Contents

DevForce does not provide user interface (UI) components but it does provide fundamental features that facilitate the **display** of entities on a variety of .NET client platforms and works well with 3rd party UI controls as well as native Microsoft controls. We mean "display" in the broad sense of presenting the entity to a user and updating the entity with user input. The core UI supporting features are the subject of this section.

Client platforms

DevForce applications run on all of the principal .NET client platforms: ASP.NET, Windows Forms, Windows Presentation Foundation (WPF), Silverlight, Windows Store, and Windows Phone 7.

Version one of Silverlight Windows Phone 7 omits critical features necessary for querying entities, diminishing the usefulness of DevForce entities on this platform. You can still build WP7 apps that get and save entity data via OData.

DevForce entities are equipped for direct presentation in the user interface (UI) primarily through support for standard .NET data annotations and data binding interfaces.

Data annotations

Data annotation define metadata about objects and their properties that the UI can read to determine how to display those objects. These annotations are described in MSDN documentation for the System.ComponentModel.DataAnnotations namespace.

Data annotations are .NET **attributes** applied to classes and property members. DevForce can generate some of them for your entities based on the Entity Data Model (EDM). The developer can add more by adding them to the custom entity class and an optional companion Entity Metadata Class as described in the Model section.

The **Display** attribute, for example, suggests text for the label next to a TextBox or the header for a grid column. UI controls that detect the **Editable** attribute can configure controls to permit or refuse user input. Numerous data annotation attributes such as Required, Range, and RegularExpression, define validation rules that the UI could enforce.

Data binding interfaces

Data binding links UI controls to properties of a data object. When a UI control is bound to a data object property, the control pulls values from the data object and pushes user entered values back into the data object automatically.

The developer sets up the binding between the UI control and data object property - we say that she "binds" the control to the data object property - and the data binding infrastructure does the work. WPF and Silverlight share common techniques for binding controls to data and it is these XAML platforms that we'll address in this section although most of what we say about the data binding interfaces applies to Windows Forms clients as well.

Data binding works if the two sides - control and data object - are implemented to support it. The UI technology you've chosen determines the control side of that equation. Whether the data object is bindable - and what data binding feature it supports - depends upon characteristics of the data object, most especially what .NET data binding interfaces it supports.

When your entities derive from the DevForce entity class, they support the following data binding interfaces:

INotifyPropertyChanged - Tells the listener when the value of a property has changed. UI controls listen for PropertyChanged and update with the latest value of the bound data property. Every DevForce generated property is wired to raise PropertyChanged when updated with a new value.

IEditableObject - A UI control can ask the entity to begin a mini-editing session during which the entity accepts all changes from that control *provisionally*. In effect, the entity maintains a snapshot of the property values as they were before the mini-session began. If the user cancels the changes made during the mini-session, the entity rolls the state of the entity back to the snapshot. When the user does something to accept the changes (that "something" is up to the control), the control tells the entity to "commit" those changes; the entity then moves the changes from "provisional" to "current" value status. Note that the entity remains in a modified state, as the changes haven't been saved to the database. The entity's original values are still available if the entity was pre-existing (i.e., it had been retrieved from the database before being modified).

INotifyDataErrorInfo- When DevForce entities are validated (see Validate), DevForce uses its implementation of this interface to make the validation results available to the UI and also notifies the UI that validation results have changed. This is a Silverlight-only interfaces as of .NET 4.

IDataErrorInfo- This is an alternative interface for making validation results available to the UI. It's different from INotifyDataErrorInfo, most conspicuously in lacking the notification event. Windows Forms and WPF controls recognize this older interface; Silverlight does not. DevForce supports all three client platforms by implementing both IDataErrorInfo and INotifyDataErrorInfo.

INotifyCollectionChanged- Entity collection navigation properties (e.g, Customer.Orders) return a list of entities. More specifically, they return a RelatedEntityList<T>. RelatedEntityList implements this interface which tells the UI when items are added or removed or the entire list is refreshed.

Design data

You can bind to DevForce entities at design time, with some caveats.

Simple techniques

1. Construct one or more detached entities:
```
return new ObservableCollection<Employee>
               {
                new Employee {EmployeeID = 1, FirstName = "Fred", LastName = "Jones"},
                new Employee {EmployeeID = 2, FirstName = "Sam", LastName = "Spade"},
                new Employee {EmployeeID = 3, FirstName = "Vera", LastName = "Nice"},
               };
```
2. Attach entities to a disconnected *EntityManager* and perform cache-only queries:
```
_mgr = new NorthwindIBEntities(false);
var emp1 = new Employee { EmployeeID = 1, FirstName = "Fred", LastName = "Jones" };
var emp2 = new Employee { EmployeeID = 2, FirstName = "Sam", LastName = "Spade" };
var emp3 = new Employee { EmployeeID = 3, FirstName = "Vera", LastName = "Nice" };
_mgr.AttachEntities(new[] { emp1, emp2, emp3 });
...
var results = _mgr.Employees.With(QueryStrategy.CacheOnly).ToList();
```
3. Add entities to a disconnected *EntityManager* and perform cache-only queries:
```
_mgr = new NorthwindIBEntities(false);
var emp1 = new Employee { FirstName = "Fred", LastName = "Jones" };
var emp2 = new Employee { FirstName = "Sam", LastName = "Spade" };
var emp3 = new Employee { FirstName = "Vera", LastName = "Nice" };
_mgr.AddEntities(new[] { emp1, emp2, emp3 });
...
var results = _mgr.Employees.With(QueryStrategy.CacheOnly).ToList();
```

- If you use an *EntityManager* it must be disconnected.
- If you attach entities to an *EntityManager*, be sure to set the values of key properties to avoid duplicates.
- You can add entities to an *EntityManager* and DevForce will assign a temporary id value.
  Several IdeaBlade presentations, videos, and code samples demonstrate techniques for designing screens and views using DevForce entities and offline *EntityManagers*. A good place to start is Cocktail.

Unknown macro: IBNote`The "IBNote" macro is not in the list of registered macros. Verify the spelling or contact your administrator.`

Limitations

Silverlight

- Code First entities have limited support at this time: The .ibmmx file of metadata cannot be loaded, so you may not attach or add entities to an *EntityManager*; however, you can construct detached entities and bind to them.
- *EntityCacheState*: At this time an *EntityCacheState* containing design-time data may not be loaded.

WPF

- You can restore an *EntityCacheState* containing design time data into an *EntityManager* but you cannot issue a query for the loaded data at this time. As a workaround, use the *FindEntities* method.

Adding DevForce entities to data sources window

You may want to use the Data Sources Window built into Visual Studio to simplify binding to your entities at design time. To do so, you must configure Visual Studio to connect to data in the object, with the followign steps:

- Build your solution (or you won't see the model project)
- Open the xaml class file (can be in either XAML or Designer view)
- Open the Data Sources window
- Click "Add New Data Source"; Data Source Configuration Wizard dialog opens
- Pick the "Object" data source type
- Open the node to the project with your model (if not shown, you may have forgotten to build)
- Check the checkbox(es) next to the entities you'll be binding (e.g, Customer, Employee)
- Click "Finish" button

This provides certain advantages by allowing you to generate more of your design with tools, but has several drawbacks. These include:

- Using the Data Sources Window can impact Visual Studio performance. It will frequently re-generate the information used to populate this window, which will slow down or pause your system.
- You cannot use this method with MVVM or similar design patterns.
- More of your application's code will be written into the XAML, which is more difficult to troubleshoot or debug.