Contents

In most cases, you will want to **perform validation** either when a property on one of your entities is *set*, or as part of a comprehensive validation of an object instance.

Property level validation

Property level validations should be run whenever the value of some property on an entity is set and that property has a verifier or verifiers associated with it. For a DevForce entity, this occurs automatically because internally DevForce calls either *VerifierEngine.ExecuteBeforeSet* or *VerifierEngine.ExecuteAfterSet* whenever a property is set. Which one is called is determined by the *VerifierExecutionModes* settings on each verifier. This occurs during the execution of the autogenerated call to the *EntityProperty.SetValue* method; an autogenerated code fragment is shown below:

```
...
public string CompanyName {
 get { ... }
 set { PropertyMetadata.CompanyName.SetValue(this, value); }
}
```

```
Public Property CompanyName() As String
 Get
  ...
 End Get
 Set(ByVal value As String)
   PropertyMetadata.CompanyName.SetValue(Me, value)
 End Set
End Property
```

This means that you will NOT have to write ANY code to perform property level validations on any DevForce generated entity type.

If you are working with a custom property or an object type that is not a DevForce entity, you will need to write some of this scaffolding behavior yourself. For example:

```
public String ShortName {
 get { return _shortName; }
  set {
    VerifierResultCollection verifierResults = _verifierEngine.ExecuteBeforeSet(this, "ShortName", value);
   if (verifierResults.Ok) {
     _shortName = value;
    } else {
    // Handle verifierResults here
    }
   }
  }
}
```

```
Public Property ShortName() As String
 Get
  Return _shortName
 End Get
 Set(ByVal value As String)
  Dim verifierResults As VerifierResultCollection = __
    verifierEngine.ExecuteBeforeSet(Me, "ShortName", value)
  If verifierResults.Ok Then
    _shortName = value
  Else
   ' Handle verifierResults here
  End If
 End Set
End Property
```

Note that the *ExecuteBeforeSet* method above takes an object instance, a property name and a proposed value; the result is a *VerifierResultCollection* that contains a collection of individual *VerifierResult*s as well as an aggregate *Ok* property. We have also assumed in the example above the availability of a *VerifierEngine* instance; *_verifierEngine*.

DevForce actually does something a good deal more complex under the covers, because it calls both the *ExecuteBeforeSet* for 'beforeSet' verifiers and *ExecuteAfterSet* for 'afterSet' verifiers and interprets and handles verifier results in accordance with the *VerifierOptions* property of each verifier. Relevant *VerifierOptions* properties include *ErrorNotificationMode*, *ErrorContinuationMode*, *TreatWarningsAsErrors* and *ShouldExitOnBeforeSetError*.

Instance level validation

Instance validation is much like property level validation except that you call the *VerifierEngine's* ***Execute*** method instead of the *ExecuteBeforeSet* or *ExecuteAfterSet* methods. As with those methods, *Execute* also returns a *VerifierResultCollection*, whose individual members are each a *VerifierResult*.

```
VerifierResultCollection aVerifierResultCollection = _verifierEngine.Execute(anEmployee);
```

```
Dim aVerifierResultCollection As VerifierResultCollection = _verifierEngine.Execute(anEmployee){{/code}}
```

Instance validation is called automatically by DevForce on the EntityServer just prior to the execution of a save, but most developers will want to call it programmatically in client side code as well.

It is also possible to execute on demand a specified collection of verifiers, or a single verifier:

```
Verifier birthDateVerifer = aVerifierEngine.GetVerifier(
 typeof(Employee), typeof(DateTimeRangeVerifier),
  Employee.EntityPropertyNames.BirthDate);
aVerifierResultCollection = aVerifierEngine.Execute(anEmployee,
 new List<Verifier>() { birthDateVerifer }, null);
```

```
Dim birthDateVerifer As Verifier = aVerifierEngine.GetVerifier(GetType(Employee), GetType _
  (DateTimeRangeVerifier), Employee.EntityPropertyNames.BirthDate)
aVerifierResultCollection = aVerifierEngine.Execute(anEmployee, New List _
  (Of Verifier)() From {birthDateVerifer}, Nothing)
```

However, this is unlikely to be a common operation in your application.

