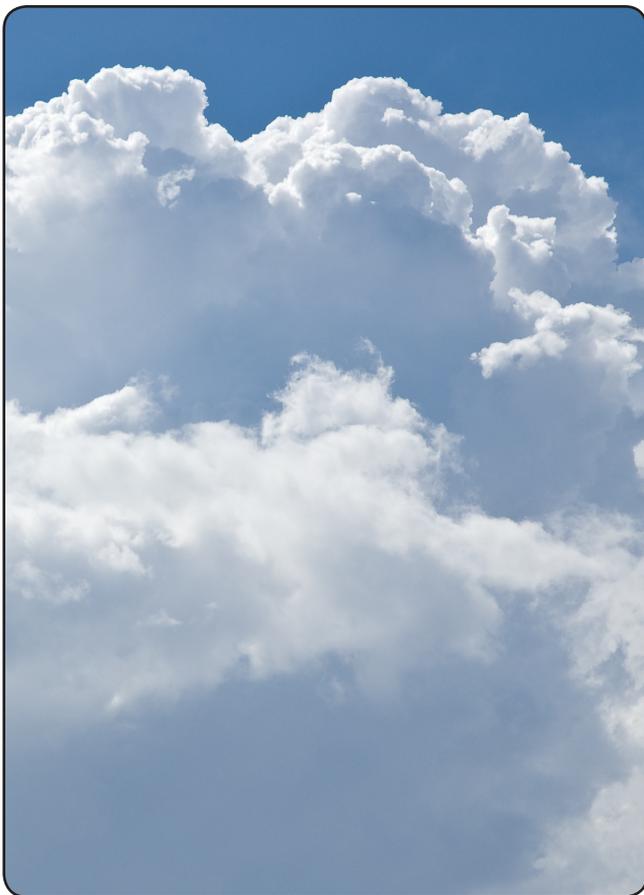


Nesta seção você encontra artigos intermediários sobre a tecnologia .Net



De que se trata o artigo

Como criar um serviço que expõe dados com ADO.NET Data Services;
Integração do ADO.NET Entity Framework com o ADO.NET Data Services;
Criação de uma aplicação que consome dados expostos com ADO.NET Data Services;

Para que serve

Criar um modelo de entidades e expô-lo na Web como um serviço. Em seguida acessar e testar as operações CRUD neste modelo.

Em que situação o tema é útil

Aplicações que se utilizam de conceitos como S+S e Cloud Computing, onde parte dos recursos é disponibilizada através de serviços na Web poderão contar com o ADO.NET Data Services para alcançar este objetivo.



Rodrigo Sendin

rodrigo.sendin@terra.com.br

É Arquiteto de Sistemas e trabalha com desenvolvimento de Software há mais de 12 anos. Tecnólogo formado pela FATEC de Americana e MCP .NET. É consultor da TauNet Consulting e escreve artigos para .net Magazine e WebMobile. Blog: <http://www.algoritma.com.br/rodrigo.sendin>

ADO.NET Data Services

Operações CRUD com o ASTORIA



Resumo do DevMan

O ADO.Net Data Services permite criar aplicações RESTful com pouquíssimo esforço, aplicando padrões de mercado, e com segurança, e integrando com fontes de dados conhecidas, como o Entity Framework. Neste artigo você vai ver como trabalhar com o ADO.NET Entity Framework em conjunto com o ADO.NET Data Services, para a construção de aplicações onde o modelo de dados é exposto como um Serviço na Web.

Neste artigo irei abordar estes assuntos sob um aspecto estritamente prático. Não vamos nos aprofundar nos conceitos teóricos que envolvem o *ADO.NET Data Services*, ou como funciona em detalhes o padrão *REST*. Neste artigo vamos utilizar os recursos que temos do *ADO.NET Data Services* para expor um modelo de Entidades feito com o *ADO.NET Entity Framework* como um serviço na internet, e posteriormente veremos como utilizar este serviço em nossas aplicações. A pergunta que você deve estar fazendo é: “Mas por quê?”.

Quem já trabalhou *WebServices* ou mesmo com o *WCF*, sabe que essa é uma necessidade muito comum nos dias de hoje. Disponibilizar dados através de serviços web é uma prática muito útil para integrar aplicações pela internet. O *ADO.NET Data Services* (também conhecido como *Astoria*, seu codinome) permite fazer isso de uma maneira muito mais simples, como veremos a seguir.

Ambiente

O *ADO.NET Data Services* expõe o que chamamos de *Modelo de Entidades*, e a forma mais fácil de criarmos um modelo de entidades é através do *ADO.NET Entity Framework (EF)*. Como veremos, com o *EF* fazemos um mapeamento das tabelas do

banco de dados, para que possamos acessá-las através do LINQ to Entities. Já escrevi alguns artigos sobre o Entity Framework nessa mesma revista em edições anteriores.

Tanto o ADO.NET Entity Framework quanto ADO.NET Data Services são recursos que vieram complementar o .NET Framework 3.5 no Service Pack 1, que foi lançado recentemente. Sendo assim, além do Visual Studio 2008 você vai precisar baixar e instalar o Service Pack 1, que pode ser encontrado neste link: <http://tinyurl.com/vs2008sp1>.

Com o ambiente já pronto, podemos começar criando uma nova Solution no Visual Studio 2008. Para isso, com o VS2008 aberto, vá até a opção File / New Project. Como iremos criar vários projetos de exemplo neste artigo, vamos criar uma solução vazia para a inclusão destes projetos. Para isso, como você confere na Figura 1, em Other Project Types selecione Visual Studio Solutions, e nos templates selecione o Blank Solution. Em seguida informe ArtigoAstoria em Name e clique em OK.

