

Contents

- [The With\(\) Extension Method](#)
- [The FirstOrNullEntity\(\) ExtensionMethod](#)
- [The ToQuery \(\) ExtensionMethod](#)

DevForce provides several additional "query support" extension methods.. All of these methods are defined as extensions to *IEntityQuery*.

The With() Extension Method

The *With()* extension method permits you to substitute a different *QueryStrategy*, a different target *EntityManager*, or both, on any existing instance of an *IEntityQuery*. The original query will be left unaltered.

When a call to *With()* is chained to a query, the result may be either a new query or a reference to the original query. Normally it will be a new query, but if the content of the *With()* call is such that the resultant query would be the same as the original one, a reference to the original query is returned instead of a new query.

If you ever want to be sure that you get a new query, use the *Clone()* extension method instead of *With()*. *With()* avoids the overhead of a *Clone()* when a copy is unnecessary.

You can pass null arguments to *With()*.

If a query does not have an *EntityManager* assigned, an exception is thrown if you attempt to execute it.

When a query has a null *QueryStrategy*, it uses the *DefaultQueryStrategy* of the assigned *EntityManager*. See the code below for more detail on the possibilities.

```
// Setup
DomainModelEntityManager em1 = null;
DomainModelEntityManager em2 = null;
IEntityQuery<Customer> query0 = em1.Customers
    .Where(c => c.CompanyName.ToLower().StartsWith("a"))
    .With(QueryStrategy.DataSourceOnly);
//Use With() to run the existing query against a different EntityManager:
List<Customer> customers = new List<Customer>(query0.With(em2));
//The next two examples use With() to run the query with a different QueryStrategy.
//The With() call in the right-hand side of the following statement
//specifies a query that is materially different from query0, in
//that it has a different QueryStrategy associated with it.
//Accordingly, the right-hand side of the statement will return
//a new query:
IEntityQuery<Customer> query1 = query0.With(QueryStrategy.CacheOnly);
//Because the content of the With() call in the right-hand side
//of the following statement doesn't result in a modification
//of query0, the right-hand side will return a reference to
//query0 rather than a new query.
IEntityQuery<Customer> query2 = query0.With(QueryStrategy.DataSourceOnly);
//If you want to be certain you get a new query, use Clone()
//rather than With():
EntityQuery<Customer> query3 = (EntityQuery<Customer>)query0.Clone();
query3.QueryStrategy = QueryStrategy.DataSourceOnly;
//Change both the QueryStrategy and the EntityManager
IEntityQuery<Customer> query4 = query0.With(em2, QueryStrategy.CacheOnly);
//Run the query against the assigned EntityManager, using that EntityManager's
//default QueryStrategy:
IEntityQuery<Customer> query7 = query0.With((QueryStrategy)null);
```

```
' Setup
Dim em1 As DomainModelEntityManager = Nothing
Dim em2 As DomainModelEntityManager = Nothing
Dim query0 As IEntityQuery(Of Customer) = em1.Customers.Where( _
    Function(c) c.CompanyName.ToLower().StartsWith("a")). _
    With(QueryStrategy.DataSourceOnly)
'Use With() to run the existing query against a different EntityManager:
Dim customers As New List(Of Customer)(query0.With(em2))
'The next two examples use With() to run the query
'with a different QueryStrategy.
'The With() call in the right-hand side of the following statement
'specifies a query that is materially different from query0, in
'that it has a different QueryStrategy associated with it.
'Accordingly, the right-hand side of the statement will return
```

```
'a new query:
Dim query1 As IEntityQuery(Of Customer) = _
    query0.With(QueryStrategy.CacheOnly)
'Because the content of the With() call in the right-hand side
'of the following statement doesn't result in a modification
'of query0, the right-hand side will return a reference to
'query0 rather than a new query.
Dim query2 As IEntityQuery(Of Customer) = _
    query0.With(QueryStrategy.DataSourceOnly)
'If you want to be certain you get a new query, use Clone()
'rather than With():
Dim query3 As IEntityQuery(Of Customer) = CType(query0.Clone(), _
    IEntityQuery(Of Customer))
query3.QueryStrategy = QueryStrategy.DataSourceOnly
'Change both the QueryStrategy and the EntityManager
Dim query4 As IEntityQuery(Of Customer) = query0.With(em2, _
    QueryStrategy.CacheOnly)
'Run the query against the assigned EntityManager, using that
'EntityManager's default QueryStrategy:
Dim query7 As IEntityQuery(Of Customer) = _
    query0.With(CType(Nothing, QueryStrategy))
```

The FirstOrNullEntity() ExtensionMethod

[LINQ](#) to Entities provides *First()* and *FirstOrDefault()* extension methods on queries. *First()* returns the first item in a collection meeting the query criteria; *FirstOrDefault()* returns that, or if no items meet the criteria, the default value for the target type. For integer target types, *FirstOrDefault()* returns a zero; for string types, it returns an empty string. For complex types or other types that have no default, it returns a null.

