Contents

All queries processed on the EntityServer including **named queries** pass through the DevForce *EntityServerQueryInterceptor* or a custom *EntityServerQueryInterceptor* if you wrote one. This topic describes how you can distinguish between named and unnamed queries and intervene as you deem appropriate.

Your custom *EntityServerQueryInterceptor* can inspect properties of the *EntityQuery* to determine where it came from and how it was composed. You can alter the query or even change the query results before they are transmitted to the client.

Everything your interceptor could do to an unnamed client query it can do to the re-composed query derived from a named query.

Cast the query as an *EntityQuery*

The *EntityServerQueryInterceptor* handles every query that implements *IEntityQuery*. A typical first step is to determine what kind of query you are intercepting and respond accordingly. A named query is necessarily an *EntityQuery* so you might start by casting to *EntityQuery* as in the following example:

```
public class MyEntityServerQueryInterceptor :
          EntityServerQueryInterceptor
{
  protected override bool FilterQuery() // called before processing the query
  {
     EvaluateEntityQuery();
    return base.FilterQuery();
  }
  private void EvaluateEntityQuery()
  {
     var entityQuery = this.Query as EntityQuery;
    if (null == entityQuery) return; // not an EntityQuery;
    ...
  }
}
```

```
Public Class MyEntityServerQueryInterceptor
Inherits EntityServerQueryInterceptor
 Protected Overrides Function FilterQuery() As Boolean
  ' called before processing the query
  EvaluateEntityQuery()
  Return MyBase.FilterQuery()
 End Function
 Private Sub EvaluateEntityQuery()
Dim entityQuery = TryCast(Me.Query, EntityQuery)
 If Nothing Is entityQuery Then ' not an EntityQuery;
  Return
 End If
 ...
 End Sub
End Class
```

Check *IsNamedQuery*

The *EntityQuery.IsNamedQuery* property returns *true* if

- the intercepted query was constructed from a named query.
- the property is called on the server inside the *EntityServerQueryInterceptor*.

Otherwise *EntityQuery.IsNamedQuery* returns *false* and the other named-query-related properties discussed here return *null* or *false*.

The *EntityServerQueryInterceptor.FilterQuery* virtual method is a good place to check for named queries. If you require that all client queries be routed through named queries, you can throw a custom exception when *IsNamedQuery* is *false*.

You could add the following to the *EvaluateEntityQuery* method from the previous example:

```
if (!entityQuery.IsNamedQuery) throw new NamedQueryException("Not a named query");
```

```
If Not entityQuery.IsNamedQuery Then
 Throw New NamedQueryException("Not a named query")
End If
```

The *EntityServer* terminates query processing and transmits the exception to the client.

A more flexible approach is to maintain a whitelist of entity types which do not require named query methods. Throw the exception if the unnamed query isn't on the list:

```
Type rootQueryType = entityQuery.QueryableType; // the entity type being queried
if (!(entityQuery.IsNamedQuery || UnnamedQueryWhitelist.Contains(rootQueryType))
 throw new NamedQueryException("Not a named query");
}
```

```
Dim rootQueryType As Type = entityQuery.QueryableType ' the entity type being queried
If Not(entityQuery.IsNamedQuery OrElse UnnamedQueryWhitelist.Contains(rootQueryType)) Then
 throw New NamedQueryException("Not a named query")
End If
```

Properties of named queries

The following properties of the EntityQuery class return meaningful values on the server when the intercepted query is a named query.

Property	Description
OriginalClientQuery	The original query from the client exactly as it was received by the server.
NamedQuery	The query output of the named query method. This is the root of the query that the interceptor is evaluating.
NamedQueryMethod	The reflection *MethodInfo* of the named query method.
NamedQueryResultIsEnumerable	Is *true* if the query method returns an *IEnumerable*; *false* if *IQueryable*

See the topics dedicated to the *EntityServerQueryInterceptor* to learn about other ways to intervene in query processing.