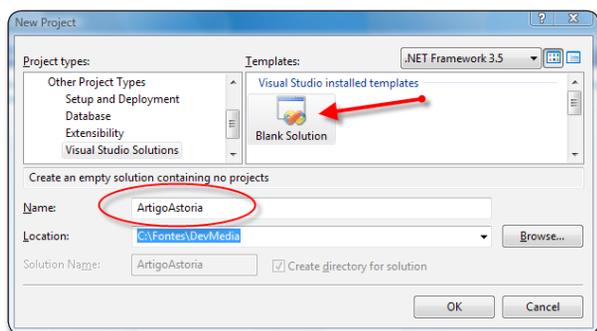


Figura 1. Criando uma solução vazia no Visual Studio

Modelo de Entidades

Agora que já temos uma solução criada, podemos começar a criar nossos exemplos. A primeira coisa que devemos fazer então é a modelagem das nossas entidades no ADO.NET Entity Framework. Uma das características deste framework é que ele permite que você crie um modelo vazio e somente depois faça o mapeamento Objeto/Relacional.

Outra possibilidade é já criar o modelo baseado em tabelas de um banco de dados, resultando em um modelo de entidades já mapeado com a base de dados relacional. Para facilitar os nossos exemplos vamos utilizar a segunda opção, e como não poderia ser diferente vamos aproveitar o database Northwind da Microsoft, que é ótimo para a construção de exemplos.

Como explicado na nota, baixe e instale o database Northwind. Neste artigo estaremos utilizando um SQL Server 2005.

Nota

O Database Northwind é um banco de dados do SQL Server para testes e com estrutura definida para uma aplicação de vendas, onde temos Produtos, Clientes, Fornecedores, Pedidos, etc. A Microsoft disponibiliza este database já com dados através do seguinte link: <http://tinyurl.com/northwinddb>

Faça o download e execute o arquivo de instalação. Ao término da instalação, os databases de exemplo do SQL Server 2000 serão instalados na pasta C:\SQL Server 2000 Sample Databases.

Você pode obviamente movê-los para o local da sua conveniência.

Agora vá até a Solution Explorer do Visual Studio, clique com o botão direito sobre a solução ArtigoAstoria e escolha a opção Add / New Project. Em Visual C# selecione o template Class Library. Informe ArtigoAstoria.Model em Name e clique em Add. Não esqueça de selecionar o .Net Framework 3.5, ou o Astoria não poderá ser adicionado ao projeto.

Este é o projeto Class Library que irá conter o modelo de entidades a ser feito com o ADO.NET Entity Framework. Você pode apagar a Class1.cs do projeto pois ela não será utilizada. Em seguida, com o botão direito sobre o projeto escolha a opção Add / New Item. Como você confere na Figura 2, na Categoria Data temos o template ADO.NET Entity Data Model. Escolha-o, informe DM_Northwind.edmx em Name, e clique em Add.

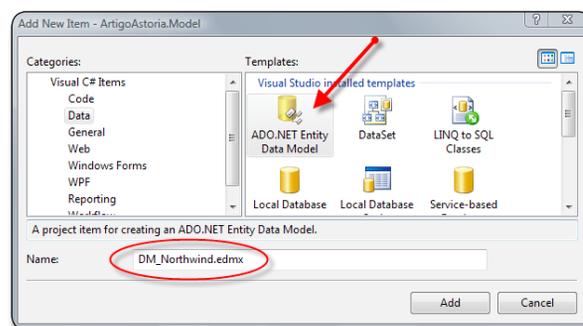


Figura 2. Criando novo Modelo de Entidades

Um Wizard será iniciado para lhe ajudar na criação do Modelo. A primeira tela desse Wizard pergunta justamente se você quer criar um modelo baseado em um banco de dados, ou se quer um modelo vazio. Vamos escolher a opção Generate from Database e clicar em Next.

Na segunda etapa do Wizard é necessário criar uma conexão com o database Northwind (veja na Figura 3). Note que a string de conexão tem um padrão um pouco diferente do que estamos acostumados no ADO.NET. Veja também que ela será armazenada no arquivo App.config com o nome NorthwindEntities. Clique em Next para prosseguir.

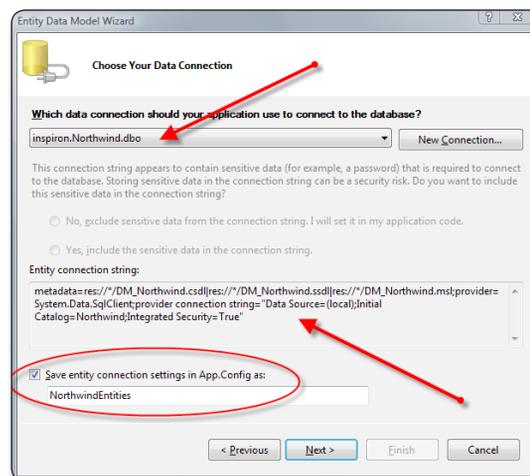


Figura 3. Criando String de Conexão com o Database Northwind

Na próxima etapa do *Wizard* precisamos selecionar as tabelas que queremos incluir em nosso modelo. Note na **Figura 4**, que estamos selecionando apenas as tabelas *Categories* e *Products* para criarmos exemplos simples e objetivos.

Veja que além de tabelas podemos incluir no modelo objetos como *Stored Procedures* e *Views*. Note também que o namespace do modelo será criado com o nome *NorthwindModel*. Em seguida basta clicar em *Finish* para concluir a criação do modelo.

Veja que o modelo é criado na *Solution Explorer* e aparece como um diagrama que pode ser alterado. Não vamos nos aprofundar nos detalhes de um modelo de entidades, para isso sugiro que você leia um dos artigos que eu e outros autores escrevemos a respeito na *.Net Magazine*.

Como você pode conferir na **Figura 5**, em um modelo de entidades temos algumas janelas novas. Além do *designer* que já é comum em modelagens de *Datasets* ou *LINQ to SQL*, temos uma janela chamada *Model Browser* onde encontramos todos os elementos que participam deste *Mapeamento Objeto/Relacional*.

Também temos uma janela chamada *Mapping Details* que é onde mapeamos cada propriedade de uma entidade com uma coluna de uma tabela no banco de dados. Veja este mapeamento da entidade *Categories*, com sua respectiva tabela.

