ActiveX.

Veřejné sestavy

Opakem soukromé sestavy je veřejná sestava. Veřejné sestavy sdílejí obálku RCW spolupráce mezi aplikacemi. Chcete-li vytvořit veřejnou sestavu, musíte umístit soubor RCW do *globální mezipaměti sestav* GAC (Global Assembly Cache), jak to uvádí obrázek 28.11.

Mezipaměť GAC najdete v adresáři `C:\Windows\assembly`. Instalace položek do GAC spočívá v prostém přetažení odpovídajících položek do této složky Průzkumníkem. Třebaže je GAC otevřena všem, nedoporučuje se slepě instalovat komponenty do této sekce, nemáte-li k tomu opravdu dobrý důvod.

Položky lze do GAC přidat také z příkazového řádku nástrojem Global Assembly Cache (`Gacutil.exe`). Ten vám dovoluje zobrazovat si obsah globální mezipaměti sestav a mezipaměti stažených prvků a manipulovat s nimi. Pohled Průzkumníku na GAC nabízí podobnou funkčnost, ovšem `Gacutil.exe` můžete využívat ve skriptech a dávkových souborech.



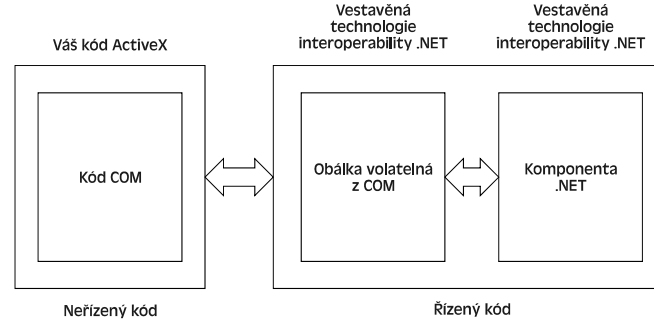
Obrázek 28.11

Jen obtížně se hledá skutečně dobrý důvod k instalování sestav spolupráce s ActiveX do mezipaměti GAC. Pokud byste to někdy dělali, pak by se muselo jednat o nějakou vysoce sdílenou komponentu ActiveX, kterou bude na jednom počítači využívat mnoho aplikací .NET. V podnikovém prostředí k tomu může dojít, když upgradujete existující firemní logiku z ActiveX na podporu .NET na serveru využívaném mnoha aplikacemi. V komerčním prostředí je lepší se používání GAC vyhýbat.

Práce s rámcem .NET z neřízeného kódu

Rámec .NET nabízí také opak spolupráce s modelem COM a umožňuje vám využívat své nově vytvořené komponenty .NET v neřízeném kódu. Tento oddíl probírá používání komponent .NET ve spustitelných souborech Visual Basicu 6. Zde zachycené techniky jsou stejné, když místo vykonatelných souborů pracujete s knihovnami OCX nebo DLL prvků ActiveX.

Ta část kódu, kterou rámec .NET Framework umožňuje neřízenému kódu komunikovat s vaší komponentou .NET, se označuje COM-Callable Wrapper (CCW – obálka volatelná z COM). Obálka CCW není na rozdíl od RCW samostatnou knihovnou DLL, kterou distribuujete společně se svou aplikací. CCW je součástí rámce .NET Framework a její instance se vytváří u každé využívané komponenty .NET.



