Contents

This sample shows DevForce validation at work in a WPF application.

- **Platform:** WPF
- **Language:** C#, VB
- **Download:** Validation (WPF)

Problem

You'd like to better understand DevForce validation and see it in action.

Solution

The sample shows a simple application displaying employees and their orders. As you modify various fields you can see property-level validation in action. For example, the *First Name* and *Last Name* fields can't contain more than 30 characters.

These are examples of simple declarative validation through standard attributes. By default when you generated the code for your model DevForce added validation attributes for string length restrictions and required fields:

```
[IbVal.RequiredValueVerifier( ErrorMessageResourceName="Employee_EmployeeID")]
public int EmployeeID {...}
[IbVal.StringLengthVerifier(MaxValue=30, IsRequired=true, ErrorMessageResourceName="Employee_LastName")]
public string LastName {..}
```

```
<IbVal.RequiredValueVerifier(ErrorMessageResourceName:="Employee_EmployeeID")>
Public ReadOnly Property EmployeeID() As Integer
End Property
<IbVal.StringLengthVerifier(MaxValue:=30, IsRequired:=True, ErrorMessageResourceName:="Employee_LastName")>
Public ReadOnly Property LastName() As String
End Property
```

These validation attributes help ensure that your simple database-defined rules are enforced.

Although not shown in the sample, you can also add custom attributes, and define rules to enforce your business logic using both predefined and fully custom verifiers.

Exceptions vs. notifications

The DevForce verification engine by default is set to an *ErrorNotificationMode* of *Notify*. This means that when a property validation fails DevForce will use the INotifyDataErrorInfo and IDataErrorInfo interfaces to *alert* the binding of the problem.

Unfortunately, WPF bindings don't recognize *INotifyDataErrorInfo* at all, and don't by default listen for *IDataErrorInfo*.

This leaves us with two choices: we can modify the DevForce default to throw validation exceptions, or we can modify the binding to listen on the *IDataErrorInfo* interface.

Let's take exceptions first. We must do both of the following:

1. Change the ErrorNotificationMode on the VerifierEngine to *ThrowException*.
2. Set the bindings to use ValidatesOnExceptions = "true".

If you dragged properties from a data source using the designer it created bindings that look something like this:

```
Text="{Binding Path=FirstName, Mode=TwoWay, ValidatesOnExceptions=true, NotifyOnValidationError=true}"
```

So by default WPF expects to use validation exceptions.

To tell DevForce to throw validation exceptions:

```
_mgr = new NorthwindIBEntities();
_mgr.VerifierEngine.DefaultVerifierOptions.ErrorNotificationMode =
  VerifierErrorNotificationMode.ThrowException;
```

```
_mgr = New NorthwindIBEntities()
_mgr.VerifierEngine.DefaultVerifierOptions.ErrorNotificationMode = _
```

```
VerifierErrorNotificationMode.ThrowException
```

Alternately, we can instead have bindings listen for errors by doing the following:

1. Leave the VerifierEngine ErrorNotificationMode at its default of *Notify*.
2. Set the bindings to use ValidatesOnDataErrors = "true".

The modified binding:

```
Text="{Binding Path=FirstName, Mode=TwoWay, ValidatesOnDataErrors=true, NotifyOnValidationError=true}"
```

The sample shows the second approach.

The UI

The default WPF validation error template will display a red box around the field in error. In the sample we've added an ErrorTemplate to instead show a red exclamation mark and a tooltip with the error message.

```
<Window.Resources>
  <Style TargetType="{x:Type TextBox}">
    <Setter Property="Validation.ErrorTemplate">
      <Setter.Value>
        <ControlTemplate>
          <DockPanel>
            <TextBlock Foreground="Red" FontSize="20">!</TextBlock>
            <AdornedElementPlaceholder/>
          </DockPanel>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
    <Style.Triggers>
      <Trigger Property="Validation.HasError" Value="true">
        <Setter Property="ToolTip" Value="{Binding RelativeSource={RelativeSource Self},
          Path=(Validation.Errors)[0].ErrorContent}"/>
      </Trigger>
    </Style.Triggers>
  </Style>
</Window.Resources>
```