Contents

There are several interesting issues that arise when DevForce tries to execute the same *EntityQuery* against both a backend datastore using the Entity Framework and its own local *EntityManager* cache. Internally DevForce uses LINQ to Entities to query entities on the server but uses LINQ to Objects (LINQ against the .NET CLR) to query these same entities from the local cache. Unfortunately, there are subtle semantic differences between LINQ to Entities and LINQ to Objects. Some of these differences have to do with the way that LINQ to Entities itself is defined and some of these differences have to do with the backend SQL database implementation that LINQ to Entities is communicating with. Ideally we want to **match the semantics of cache queries to database queries**.

QueryStrategy and CacheQueryOptions

In general, it is DevForce's goal to insure that the execution of a query against the local EntityManager cache is interpreted identically to that same query run against Linq to Entities on the server. This is accomplished by modifying the way that standard LINQ to Objects queries are interpreted when run against the DevForce EntityManager cache. The goal is to mirror LINQ to Entities semantics for these queries. While much of this can be performed automatically by DevForce, there is some logic that is dependent on the settings of whatever backend SQL database is being used on the server.

The *IdeaBlade.EntityModel.CacheQueryOptions* property contains the settings that allow a developer to inform DevForce of any database settings that may effect LINQ to Entities behavior, and that will therefore require DevForce to modify the way it executes those same queries when run against the local cache as well. Typically, within a single application, these settings will be the same for all queries and therefore the *CacheQueryOptions* class offers a static (shared in VB) 'Default' property that will be used by every query unless explicitly overridden on that query's *QueryStrategy*. By default this property returns an instance of *CacheQueryOptions* that is appropriate for most standard SQL Server implementations. (See *CacheQueryOptions.DefaultSqlServerCompatibility*)

Instances of the *CacheQueryOptions* class are immutable and can be created via the following constructor

```
public CacheQueryOptions(StringComparison stringComparison,
 bool useSql92CompliantStringComparison, GuidOrdering guidOrdering)
```

```
public CacheQueryOptions(StringComparison stringComparison, _
 Boolean useSql92CompliantStringComparison, GuidOrdering guidOrdering)
```

Each of the parameters to the constructor is described below:

stringComparison: Because many SQL databases are configured to allow queries against string columns to be performed in a case insensitive manner, it is necessary to inform DevForce that it should mirror this behavior when performing queries against its local cache. Note that this is different than standard Linq to Objects (.NET CLR behavior) which is to perform case sensitive comparisons. The default for this is *StringComparison.OrdinalIgnoreCase*. (i.e. case insensitive comparisons).

useSql92CompliantStringComparison: The ANSI/ISO SQL-92 specification (Section 8.2, <Comparison Predicate>, General rules #3) describes how to compare strings with spaces. The ANSI standard requires padding for the character strings used in comparisons so that their lengths match before comparing them. The padding directly affects the semantics of WHERE and HAVING clause predicates and other string comparisons. For example, ANSI-SQL considers the strings '**abc**' and '**abc** ' to be equivalent for most comparison operations. This is the behavior that DevForce mimics for local cache queries when this value is set to true, otherwise normal .NET CLR comparison semantics are used (i.e. no padding). The default value for this flag is true.

guidOrdering: SQL Server currently sorts 'Guids' according to different rules than those used by the .NET CLR. This setting allows the local cache to match SQL Server's sorting behavior in any LINQ query that involves grouping or ordering of 'Guid's. The default for this property is ***GuidOrdering.SqlServer***. The only other option currently available is ***GuidOrdering.CLR***.