Obrázek 28.12

Obrázek 28.12 ukazuje, jak CCW převádí komunikaci mezi neřízeným kódem a kompo-

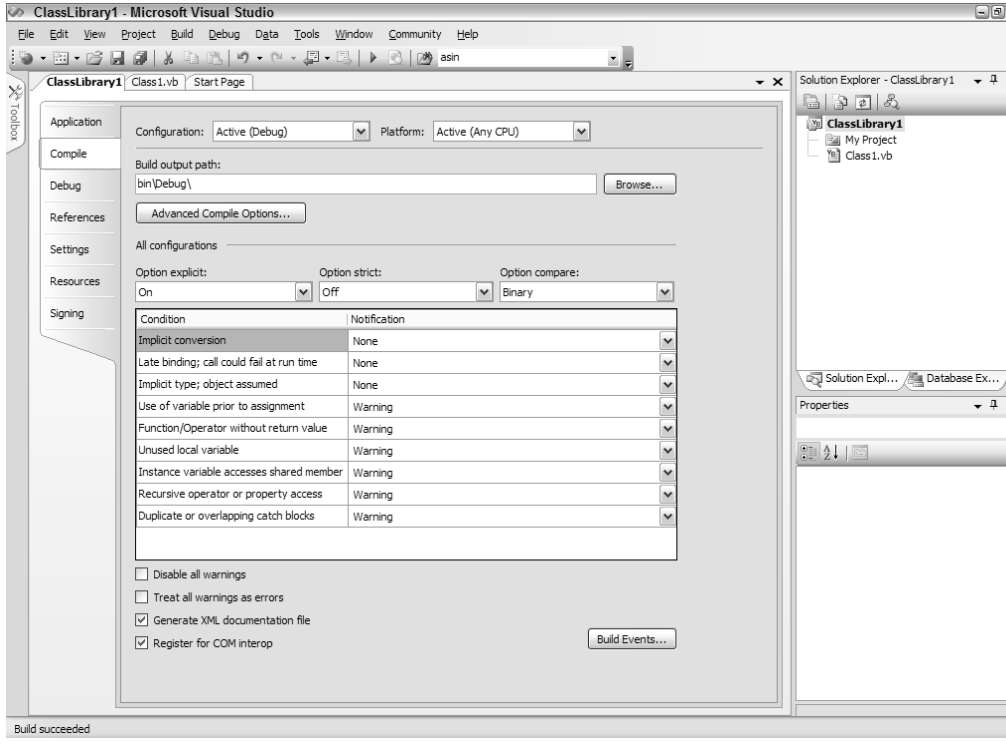
nentou .NET podobným způsobem, jako obálka RCW převádí komunikaci mezi řízeným kódem a kódem modelu COM.

Obálka volatelná z COM

Jak již bylo řečeno, obálka volatelná z COM neboli CCW není samostatnou knihovnou DLL jako obálka RCW. CCW používá speciálně vytvořenou knihovnu typů vycházející z komponenty .NET. Tato knihovna typů se označuje Interop Type Library (knihovna typů spolupráce neboli interoperability) a je staticky napojená na neřízený kód, který pak může komunikovat s obálkou CCW a komponentou rámce .NET zahrnutou v aplikaci.

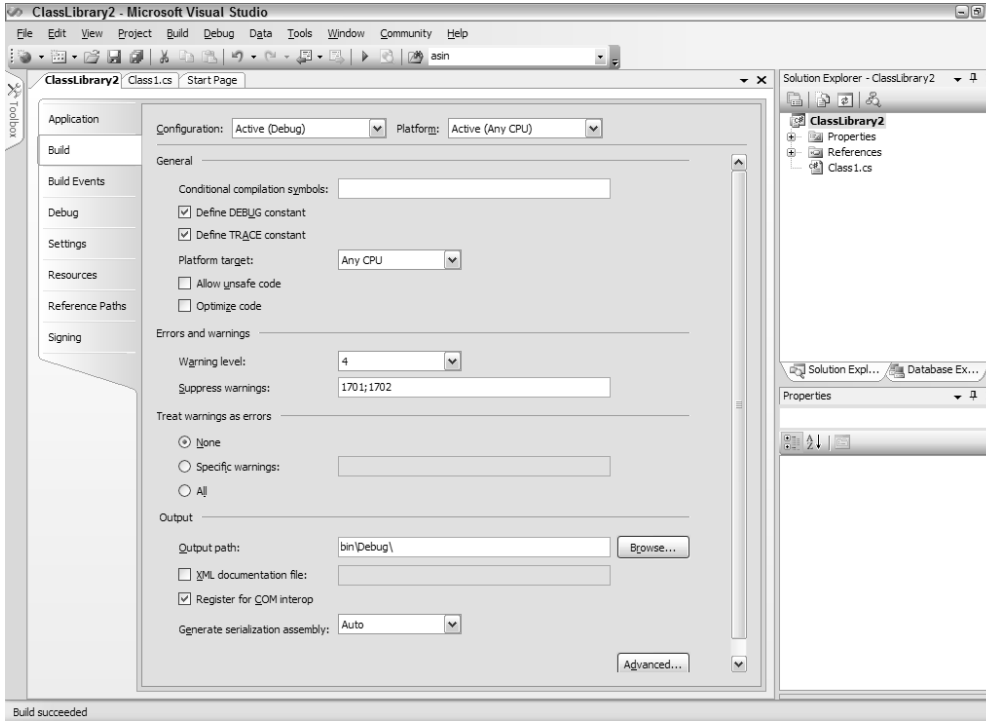
Aby se z nějaké komponenty .NET vygenerovala knihovna typů interoperability, musíte ji od Visual Studia 2008 požadovat během sestavování. Projekty Visual Basicu i C# nabízejí odpovídající nastavení v sekci vlastností kompilace v dialogu vlastností projektu knihovny tříd.

