

## Contents

- [Coordinating entity definitions](#)
- [Linking](#)
  - [Automatic linking](#)
  - [Manual linking](#)
- [Preserving differences deliberately](#)

This topic describes how to construct a non .NET client application entity model project by linking to source code files in a full .NET project ... and why you'd want to do that. The discussion applies both to entities [generated from an EDM](#) and entities coded by hand in "Code First" style.

We can't use .NET entity classes in environments such as Silverlight, Windows Store and Windows Phone. While written in the same .NET languages, they don't share the same supporting libraries. While they both have libraries with the same names and the same functions, these are not the same assemblies. These applications can't consume the entity classes in your full .NET entity model.

This fact causes serious pain. When you are responsible for building an entire data-driven, business application, back to front, from server to client, you want one implementation of each conceptual entity. You don't want two versions: one on the client and one in full .NET.

## Coordinating entity definitions

You could rely on code-generation magic to automatically write and rewrite the sort-of-similar-but-not-really Silverlight versions of the full .NET entities. This is the approach taken by WCF RIA Services.

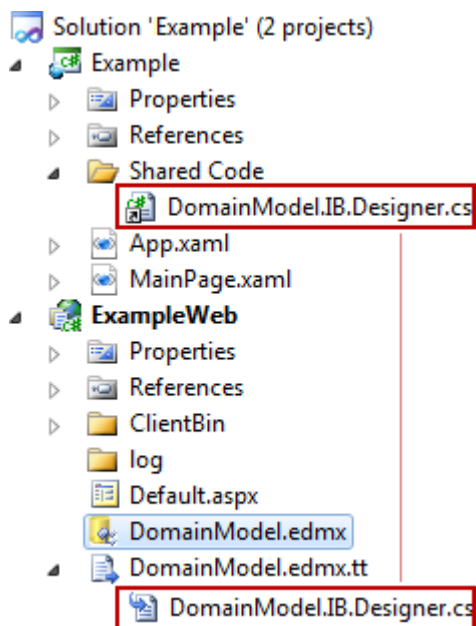
The DevForce takes a different tack. DevForce strives for perfect fidelity between client and .NET class definitions by ensuring that the source code is the same source code for entities compiled on both sides.

That guarantee is made possible by link items in the client model project that point to corresponding entity source code files in the .NET model project. Changes to an entity class file, whether made from within the client project or from within the .NET project, are actually changes to the same physical source code file.

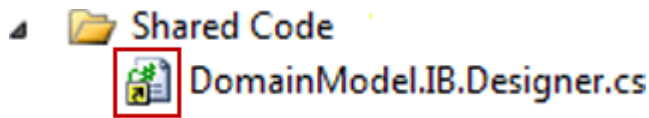
When the projects are (re)compiled, both client and .NET entity classes are built from the same source code. The resulting classes are as identical as they can be.

## Linking

This screenshot shows generated entity class source files in a web application project linked to the Silverlight application project:



Notice the shortcut symbol on the Silverlight project item name indicating that the item is a link.



### Automatic linking

DevForce will automatically generate a link to your auto-generated model and developer classes, using the following rules:

1. If only a single Silverlight project is in the solution, it will be the link target.
2. Assembly name "matching" is done: DevForce will look for the first Silverlight project (in assembly name order) whose assembly name starts with the server project's "root" assembly name. This "root" name is the server project's assembly name with any trailing "Web" or ".Web" removed. For example, if the server project assembly name is "MyApplication.Web", DevForce will look for a Silverlight project whose assembly name starts with "MyApplication".
3. If you've manually created or moved a link DevForce will honor that link and not remove or modify it.

This automatic linking logic is also performed for Windows Store and Windows Phone application types.

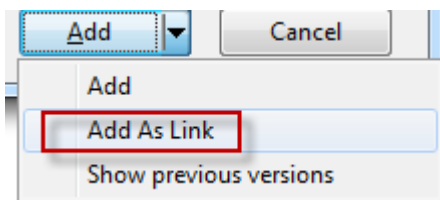
If a target project can't be found then a message will be written to the output window. You can still manually create the link, as discussed below.

### Manual linking

When DevForce can't determine the project to be linked, or you have additional files to be linked, the developer must *manually* link the source code files - such as [partial class files](#) - that he creates on his own. You create the link - once - by doing the following:

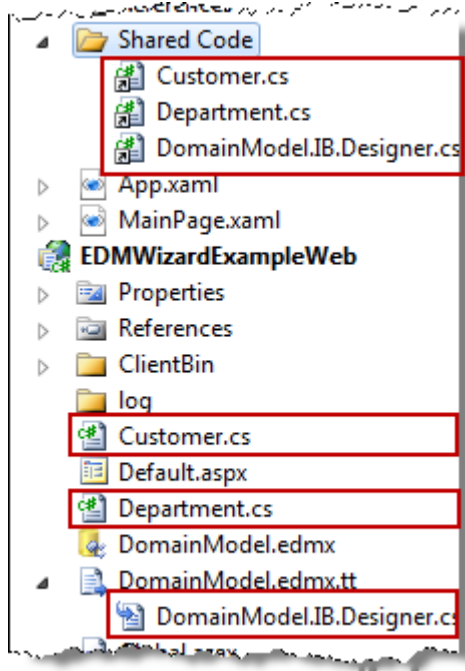
- **Select** the client project
- **Right-mouse-click** and select **Add | Existing item...** (or press **Ctrl-Shift-A**)
- **Navigate** to the model project's directory
- **Scatter select** the code files to link (hold down Ctrl key while clicking each file)

**Do not click the Add button.** To the right of the Add button is a drop-down arrow; click that to reveal "Add" menu choices



#### Pick "Add As Link"

The selected files are now linked in the client application as we see here with *Customer* and *Department* partial class files.



## Preserving differences deliberately

Sometimes you have business logic that should only execute on one side or the other. DevForce makes full type fidelity **possible** but it doesn't **mandate** it.

You could use compiler directives within a source code file to include or exclude statements that only apply in Silverlight, Windows Store or .NET. The compiler directive defined in all Silverlight projects is "SILVERLIGHT". Windows Store projects all define "NETFX\_CORE" to indicate the environment. Windows Phone uses both "WINDOWS\_PHONE" and "SILVERLIGHT" directives.

A cleaner and easier way is to create additional partial class files, either in the .NET project or in the client project, and deliberately **not link** them.

It's a good idea to give the files special names that announce your intention and prevent confusion. IdeaBlade uses the ".Desktop" and ".SL" extension conventions for this purpose. *Customer.Desktop* and *Customer.SL* are suggested names for *Customer* class files containing code to run only in full .NET or only in Silverlight, while *Customer.WinRT* can indicate a Windows Store application source file.