Contents

The following is a summary of the methods available on the *EntityManager* arranged by task. For a complete list of methods, please see the EntityManager API documentation *EntityManager API documentation* .

Note: the terms "manager" and "EntityManager" are interchangeable here as they are almost everywhere in DevForce documentation.

Query

The *EntityManager* members related to querying entities, either from remote data sources or from its entity cache.

Many of the members that can fetch entities from the server come in both synchronous and asynchronous versions. The asynchronous versions can be used on any client EntityManager. The synchronous versions are only available for regular .NET clients and also can be used by custom server methods calling into a server-side EntityManager. You cannot use (or even see) these synchronous methods on an EntityManager used in other client application environments which do not support synchronous communications.

Members that read exclusively from the entity cache are always synchronous and are available to all EntityManagers, including Silverlight, Windows Store and Windows Phone application EntityManagers.

Member	Summary
ExecuteQuery	Synchronously executes its *IEntityQuery* query argument, returning an untyped *IEnumerable* of entities.
ExecuteQuery<T>	Synchronously executes its strongly typed *IEntityQuery<T>* query argument, returning an *IEnumerable<T> of T-type entities*.
ExecuteQueryAsync	Executes asynchronously its *IEntityQuery* query argument. An untyped *IEnumerable* of entities is returned in the completed task.
ExecuteQueryAsync<T>	Executes asynchronously its strongly typed *IEntityQuery<T>* query argument. An *IEnumerable<T>* of "T" entities is returned in the completed task.
FindEntities<T>	Searches the entity cache for entities of type "T" which also have an entitystate that matches one of the flag values in the EntityState argument.
FindEntities	Searches the entity cache for **all** entities whose entitystate matches one of the flag values the EntityState argument. Can optionally limit the search to entities of a specified type.
FindEntity	Searches the entity cache for an entity with a given EntityKey. By default the search would not return a deleted entity with that key but you can indicate with a boolean flag that you want the entity if it is deleted.
FindEntityGraph	Searches the entity cache for the entity "roots" passed as arguments to the method **and also** returns the entities related to those root entities. The "entity graph" is the combination of a root and its related entities. The extent of the graph is defined by the relationships identified in the *EntitySpan* argument.
GetNullEntity<T>	Get the "null entity" (aka, the "nullo") for the entity type "T" that represents the "entity not found". Reference navigation from an entity-in-cache (e.g., anOrder.Customer) returns the "nullo" rather than null if the entity can't be found.
GetNullEntity	The untyped variant of *GetNullEntity<T>*.
GetQuery<T>	Returns an *EntityQuery<T>* that, if executed, would query for all entities of type "T". The statement, manager.GetEntity<Customer>(), returns a query that, if executed, would retrieve all customers in the database. It is typically the

	foundation query to which the developer adds filter clauses to narrow the search.
RefetchEntities	Refreshes the entities identified in the method arguments with the most recent data available from the database.
RefetchEntitiesAsync	The asynchronous version of *RefetchEntities*.
TryExecuteQuery	Synchronously executes its *IEntityQuery* query argument, returning an untyped *QueryResult* containing the retrieved entities and other information about the processed query.
TryExecuteQuery<T>	Synchronously executes its strongly typed *IEntityQuery<T>* query argument, returning a *QueryResult<T>* containing the retrieved entities and other information about the processed query.
TryExecuteQueryAsync	Executes asynchronously its *IEntityQuery* query argument. An untyped *QueryResult* is returned in the completed task.
TryExecuteQueryAsync<T>	Executes asynchronously its strongly typed *IEntityQuery<T>* query argument. A *QueryResult<T>* is returned in the completed task.

Some *EntityManager* members notify subscribers about query activity.

Member	Summary
Querying	A cancellable event raised when the manager is about to process a query. The manager has not yet determined if it will satisfy the query from its entity cache or will have to go to the server for entities. It is raised before *Fetching*.
OnQuerying	The virtual method that a derived EntityManager can override to raise the *Querying* event.
Fetching	A cancellable event raised when the manager, while processing a query, has determined that it must go to the server for entities. It is raised before sending a query to the server. It won't be raised if the manager expects to satisfy the query entirely from its entity cache. Compare to *Querying*.
OnFetching	The virtual method that a derived EntityManager can override to raise the *Fetching* event.
Queried	An event raised by the manager after it has processed a query request. The event is raised even if the query was cancelled or failed. The query may or may not have fetched data from the server. If it did, the event is raised after entities were retrieved successfully and merged into the entity cache. The *EventArgs* can tell you whether the query involved a trip to the server and, if so, which of the entities retrieved from the server (including related entities) resulted in changes to the entity cache.
OnQueried	The virtual method that a derived EntityManager can override to raise the *Queried* event.

Finally, a few query-control members.

