Introducción a ADO.NET Entity Framework en Delphi Prism

Dirigido a Desarrolladores .NET con Delphi Prism | Área: Programación
Autor: Miguel Mesa | Colaborador Servicios Profesionales Danysoft

Resumen

En Microsoft ADO.NET permite crear aplicaciones de acceso a datos mediante modelos conceptuales en lugar de utilizar modelos relacionales. El objetivo es, reducir la cantidad de código y tiempo de mantenimiento a la hora de desarrollar estos tipos de aplicaciones. Visual Studio 2010 cuenta con una plantilla Entity Data Model de ADO.NET que nos permite de forma visual desarrollar un modelo conceptual, y un modelo de almacenamiento e información de asignación, de hecho el archivo .edmx que se genera define un conjunto de asignaciones entre entidades y tablas, entre el modelo conceptual y la base de datos; **Delphi Prism** es instalado como un plug-in dentro del Visual Studio pero en su caso no define ninguna plantilla relacionada con Entity Data Model ADO.NET . Nuestro objetivo será la generación e instalación de una plantilla similar en el entorno de Delphi Prism.

Introducción

En los esquemas tradicionales los desarrolladores invertían tiempo y esfuerzo a la hora del desarrollo de aplicaciones orientadas a bases de datos debido a la filosofía de trabajo entiéndase diseño de tablas, relaciones entre estas, desarrollo de procedimientos almacenados y vistas, así como la implementación de esquemas de diseño apropiados.

Con la introducción de Entity Framework esta filosofía cambia, ya que permite que los desarrolladores trabajen con datos en forma de objetos y propiedades específicas del dominio, por ejemplo, con clientes y direcciones, sin tener que pensar en las tablas de las bases de datos subyacentes y en las columnas donde almacenan estos datos.

Con Entity Framework, se pueden trabajar en un nivel más alto de abstracción cuando se tratan con datos, y se pueden crear y mantener aplicaciones orientadas a datos con menos código que en las aplicaciones tradicionales. Dado que Entity Framework es un componente de .NET Framework, las aplicaciones se pueden ejecutar en cualquier equipo en el que esté instalado .NET Framework a partir de la versión 3.5 SP1.

Delphi Prism no cuenta con un asistente para Entity Data Model que nos ayude a generar un archivo .edmx donde se almacenara nuestro modelo conceptual y de asignación. Por ello he realizado una investigación (búsquedas bibliográficas, consultas online) para ver si existía algún mecanismo para importar la plantilla predeterminada en Visual Studio (en este caso para c#).

Ron Grove (3) en su artículo nos describe como realizar la importación de una plantilla a Delphi Prism; aunque si nos remarca el hecho que la plantilla obtenida no trabajara de la misma forma que lo hace la plantilla tradicional, en esta caso para C# ya que no genera los archivos *.designer.pas ni *.metadata.pas en el caso de querer realizar operaciones de actualización a la base de datos.

Importando nuestra plantilla a Delphi Prism

Los pasos a seguir para importar una plantilla predefinida en Visual Studio a Delphi Prism son los siguientes:

- Seleccione la plantilla en C# que deseamos importar
- Copie la plantilla en algún subdirectorio creado al efecto y descomprímala.
- Copie una plantilla Prism (Oxygene) en la misma ubicación.

- Compare los archivos .vstemplate de C# y Prism para ver qué cambios son necesarios hacer.
- Si existe algún código en C# en la plantilla que vamos a importar también será necesario trasladarlo.
- Comprima la nueva plantilla y colóquela en la ubicación correcta dentro del directorio de Visual Studio IDE template
- Abra línea de comandos del DOS con derechos administrativos si estamos usando Windows 7 o Vista; aunque en XP trabajara sin problemas.
- Ir al directorio Visual Studio 2010 IDE donde se localiza devenv.exe
- Ejecute *devenv.exe /installvstemplates*