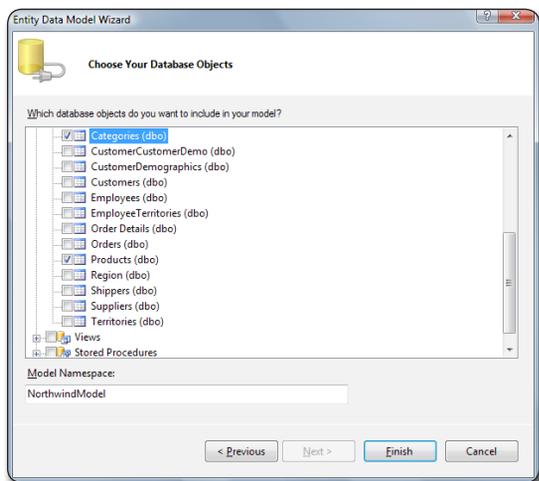


Figura 4. Selecionando Tabelas do Database Northwind para a criação do Modelo

Quando criamos um modelo de entidades baseado em um banco de dados já existente, não precisamos nos preocupar com as questões que envolvem o *Mapeamento Objeto/Relacional*, pois isso é feito automaticamente pelo modelo.

Testando o Modelo sem o Astoria

Para que possamos comprovar alguns benefícios do *ADO.NET Data Services*, antes de realizarmos um exemplo diretamente com ele, vamos acessar o nosso modelo de Entidades da maneira tradicional com uma referência direta ao projeto.

Para isso crie um novo projeto na *Solution* do tipo *Console Application*, e chamado *ArtigoAstoria.ConsoleDireto*. Como o próprio nome sugere a idéia é acessar o modelo de entidades diretamente, sem passar por um serviço do *Astoria*.

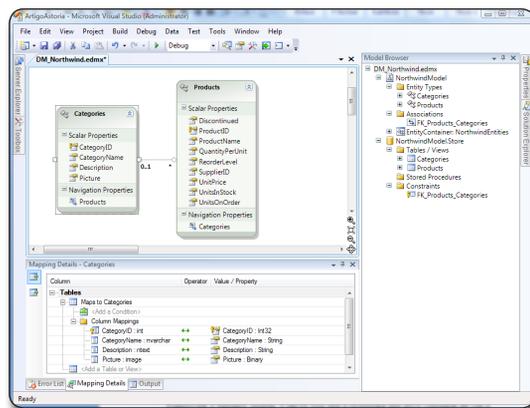


Figura 5. Modelo de Entidades criado com o ADO.NET Entity Framework

Este projeto *Console Application* é basicamente o que precisamos para disparar consultas no nosso modelo e verificar o resultado em uma janela de console. Para que possamos acessar o modelo, no entanto, é preciso antes fazer algumas coisas.

Primeiro é necessário adicionar uma referência de projeto, que diz que o projeto *ArtigoAstoria.ConsoleDireto* acessa o projeto *ArtigoAstoria.Model*. Para isso clique com o botão direito sobre o projeto recém criado e escolha a opção *Add Reference*.

Veja que nesta janela temos uma aba chamada *Projects*, e dentro dela temos o projeto *ArtigoAstoria.Model*. Basta selecioná-lo e clicar em *OK*. Com isso temos acesso ao nosso modelo.

Além disso, para podermos executar *queries* no modelo de entidades, vamos precisar adicionar uma referência ao namespace *System.Data.Entity* do *.NET 3.5*, que não vem adicionado por default em aplicações console.

Sendo assim, clique novamente com o botão direito sobre o projeto *ArtigoAstoria.ConsoleDireto* e escolha a opção *Add Reference*. Agora, na aba *.NET* selecione o namespace *System.Data.Entity* como mostra a **Figura 6**.

Uma terceira configuração ainda precisa ser feita. Precisamos configurar a *string* de conexão que foi criada no Modelo, aqui no projeto Console. Para isso você deve criar um novo item do tipo *Application Configuration File (App.config)* no projeto *ArtigoAstoria.ConsoleDireto*. E dentro deste arquivo você deve incluir o mesmo conteúdo que tem no *App.config* do projeto *ArtigoAstoria.Model*, veja um exemplo na **Listagem 1**.

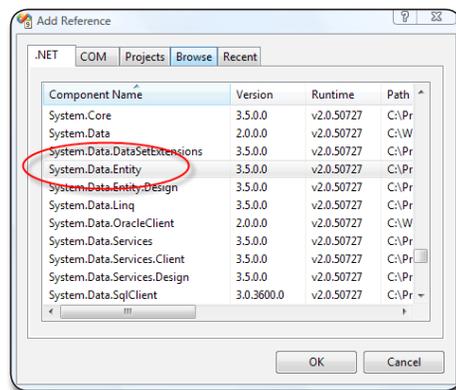


Figura 6. Adicionando referencia ao namespace System.Data.Entity



Nota do DevMan

Mapeamento objecto-relacional (ou O/RM) é uma técnica de desenvolvimento utilizada para reduzir a diferença de impedância da programação orientada a objetos com relação a bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; irá usar uma interface de programação simples que faz todo o trabalho de persistência.

Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa. A relação entre as tabelas onde originam os dados e o objeto que os disponibiliza é configurada pelo programador, isolando o código do programa das alterações à organização dos dados nas tabelas do banco de dados.

A forma como este mapeamento é configurado depende da ferramenta que será utilizada. Como exemplo, o programador que use Hibernate na linguagem Java (ou NHibernate para .NET) pode usar arquivos XML ou o sistema de anotações que a linguagem providencia.

Alguns exemplos de Ferramentas O/RM são:

- ADO.NET Entity Framework: Ferramenta O/RM da Microsoft baseada na linguagem LINQ
- SQLAlchemy: Um poderoso sistema ORM para Python

- Hibernate: Uma ferramenta de mapeamento objeto relacional para Java
- NHibernate: Uma ferramenta de mapeamento objeto relacional para .NET (portada do Hibernate)
- OJB: Uma ferramenta de mapeamento objeto relacional para Java, da Apache Software Foundation
- Django (framework web): Framework de desenvolvimento web escrito em Python que possui um ORM próprio.