Member	Summary
DefaultQueryStrategy	Get and set the QueryStrategy the manager should use if the query itself does not specify a *QueryStrategy*.
QueryCache	Get the manager's *QueryCache*, the special cache that remembers previously issued queries. The manager may decide to fulfill a new query request entirely from its **entity cache** if the manager finds the query in its ***QueryCache***. Adding and removing queries from the *QueryCache* is largely automatic but you can manipulate it directly once you gain access to it through this property.

Save

The *EntityManager* members related to saving cached entities with pending changes.

Member	Summary
DefaultSaveOptions	Gets or sets the *SaveOptions* to be used as a default for any save with unspecified SaveOptions.

HasChanges	Get whether the manager's entity cache contains a changed entity. More specifically whether the entitystate of any cached entity is other than *Unchanged*.
SaveChanges	Save synchronously all changed entities in cache. Overloads accept optional parameters such as /SaveOptions.
SaveChangesAsync	Save asynchronously all changed entities in cache. Same overloads as SaveChanges.
TrySaveChanges	Save synchronously all changed entities in cache. Overloads accept optional parameters such as *SaveOptions*. The returned *SaveResult* provides information about success or failure and the entities that were saved.
TrySaveChangesAsync	Save asynchronously all changed entities in cache. Same overloads as SaveChange. The returned *SaveResult* provides information about success or failure and the entities that were saved.

Some *EntityManager* members notify subscribers about save activity.

Member	Summary
Saving	A cancellable event raised when the manager is about to save. The manager provides the handler with a list of entities it intends to save.
OnSaving	The virtual method that a derived EntityManager can override to raise the *Saving* event.
Saved	Raised when the manager has completed save processing. Will also be raised if the save failed or was cancelled. The event args provide information about success or failure and the entities that were saved.
OnSaved	The virtual method that a derived EntityManager can override to raise the *Saved* event.

Finally, a few save-control members for special situations.

Member	Summary
ForceIdFixup	Immediately replace the temporary ids of newly-created entities with server-generated permanent ids for entities whose keys are generated by a custom method. The save process performs this "fix-up" automatically but you can force it to happen sooner.
ForceIdFixupAsync	The asynchronous version of *ForceIdFixup*.

Access the entity cache

Some *EntityManager* members yield insights into the manager's entity cache.

Member	Summary
IsEntityLoaded(key)	Whether an entity with given EntityKey is in the manager's cache.
CacheStateManager	Get the manager's *CacheStateManager* through which you can save and restore a copy of the entity cache in the form of an *EntityCacheState*.
GetEntityGroup,T>	Get the *EntityGroup* inside the cache that holds the cached instances of type "T".
GetEntityGroup	The non-generic version of *GetEntityGroup<T>* takes the entity type as a parameter.
GetEntityGroups	Get an array of all *EntityGroups* in the cache. There will be a group for every type the cache has ever seen.

Many operations move entities into and out of a manager's entity cache automatically. You can manipulate this cache directly using these *EntityManager* members.

Member	Summary
AddEntities	Add entities to the cache as if they were new, *Added* entities that do not exist in the database and will be inserted if saved.
AddEntity	Add an entity to the cache as if it were a new, *Added* entity that does not exist in the database and will be inserted if saved.
AttachEntities	Attach entities to the cache as if they were pre-existing, *Unchanged* entities that might have been queried from the database.

AttachEntity	Attach an entity to the cache as if it were a pre-existing, *Unchanged* entity that might have been queried from the database.
Clear	Clear the cache, emptying it of all entities. Also clears the *QueryCache* described in the *Query* section above.
Cleared	Event raised when *Clear* is called.
RemoveEntities (entityState, ...)	Remove all entities with the given entitystate, optionally clearing the *QueryCache*.
RemoveEntities (entities, ...)	Remove the entities in the argument, optionally clearing the *QueryCache*.
RemoveEntities (entityType, entityState)	Remove entities of a particular and entitystate, optionally clearing the *QueryCache*.
RemoveEntity	Remove the entity, optionally clearing the *QueryCache*.
ImportEntities	Import shallow copies of the entities into the cache, merging them according to the provided *MergeStrategy*.

Create, modify and delete

These methods assist the developer in the course of creating, modifying and deleting entities.

