Contents

You can perform an **asynchronous save** to help enhance the overall responsiveness of your application.

Async/await

With the new task-based asynchronous programming model, an asynchronous save is as easy as a synchronous one.

```
private async void DoSaveAsync() {
 try {
    await _manager.SaveChangesAsync();
 catch (EntityManagerSaveException) {
    MessageBox.Show("Save error");
  }
}
```

Be sure to wrap your async call in a try/catch to handle the save errors which inevitably occur. Or if you prefer, you can "try" a save and examine the save results:

```
private async void DoSaveAsync() {
  SaveResult saveResult = await _manager.TrySaveChangesAsync();
 if (!saveResult.Ok) {
     MessageBox.Show("save did not succeed");
  }
}
```

You'll note we added the async (or Async in Visual Basic) modifier to indicate that the method contains asynchronous code, and the *await* (or *Await*) keyword to indicate that further processing in the method should be suspended until the asynchronous task completes.

Since the entire entity cache will be saved during a call, there's no reason to issue multiple concurrent asynchronous saves from the same *EntityManager*.

Either a Task or *Task<SaveResult>* is returned from an async save. The task represents the asynchronous operation, and will indicate the status of the operation, the results of a completed operation, and whether the operation was cancelled or failed.

Note that the task returned from a DevForce async method is "hot": it has already started and is scheduled for execution.

Cancelling a save

The asynchronous save task cannot itself be cancelled. You can, however, cancel a save before it starts in an *EntityMaager Saving* event handler. You can also cancel the save in a custom *EntityServerSaveInterceptor* before the save begins on the server.

In either case, the save task will be cancelled. If you await the *SaveChangesAsync* call, an *OperationCanceledException* is thrown. If you've called the *TrySaveChangesAsync* method, the *SaveResult* returned will indicate the save was cancelled.

Error handling

An *awaited* task will throw an exception if it's either faulted or cancelled. This is why you should wrap any await calls in a try/catch.

Save processing exceptions are passed to the *EntityManager's EntityServerError* handler if one is defined. If you do mark the error as handled the exception will not be rethrown.