Contents

An n-tier DevForce client application may experience sometimes transient communication-related failures. For some of these failures DevForce will automatically retry based on default settings, but you can also **configure auto retry** options.

Default behavior

For *EntityManager* actions other than *SaveChanges*, such as *Connect* and query, and for *Authenticator.Login*, the *CommunicationSettings.Default.DefaultRetryPolicy* is used. This in turn defaults to a *FixedRetry* policy which will attempt 3 retries with a half-second delay between each attempt.

For *EntityManager* save operations, including *ForceIdFixup*, the *CommunicationSettings.Default.DefaultRetryPolicyForSave* is used, which defaults to a *NoRetry* policy.

To set your own application-wide defaults you can change either the *DefaultRetryPolicy* or *DefaultRetryPolicyForSave* as needed.

For example, to set the *DefaultRetryPolicy* to perform 2 retries with a 1 second delay between tries:

```
using IdeaBlade.Core;
using IdeaBlade.Core.Wcf.Extensions;
CommunicationSettings.Default.DefaultRetryPolicy = new FixedRetry { MaxTries = 2, Delay = TimeSpan.FromSeconds(1) };
```

A retry for a save operation is not advised, unless you can inspect the exception to ensure that the request never reached the server. You can implement a custom *IRetryPolicy* to inspect the exception to determine if a retry can be done.

For example:

```
using IdeaBlade.Core;
public class MyRetryPolicy : IRetryPolicy  {
 // Allow 2 retries for EndpointNotFound errors.
 public bool ShouldRetry(int currentRetryCount, Exception exception, out TimeSpan delay) {
   delay = TimeSpan.FromSeconds(1);
  if (exception is System.ServiceModel.EndpointNotFoundException) {
   return currentRetryCount < 2;
   } else {
   return false;
   }
  }
}
```

Per request overrides

Several *EntityManager* actions can also use a custom retry policy for the request.

For queries, the *QueryStrategy.CommunicationRetryPolicy* can be set.

For example:

```
var qs = QueryStrategy.Normal.With(new FixedRetry { MaxTries = 5, Delay = TimeSpan.FromMilliseconds(100) });
var customers = await entityManager.Customers.With(qs).ExecuteAsync();
```

Individual save operations can also use a custom policy, using *SaveOptions.CommunicationRetryPolicy*. As noted above, use caution if you implement a retry for save operations.

Calls to remote server methods may use a custom retry policy when using the *InvokeServerMethod* (or *InvokeServerMethodAsync*) override which accepts an *InvokeServerMethodArgs* parameter.

For example:

```
// Call an RPC method with NoRetry
var args = new InvokeServerMethodArgs {
  FullTypeName = "DomainModel.ServerMethods, DomainModel",
  MethodName = "GetNumberOfOrders",
  ClientArgs = new object[] { 10, new DateTime(1995, 1,1), new DateTime(1999, 1,
  CommunicationRetryPolicy = new NoRetry()
};
```

```
var num = (int)entityManager.InvokeServerMethod(args);
```