Client-side query life cycle events on the *EntityManager* include *Querying*, *Fetching*, and *Queried*. These are summarized in the table below.

DevForce raises the client-side *Querying* event as query execution begins. The *Fetching* event is raised before sending the query to the *EntityServer*; if a query can be satisfied from cache, then *Fetching* is not raised. The *Queried* event is raised just before returning query results. We can listen to these events by attaching a custom handler.

Both the *Querying* and the *Fetching* events provide the query object. Your handler can examine the object (it implements *IEntityQuery* and choose to let the query continue as is, continue after modification, or cancel it. If you cancel the query, the Entity Manager method returns as if it found nothing. If the method returns a scalar entity, it yields the return entity type's Null Entity; otherwise, it returns a null entity list. Beware of canceling an entity navigation list query method.

The *Queried* event fires just before the query method returns. Entities have been fetched and merged into the cache. The event arguments include a list, *EntitiesChanged*, of any entities added to or modified in the cache. Note that this list could include entities of the target entity type – the kind we expected returned from the query – but also of other types brought in via *Includes* or by virtue of *query inversion*.

Client-Side Life-Cycle Events

Event	EventArgs	Comments	Typical Uses of the Corresponding Event Handler
Querying	EntityQueryingEventArgs	Raised as query execution begins. Provides access to the submitted query.	Modify the query being submitted, or refuse the request for data.
Fetching	EntityFetchingEventArgs	Raised before sending the query to the EntityServer. Provides access to the submitted query. If a query can be satisfied from cache, Fetching is not raised.	Modify the query being submitted, or refuse the request for data.
Queried	EntityQueriedEventArgs	Raised just before returning query results. Provides access to the complete result set, as well as the subset representing changes to the cache.	Modify the objects that were returned by the query

Corresponding to each of these events is a protected virtual method on the EntityManager that can be used to perform the same task directly in a subclassed *EntityManager*.

Member	Summary
OnQuerying	The virtual method that a derived *EntityManager* can override to raise the *Querying* event.
OnFetching	The virtual method that a derived *EntityManager* can override to raise the *Fetching* event.
OnQueried	The virtual method that a derived *EntityManager* can override to raise the *Queried* event.

EntityQueryingEventArgs

Property	PropertyType	Access	Description
Query	IEntityQuery	get, set	The query about to be executed. The query can be modified here, but the final shape must remain the same.
Cancel	bool	get, set	Allows the query to be canceled.

EntityFetchingEventArgs

Property	PropertyType	Access	Description
Query	IEntityQuery	get	The query about to be executed. The query can no longer be modified here, use the Querying event if you need modification.
Cancel	bool	get, set	Allows the query to be canceled.

EntityQueriedEventArgs

Note that this is the non-generic version of the *EntityQueriedEventArgs<T>* class. As a result the Query and Results properties here are non-generic as well. i.e. *IEntityQuery* vs *IEntityQuery<T>*, *IEnumerable* vs *IEnumerable<T>*.

Property	PropertyType	Access	Description
Query	IEntityQuery	get	The query just executed.
Results	IEnumerable	get	The results of the query.
ChangedEntities	IList<Object>	get	The list of every entity that was either added or modified in the EntityManager's cache as a result of this query. The *ChangedEntities* list may differ from the *Results* property since the *Results* will include only those entities directly queried, and not entities fetched due to query inversion or use of an Include.
WasFetched	bool	get	Whether the operation actually required a trip to the database. Many queries can be completed without having to go to the database.
ResolvedFetchStrategy	FetchStrategy	get	The *FetchStrategy* actually used to process the query. This is really only useful when an 'optimized' *FetchStrategy* was stipulated when executing the query. What is returned here is the query strategy that 'optimized' determined was most appropriate.