Além dessas, podemos citar as seguintes ferramentas para .NET, que de uma forma ou outra tentam atender à essa necessidade: .NET Persistence, BBADDataObjects, DataObjects.NET, Data Tier Modeler for .NET, DotNorm, Eldorado.NET, Entity Broker, eXpress Persistent Objects for .NET, FastObjects.NET, JC Persistent Framework, LBLGen Pro, ModelWorks, No-lics.NET, Norm, Norpheme, ObjectBroker, ObjectSpaces, ObjectSpark, Objectz.NET, OJB.NET, OPF.NET (Object Persistent Framework), ORM.NET, Pragmatier Data Tier Builder, RapTier, Sisyphus Persistence Framework, TierDeveloper, Bob.NET, ObjectPistor.NET, Genome.

Esta definição foi baseada na definição de O/RM do Wikipédia: http://pt.wikipedia.org/wiki/Mapeamento_objeto-relacional

A lista de ferramentas O/RM para .NET Framework, foi retirada do seguinte link: <http://weblogs.asp.net/yreynhout/archive/2003/10/07/30798.aspx>

Listagem 1. Arquivo App.config para acesso ao Modelo de Entidades

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="NorthwindEntities" connectionString="metadata=res://*/DM_Northwind.csdl|res://*/DM_Northwind.ssdl|res://*/DM_Northwind.msl;provider=System.Data.SqlClient;provider connection string="Data Source=(local);Initial Catalog=Northwind;Integrated Security=True;MultipleActiveResultets=True";" providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Agora já podemos fazer nossos primeiros testes com acesso direto ao modelo de entidades. Abra o arquivo *Program.cs*, e conforme você pode ver na **Listagem 2**, vamos incluir o código necessário para fazer uma simples consulta na tabela de Categorias.

Primeiro note que foram adicionados dois *namespaces* na área de *using*. O primeiro aponta para o projeto *ArtigoAstoria.Model* e o segundo é o namespace *System.Data.Object* que é necessário para utilizarmos a classe *ObjectQuery*, que veremos a seguir.

Em seguida dentro da classe criamos uma instância do modelo, que foi chamada simplesmente de *dm*. Dentro do método *Main* estamos chamando o método *SelectCategories* que foi definido mais abaixo.

Em *SelectCategories* estamos disparando uma *query LINQ* no modelo, que irá listar todos os *IDs* e nomes da entidade *Categories*. Logo abaixo fazemos um *cast* do resultado para um objeto da classe *ObjectQuery*, para que seja possível utilizar o método *ToTraceString*. Este método irá nos mostrar qual foi o comando *T-SQL* gerado pelo *LINQ to Entities*, e que será

disparado no *database*. Isso é muito importante para que você saiba com que qualidade a *query* está sendo gerada.

E logo abaixo estamos percorrendo todas as categorias encontradas pela *query* e exibindo-as no *Console*. Para fazer um teste basta salvar, compilar e executar a aplicação. O resultado você confere na **Figura 7**. Com isso temos certeza que o Entity Framework está funcionando conforme o esperado.

```
file:///C:/Fontes/DevMedia/ArtigoAstoria/ArtigoAstoria.ConsoleDireto/bin/Debug/ArtigoAstoria.Co...
SELECT
1 AS [C1],
[Extent1].[CategoryID] AS [CategoryID],
[Extent1].[CategoryName] AS [CategoryName]
FROM [dbo].[Categories] AS [Extent1]

LISTA DE CATEGORIAS::
1 - Beverages
2 - Condiments
3 - Confections
4 - Dairy Products
5 - Grains/Cereals
6 - Meat/Poultry
7 - Produce
8 - Seafood
```

Figura 7. Exemplo de Acesso direto ao Modelo de Entidades

Expondo o Modelo como um Serviço

Para expormos o modelo de entidades como um serviço, temos que criar este serviço em uma aplicação Web, e portanto temos que criar um novo projeto Web na Solução. Clique com o botão direito sobre a solução na *Solution Explorer* e escolha a opção *Add / New Project*.

Em seguida selecione o template *ASP.NET Web Application*, informe *ArtigoAstoria.Servico* em *name* e clique em *OK*. Neste projeto iremos acessar o nosso modelo de entidades que se encontra no projeto *ArtigoAstoria.Model* e expô-lo como um serviço web.

Listagem 2. Alterações na classe Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

// Referência ao Modelo de Entidades
using ArtigoAstoria.Model;

// Vamos utilizar o ObjectQuery para poder
// ver as queries que são geradas pelo LINQ
using System.Data.Objects;

namespace ArtigoAstoria.ConsoleDireto
{
    class Program
    {
        // Criando uma instância do Data Model
        public static NorthwindEntities dm = new
            NorthwindEntities();

        static void Main(string[] args)
        {
            // Chamada ao Método SelectCategories
            SelectCategories();
        }

        // Método que Lista Todas as Categorias
        private static void SelectCategories()
        {
            // Querie que lista Todas as Categorias
            var result = from c in dm.Categories
                select new { c.CategoryID, c.CategoryName };

            // Imprimindo o comando T-SQL que foi gerado
            // e será disparado no banco
            Console.WriteLine((result as ObjectQuery).
                ToTraceString());

            // Imprimindo o resultado da Query
            Console.WriteLine();
            Console.WriteLine("LISTA DE CATEGORIAS:");
            foreach (var item in result)
            {
                Console.WriteLine("{0} - {1}",
                    item.CategoryID,
                    item.CategoryName);
            }

