Contents

Undo pending (unsaved) changes to a single entities, selected entities, or all entities in cache by calling one of the ***RejectChanges*** methods. In addition, DevForce entities also support the *IEditableObject* interface which allows for the undo of changes within an "editing" context"

The RejectChanges method

An application can undo changes made to individual entity via the *EntityAspect.RejectChanges* method. The same operation can be performed against all of the relevent entities within an EntityManager (all modified, deleted, or added entities) via the *EntityManager.RejectChanges* method.

This is a single level undo. Undoing a pre-existing object, whether added, modified or marked for deletion, restores it to its state when last retrieved from the data source;

The following table describes what happens to an entity in any given state when a RejectChanges call is made on it.

Original EntityState	EntityState After RejectChanges	Other side effects
Modified	Unchanged	Original values become current values
Added	Detached	Entity gets removed from the cache (as if it had never been Added).
Deleted	Unchanged	Entity gets "undeleted"
Unchanged	Unchanged	Nothing actually happens here.
Detached	Detached	Nothing actually happens here.

There is no undo of an undo.

DevForce Entities also implement the .NET ***IEditableObject*** interface. This interface provides for a second form of undo that is commonly utilized by UI controls when binding directly to an entity. By convention this interface is implemented explicitly and therefore requires casting.

Access to "Undo" information

DevForce effectively maintains a copy of each entities "original" version. When you access the properties of an entity, either via a get or set operation, you are accessing the "default" version of the entity. The property values of the "original", as well as a "proposed" version when inside of an *IEditableObject* edit session, however, are available as well. The values for any data property of an object are available via either of the following two overloads of the ***EntityAspect.GetValue*** method. This is discussed in more detail here.

```
public Object GetValue(String propertyName, EntityVersion version);
public Object GetValue(DataEntityProperty dataEntityProperty, EntityVersion version);
```

```
Public Function GetValue(ByVal propertyName As String, ByVal version As EntityVersion) As Object
End Function
Public Function GetValue(ByVal dataEntityProperty As DataEntityProperty, ByVal version As EntityVersion) As Object
End Function
```

This value returned from the original version is usually exactly what was queried from the database, although it is possible to update the "original" to match the "current" version via either the *EntityAspect.AcceptChanges* or *EntityManager.AcceptChanges* method calls.

EntityVersion is an enumeration that with the following values:

Value	Description
Default	The default version for the state of the entity. This is a version that is used to actually reference another version based on the state of the entity. A version of **Default** will almost always mean **Current** with the two exceptions being • when an object is within an *IEditableObject* session, the default version is **Proposed**. • when an object has an *EntityState* of **Deleted**, its default version is **Original**.

Original	The original version of the entity from when it was last queried, saved or had *AcceptChanges* called on it.
Current	The most recent version of the entity, outside of an *IEditableObject* session.
Proposed	The value of the entity when within an *IEditableObject* session.

Undo of a change to a navigation property that returns a list

The RejectChanges method basically returns an entity to the state it was in when it was last queried or saved. If other related objects are associated with this entity by being added to or removed from a "list" navigation property then the RejectChanges call will not have any effect on the collection returned by the property. This is because information about what entities are associated with another entity is not considered part of the persistent state of the entity except in the case where a foreign key reference id exists (or in case of a many-many relationship). Since most relationships do have a foreign key defined on one side or the other of the relationship it is often sufficient to call RejectChanges on the entities on both sides of a relationship in order to restore the "graph" to an earlier state.

Another approach is to use the rollback mechanism described under the "Snapshot changes" topic.