

8: Entity Framework - N-Tier Applications

Entity Framework: N-Tier Applications

Building distributed applications with the
Entity Framework



8: Entity Framework - N-Tier Applications

Objectives

- **Service-oriented architectures**
 - Data Access Layer (DAL)
 - Data Transfer Objects (DTOs)
 - Self-Tracking Entities (STEs)
- **Disconnected object persistence**
 - retrieval and loading options
 - inserts, updates, deleted
 - client change-tracking



developmentor

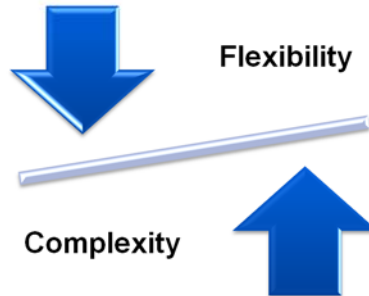


2

8: Entity Framework - N-Tier Applications

Motivation [n-tier architectures]

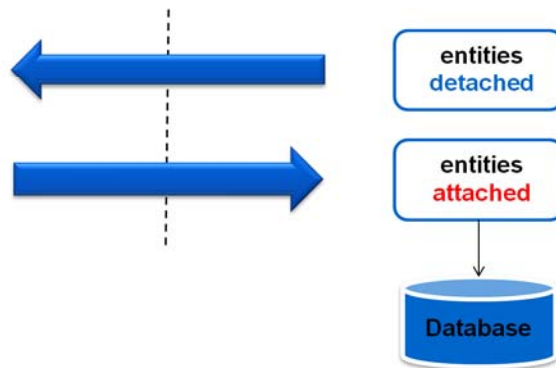
- N-Tier architectures involve trade-offs
 - flexibility, scalability, maintainability
 - requires a lot more work!



8: Entity Framework - N-Tier Applications

Solution [n-tier support in L2E]

- Entity Framework can persist disconnected entities
 - entities **detached** on serialization (or manually)
 - can later be **attached** to object context and persisted



8: Entity Framework - N-Tier Applications

WCF Overview

- **Unifies legacy API's (ASMX, Remoting, COM+, MSMQ)**
 - protocols highly configurable
- **Supports service-orientation**
 - loosely coupled, autonomous systems
 - interoperability standards (security, tx, reliability, etc)
- **Implements bindings for REST**
 - Plain Old XML, JSON formats (soap-less)
 - used by ADO.NET Data Services

developer



8: Entity Framework - N-Tier Applications

WCF Primer

A. Steps to **create** a WCF service

1. Write an interface marked with WCF attributes
2. Configure the service with one or more endpoints
3. Enable metadata
4. Host the service (exe, IIS, NT Service)