            System.Console.ReadKey();
        }
    }
}
```

Sendo assim precisamos adicionar uma referência ao projeto de modelo. Clique com o botão direito sobre a *Web Application*, selecione a opção *Add Reference*, em *Projects* marque o projeto *ArtigoAstoria.Model* e clique em *OK*.

Clique novamente com o botão direito sobre a aplicação web, mas escolha agora a opção *Add/New Item*. Como você confere na **Figura 8**, na Categoria *Web* selecione o template *ADO.NET Data Service*, informe *NWService.svc* em *Name* e clique em *Add*.

Com isso será criado o serviço *NWService.svc* juntamente com seu *code-behind* *NWService.svc.cs*, que você pode conferir como é gerado na **Listagem 3**. Note que a classe *NWService* herda de *DataService<T>* que usa *Generics*. Veja que temos um comentário acusando um *TODO* bem aqui. No lugar deste comentário vamos precisar informar a classe onde está o nosso modelo de entidades.

Além disso, veja que temos um método *InitializeService*, e dentro dele temos uma série de comentários que falam a respeito da questão da segurança do acesso ao Serviço. Por padrão, o serviço está totalmente fechado, não dando direito a nenhum acesso ao modelo. A liberação do acesso deve ser feita explicitamente aqui, e é altamente recomendado que você tenha um cuidado extra ao fazer a configuração do acesso em ambientes de produção.

Em ambientes de testes, como é o nosso caso, podemos liberar o acesso completo ao modelo, para que possamos avaliar todas as possíveis operações. Veja na **Listagem 4** como vai ficar nossa classe *NWService.svc.cs* para que o serviço exponha o modelo de entidades e dê plenos direitos de acesso ao modelo.

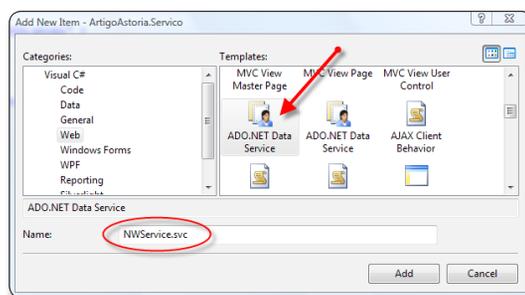


Figura 8. Criando novo serviço com ADO.NET Data Services

Listagem 3. Código do NWService.svc gerado automaticamente

```
using System;
using System.Collections.Generic;
using System.Data.Services;
using System.Linq;
using System.ServiceModel.Web;
using System.Web;

namespace ArtigoAstoria.Servico
{
    public class NWService :
        DataService< /* TODO: put your data source class name here */ >
    {
        // This method is called only once to initialize
        // service-wide policies.
        public static void InitializeService(IDataServiceConfigu
            ration config)
        {
            // TODO: set rules to indicate which entity sets and
            // service operations
            // are visible, updatable, etc.

            // Examples:
            // config.SetEntitySetAccessRule("MyEntityset",
            // EntitySetRights.AllRead);

            // config.SetServiceOperationAccessRule("MyService
            // Operation",
            // ServiceOperationRights.All);
        }
    }
}
```

Note também que foi definido para a classe o atributo *ServiceBehavior* com o parâmetro *IncludeExceptionDetailInFaults* igual a *True*. Isso é importante, pois caso ocorra qualquer exceção na execução do serviço, nós tenhamos um retorno detalhado do erro ocorrido.

Listagem 4. Apontando serviço ao modelo de entidades e configurando regras de acesso

```
namespace ArtigoAstoria.Servico
{
    [System.ServiceModel.ServiceBehavior(IncludeExceptionDetailInFaults = true)]
    public class NWService : DataService<ArtigoAstoria.Model.NorthwindEntities>
    {
        public static void InitializeService(IDataService Configuration config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
        }
    }
}
```

Além disso, para que nosso serviço consiga acessar com sucesso o modelo de entidades, é necessários configurarmos a connection string no arquivo *Web.config*, assim como fizemos na *App.config* da aplicação console.

Para isso abra o *Web.config* e inclua na seção *ConnectionString* a nossa *NorthwindEntities*, assim como mostra a **Listagem 5**.

Listagem 5. ConnectionString no Web.config