Klepněte pravým tlačítkem myši na projekt v okně Solution Explorer a zadejte příkaz Properties. Tím si zobrazíte vlastnosti projektu. Obrázek 28.13 zachycuje vlastnosti projektu aplikace knihovny tříd Visual Basicu 2008. Vše se objevuje přímo v dokumentovém okně Visual Studia.



Obrázek 28.13

Jazyk C# nabídne trochu odlišné okno, jak je zřejmé z obrázku 28.14. V obou dialozích se však potřebná vlastnost nazývá Register for COM Interop (zaregistrovat pro spolupráci s modelem COM). Ve Visual Basicu ji najdete na kartě Compile; v C# se nachází na kartě Build stránek vlastností.



Obrázek 28.14

Jakmile volbu zadáte zaškrtnutím políčka, pak se při sestavování projektu vygeneruje kromě souboru DLL také soubor knihovny typů (Type Library – `.tlb`). Ten je klíčem k zahrnutí komponent .NET do aplikací COM.

Když doplňujete odkaz na knihovnu DLL ve Visual Basicu, navigujete ze sekce References projektu a vyhledáváte přidávanou knihovnu DLL prvků ActiveX. Používáte-li komponenty .NET, pak se na ně nemůžete řádně tímto způsobem odkazovat, protože se nejedná o prvky ActiveX. Místo toho se odkazujete na knihovnu typů interoperability, která vaší aplikaci zpřístupní funkčnost příslušné komponenty rámce .NET.

Rámec .NET Framework také nabízí metodu manuálního vytváření knihoven typů interoperability komponent .NET. K tomu slouží nástroj příkazového řádku nazvaný Type Library Exporter (porovnejte tento název s nástrojem Type Library Importer používaným k zajištění spolupráce s COM). Type Library Exporter se volá pomocí spustitelného souboru `tlbexp.exe`.

Chcete-li kupříkladu vytvořit knihovnu typů interoperability pro knihovnu `NameComponent.dll` podle dále uvedeného příkladu, pak použijete následující příkaz:

```
tlbexp NameComponent.dll /out:NameComponentEx.tlb
```

Parametr `/out`: specifikuje název vytvářené knihovny typů interoperability. Pokud jej vynecháte, obdržíte soubor s názvem komponenty .NET, ale s příponou `.tlb`.

Nástroj Type Library Exporter je užitečný, když jako své vývojové prostředí nepoužíváte Visual Studio 2008, chcete-li mít větší kontrolu nad vytvářenými sestavami nebo automatizujete-li proces napojování na komponenty .NET.

Používání komponent .NET v objektech COM

Následující příklad ukazuje, jak lze komponenty .NET používat v kódu modelu COM. Nejprve vytvořte a zkompilujte kód .NET podle výpisu 28.5, a to buď ve Visual Basicu, nebo v C#.

Jakmile kód zadáte do projektu knihovny tříd, komponentu sestavte pod názvem `NameComponent`. Nezapomeňte zaškrtnutím příslušného políčka na stránkách vlastností projektu zajistit podporu spolupráce s COM. To ukazuje obrázek 28.13 pro kód Visual Basicu a obrázek 28.14 pro kód C#. Nepoužíváte-li Visual Studio 2008, můžete k manuálnímu vygenerování knihovny typů interoperability použít `tblexp.exe`, jak bylo popsáno.