DevForce adds a *FirstOrNullEntity* extension method that can be used when you are querying for target types that inherit from *IdeaBlade.EntityModel.Entity*. If no entity meets the specified criteria, *FirstOrNullEntity()* returns the DevForce *NullEntity* for the target type. The *NullEntity* is a non-saveable, immutable, syntactically correct instance of an entity represents "nothing there" but will not trigger an exception.

The screenshot shows a Visual Studio IDE with a C# code snippet and its runtime results in the Watch window.

Code Snippet:

```
//// The following results in an exception, as there is no Company named "kjkkkj":
//_em1.Customers.Where(c => c.CompanyName == "kjkkkj").First();

//// This returns a null for aCustomer2
//Customer aCustomer2 = _em1.Customers.Where(c => c.CompanyName == "kjkkkj").FirstOrDefault();

// This returns a NullEntity Customer for aCustomer2
Customer aCustomer2 = _em1.Customers.Where(c => c.CompanyName == "kjkkkj").FirstOrNullEntity();
```

Watch Window:

Name	Value
aCustomer2	{DomainModel.Customer}
base	{DomainModel.Customer}
Address	null
City	null
CompanyName	""
ContactName	null
ContactTitle	null
Country	null
CustomerID	{00000000-0000-0000-0000-000000000000}
CustomerID_OLD	null
EntityManager	{DomainModel.DomainModelEntityManager}
Fax	null
NumberOfOrders	0
Orders	{IdeaBlade.EntityModel.v4.RelatedEntityList<DomainModel.Order>}
Orders_Reference	{IdeaBlade.EntityModel.v4.ListEntityReference<DomainModel.Order>}
Phone	null
PostalCode	null
Region	null
RowVersion	null
Static members	

The ToQuery () ExtensionMethod

Every *IdeaBlade.EntityModel.Entity* has a [ToQuery\(\)](#) extension method that returns an *IEntityQuery<T>* where T is an Entity type. This *IEntityQuery<T>* specifies the Entity on which it was based using its *EntityAspect.EntityKey*, and can be extended to perform various useful operations. Consider, for example, the following statements:

```
Customer aCustomer = _em1.Customers.FirstOrDefaultEntity();
var query = aCustomer.ToQuery<Customer>()
    .Include(Customer.PathFor(c => c.Orders));
query.With(QueryStrategy.DataSourceOnly).ToList();

Dim aCustomer As Customer = _em1.Customers.FirstOrDefaultEntity()
Dim query = aCustomer.ToQuery().Include(Customer.PathFor(Function(c) c.Orders))
query.With(QueryStrategy.DataSourceOnly).ToList()
```

Here, from a Customer entity, we have created a query that will retrieve that same Customer. We have then extended with a call to *Include()* it to create a span query that will also retrieve all of that Customer's associated Orders. We do not otherwise have so convenient a way to accomplish this goal.

The *ToQuery()* extension method is also provided on any *IEnumerable<T>* collection, when T is an Entity. Thus you can turn an arbitrary list of Customers into a query that will return the same set of Customers. The *Where()* clause on the resultant query will specify a series of *OR'd* key values. For example, consider the following statements:

```
List<Customer> customers = _em1.Customers
    .Where(c => c.CompanyName.ToLower().StartsWith("a")).ToList();
var query2 = customers.ToQuery<Customer>();

Dim customers As List(Of Customer) = _em1.Customers _
    .Where(Function(c) c.CompanyName.ToLower().StartsWith("a")).ToList()
Dim query2 = customers.ToQuery()
```

Placing *query2* in a watch window reports its value as the following:

```
{ value(IdeaBlade.EntityModel.EntityQueryProxy`1[DomainModel.Customer]).Where(t => (((t.CustomerID = 785efa04-cbf2-4dd7-a7de-083ee17b6ad2) || (t.CustomerID = b61cf396-206f-41a6-9766-168b5cbb8edd)) || (t.CustomerID = f214f516-d55d-4f98-a56d-7ed65fd79520)) || (t.CustomerID = 256d4372-baa7-4937-9d87-d9a4e06146f8)))}
```

The first query evidently placed four Customers in the customers list; the query returned by *ToQuery()* specifies those four by their (GUID) key values.