Contents

You can **inspect the original version** of a data property value even if the entity is in a modified or deleted state.

Normally when you read an entity data property, DevForce returns the "current" value. Internally DevForce maintains several versions of each data property's value: the current version (with a pending change if any), its original version (the value as it was when the entity was last retrieved or saved), and a proposed version (for a change that is no yet recorded). This topic covers the ***EntityVersion*** enumeration and how you use it in combination with the ***GetValue*** method to inspect specific versions of property data.

Data Property Versioning

When you undo changes you've made to a modified entity, DevForce reverts the entity's data properties to their "original" values.

DevForce maintains more than one value per data property. In fact, it may hold up to *three versions of the property value* : the **current**, the **original**, and a **proposed** version.

The *Current* version is the one you see most of the time. You won't often want to look at other versions. You'll be happy to know the original is there somewhere, standing by in case you "undo". But there may come a time, perhaps while writing validation logic that concerns transitions from one state to another, when you want to examine the other versions. This topic explains the versions in some detail and shows how you can inspect them.

DevForce only versions "data properties", the properties that you persist to the database. It doesn't version navigation properties nor the custom properties that you wrote.

Seeing is believing

This is easier to talk about with an example. Let's write a simple statement using the *CompanyName* property of an umodified *Customer* object:

```
originalName = testCustomer.CompanyName;
```

```
originalName = testCustomer.CompanyName
```

The "*originalName*" variable is assigned the ***Current*** version of the *CompanyName* property.

Technically, the *CompanyName* property returns the *Default* version which might be other than the *Current* under different circumstances. See below.

Because this entity is unmodified, the *Current* version is the same as the *Original* version which is why we called the variable, *originalName*.

Let's change the *Current* version:

```
testCustomer.CompanyName = newName =  "New Name";
```

```
testCustomer.CompanyName = newName =  "New Name"
```

The current *CompanyName* will be *newName*; the original will remain *originalName*. To prove it, we'll use the EntityAspect to go under the hood. EntityAspect has a *GetValue* method with an overload that accepts one of the *EntityVersion* enums.

```
// Get the current and original values
var curVal  = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current);
 var origVal = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original);
 Assert.AreEqual(newName, curVal);
 Assert.AreEqual(originalName, origVal);
 Assert.AreNotEqual(curVal, origVal); // driving the point home.
```

```
' Get the current and original values
Dim curVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current))
Dim origVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original))
 Assert.AreEqual(newName, curVal)
 Assert.AreEqual(originalName, origVal)
Assert.AreNotEqual(curVal, origVal) ' driving the point home.
```

Let's detach the entity and test the versions again:

```
// Hold onto its current manager
var manager = testCustomer.EntityAspect.EntityManager;
// Detach
testCustomer.EntityAspect.RemoveFromManager();
// Get the current and original values again
curVal  = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current);
 origVal = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original);
// Both values are available, even in the detached state
Assert.AreEqual(newName, curVal);
 Assert.AreEqual(originalName, origVal);
```

```
 ' Hold onto its current manager
Dim manager = testCustomer.EntityAspect.EntityManager
' Detach
testCustomer.EntityAspect.RemoveFromManager()
' Get the current and original values again
curVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current))
 origVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original))
' Both values are available, even in the detached state
Assert.AreEqual(newName, curVal)
 Assert.AreEqual(originalName, origVal)
```

DevForce entities are "self-tracking". They retain both their current and original values even when detached from an EntityManager. We can serialize them, deserialize them, hand them around, import them in another EntityManager, or re-attach them to their former manager like this:

```
// Re-attach **and** ensure its //[[EntityState>>EntityState]]// is "modified"
manager.AttachEntity(testCustomer);
 testCustomer.EntityAspect.SetModified();
// Get the current and original values again
curVal =  (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current);
 origVal = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original);
// We've restored the modified entity
Assert.IsTrue(testCustomer.EntityAspect.EntityState.IsModified());
 Assert.AreEqual(newName, curVal);
 Assert.AreEqual(originalName, origVal);
```

```
' Re-attach **and** ensure its //[[EntityState>>EntityState]]// is "modified"
manager.AttachEntity(testCustomer)
 testCustomer.EntityAspect.SetModified()
' Get the current and original values again
curVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current))
 origVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original))
' We've restored the modified entity
Assert.IsTrue(testCustomer.EntityAspect.EntityState.IsModified())
 Assert.AreEqual(newName, curVal)
 Assert.AreEqual(originalName, origVal)
```