Výpis 28.5: Komponenta .NET

VB

```
Public Class NameFunctions

  Private m_FirstName As String
  Private m_LastName As String

  Public Property FirstName() As String
    Get
      Return m_FirstName
    End Get
    Set(ByVal Value As String)
      m_FirstName = Value
    End Set
  End Property

  Public Property LastName() As String
    Get
      Return m_LastName
    End Get
    Set(ByVal Value As String)
      m_LastName = Value
    End Set
  End Property

  Public Property FullName() As String
    Get
      Return m_FirstName + " " + m_LastName
    End Get
    Set(ByVal Value As String)
      m_FirstName = Split(Value, " ")(0)
      m_LastName = Split(Value, " ")(1)
```

```
    End Set
  End Property

  Public ReadOnly Property FullNameLength() As Long
    Get
      FullNameLength = Len(Me.FullName)
    End Get
  End Property
End Class
```

C#

```
using System;
using System.Runtime.InteropServices;

namespace NameComponent
{
   [ComVisible(true)]
  public class NameFunctions
  {
    private string m_FirstName;
    private string m_LastName;

    public string FirstName
    {
      get
      {
        return m_FirstName;
      }
      set
      {
        m_FirstName=value;
      }
    }

    public string LastName
    {
      get
      {
        return m_LastName;
      }
      set
      {
        m_LastName=value;
      }
    }

    public string FullName
    {
      get
      {
        return m_FirstName + " " + m_LastName;
      }
```

```
      set
      {
        m_FirstName=value.Split(' ')[0];
        m_LastName=value.Split(' ')[1];
      }
    }

    public long FullNameLength
    {
      get
      {
        return this.FullName.Length;
      }
    }
  }
}
```

Jakmile máte komponentu .NET, můžete vytvořit spotřebovávající kód Visual Basicu 6 podle výpisu 28.6.

Výpis 28.6: Kód VB6 využívající komponentu .NET

```
Option Explicit

Public Sub Main()
  Dim o As NameComponent.NameFunctions

  Set o = New NameComponent.NameFunctions

  o.FirstName = "Bill"
  o.LastName = "Evjen"

  MsgBox "Celé jméno zní: " + o.FullName

  MsgBox "Délka celého jména je: " + CStr(o.FullNameLength)

  o.FullName = "Scott Hanselman"

  MsgBox "Jméno zní: " + o.FirstName

  MsgBox "Příjmení zní: " + o.LastName

  o.LastName = "Evjen"

  MsgBox "Celé jméno zní: " + o.FullName

  Set o = Nothing
End Sub
```

Nezapomeňte doplnit odkaz na komponentu .NET. Zadejte příkaz Project → Project References a vyberte soubor `.tlb` oné již dříve ve Visual Studiu vytvořené komponenty .NET.

Když spustíte kód ve výpisu 28.6, uvidíte, že Visual Basic 6 nemá s komunikací s komponentou rámce .NET žádný problém.

Sestavy si také můžete registrovat sami. Již dříve jsme si ukázali jak manuálně vytvářet knihovny typů interoperability nástrojem Type Library Exporter. Ten vytvořené sestavy neregistruje, ale generuje pouze knihovnu typů.

Chcete-li si sestavy zaregistrovat sami, použijte nástroj Assembly Registration Tool (`regasm.exe`). Funguje vlastně jako `regsvr32.exe`, ovšem pro komponenty .NET.

Nástroj `regasm.exe` použijte s příkazy zachycenými v následujícím příkladu:

```
regasm NameComponent.dll /tlb:NameComponentEx.tlb /regfile:NameComponent.reg
```

Přepínač `/tlb:` specifikuje název knihovny typů a volba `/regfile:` určuje název vytvářeného souboru registru, který lze používat později při instalaci a zavádění aplikace.

Časná a pozdní vazba

Předchozí příklad ilustruje práci s časným vázáním, což je technika, na kterou je zvyklá většina vývojářů Visual Basicu 6. V některých případech je ovšem žádoucí používat pozdní vazby. Zajištění pozdní vazby komponent .NET je podobné jako přístup ke komponentám ActiveX, jak to dokládá výpis 28.7.