```
<connectionStrings>
<add name="NorthwindEntities"
connectionString="metadata=res://*/DM_Northwind.csd|res://*/DM_Northwind.ssd|res://*/DM_Northwind.msl;provider=System.Data.SqlClient;providerconnection string=&quot;Data Source=(local);Initial Catalog=Northwind;Integrated Security=True;MultipleActiveResultSets=True&quot;; providerName="System.Data.EntityClient" />
</connectionStrings>
```

Com isso já podemos fazer nosso primeiro teste do serviço. Configure o projeto Web como o projeto de início da solução, salve, compile e execute. O resultado deve ser o mostrado aqui na **Figura 9**, onde temos um *XML* mostrando que o nosso modelo contém as entidades *Categories* e *Products*.

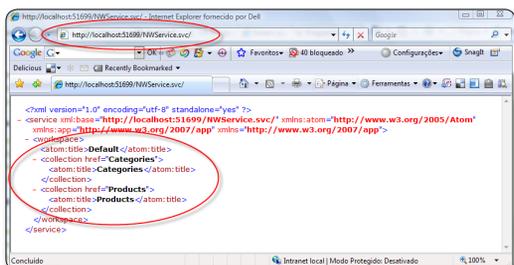


Figura 9. Entidades existentes no Modelo exposto pelo Serviço

Caso você não esteja enxergando o *XML* nessa formatação, é necessário desabilitar o recurso de renderização de *Feeds* do *Internet Explorer* (ou outro *browser* que você utilize). No *Internet Explorer 7* vá em *Ferramentas / Opções da Internet / Conteúdo / Feeds / Configurações*, e desmarque a opção de *Ativar o modo de exibição de leitura de feed*. Confira na **Figura 10**.

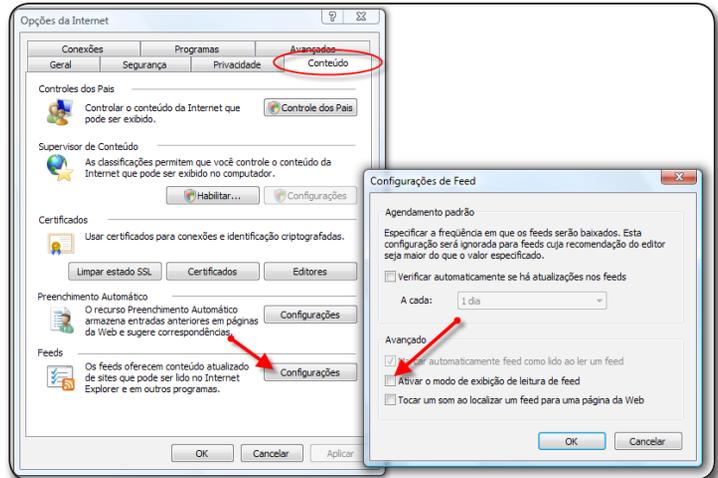


Figura 10. Desabilitando a leitura de feed do Internet Explorer

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

COBREBEMX

-  56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
-  GERAÇÃO DE BOLETOS ON LINE;
-  GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
-  MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM WWW.COBREBEM.COM

obrebem
Tecnologia

REST

Com a aplicação executando você já pode ver na própria **Figura 9** que temos acesso ao modelo de entidades. Este acesso é feito via *HTTP* através do *REST*, que é um padrão para acesso de recursos através da *URI*. Não vou entrar nos meandros deste padrão para não alongar demais este artigo e desviar do assunto.

Para você entender os benefícios do *REST* basta fazer alguns exemplos de pesquisa no browser. Utilize as URLs da **Tabela 1**, e comprove você mesmo os resultados que irá obter no browser em XML. Veja por exemplo na **Figura 11**, o resultado obtido com a URI: `http://localhost:51699/NWService.svc/Categories(1)`. A porta, no caso 51699, é gerada automaticamente pelo Visual Studio, e provavelmente a sua é diferente. Acerte então a porta de acordo.

URI	O que retorna:
<code>http://localhost:51699/NWService.svc/Categories</code>	Todas as Categorias cadastradas
<code>http://localhost:51699/NWService.svc/Products</code>	Todos os Produtos cadastrados
<code>http://localhost:51699/NWService.svc/Categories(1)</code>	Somente a Categoria com Id = 1
<code>http://localhost:51699/NWService.svc/Products(7)</code>	Somente o Produto com Id = 7
<code>http://localhost:51699/NWService.svc/Products(7)/Categories</code>	A Categoria do Produto 7
<code>http://localhost:51699/NWService.svc/Categories(1)/Products</code>	Todos os Produtos da Categoria 1

Tabela 1. Exemplos de pesquisas no Modelo com REST

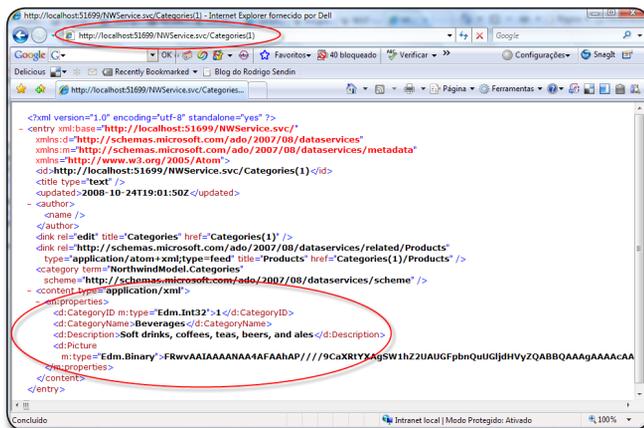


Figura 11. Listando apenas a Categoria com ID = 1

Note que nada disso foi codificado. Todas essas funcionalidades já são nativas do próprio *ADO.NET Data Services* em conjunto com o *Entity Framework*.

E estes são os exemplos mais simples do que conseguimos fazer com o *REST* no *ADO.NET Data Services*. Você encontra uma lista bem mais detalhada das possibilidades de consulta e sintaxe neste artigo publicado no *MSDN*: <http://msdn.microsoft.com/en-us/library/cc907912.aspx>

Testando o Serviço em Console

Mas não é exatamente assim que queremos utilizar o nosso serviço. O ideal é vermos como acessar o serviço em uma aplicação *.NET*. Para concluirmos o nosso raciocínio, vamos

fazer um novo teste em uma aplicação *Console*, porém agora não vamos acessar o modelo diretamente, vamos acessá-lo através do serviço criado com o *ADO.NET Data Services*.

Clique com o botão direito sobre a Solução na *Solution Explorer* e escolha a opção *Add / New Project*. Selecione o template *Console Application*, informe *ArtigoAstoria.ConsoleServico* em *Name* e clique em *OK*.

Como vamos agora acessar o modelo através do serviço, não podemos adicionar uma referência direta ao modelo. Ao invés disso precisamos adicionar uma referência ao serviço, que basicamente vai gerar um *Proxy* para acessarmos o modelo.

Antes da versão final do *ADO.NET Data Services* no *Service Pack 1*, era preciso utilizar um aplicativo externo para a geração deste *Proxy*, chamado de *datasvcutil.exe*. Porém, agora na versão final, como esperado, essa funcionalidade passou para dentro do *Visual Studio*.