Finally let's undo the pending changes and check the values again:

```
 testCustomer.EntityAspect.RejectChanges(); // undo
// Get the current and original values again
curVal  = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current);
 origVal = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original);
// Back to the initial unmodified entity
Assert.IsTrue(testCustomer.EntityAspect.EntityState.IsUnchanged());
 Assert.AreEqual(originalName, curVal); // current restored to original
Assert.AreEqual(originalName, origVal);
 Assert.AreEqual(curVal, origVal);      // driving the point home.
```

```
testCustomer.EntityAspect.RejectChanges() ' undo
' Get the current and original values again
curVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Current))
 origVal = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Original))
' Back to the initial unmodified entity
Assert.IsTrue(testCustomer.EntityAspect.EntityState.IsUnchanged())
 Assert.AreEqual(originalName, curVal) ' current restored to original
Assert.AreEqual(originalName, origVal)
 Assert.AreEqual(curVal, origVal) ' driving the point home.
```

The Original version

The *Original* value is the value retrieved from the database. Newly added entities don't have an *Original* value. They have a *Current* value. They get an *Original* value when you save.

You can simulate a save by calling *EntityAspect.AcceptChanges*. You should be wary of doing that although it can be useful in tests.

The Proposed version

DevForce entities implement *System.ComponentModel.**IEditableObject***. That means they have an additional level of do-undo.

IEditableObject has three methods:

Methods	Summary
BeginEdit()	Begin editing the object. An edit session is open. While open, changes go into the *Proposed* version.
CancelEdit()	Cancel the edit. The edit session closes and the *Proposed* versions are discarded.
EndEdit()	Ends the edit session and pushes the *Proposed* version values into the *Current* version.

You can inquire about the *Proposed* version any time.

```
var proposed = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Proposed);
```

```
Dim proposed = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Proposed)
```

The Default

There is a "*Default*" EntityVersion enum. *Default* is not really a version. Rather, it maps to the version whose value would be returned by calling the property directly as we see in this example:

```
var value   =  testCustomer.CompanyName;
var default = (string) testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Default);
Assert.AreEqual(value, default );  // Always true.
```

```
Dim value = testCustomer.CompanyName
Dim default1 = CStr(testCustomer.EntityAspect.GetValue("CompanyName", EntityVersion.Default))
Assert.AreEqual(value, default1) ' Always true.
```

That's a tautology. You want to know how the *Default* is mapped.

Default maps to ...	when ...
Current	the entity is added, modified, unchanged, or detached.
Original	the entity is deleted.
Proposed	the entity is in the middle of an IEditableObject editing session.

Asking for a version that isn't defined

What if you ask for the *Original* of an added entity? An added entity doesn't have an *Original* value. Rather than throw an exception or return a strange value, DevForce returns the *Default* version.

DevForce always returns the *Default* version if the version asked for is undefined.

To give another example, the *Proposed* version is undefined unless the entity is in an open IEditableObject edit session. If there is no open edit session and you ask for the *Proposed* value, you get ... the *Default* value.

The EntityVersion Enum

EntityVersion is an enum whose values you should now appreciate.

Version	Summary
Current	The value that would be saved to the database.
Original	The value as it was when the entity was last queried or saved.

Proposed	A changed value within an open IEditableObject session. It becomes the current value when the session is closed by a call to EndEdit.
Default	Always the same as the property value. It is usually *Current*; it is *Proposed* during an IEditableObject edit session and *Original* for a *Deleted* entity.

EntityVersion is a flag enumeration, meaning you can 'OR' the values together to form a composite EntityVersion value for search purposes.