Výpis 28.7: Pozdní vazba ve VB6

```
Option Explicit

Public Sub Main()
  Dim o As Object

  Set o = CreateObject("NameComponent.NameFunctions")

  o.FirstName = "Bill"
  o.LastName = "Evjen"

  MsgBox "Celé jméno zní: " + o.FullName

  MsgBox "Délka celého jména je: " + CStr(o.FullNameLength)

  o.FullName = "Scott Hanselman"

  MsgBox "Jméno zní: " + o.FirstName

  MsgBox "Příjmení zní: " + o.LastName

  o.LastName = "Evjen"

  MsgBox "Celé jméno zní: " + o.FullName

  Set o = Nothing
End Sub
```

Zpracování chyb

Zpracování chyb vyvolaných v komponentě .NET ve Visual Basicu 6 snadno zajistí funkčnost interoperability. Výpis 28.8 ukazuje kód Visual Basicu i C# vyvolávající vlastní výjimku. Když je parametr dělence nebo dělitele ve funkci `Divide` větší než 1 000, tak se volajícímu kódu, kterým je v našem případě Visual Basic 6, předá vlastní výjimka.

Všimněte si, že se tu nezpracovává možnost dělení nulou. To je úmyslné, aby bylo patrné, jak interoperabilita pracuje s nezpracovanými chybami.

Výpis 28.8: Vyvolávání chyb

VB

```
Public Class CustomException
  Inherits Exception

  Sub New(ByVal Message As String)
    MyBase.New(Message)
  End Sub
End Class

Public Class DivideFunction
  Public Function Divide(ByVal Numerator As Double, _
    ByVal Denominator As Double) As Double

    If ((Numerator > 1000) Or (Denominator > 1000)) Then
      Throw New CustomException("Dělenec ani dělitel nesmí " & _
        "být větší než 1000.")
    End If

    Divide = Numerator / Denominator
  End Function
End Class
```

C#

```
using System;

namespace DivideComponent
{
  public class CustomException:Exception
  {
    public CustomException(string message):base(message)
    {
    }
  }

  public class DivideFunction
  {
    public double Divide(double Numerator, double Denominator)
    {
      if ((Numerator > 1000) || (Denominator > 1000))
        throw new CustomException("Dělenec ani dělitel " +
```

```
        "nesmí být větší než 1000.");

      return Numerator / Denominator;
    }
  }
}
```

Máte již kód komponenty rámce .NET, takže ji zkompilujte s příznakem registrace pro spolupráci s COM a nazvěte ji `DivideComponent`.

Kód Visual Basicu 6 pracující s touto komponentou najdete ve výpisu 28.9. Nezapomeňte doplnit odkaz na knihovnu typů interoperability komponenty `DivideComponent` vygenerované Visual Studiem.

Výpis 28.9: VB6 zakoušející chyby rámce .NET

```
Option Explicit

Public Sub Main()
  Dim o As DivideComponent.DivideFunction

  Set o = New DivideComponent.DivideFunction

  MsgBox "1 děleno 3: " + CStr(o.divide(1, 3))

  MsgBox "1 děleno 0: " + CStr(o.divide(1, 0))

  MsgBox "2000 děleno 3000: " + CStr(o.divide(2000, 3000))

  Set o = Nothing
End Sub
```

Kód Visual Basicu 6 v ukázkovém výpisu 28.9 sice nezpracovává chyby vyvolané komponentou .NET, můžete to ovšem snadno zajistit pomocí `On Error`, což je metoda Visual Basicu 6 sloužící k zachytávání vyvolaných chyb.

Chyby nezachytávejte, ale zajistěte volbu nastavení Error Trapping v dialogu Options aplikace Visual Basic 6 na hodnotu Break.

Po spuštění aplikace správně proběhne příklad dělení jedničky trojkou; výstup je bez problémů. Pro druhý příklad, u něhož lze očekávat chybu dělení nulou, to neplatí. Místo toho se Visual Basicu 6 vrátí neplatná hodnota vlastnosti. Poslední příklad, který neprojde definovanou chybou v komponentě .NET, vyvolá podle očekávání chybu Visual Basicu.

