In production, your client application should not receive error messages from the server which might contain detailed or sensitive information. You can **intercept server exceptions** to examine and optionally replace the exception to be sent to the client.

# EntityServerErrorInterceptor

Implement a custom *EntityServerErrorInterceptor* to intercept exceptions thrown on the server. With the *OnError* method of the interceptor you'll have an opportunity to inspect all exceptions - thrown by login, queries, saves, and unhandled exceptions from RPC methods.

The default interceptor writes a trace message of the the full exception text to the log, but does not modify the original exception. To avoid sending sensitive information to a client application - for example, when a save fails it might indicate the database table or user - you can replace the exception to be returned to the client.

Note that you can't entirely "swallow" the exception and return no error to the client. If you set the exception to null DevForce will send the original exception back to the client.

The error interceptor is easy to implement, simply sub-class the base *EntityServerErrorInterceptor* and override the *OnError* method. Be sure to place your class in an assembly deployed to the server.

Here's a simple example:

```
using System;
using IdeaBlade.Core;
using IdeaBlade.EntityModel;
using IdeaBlade.EntityModel.Server;
namespace Samples {
 class SampleErrorInterceptor : EntityServerErrorInterceptor {
  public override void OnError(ErrorInterceptorArgs args) {
    // Log the real exception
    TraceFns.WriteLine(string.Format("Error {0} on thread {1} for user {2}",
      args.Exception.ToString(), System.Threading.Thread.CurrentThread.ManagedThreadId, base.Principal.Identity.Name));
    // Add more detail if possible:
    var ese = args.Exception as EntityServerException;
    if (ese != null) {
      TraceFns.WriteLine(string.Format("Failure type: {0}, operation type: {1}", ese.FailureType, ese.OperationType));
     }
    // If the user is a guest, then scrub the exception.
    if (!base.Principal.Identity.IsAuthenticated) {
      args.Exception = new ApplicationException("The request could not be completed.");
     }
   }
  }
}
```

The exception will be wrapped in an EntityServerException when returned to the client, with the *RemoteException* information describing your exception. Because the actual exception you throw is not returned to the client application, you can return any exception type, even it that type is not defined on the client or is not serializable.

Remember that if you do scrub the exception, the try/catch logic in your client code should be able to handle it.