Você pode adicionar uma referência ao serviço do *Astoria* como se fosse um serviço *WCF* qualquer. Clique com o botão direito sobre o projeto *ArtigoAstoria.ConsoleServico* e escolha a opção *Add Service Reference*. E como você pode conferir na **Figura 12**, para “descobrir” os serviços que estão na mesma *solution*, basta clicar no botão *Discover / Services in Solution*. Em seguida o endereço do serviço irá aparecer juntamente com alguns detalhes do modelo que ele expõe.

Para completar, informe *SV_Northwind* em *Namespace* e clique em *OK*. Note que estamos adicionando uma referência a um serviço que está na mesma *solution*, porém o acesso é feito pelo endereço que você vê na figura, e portanto você pode adicionar referências à qualquer serviço que esteja publicado na *internet* ou na sua *intranet*.

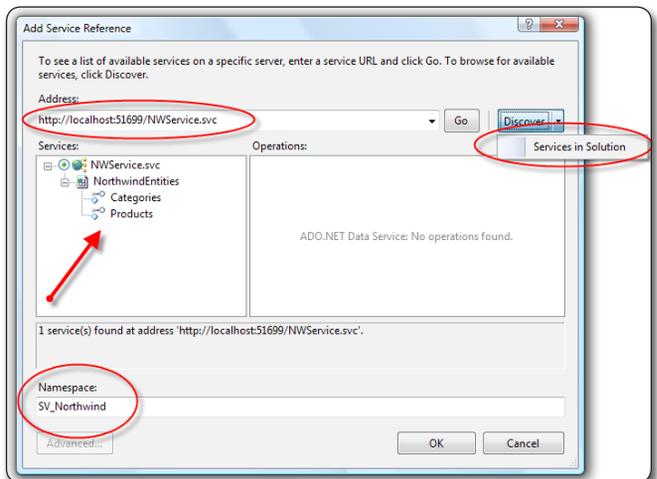


Figura 12. Adicionando referência ao serviço

Lembre-se que no exemplo anterior com acesso direto ao modelo, foi preciso adicionar uma referência ao *namespace System.Data.Entity*. Pois bem, esse é o *namespace* necessário para utilizarmos o *LINQ to Entities*.

Agora vamos utilizar um outro *provider LINQ*, que irá fazer consultas sobre o *ADO.NET Data Services*, *provider* este

que algumas pessoas estão chamando de *LINQ to REST*. A diferença é simples, enquanto no *LINQ to Entities* as *queries LINQ* são convertidas em comandos *T-SQL*, no *LINQ to REST* as *queries LINQ* são convertidas em *URIs* no padrão *REST*, como nos exemplos acima.

Para podermos usar o *LINQ* no *ADO.NET Data Services*, isso precisamos incluir uma referência ao *namespace System.Data.Services.Client*, como você pode conferir na **Figura 13**.

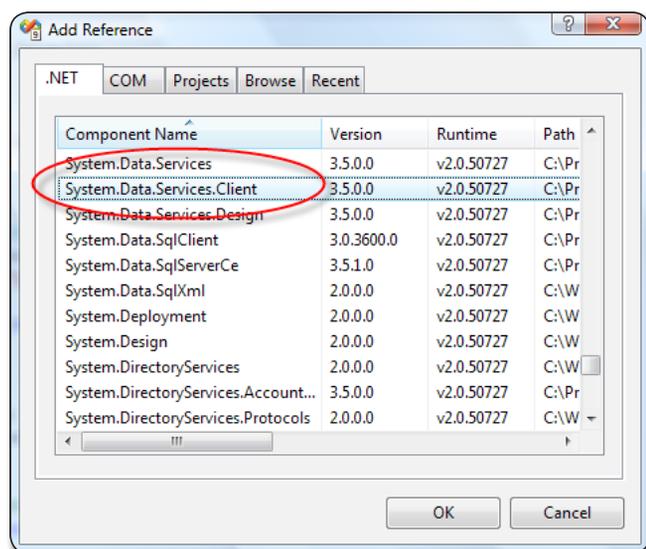


Figura 13. Adicionando referência ao *System.Data.Services.Client*

Agora que temos o serviço feito e todas as referências necessárias adicionadas, vamos testá-lo. Abra a classe *Program.cs* da aplicação *ArtigoAstoria.ConsoleService*, e alterá-la de forma que fique como mostra a **Listagem 6**.

A primeira coisa que você vai notar neste código é que ele é muito parecido com o código da **Listagem 2**, que fizemos para acessar o modelo diretamente. Vamos então às diferenças. A primeira é que aqui não importamos nenhum namespace.

Em seguida a instância do modelo é criada através do Serviço, de forma muito similar ao feito anteriormente. A diferença é que aqui precisamos definir a *URI* da onde o serviço está publicado.

Depois, no método *SelectCategories* estamos disparando uma *query LINQ*, com a única diferença que aqui não podemos gerar um tipo anônimo no comando *SELECT*, isso não é suportado pelo *ADO.NET Data Services*. Todos os campos devem ser consultados.

Em seguida estamos exibindo na *Console* a *URI* que é gerada através da *query LINQ*. Fazemos isso simplesmente chamando o método *ToString()* do *result*. E para finalizar, estamos listando o resultado da *query* assim como foi feito antes, sem mudança nenhuma. Salve, compile e execute. Veja o resultado na **Figura 14**.



Figura 14. Resultado da Query feita através do *ADO.NET Data Services*

Note que ao invés de gerar um comando *T-SQL*, nossa *query LINQ* gerou uma *URI* no padrão *REST*. Se copiarmos e colarmos essa *URI* no *Browser*, nós teremos o mesmo resultado no formato *XML*.

Listagem 6. Acessando o modelo através do serviço criado

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArtigoAstoria.ConsoleService
{
    class Program
    {
        // Criando uma instância do Data
        Model exposto pelo serviço
        public static SV_Northwind.
        NorthwindEntities dm =
            new ArtigoAstoria.
            ConsoleService.SV_
            Northwind.NorthwindEntities(
                new Uri("http://localhost:
                51699/NWService.svc"));

        static void Main(string[] args)
        {
            // Chamada ao Método
            SelectCategories
            SelectCategories();
        }

        // Método que Lista Todas as
        Categorias
        private static void SelectCategories()
        {
```

```
// Querie que lista Todas as
Categorias
var result = from c in
    dm.Categories
    select c;

// Imprimindo o comando T-SQL
que foi gerado
// e será disparado no banco
Console.WriteLine(result.
    ToString());

// Imprimindo o resultado
da Query
Console.WriteLine();
Console.WriteLine("LISTA DE
CATEGORIAS::");
foreach (var item in result)
{
    Console.WriteLine("{0} - {1}",
        item.CategoryID,
        item.CategoryName);
}

System.Console.ReadKey();
}
}
```

