Contents

DevForce provides a mechanism to intercept and either modify or extend the behavior of any property on a DevForce entity. This attribute interception is intended to replace, and expand upon, the technique of marking properties as virtual and overriding them in a subclass. This facility is a lightweight form of what is termed Aspect-Oriented Programming.

Interception can be accomplished either statically, via attributes on developer-defined interception methods, or dynamically, via runtime calls to the *PropertyInterceptorManager*. Attribute interception is substantially easier to write and should be the default choice in most cases.

Attribute interception

DevForce supplies four attributes that are used to specify where and when property interception should occur. These attributes are:

- *BeforeGetAttribute*
- *BeforeSetAttribute*
- *AfterGetAttribute*
- *AfterSetAttribute*

Under most conditions these attributes will be placed on methods defined in the custom partial class associated with a particular DevForce entity. For example, the code immediately below represents a snippet from the auto-generated Employee class.

(Generated code)

```
public partial class Employee : IdeaBlade.EntityModel.Entity {
 public string LastName {
  get {
   return PropertyMetadata.LastName.GetValue(this);
  }
  set {
   PropertyMetadata.LastName.SetValue(this, value);
  }
 }
```

```
Partial Public Class Employee
 Inherits IdeaBlade.EntityModel.Entity
 Public Property LastName() As String
  Get
   Return PropertyMetadata.LastName.GetValue(Me)
  End Get
  Set(ByVal value As String)
   PropertyMetadata.LastName.SetValue(Me, value)
  End Set
 End Property
```

Property interception of the get portion of this property would be accomplished by adding the following code fragment to a custom Employee partial class definition:

```
[AfterGet(EntityPropertyNames.LastName)]
public void UppercaseNameAfterGet(PropertyInterceptorArgs<Employee, String> args) {
  var lastName = args.Value;
 if (!String.IsNullOrEmpty(lastName)) {
   args.Value = args.Value.ToUpper();
  }
}
```

```
<AfterGet(EntityPropertyNames.LastName)>
Public Sub UppercaseNameAfterGet(ByVal args As PropertyInterceptorArgs(Of Employee, String))
 Dim lastName = args.Value
 If Not String.IsNullOrEmpty(lastName) Then
   args.Value = args.Value.ToUpper()
 End If
End Sub
```

DevForce will insure that this method is automatically called as part of any call to the *Employee.LastName* 'get' property. The "AfterGet" attribute specifies that this method will be called internally as part of the 'get' process "after" any internal get

operations involved in the get are performed. The effect is that the LastName property will always return an uppercased result. For the remainder of this document, methods such as this will be termed interceptor actions.

The corresponding 'set' property can be intercepted in a similar manner.

```
[BeforeSet(EntityPropertyNames.LastName)]
public void UppercaseNameBeforeSet(IbCore.PropertyInterceptorArgs<Employee, String> args) {
  var name = args.Value;
 if (!String.IsNullOrEmpty(name)) {
    args.Value = args.Value.ToUpper();
  }
}
```

```
<BeforeSet(EntityPropertyNames.LastName)> _
Public Sub UppercaseLastName(ByVal args As PropertyInterceptorArgs(Of Employee, String))
 Dim lastName = args.Value
 If Not String.IsNullOrEmpty(lastName) Then
    args.Value = args.Value.ToUpper()
 End If
End Sub
```

In this case we are ensuring that any strings passed into the 'LastName' property will be uppercased before being stored in the Employee instance (and later persisted to the backend datastore). Note that, in this case, the interception occurs "before" any internal operation is performed.

In these two cases we have implemented an 'AfterGet' and a 'BeforeSet' interceptor. *BeforeGet* and *AfterSet* attributes are also provided and operate in a similar manner.