

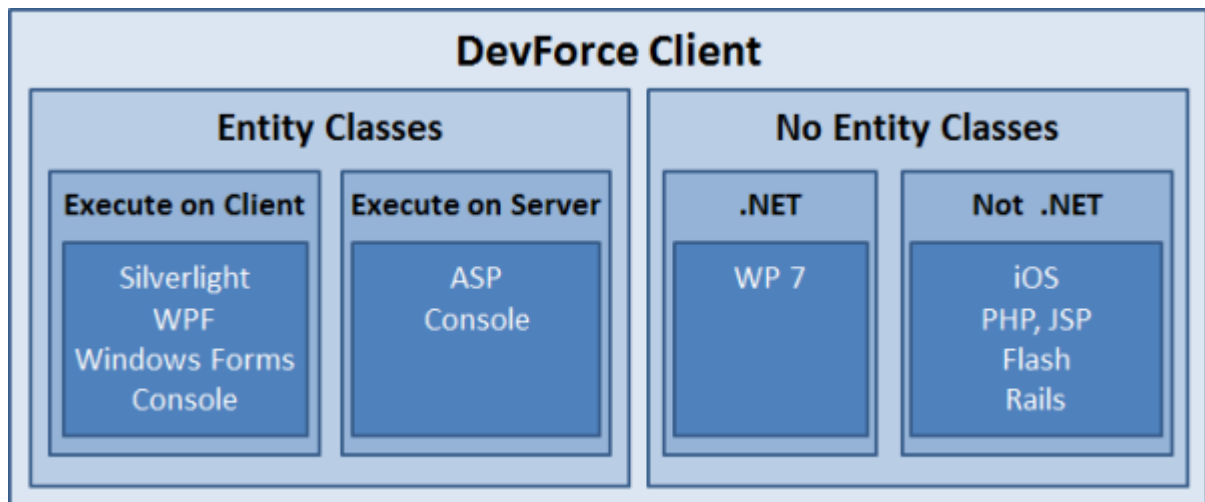
Contents

- [With entity classes](#)
 - [Executing on the client](#)
 - [Executing on the server](#)
 - [Tiered deployment options](#)
- [Clients without entity classes](#)

Any code that makes requests of the DevForce [EntityServer](#) is a DevForce **client**. This topic categorizes the diversity of client types and explores some of their differences.

DevForce clients come in many flavors, each with their own characteristic experiences and development patterns. Some clients use DevForce heavily; others consume DevForce-backed server resources without any awareness of DevForce itself. The factor that most influences where a client falls on this spectrum is the technology with which it is built.

The following diagram illustrates that point:



With entity classes

The most fundamental distinction is whether or not the client technology can support .NET entity classes.

The DevForce mission is to enable easy development of rich, data-driven internet applications by offering a unified programming model. Perhaps the most valuable aspect of DevForce programming is the ability to use the same entity model classes on both client and server. Accordingly, most DevForce applications are deployed to .NET client that can support the entity class model and use the [EntityManager](#) to coordinate communications with the server.

The .NET clients in this group split into two groups: those which execute on the client machine and those which execute on a server.

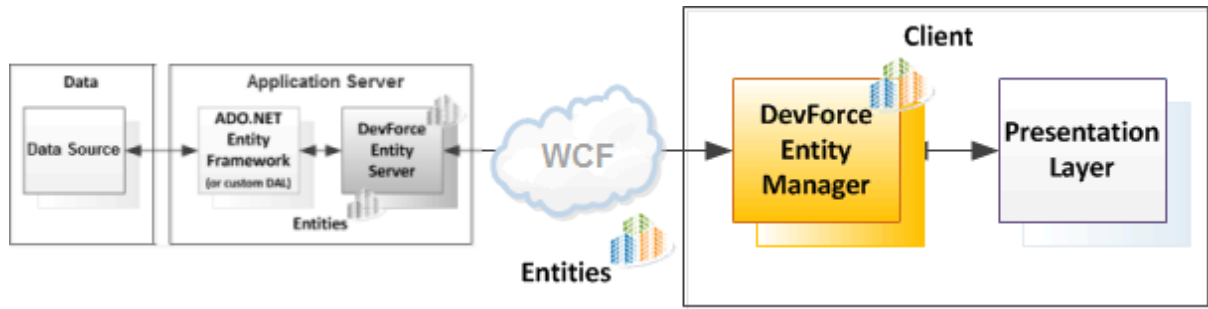
Executing on the client

The typical DevForce client is a smart .NET UI such as a Silverlight, Windows Store, Windows Presentation Foundation (WPF), or Windows Forms application. It could even be a command-line console application. These are technologies that execute .NET code on the client machines themselves.