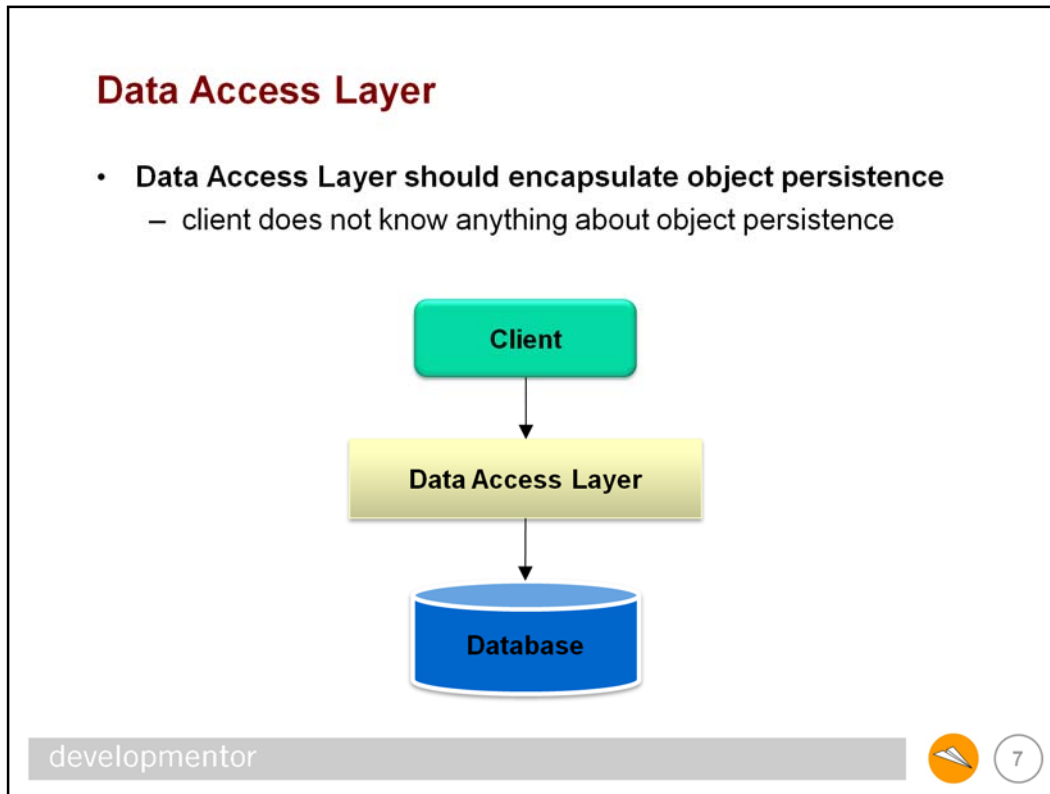
B. Steps to **consume** a WCF service

1. Set a service reference to metadata endpoint
2. Instantiate proxy class (in a using block)
3. Invoke methods on the proxy

developmentor

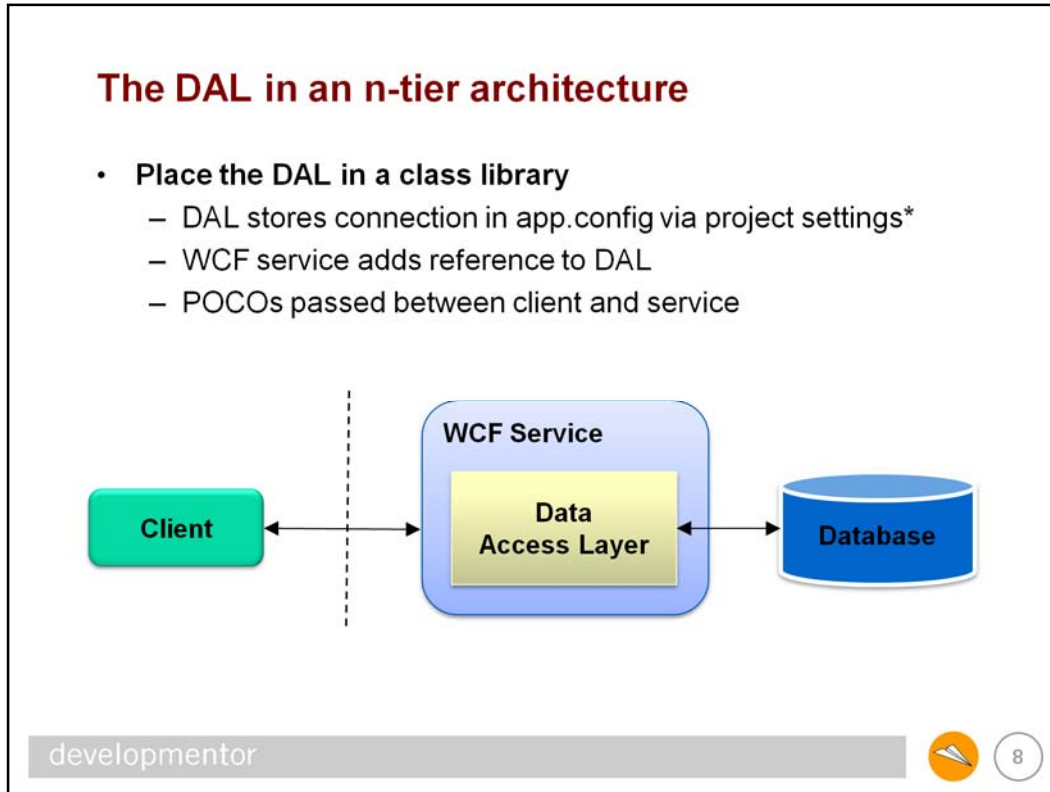


8: Entity Framework - N-Tier Applications



In an n-tier application, the client to the DAL could be a WCF service.