Nasazení komponent .NET s aplikacemi COM

Nasazení komponent .NET s aplikacemi COM se podobá nasazení komponent COM. Schéma nasazení může nabýt dvou scénářů:

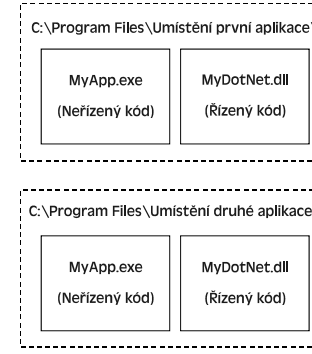
- použití soukromých sestav,
- využití sdílených sestav.

Následující oddíly popisují oba uvedené scénáře.

Soukromé sestavy

Soukromé sestavy znamenají, že se nasazení komponenty .NET instaluje v každém jednotlivém adresáři, kde se aplikace nachází, a to i na stejném počítači. Jedinou potřebnou komponentou je knihovna DLL rámce .NET a volající aplikace. Knihovna typů interoperability, kterou vytvoříte pomocí Visual Studia 2008 nebo nástroje `tlbexp.exe`, je staticky napojena na komponentu či aplikaci, jež se na danou komponentu rámce .NET odkazuje.

Jediným dodatečným úkolem je řádné zaregistrování sestavy .NET pomocí `regasm.exe`. Jedná se o doplňkový krok, který ovšem není nezbytný ve všech aplikacích .NET; je zapotřebí pouze v případě interoperability neřízeného kódu s odkazy na řízený kód. Obrázek 28.15 ilustruje použití soukromých sestav.

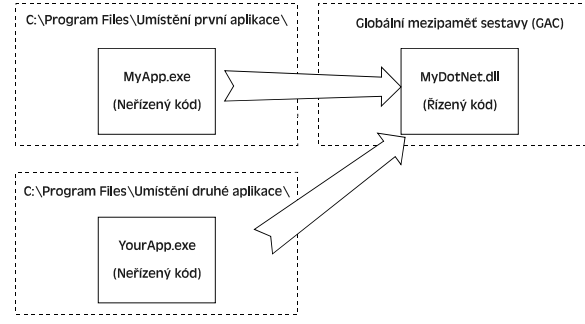


Obrázek 28.15

Veřejné sestavy

Použití veřejné sestavy ukazuje obrázek 28.16. Tento scénář zahrnuje instalaci komponenty .NET do globální mezipaměti sestav (Global Assembly Cache – GAC).

Podobně jako u soukromých sestav jsou komponenta .NET a spotřebovávající neřízený kód jedinými nezbytnostmi k nasazení – jen je ještě zapotřebí zaregistrovat sestavu interoperability pomocí `regasm.exe`.[3]



Obrázek 28.16

[3] Poznámka českého vydavatele: Nezapomeňte při tomto nasazení mít na cílovém počítači nainstalován rámec .NET Framework.

Souhrn

Při uvádění rámce .NET se někteří vývojáři obávali o osud svých existujících ovládacích prvků ActiveX a jejich místo ve vizi budoucího vývoje komponent společnosti Microsoft. Tato firma však ihned zareagovala a nabídla robustní a solidní funkčnost spolupráce s modelem COM, který zajišťuje nejen prostředky komunikace z řízeného kódu .NET k neřízenému kódu COM, ale také z neřízeného kódu modelu COM k řízenému kódu .NET. Ta druhá možnost se ani neočekávala, pro mnoho vývojářů Visual Basicu 6 a budoucích komponent .NET se však jednalo o velmi vítané rozšíření.

Vrstva interoperability umožnila společnosti Microsoft nabízet vývoj komponent .NET jako řešení nejen pro nově vytvářené aplikace, ale také pro právě vyvíjené aplikace i ty, které již byly uvolněny a jsou nyní ve fázi údržby.

Interoperabilita dala vývojářům .NET prostředky potřebné k postupné aktualizaci aplikací, aniž by je bylo nutné celé přepisovat. Umožnila jim začínat nové projekty .NET bez čekání na vývoj všech podpůrných komponent v rámci .NET.