Operações CRUD

Até agora vimos apenas como realizar *queries* (SELECT) através do *ADO.NET Data Services*, mas não é apenas disso que ele é capaz. Podemos realizar qualquer uma das operações *CRUD* através do *Data Services*, utilizando o próprio padrão *REST*.

É claro que o *Client* do *ADO.NET Data Services* encapsula essa lógica, abstraíndo isso no modelo de entidades a que estamos mais acostumados. Vamos a estes exemplos então.

Inclua o método da **Listagem 7** na classe *Program.cs* do projeto *ArtigoAstoria.ConsoleServico*. Este método, como o próprio nome sugere, vai incluir uma nova entidade *Categories*, o que deve resultar em um novo registro na tabela *Categories* do *Northwind*.

Listagem 7. Método para Inclusão de Registros

```
// Método que Inclui nova Categoria
private static void InsertCategories()
{
    try
    {
        // Criando instância do novo objeto
        SV_Northwind.Categories cat = new
            ArtigoAstoria.ConsoleServico.SV_Northwind.Categories();

        // Setando o nome da categoria
        cat.CategoryName = "NOVA CATEGORIA";

        // Adicionando a categoria no modelo
        dm.AddToCategories(cat);

        // Salvando mudanças no Modelo
        dm.SaveChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Veja que a lógica é a mesma que temos no *Entity Framework*. Instanciamos um objeto da entidade, *setamos* as propriedades da nova entidade, adicionamos a nova entidade no modelo e salvamos as alterações.

Para testar faça uma chamada no método *main*, antes do *Select* para que possamos ver o resultado. Veja na **Listagem 8**. Salve, compile, execute e confira o resultado na **Figura 15**.

Listagem 8. Chamada do Insert no método Main

```
static void Main(string[] args)
{
    // Chamada ao Método InsertCategories
    InsertCategories();

    // Chamada ao Método SelectCategories
    SelectCategories();
}
```



Figura 15. Incluindo novo Registro

Como o *CategoryID* da tabela *Categories* é um *IDENTITY* no *database*, ele gerou automaticamente o *ID* e incluiu a nossa "NOVA CATEGORIA". Veja que isso foi feito através do serviço que foi criado no *ADO.NET Data Services*, e só foi possível pois configuramos as permissões adequadas.

Para realizar as operações de *Update* e *Delete* também é muito simples, veja na **Listagem 9** como ficam os métodos *UpdateCategories* e *DeleteCategories*.

Vejam que em ambos os métodos estamos recuperando a categoria com *ID* igual a 10, através de uma *query LINQ*. No método *UpdateCategories* estamos atualizando o nome da categoria e chamando o método *UpdateObject*, para que a alteração seja persistida no modelo.

No método *DeleteCategories* estamos simplesmente chamando o método *DeleteObject* que irá excluir a categoria em questão do modelo. Para que as alterações sejam feitas no banco de dados, em ambos os casos chamamos o método *SaveChanges*.

Altere as chamadas à estes métodos lá no método *Main* da classe *Program.cs*. Veja que você poderá Alterar e Excluir registros através do *ADO.NET Data Services*, bem como Incluir e consultar.

Listagem 9. Métodos UpdateCategories e DeleteCategories

```
// Método que Altera uma Categoria
private static void UpdateCategories()
{
    try
    {
        // Recuperando a Categoria com ID = 10
        SV_Northwind.Categories cat = (from c in dm.Categories
            where c.CategoryID == 10
            select c).First();

        // Alterando o Nome da Categoria
        cat.CategoryName = "CAT. ALTERADA";

        // Atualizando Objeto no Modelo
        dm.UpdateObject(cat);

        // Salvando mudanças no Modelo
        dm.SaveChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}

// Método que Deleta uma Categoria
private static void DeleteCategories()
{
    try
    {
        // Recuperando a Categoria com ID = 10
        SV_Northwind.Categories cat = (from c in dm.Categories
            where c.CategoryID == 10
            select c).First();

        // Deletando Objeto no Modelo
        dm.DeleteObject(cat);

        // Salvando mudanças no Modelo
        dm.SaveChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Conclusão

Como vimos, com o *ADO.NET Data Services* e o *ADO.NET Entity Framework* ficou muito simples criar um modelo de entidades e expô-lo como um serviço na Web. Se tivéssemos que criar um serviço “na mão” para permitir a realização de operações *CRUD*, acabaríamos tendo que criar um método para cada operação, e estaríamos restritos a opções limitadas de consultas.

Já com o *ADO.NET Data Services*, que utiliza o padrão *REST*, temos um número muito maior de possibilidades na hora de consultarmos os dados através do serviço. Essa é sem dúvida a grande vantagem dessa arquitetura.

Nós vimos aqui apenas os exemplos mais simples de operações que podemos realizar no modelo, porém já vimos que o modelo é o que chamamos de “consultável” ou “queriable”. Isso quer dizer que podemos utilizar a linguagem *LINQ* em modelos expostos pelo *ADO.NET Data Services*. O *Provider* que possibilita isso é o *LINQ to ADO.NET Data Services*, também chamado por alguns de *LINQ to REST*.

E a partir do momento em que podemos usar *LINQ* com o *ADO.NET Data Services*, sabemos que temos todos os recursos dessa linguagem para criar aplicações que acessam dados através deste modelo, sejam aplicações *Web ASP.NET*, *Windows Forms*, *WPF*, *Silverlight*, etc.

Espero que tenham gostado do artigo e principalmente do *ADO.NET Data Services*. Aguardem que em breve teremos muito mais novidades a respeito dessa tecnologia. Grande abraço e até a próxima! ●

Dê seu feedback sobre esta edição!

A .NET Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/netmagazine/feedback



O seu mundo gira mais rápido com a tecnologia de banco de dados c-tree®.

Soluções embarcadas, como controladores de tráfego inteligentes, utilizam tecnologia c-tree.



12:00

Soluções baseadas em c-tree garantem milhares de tps em sistemas de cobrança eletrônica.



Transações financeiras e sistemas de detecção de fraudes, que necessitam de alta performance e confiabilidade, são baseadas em tecnologia c-tree.



10:30



Câmeras digitais mantêm seus álbuns de fotos organizados graças à distribuição transparente da tecnologia c-tree.



Seu mundo gira mais rápido (e você dorme mais tranquilo!) com a tecnologia de bancos de dados c-tree!

A FairCom desenvolve tecnologia de gerenciamento de dados de alta performance e baixa manutenção. Nossos clientes - que vão de micro empresas a corporações multinacionais - estão aptos a solucionar dilemas de performance específicos de cada aplicativo, porque o c-tree oferece total e preciso controle sobre as operações relacionadas ao armazenamento de dados. Turbine seu aplicativo e simplifique sua distribuição! Baixe hoje mesmo sua versão de avaliação.



FairCom® Baixe hoje mesmo o **c-tree Plus da FairCom**

www.faircom.com/go/?usesDLDBr

High Performance Database Technology • 11-3872-9802

Outras empresas e nomes de produtos são marcas ou marcas registradas de seus respectivos proprietários. © 2007 FairCom Corporation