8: Entity Framework - N-Tier Applications



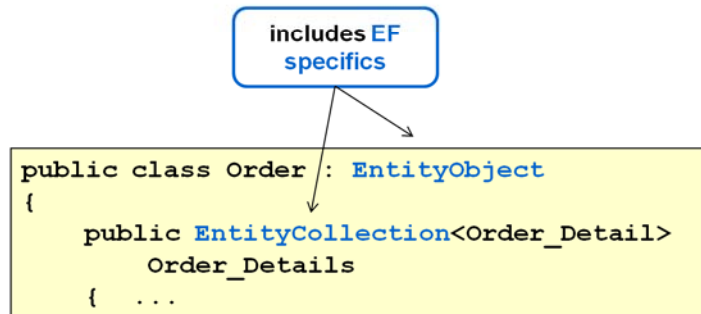
* As of .NET 2.0 you can use project settings to store items in an config file associated with a class library. However,ObjectContext's default constructor attempts to retrieve the connection string from the config file of the *executing* assembly (EXE). If you want to decouple the service layer from DAL, you probably should use DAL's project settings to store the connection string in the DAL's config file, then supply it to the ObjectContext constructor, like so:

```
using (NorthwindEntities ctx = new NorthwindEntities  
    (Settings.Default.NorthwindEntityConnection))  
{ ...
```

8: Entity Framework - N-Tier Applications

EF-Specific Entities

- Do not expose EF-Specific Entities
 - contain irrelevant goo (data binding, validation, etc)
 - EF entities contain **framework-specific** types
 - instead use POCOs (Plain Old CLR Objects)



developer



8: Entity Framework - N-Tier Applications

Data Transfer Objects

- DTOs provide separation from the data model
 - can have a **different shape** than **EF entities**
 - they are POCOs (only properties, no methods)
 - need to manually translate between DTOs and EF entities

```
public class Order
{
    public int OrderID { get; set; }
    public string CustomerName { get; set; }
    public DateTime? OrderDate { get; set; }
    public decimal OrderTotal { get; set; }
}
```

CustomerName
comes from
Customer table

developmentor



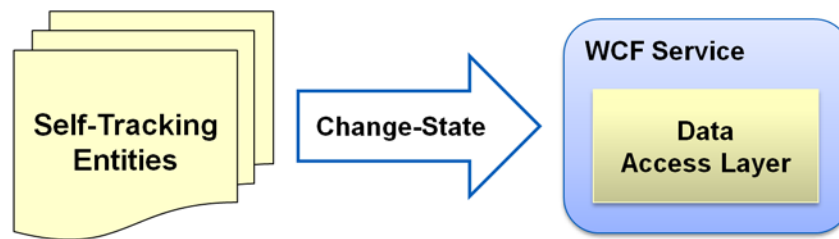
10

For an example of using Data Transfer Objects with Entity Framework 1.0, see Tony Sneed's article on SOA Data Access: <http://msdn.microsoft.com/en-us/magazine/dd263098.aspx>.

8: Entity Framework - N-Tier Applications

Self-Tracking Entities

- STEs know how to **track** their own **change-state**
 - change-state serializable – just like Datasets
 - persistence and platform ignorant – just like POCOs
 - do not require manual translation – unlike DTOs



developmentor



11

Danny Simmons, an architect on the EF team, wrote an article highlighting the differences between Data Transfer Objects and Self-Tracking Entities: <http://msdn.microsoft.com/en-us/magazine/ee321569.aspx>.

8: Entity Framework - N-Tier Applications

Self-tracking entities [change-tracking]

- Use “Self-Tracking Entities” T4 template
 - entities implement `IObjectWithChangeTracker`
 - includes `ChangeTracker` property and methods to set `ObjectState`
 - change-tracking takes place on the client
 - change-state persisted by the service



developmentor



12

Entities implement `INotifyPropertyChanged` both to facilitate change-tracking and to support data binding. Collection properties use `TrackableCollection<T>`, which derives from `ObservableCollection<T>`, for the same purpose. Entities also include code to handle relationship fix-up, which keeps both sides of an association in synch when one of the sides changes.