Examine the validation results

You can examine a collection of *VerifiersResults*. Each *VerifierResult* consists of the following readonly properties:

Property	Property Type	Description
ResultCode	*VerifierResultCode*	A classification of the result of this validation, primarily a classification of the type of the success or failure of the operation.
Message	String	A description of the result.
Verifier	Verifier	The verifier that this VerifierResult resulted from. **This will be null (Nothing in VB) if this is a "remote" VerifierResult**.
TargetInstance	Object	The object instance that was validated to get this result.
VerifierOptions	VerifierOptions	The VerifierOptions on the Verifier that was executed to get this result
VerifierContext	VerifierContext	The VerifierContext under which this validation was run. **This will be null (Nothing in VB) if this is a "remote" VerifierResult**.
TriggerContext	TriggerContext	The TriggerContext under which this validation was run. The TriggerContext contains information about the timing (BeforeSet, AfterSet) of the validation among other things.
PropertyNames	ICollection<String>	List of names of properties involved in this validation.

The ***VerifierResultCode*** enumeration consist of the following values:

Value	Description
Ok	Was this a successful validation.

Error	Was this a failed validation.
OkWarning	Was this is a "warning" result. Whether this is treated as a success or failure depends on the VerifierOptions.ShouldTreatWarningsAsErrors flag.
OkNotApplicable	Whether this result occured because the validation was not applicable to the data;
ErrorInsufficientData	Whether this result occured because of insufficient data.

The following methods on the *VerifierResult* class provide a shorthand mechanism for accessing the actual enumerated values shown above:

Method	Success or Failure	Description
IsOk	Success	Was this a successful validation.
IsError	Failure	Was this a failed validation.
IsWarning	Either	Was this is a "warning" result. Whether this is treated as a success or failure depends on the VerifierOptions.ShouldTreatWarningsAsErrors flag.
IsNotApplicable	Success	Whether this result occured because the validation was not applicable to the data;
IsInsufficientData	Failure	Whether this result occured because of insufficient data.

So for example the following two expressions mean the same thing:

```
ValidationResultCode vrc = ...
var isError = (vrc == ValidationResultCode.Error);
// is same as
var isError = vrc.IsError();
```

```
Dim vrc As ValidationResultCode =...var isError = (vrc Is ValidationResultCode.Error)
' is same as
Dim isError = vrc.IsError()
```

Note that both the *Verifier* and the *VerifierContext* properties will be null for any *VerifierResult* that occurs as a part of a "remote" validation. A "remote" validation is one where the validation occurs on the *EntityServer* instead of on the client. This is because, in such validations, the *Verifier* and its *VerifierContext* only exist on the EntityServer and we may not want these verifiers to be accessible to the client. In this case, we only want the results of the validation.

Example

The following code, for example, iterates through a collection of *VerifierResults*, captures the *Message* property value of each one that represents an error. (Note that a *Verifier* always returns a *VerifierResult* when executed, regardless of whether an error was found.)

```
bool foundErrors = false;
foreach (VerifierResult aVerifierResult in aVerifierResultCollection) {
if (aVerifierResult.IsError) {
    foundErrors = true;
    _localOutput.Append(string.Format("\tValidation Failure: {0}\n",
    aVerifierResult.Message));
  }
}
```

```
Dim foundErrors As Boolean = False
For Each aVerifierResult As VerifierResult In aVerifierResultCollection
 If aVerifierResult.IsError Then
    foundErrors = True
    _localOutput.Append(String.Format(vbTab & _
     "Validation Failure: {0}" & vbLf, aVerifierResult.Message))
 End If
Next aVerifierResult
```