Contents

When requests to the *EntityServer* fail (including query and save requests), the server-side exception is wrapped in an *EntityServerException* and sent back to the client. **Understanding the *EntityServerException*** can help in debugging application problems. You may be able to catch the exception and recover - an option that begins with listening to the *EntityManager.EntityServerError* event.

If a request to the *EntityServer* fails the exception returned to the client application will be wrapped in an *EntityServerException*. Your application might receive the base *EntityServerException*, or one of the custom sub-types:

- *EntityManagerSaveException* for save failures.
- *EntityServerConnectionException* for a subset of *EntityServer* connection failures.
- *LoginException* for login failures.
- *PersistenceSecurityException* for a subset of unauthorized access attempts.

Details

While the text of the exception message isn't always helpful, since exceptions can bubble up through many layers, in not only DevForce but in .NET and other assemblies, you'll often have to look at other details in the exception to understand the problem.

All *EntityServerExceptions* will have this information:

- *FailureType* will indicate the general reason for the failure: for example *Data* will indicate a query or save failed at the database, while *Connection* will indicate a communications failure. There are a handful of other reasons too.
- *OperationType* indicates the operation being performed at the time of failure, for example *Query* or *Save*, among others.

For failures occurring on the EntityServer in an n-tier application the *InnerException* will often be empty because that information is not serialized to the client. Instead, several other properties will contain this information:

- *RemoteExceptionDetails* - The *ToString()* representation of the original exception.
- *RemoteExceptionName* - The exception class name.
- *RemoteSource* - The source of the original exception.
- *RemoteStackTrace* - The stack trace of the original exception.

If the *InnerException* is present, drilling down into it and all nested inner exceptions can help determine the original cause of the failure.

Catching the exception

You should wrap all calls to the *EntityServer* with **try/catch** logic.

For queries and saves, you can optionally use one of the "try/result" methods:

- *TryExecuteQuery* and *TryExecuteQueryAsync* will return a *QueryResult* instead of throwing an exception. You can then examine the *Error* property.
- *TrySaveChanges* and *TrySaveChangesAsync* will return a *SaveResult* instead of throwing an exception. You can then examine the *Error* property.

The *EntityManager* also provides a central error handling facility for *EntityServerExceptions*. The *EntityServerError* event will be raised as a "first chance" handler for any exceptions before they are thrown. In your event handler you can mark the exception as handled so that it will not be re-thrown.