Una vez que hemos copiado la plantilla a un directorio y previamente descomprimida, editamos el archivo .vstemplate; esto lo podemos hacer con cualquier editor de texto como por ejemplo notepad. Este es un archivo XML en nuestro caso editamos el archivo ModelObjectItemCS.vstemplate:

```
<VSTemplate Version="3.0.0" xmlns="http://schemas.microsoft.com/developer/vstemplate/2005"
Type="Item">
 <TemplateData>

  <Name Package="{8889e051-b7f9-4781-bb33-2a36a9bdb3a5}" ID="500" />
  <DefaultName>Model.edmx</DefaultName>
  <Description Package="{8889e051-b7f9-4781-bb33-2a36a9bdb3a5}" ID="501" />
  <ProjectType>CSharp</ProjectType>
  <Icon Package="{8889e051-b7f9-4781-bb33-2a36a9bdb3a5}" ID="502" />
  <ProvideDefaultName>true</ProvideDefaultName>
  <AppendDefaultFileExtension>true</AppendDefaultFileExtension>
  <NumberOfParentCategoriesToRollUp>1</NumberOfParentCategoriesToRollUp>
  <RequiredFrameworkVersion>3.5</RequiredFrameworkVersion>
  <TemplateID>Microsoft.Data.Entity.Design.VSTemplate.CS</TemplateID>
  <!-- Template ID max length is 63 characters -->
 </TemplateData>

 <TemplateContent>
  <References>
   <Reference>
    <Assembly>System</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Data</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Data.Entity</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Runtime.Serialization</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Xml</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Core</Assembly>
   </Reference>
   <Reference>
    <Assembly>System.Security</Assembly>
```

```
    </Reference>
   </References>

   <ProjectItem OpenInEditor="false" SubType="" TargetFileName="$fileinputname$.edmx"
ReplaceParameters="true">ProjectItem.edmx</ProjectItem>

  </TemplateContent>

  <WizardExtension>
   <Assembly>Microsoft.Data.Entity.Design, Version=10.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a</Assembly>
   <FullClassName>Microsoft.Data.Entity.Design.VisualStudio.ModelWizard.ModelObje
ctItemWizard</FullClassName>
  </WizardExtension>

</VSTemplate>
```

En este archivo tenemos que cambiar las líneas:

<ProjectType>CSharp</ProjectType>
Por:
<ProjectType>Oxygene</ProjectType>

y

<TemplateID>Microsoft.Data.Entity.Design.VSTemplate.CS</TemplateID>
Por:
<TemplateID>Microsoft.Data.Entity.Design.VSTemplate.pas</TemplateID>

Seguidamente guardamos el archivo con otro nombre por ejemplo oxModelObjectItem.vstemplate.

El siguiente paso sería comprimir los archivos oxModelObjectItem.vstemplate y ProjectItem.edmx (la compresión tiene que ser en formato zip); una vez comprimidos los dos archivos, los copiamos en el directorio donde están las plantillas, en mi caso sería:

C:\Archivos de programa\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates\Oxygene\Data\1033

Y solo nos quedara instalarla, para ello nos vamos a la línea de comandos del sistema y como se mencionó anteriormente, me posiciono o le doy la ruta donde se ubica el archivo devenv.exe, y ejecuto el comando devenv.exe /installvstemplates

Una vez realizadas estas acciones, abrimos en Visual Studio un proyecto nuevo de **Delphi Prism** y ya seriamos capaces de ver la nueva plantilla al adicionar un nuevo Items.

Realizando pruebas con la nueva plantilla.
Para realizar algunas pruebas me he creado una base de datos muy simple llamada EJEMPLO con una tabla (USUARIOS) que tiene dos campos (ID,Nombre) y sobre la misma realizaremos nuestras pruebas. Vamos a seguir los siguientes pasos:

Campo	Tipo
ID	Int
Nombre	String

Definiendo nuestra clase

Primero vamos añadir un nuevo proyecto a nuestra solución del tipo Class Library donde colocaremos nuestro modelo:

1. Posicionamos el cursor en el nombre de la solución seleccionamos la opción Add-New Project y seleccionamos la plantilla Class Library. Le podemos poner como nombre PrismModel.
2. Renombramos la clase y el archivo .pas que nos muestras por defecto, a Usuario.
3. Y definimos cuatro propiedades ID y Nombre.

```
USUARIO = public class
  private
  protected
  public
  property ID: System.Int32; virtual;
  property Nombre: System.String; virtual;
  end;
```

Definiendo el modelo de datos

Una vez realizado esto es hora de añadir nuestra plantilla ADO.NET Entity Data Model, para ello añadiremos un nuevo proyecto de tipo Class Library donde colocaremos nuestro modelo de datos y lo nombraremos PrismDM:

1. .Posicionemos el cursos sobre la solución clic derecho Add-New Project y seleccionamos Class Library. Le ponemos como nombre PrismDM.
2. Una vez añadido el nuevo proyecto es hora de añadir nuestra plantilla para ello click derecho sobre el nuevo proyecto creado, Add-New Item y seleccionamos ADO.NET Entity Data Model. A continuación se nos muestra el asistente estándar de Microsoft y nos conectamos con la base de datos EJEMPLO.

