Contents

This chapter describes the backward-compatibility features of Punch to help with migration of an existing code base.

Introduction

One of the bigger breaking changes in Punch and DevForce 2012 is the drop of the Coroutine-style Asynchronous Pattern in favor of the more modern Task-based Asynchronous Pattern (TAP). To ease migration of legacy code to TAP, both DevForce and Punch provide Compatibility Packs with limited backwards-compatibility for the Coroutine-style Asynchronous Pattern.

Preparing the project

To add backwards-compatibility support to your projects, install the "Punch Compatibility Pack NuGet package to each project, this will add a reference to Cocktail.Compat.dll. In addition, the "DevForce 2012 Compatibility Pack is automatically installed as a required dependency, which adds a reference to IdeaBlade.EntityModel.Compat.dll.

What's included?

The "Punch Compatibility Pack" provides the following features.

- Contains the OperationResult and OperationResult<T> types.
- Implicit and explicit conversions from OperationResult to Task and back.
- Provides *await* support for OperationResult and DevForce operations.
- Extension methods with backwards-compatible signatures.

Implicit conversions

In many cases, OperationResult and Task can be used interchangeably. The compiler implicitly converts one to the other if possible. Consider the following example.

```
public OperationResult<bool> SomeMethodAsync()
{
  return OperationResult.FromResult(true);
}
public Task<bool> SomeMethodTaskAsync()
{
  // Implicit conversion from OperationResult<bool> to Task<bool>
  return SomeMethodAsync();
}
```

Similarly, OperationResult can be *awaited* in an asynchronous method as demonstrated in the following example.

```
public OperationResult<bool> SomeMethodAsync()
{
  return OperationResult.FromResult(true);
}
public async Task<bool> SomeMethodTaskAsync()
{
  // Await OperationResult<bool> and return result value.
  return await SomeMethodAsync();
}
```

The opposite is true as well as demonstrated in the following example.

```
public OperationResult<bool> SomeMethodAsync()
{
  // Implicit conversion from Task<bool> to OperationResult<bool>
  return SomeMethodTaskAsync();
```

```
}
public Task<bool> SomeMethodTaskAsync()
{
  return Task.FromResult(true);
}
```

Explicit conversions

In cases where the compiler cannot implicitly convert between OperationResult and Task, we can help it along with explicit conversions as demonstrated in the following example.

```
public Task<bool> SomeMethodAsync()
{
  return Task.FromResult(true);
}
public OperationResult LegacyCoroutine()
{
  return Coroutine.Start(LegacyCoroutineCore).AsOperationResult();
}
public IEnumerable<INotifyCompleted> LegacyCoroutineCore()
{
  // Explicit conversion to OperationResult<bool> required
  yield return SomeMethodAsync().AsOperationResult();
}
```

Backwards-compatible extension methods

Last but not least, both the "Punch Compatibility Pack" and "DevForce Compatibility Pack" provide a series of extension methods with backwards-compatible method signatures. Despite these extension methods, minor code changes are unavoidable. In order to disambiguate the legacy methods from the TAP methods, certain parameters that used to be optional are no longer optional and must be provided with a value or null.

Consider the following example.

```
public IEnumerable<INotifyCompleted> UpdateCompanyName()
{
   var provider = new EntityManagerProvider<EntityManager>();
   var uow = new UnitOfWork<Customer>(provider);
  // Retrieve customer
  var id = new Guid( "ed99343d-b368-4007-90a2-48ccf2699d44");
   OperationResult<Customer> operation;
  yield return operation = uow.Entities.WithIdAsync(id); // Implicit conversion to OperationResult<Customer>
  // Update company name
  operation.Result.CompanyName = "Test" ;
  // Commit changes, using legacy extension method instead of explicit conversion
  // yield return uow.CommitAsync().AsOperationResult();
  yield return uow.CommitAsync(null, null);
}
```

Await support for DevForce operations

As with OperationResult, the "Punch Compatiblity Pack" enables *await* for DevForce legacy operations as demonstrated in the following example.

```
// Legacy method returning DevForce EntityQueryOperation from IdeaBlade.EntityModel.Compat
private EntityQueryOperation<Customer> GetCustomersAsync()
{
   var query = new EntityManager().GetQuery<Customer>();
  return query.ExecuteAsync(null);
}
public async Task<IEnumerable<Customer>> GetCustomersTaskAsync()
{
  return await GetCustomersAsync();
}
```

Other breaking changes

The following enhancements require code changes when migrating from Cocktail 2010 to Punch.

- **FrameworkBootstrapper** was replaced by **CocktailMefBootstrapper**. For more info, visit the Boostrappping topic.
- Due to significant platform differences introduced by Windows 8 Store apps, the Punch Composition API was generalized and now architecturally supports alternative IoC implementations. For more info, visit the API documentation.
- In order to provide a consistent UI navigation API across all platforms supported by Punch, the NavigationService class was replaced with the new Navigator class. For more info, visit the Navigation topic.
- For DevForce breaking changes see Migrating from DevForce 2010.