Member	Summary
AcceptChanges	Treat all entities with pending changes as if they had been saved. They should appear as if they had been freshly retrieved from the database. *Added* and *Modified* entities become *Unchanged*, their current version values overwrite their original version values. *Deleted* entities are removed from cache, becoming *Detached*.
RejectChanges	Rollback all entities with pending changes. They should appear as if they had been freshly retrieved from the database. *Deleted* and *Modified* entities become *Unchanged*, their current version values restored from their original version values. *Added* entities are removed from cache, becoming *Detached*.
CreateEntity<T>	Create new entity of type "T"; the entity remains detached although the entity retains the inaccessible memory of the manager that created it.
CreateEntity	Create an entity dynamically with this non-geneneric version of *CreateEntity*.
EntityChanging	Cancellable event raised when a cached entity is about to change, e.g., an entity is about to be added, modified, attached, queried, imported, deleted, detached, committed or rolled back.
EntityChanged	Event raised when a cached entity has changed, e.g., an entity was added, modified, attached, queried, imported, deleted, deEtached, committed or rolled back.
GenerateId	Generate a temporary id for a newly created entity that requires custom Id generation; that temp id becomes permanent during save as result of an id fix-up process.

Configure

The *EntityManager* members with which to configure a new *EntityManager* and learn about its current configuration. Constructors have been omitted.

A few members you can set or attach to:

Member	Summary
AuthorizedThreadId	Get and set the id of the thread on which this manager can run. See the topic on thread safety for more information.
EntityManagerCreated	Static event raised whenever a new EntityManager is created. Useful for tracking and uniformly configuring the managers created in multiple EntityManager scenarios.
EntityServerError	Event raised when the *EntityServer* returns an exception for any reason.
DefaultEntityReferenceStrategy	Gets or sets the *EntityReferenceStrategy* to be used as a default for any navigation properties with an unspecified EntityReferenceStrategy

DefaultQueryStrategy	Gets or sets the *QueryStrategy* to be used as a default for any queries with an unspecified QueryStrategy.
DefaultSaveOptions	Gets or sets the *SaveOptions* to be used as a default for any save with unspecified SaveOptions.
Options	Access to the *EntityManagerOptions* controlling miscellaneous aspects of the EntityManager's behavior.
Tag	Get and set an arbitrary object. Useful for distinguishing one manager from another in multiple EntityManager scenarios.
VerifierEngine	Get and set the DevForce validation engine assigned to this manager. This is the validation engine used by automatic property validation within a cached entity.

Many of the configuration members you can only read:

Member	Summary
CompositionContext	Gets the *CompositionContext* instance that helps determine how MEF will compose some of the components that support this manager, a key part of the DevForce extensibilty story. You establish the manager's *CompositionContext* when you construct it.
DataSourceExtension	Get the string that identifies the data source targeted by this manager. You set the manager's *DataSourceExtension* when you construct it.
EntityServiceOption	The enumeration indicating whether the manager connects to a remote service (n-Tier), a local service (2-tier) or uses the setting from the configuration file. You establish the manager's *EntityServiceOption* when you construct it.
IsClient	Whether the manager is running on a client or on the server. The valus is usually *true* - meaning the manager is running on the client - but a number of processes on the server are provided with a server-side EntityManager; the *EntityServer* interceptor classes and your custom remote service methods are good examples. The value would be *false* for these server-side EntityManagers.
MetadataStore	Get the application-wide *EntityMetadataStore,* the store of metadata about all currently known entity types in the application.
UsesDistributedEntityService	Get whether this EntityManager communicates with a remote server as it must for a Silverlight, Windows Store or Windows Phone application EntityManager. It is *false* for 2-tier deployments.

Miscellaneous server calls

The *EntityManager* is often the application's sole channel for all communications with the server. Most communications concern data. Security and other kinds of server communications are addressed by the following members.

Connectivity

By default, the *EntityManager* automatically and immediately tries to connect to the server upon construction. But you may want to connect and disconnect for a variety of reasons with a method listed here:

Member	Summary
Connect	Connect to the *EntityServer* explicitly and synchronously.
ConnectAsync	Connect to the *EntityServer* explicitly asynchronously.
Disconnect	Disconnect the manager deliberately and continue running offline.
IsConnected	Get whether this manager believes it is connected to the server.
ConnectionStateChanged	Event raised when the connection state of the manager to the EntityServer changes.

Secure

We devote a full topic to security elsewhere. In brief, all DevForce *EntityManager* instances must have acquired a local security context before performing most server operations.

Member	Summary
AuthenticationContext	Get or set the security context used by this manager.
IsLoggedIn	Get whether this manager is currently authenticated.

Remote Service Method

The client may call a custom application method on the application server using one of these two *Invoke...* methods. These methods are untyped, affording the developer the ultimate luxury of sending almost anything as parameters and receiving almost any kind of result. The developer should consider wrapping these commands in a type-safe manner before exposing them to higher level application layers.

Member	Summary
InvokeServerMethod	Call a remote server method synchronously, passing in optional service method parameters of any type. The parameters must be both serializable and understood by the server method. *InvokeServerMethod* returns the *object* result that was itself returned by the remote server method. The client may cast the result or otherwise interpret it as best it can.
InvokeServerMethodAsync	The asynchronous variant of *InvokeServerMethod*, available on all .NET clients.