Entities are usually placed in a separate assembly than the DAL assembly and are referenced both by the DAL and the client. There are two T4 templates added for STE's, one for the `ObjectContext` and another for the entities themselves. Both can reside in the DAL assembly, but the entities T4 template can be added as a link to the entities assembly. For more info see <http://blogs.msdn.com/adonet/pages/feature-ctp-walkthrough-self-tracking-entities-for-the-entity-framework.aspx>.

8: Entity Framework - N-Tier Applications

Self-tracking entities [serialization]

- Place self-tracking entities in a separate class library
 - no dependency on the Entity Framework
 - both client and service reference the class library
 - entities marked with `[DataContract]` for serialization*
 - **change-state** also serialized and passed to the service

```
[DataContract(IsReference = true)]
public class Order
{
    [DataMember]
    public int OrderID { get; set; }
    [DataMember]
    public Customer Customer { get; set; }
    [DataMember] public ObjectChangeTracker
        ChangeTracker { get; set; }
}
```

developer



13

* The `IsReference` property on the `DataContract` attribute allows WCF to serialize bidirectional graphs with cycles.

8: Entity Framework - N-Tier Applications

Service DAL methods [retrieve]

- Load related entities explicitly
 - use `Include` operator to eagerly load **related** items

```
public Order GetOrderById(int orderId)
{
    using (var ctx = new NorthwindEntities())
    {
        Order order =
            (from o in ctx.OrderSet
             .Include("Customer")
             .Include("OrderDetails.Product")
             where o.OrderID == orderId
             select o).First();
        return order;
    }
}
```



8: Entity Framework - N-Tier Applications

Service DAL methods [save]

- Invoke **ApplyChanges** on the root entity set
 - extension method in Microsoft.Data.Entity*
 - informs ObjectStateManager of inserts, updates, deletes
- Call **AcceptChanges** to restore state to Unchanged
 - useful when returning saved entity

```
public Order SaveOrder(Order order)
{
    using (var ctx = new NorthwindEntities())
    {
        ctx.Orders.ApplyChanges(order);
        ctx.SaveChanges();
        order.AcceptChanges(); ←
        return order;
    }
}
```

Return entity
with identity and
concurrency values

developer



15

* For the CTP, **ApplyChanges** can be found in the **ObjectContextExtensions** class in the **Microsoft.Data.Entity.CTP** assembly, which your service DAL should reference. You need to add a **using** directive to the **Microsoft.Data.Entity** namespace in order to bring the extension method into scope.

AcceptChanges is a code-generated method in the **ObjectChangeTracker** class, located in the **xxxTypes.cs** file. It sets **ObjectState** to **Unchanged** and removes change-tracking state such as **OriginalValues** and objects added to or removed from collection properties.

8: Entity Framework - N-Tier Applications

Service DAL methods [delete]

- Call **MarkAsDeleted** on both parent and child entities
 - cache details in a `List<T>` before marking as deleted
 - then mark parent as deleted

```
public static void DeleteOrder(Order order)
{
    using (var ctx = new NorthwindEntities())
    {
        foreach (OrderDetail od in
            order.OrderDetails.ToList())
            od.MarkAsDeleted();
        order.MarkAsDeleted();

        ctx.Orders.ApplyChanges(order);
        ctx.SaveChanges();
    }
}
```

Calling `ToList`
required to prevent
exceptions



8: Entity Framework - N-Tier Applications

Client change-tracking

- Changes are tracked on the client
 - reference STEs assembly before adding service reference
 - items marked as **Modified** or **Added** when inserted, changed
 - you must call `MarkAsDeleted` explicitly

```
static void ChangeOrder(Order order)
{
    order.OrderDetails[0].Quantity++;
    OrderDetail detail = new OrderDetail
        { Product = addedProduct, Quantity = 3 };
    order.OrderDetails.Add(detail);
    order.OrderDetails[1].MarkAsDeleted();
}
```



8: Entity Framework - N-Tier Applications

Summary

- **Encapsulate object persistence in a Data Access Layer**
 - should only expose POCOs – not EF-specific entities!
 - DTOs provide greater flexibility
 - STEs are easier to use
- **Use self-tracking entities T4 template**
 - includes change-tracking code
 - change-state is serializable
- **API's provided to persist change-state**
 - ApplyChanges informs ObjectStateManager of changes
 - AcceptChanges restores entity state to Unchanged

