Contents

Use the **DevForce Compatibility Pack** to ease the migration of existing DevForce 2010 projects using the "operation/callback" asynchronous API or other capabilities not included in the new API.

Overview

For existing DevForce 2010 .NET and Silverlight projects which use the "operation/callback" asynchronous API or the "partial save" capability, you can install the DevForce Compatibility Pack NuGet package.

Install the compatibility pack to each project which will use the backwards compatibility API. The package adds the required dependency, *IdeaBlade.EntityModel.Compat*, to the project.

The compatibility pack is only necessary if you are upgrading an existing DevForce 2010 application which uses the older asynchronous API. If you are developing a new DevForce application please do not use the compatibility pack, as the new Task-based asynchronous pattern is much richer and easier to use.

What's Included

To use the backwards compatibility API, add a using/Imports statement to your code file for ***IdeaBlade.EntityModel.Compat***. This namespace contains the extension methods and Coroutine support for the compatibility API.

Coroutines

Support for serial and parallel coroutines is included in the compatibility API.

The Coroutine.Start and Coroutine.StartParallel method overloads are unchanged from DevForce 2010. What is different is the asynchronous method signatures you'll use within your coroutines.

For example:

```
public void SampleCoroutine() {
  var op = Coroutine.Start(() => SampleCoroutineCore());
  op.Completed += (s, e) => {
   if (e.HasError) {
     e.MarkErrorAsHandled();
     MessageBox.Show("an error occurred");
    }
  };
}
private IEnumerable<INotifyCompleted> SampleCoroutineCore() {
 yield return _entityManager.Customers.Where(c => c.Country == "USA").ExecuteAsync(userCallback: null);
 yield return _entityManager.Employees.Where(e => e.Country == "USA").ExecuteAsync(userCallback: null);
}
```

Here we see that the ExecuteAsync methods require a *userCallback* argument, which was optional in DevForce 2010. This argument is required in order to disambiguate the method signature from the other, Task-based, signatures in the new asynchronous API.

Partial Save

A "partial" save can be used to persist a subset of the changed entities in the *EntityManager* cache. It is not a recommended practice and was not included in the newer DevForce API. It is provided here in the compatibility API for applications written in DevForce 2010 which still require this capability during their migration.

The partial save API, while providing an "older" feature, uses the "newer" DevForce API and does not return a *SaveResult* unless requested. The following *EntityManager* extension methods are provided:

Synchronous methods:

- void SaveChanges(IEnumerable entities, SaveOptions saveOptions = null)
- SaveResult TrySaveChanges(IEnumerable entities, SaveOptions saveOptions = null)

Asynchronous methods:

- Task SaveChangesAsync(IEnumerable entities, SaveOptions saveOptions = null)
- Task<SaveResult> TrySaveChangesAsync(IEnumerable entities, SaveOptions saveOptions = null)
- EntitySaveOperation SaveChangesAsync(IEnumerable entities, SaveOptions saveOptions, Action<EntitySaveOperation> userCallback, object userState = null)

Async API

Asynchronous operations without a return result, such as *ConnectAsync* and *ForceIdFixupAsync*, now return a BasicOperation instead of a *BaseOperation*.

In order to disambiguate method signatures from their task-based counterparts in DevForce, all async methods now require the *userCallback* argument. You can pass null/Nothing for the argument value.

Asynchronous operations may be cancelled, where the operation supports it, using the *Cancel* method on the operation.

EntityManager extension methods

The familiar asynchronous methods on the *EntityManager* in DevForce 2010 are provided, with a few minor signature changes.

Remember that *LoginAsync* and *LogoutAsync* from the *EntityManager* were deprecated in DevForce 2010 and are now available on the Authenticator.

In order to disambiguate method signatures from their task-based counterparts in DevForce, all methods now require the *userCallback* argument. You can pass null/Nothing for the argument value.

For example, an asynchronous query:

```
var query = entityManager.Customers.OrderBy(c=> c.CompanyName).Take(10);
var op = entityManager.ExecuteQueryAsync(query, userCallback: null);
op.Completed += (o, e) => {
  var customers = e.Results;
};
```

```
Dim Query = Manager.Customers.OrderBy(Function(c) c.CompanyName).Take(10)
Dim Op = Manager.ExecuteQueryAsync(Query, userCallback:=Nothing)
AddHandler Op.Completed, Sub(o, e)
 Dim Customers = e.Results
End Sub
```

And an asychronous save:

```
var op = entityManager.SaveChangesAsync(userCallback: null);
op.Completed += (o, e) => {
  var result = e.SaveResult;
};
```

```
Dim Op = Manager.SaveChangesAsync(userCallback:=Nothing)
AddHandler Op.Completed, Sub(o, e)
 Dim Result = e.SaveResult
End Sub
```

EntityQuery and EntityScalarQuery extension methods

The *ExecuteAsync* extension methods on the *IEntityQuery* and *IEntityQuery<T>* are supported, and as with other extension methods in the compatibility pack, require a *userCallback* argument.

```
var query = _entityManager.Customers.Where(c => c.Country == "France");
var op = query.ExecuteAsync(userCallback: null);
op.Completed += (o, e) => {
  var customers = e.Results;
};
```

The many scalar async extension methods are provided: First, FirstOrDefault, FirstOrNullEntity, Single, SingleOrDefault, SingleOrNullEntity, Count, LongCount, Min, Max, Sum, Average, Any, All. They too require a *userCallback* argument to disambiguate the methods from their task-based counterparts.

For example:

```
var op = _entityManager.Employees.AsScalarAsync().Count(userCallback: null);
op.Completed += (o, e) => {
  var employeeCount = e.Result;
};
```

Authenticator extension methods

The *LoginAsync* and *LogoutAsync* methods are defined for the Authenticator. As with other methods in the compatibility pack, the *userCallback* argument is now required.

The *LogoutAsync* method now returns a *BasicOperation*, instead of a *BaseOperation*.