3. El siguiente paso sería relacionar las entidades con el código ya que este no me genera el archivo .designer.* de forma automática como es el caso de C#. para ello definimos una clase que hereda de ObjectContext con una propiedad Usuarios de solo lectura del tipo IObjectSet<USUARIOS>.

```
EJEMPLOContext = public class(ObjectContext)
  private
    _usuarios: IObjectSet<USUARIO>;
  public
    constructor;
    property Usuarios: IObjectSet<USUARIO> read _usuarios;
  end;


constructor EjemploContext;
begin
  inherited constructor('name=EJEMPLOEntities', 'EJEMPLOEntities');
  _usuarios := CreateObjectSet<USUARIO>();
end;
```

Como tercer paso sería incluir otro proyecto en la solución del mismo tipo que los anteriores y lo denominaríamos PrismRepository , en este definiríamos una propiedad privada del tipo EJEMPLOContext y método para obtener los usuarios

```
CustomerRepository = public class
  private
    _context : EJEMPLOContext;
  protected
  public
    constructor();
    constructor(context: EJEMPLOContext);
    method GetAllUsuarios: List<USUARIOS>;
```

```
  end;

constructor UsuarioRepository();
begin
  _context := new EJEMPLOContext();
end;

constructor UsuarioRepository(context: EJEMPLOContext);
begin
  _context := Context;
end;

method UsuarioRepository.GetAllUsuarios: List<USUARIO>;
begin
  result := _context.Usuarios.ToList;
end;
```

El último paso sería mostrar los datos. Para ello en el proyecto principal vamos a la ventana principal de la aplicación (MainForm). Arrastramos un DataGridViews, un bindingSource y un botón. Primero relacionamos la propiedad BindingSource del DataGridView con bindingSource1 y seguidamente generamos un evento para el botón de tipo Click, y llamaremos al método GetAllUsuarios. El listado se muestra a continuación.

```
MainForm = partial class(System.Windows.Forms.Form)
  private
   _usuarioRepository: UsuarioRepository;
    method button1_Click(sender: System.Object; e: System.EventArgs);
    method ResetGrid();
  protected
    method Dispose(disposing: Boolean); override;
  public
    constructor;
  end;


constructor MainForm;
begin
  InitializeComponent();
  _usuarioRepository := new UsuarioRepository();
end;

method MainForm.Dispose(disposing: Boolean);
begin
  if disposing then begin
    if assigned(components) then
      components.Dispose();
  end;
  inherited Dispose(disposing);
end;

method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
begin
  self.ResetGrid;
  var lUsuarios := _usuarioRepository.GetAllUsuarios;
  bindingSource1.DataSource := lUsuarios;
end;

method MainForm.ResetGrid();
begin
  dataGridView1.Rows.Clear;
  dataGridView1.AutoGenerateColumns := True;
  bindingSource1.Clear;
end;
```

Importando *.designer.cs a Delphi Prism.

Existe otra forma de ahorrarnos todo lo anterior, para ello abramos un nuevo proyecto en **Delphi Prism** pero en este caso de Consola. Agregamos nuestra plantilla ADO.NET Entity Data Model y configuramos nuestro modelo igual que en la sección anterior. Paralelamente a esto iniciamos otra sesión del Visual Studio, abrimos un nuevo proyecto también de tipo Consola, pero en este caso de C# y agregamos un nuevo elemento ADO.NET Entity Data Model y lo configuramos igual que en caso del proyecto de **Delphi Prism**; el asistente si nos genera un archivo ***.designer.cs** y **App.Config** el primer archivo lo utilizaremos para importarlo a nuestro proyecto de Delphi Prism.

Esto lo realizaremos siguiendo estos pasos:

1. Nos posicionamos sobre el proyecto. Click derecho Add-Import C# y seleccionamos el archivo que queremos importar en este caso ***.designer.cs** el asistente de importación nos convertirá nuestro archivo cs en un archivo pas.
2. Dentro del archivo obtenido hay métodos declarados como **partial** y que no tienen implementación, por ello hay que añadir un **empty;** sino el compilador nos devolverá un error.
3. Tendríamos también que importar el archivo **App.Config,** ya que en el mismo se almacenan los parámetros de conexión.

Conclusión

Aunque la plantilla generada no resulte gran cosa ya que no generará el código de *.desgner.pas ni App.Config nos puede ser ayuda a la hora de visualizar nuestro modelo de datos.

Bibliografía.

1. http://msdn.microsoft.com. ADO.NET Entity Framework
2. O'Reilly, Julia Lerman. Programming Entity FrameWork, 2010.
3. Ron Grove Install ItemTemplate into DelphiPrism. http://www.rongrove.com/%28A%28Gg1z0o5LywEkAAAAMGQ0YzFmODMtMWVlYS00ZDRmLTljMGMtMWE5OTY3Y2RmMDAzZSHJYIZI7n8WRb813IFdYa78Kv41%29%29/Article/ViewArticle/31/install-itemtemplate-into-delphi-prism