Contents

Listen for changes to specific entities or for changes to groups of entities in cache by attaching handlers to Entity and EntityManager events.

EntityManager events

The *EntityManager* has a number of events that provide access to a variety of information regarding changes to both the Entities within an EntityManager as well as the EntityManager itself.

Instance events:

Event	Description
Cleared	Fired whenever the *EntityManager.Clear* method is called.
EntityChanging	Fired just before any entity within the EntityManager is about to be changed. The change can be cancelled from within the event handler. Information about the change is available via an *EntityChangingEventArgs* parameter passed to the event handler.
EntityChanged	Fired just after any entity within the EntityManager is changed. Information about the change is available via an *EntityChangedEventArgs* parameter passed to the event handler.
EntityServerError	Fired when an error occurs during an operation that passed data to the EntityServer. The *IdeaBlade.EntityModel* allows the exception to be examined and optionally "handled" so that the exception does not propogate further. This can be used as a "last resort" error handler for any EntityServer errors.
Querying	Discussed here
Fetching	Discussed here
Queried	Discussed here
Saving	Discussed here
Saved	Discussed here

Both the *EntityChanging* and *EntityChanged* event handlers are passed an event args variable that in turn contains an instance of the *EntityAction* enumeration. This action may be used to determine the kind of change that is either about to occur or has just occured. The descriptions below are phrased in the past tense but should be translated to the future tense in the case of the *EntityChanging* event. *EntityAction* is a ***flag*** enumeration meaning that more than one action may set at the same time. Where this occurs, it is called out.

EntityAction	Description
Remove	The entity has been removed. Either via a call to *RemoveEntity* or after the save of a *Deleted* entity.
Change	The entity has changed as a result of a change to some property on the entity.
Rollback	The most recent change to the entity has been rolled back. This occurs as a result of a *RejectChanges* call.
Commit	The changes to the entity have been committed. This occurs as a result of a *AcceptChanges* call.
Add	The entity has just been attached to the EntityManager.
ChangeOriginal	The original version of the entity has been changed. This occurs as a result of a Merge operation (via Query or ImportEntities) with PreserveChangesUpdateOriginal strategy.
ChangeCurrentAndOriginal	The original and the current versions of the entity have been changed. This occurs on any Merge operation other than the one above.
AddOnQuery	The entity has been attached as a result of an query operation. (This is also an **Add** *EntityAction*)
AddOnImport	The entity has been attached as a result of an import operation. (This is also an **Add** *EntityAction*)

AddOnAttach	The entity has been attached to as a result of an *AttachEntity* call but with an *EntityState* other than **Added**. (This is also an **Add** *EntityAction*)
Delete	The entity was marked for deletion

By default, if your handler for the *EntityChanging* or *EntityChanged* event throws an exception when an entity is added by either query or import, DevForce will "eat" the exception and your application will never see the problem. DevForce handles these exceptions in order to insure that the state of the EntityManager remains consistent. To override this behavior, set the *EntityManagerOptions.ThrowAllLoadExceptions* flag to true. For example:

```
entityManager.Options.ThrowAllLoadExceptions = true;
```

Note that this flag also controls the behavior of exception handling for these events on the *EntityGroup*, and *PropertyChanged* events on the *Entity* and *EntityAspect*.

Static (class level) events:

Event	Description
EntityManagerCreated	A class level event that is fired anytime a new EntityManager is created

EntityGroup events

In DevForce, every EntityManager contains a collection of *EntityGroups*. Each *EntityGroup* in turn contains all of the entities of a single entity type that are part of this EntityManager. An EntityGroup can be retrieved via either of the following methods on an EntityManager.

```
public EntityGroup GetEntityGroup(Type entityType);
public EntityGroup<T> GetEntityGroup<T>();
```

```
Public Overloads Function GetEntityGroup( _
  ByVal entityType As Type _
) As EntityGroup
Public Overloads Function GetEntityGroup(Of T)() As EntityGroup(Of T)
```

Each EntityGroup has a number of events that may be handled:

Event	Description
EntityPropertyChanging	Fired just before a property of an entity changes. The associated *EntityPropertyChangingEventArgs* contains information regarding the entity being changed, the property being changed and the "proposed value" for the change. The change may be cancelled, in which case the property set operation will not occur.
EntityPropertyChanged	Fired just after a property of an entity changes. The associated *EntityPropertyChangedEventArgs* contains information regarding the entity that was changed, the property that was changed and the "value" for the change.
EntityChanging	This is basically the same as the *EntityManager.EntityChanging* event simply fired at a different level.
EntityChanged	This is basically the same as the *EntityManager.EntityChanged* event simply fired at a different level.

There may be cases where a developer wants to suppress all "change notification" within an EntityGroup for a period of time. This can be accomplished by setting the *ChangeNotificationEnabled* property to 'false'. Note that this will affect change notification at the EntityManager level (*EntityChanging*, *EntityChanged*) as well as at the Entity level (*INotifyPropertyChanged*).

Entity events

Event	Description
PropertyChanged	Implementation of *INotifyPropertyChanged*
ErrorsChanged	Explicit implementation of *INotifyDataErrorInfo*

EntityAspect events

Most changes that relate to an Entity are handled by the events listed previously on this page; however, the *EntityAspect* also implements *INotifyPropertyChanged* to raise a change notification for several of its properties.

Event	Description
PropertyChanged	Part of *INotifyPropertyChanged* implementation. Properties on the EntityAspect that are subject to changes and therefore available via the PropertyChanged notification are EntityState, EntityKey, *IsChanged*, and *HasErrors*.