Many of these applications are semi-autonomous, running most of their code on the local machine rather than on the server. They are often capable of running disconnected from the server for hours, days or weeks.

Such clients tend to split their code into a presentation layer and a model layer. The presentation layer manages the user experience – what the user sees and does. Some of what the user sees is information held in [entities](#) – the same entities found on the [application server](#).

In a DevForce application, the presentation layer delegates all matters relating to entities to the model layer. The primary interface to the model layer is the DevForce [EntityManager](#), as shown in this diagram.



In this way, the presentation layer is relieved of many distracting details. It doesn't know and shouldn't know how the model layer gets and saves entities. It simply makes requests of the *EntityManager*. When it needs entities, it asks the *EntityManager* for them. If the user changes information in the presentation that should be reflected in the entities, the *EntityManager* remembers that the changes are pending. When the user wants to save the changes, the *EntityManager* gathers the changed entities in a change set and sends them to the server.

Please look [here](#) for an overview of this development style.

Executing on the server

Sometimes the "client" of a DevForce Application Server executes on a server rather than an end user machine. You could write a service that reads, calculates, updates the database using the same entity model classes, the same *EntityManager*, the same techniques as you would when writing a remote UI client. It's just a "service client" instead of a "UI client". The pertinent difference is where you host it; you host a "service client" on a server.

With DevForce you have the full range of deployment options. You could host in IIS, as a console application, or as a Windows Service. You could run it on the same box as the Application Server, or a different box, or in the cloud.

The most familiar example of a server-side DevForce application is the ASP.NET client. Whether its ASP.NET Web Forms or ASP.NET MVC, the code that handles end-user requests and prepares HTML responses runs on a server. That code can use the same facilities - the entity classes, the *EntityManager*, change-set management - as a remote Silverlight or WPF client. The presentation layer is different, the interaction demands are different, but the exchanges between presentation and model layers can be the same. The similarities are advantageous in mixed-mode applications that present to external users in HTML and to internal workers with WinForms, WPF, or Silverlight UIs. DevForce offers a common foundation across .NET client technologies.

Which is why the [discussion of remote client development style](#) applies equally to server-side clients.

Tiered deployment options

We're usually thinking about multi-tiered applications deployed to at least 3 tiers

1. A data tier consisting of a database running on its own server.
2. A middle tier running the DevForce *EntityServer* on its own server.
3. A client with entity classes governed by an *EntityManager* executing either on a remote client machine or (as with ASP) on a web server.

In fact, you can run all three layers - data, middle, and client - in a single process on a single client machine!

DevForce offers a two-tier mode in which the *EntityServer* and your client application (with the *EntityManager*) execute in the same process. While even SQL Server Express runs in a separate process, you can configure your application to use a simulated database, the DevForce in-memory [fake backing store](#). Now all three layers are running in the same process.

The two-tier mode is the preferred choice for most non-Silverlight development work because it eliminates the complexities and hassles of depending upon external software, network, and hardware that are irrelevant to the client application itself. Of course you should design for and repeatedly test the application as you intend it to be deployed. But you can save time, complexity, and money by developing two tier, with or without the fake backing store.

Simple changes to your [configuration](#) are all it takes to switch between a two-tier development environment and an n-tier test, stage, or production environment.

Clients without entity classes

A non-.NET technology obviously cannot make use of .NET classes. Non-.NET technologies can consume DevForce services hosted on a [DevForce Application Server](#) but they cannot work directly with the entity classes themselves.

Apple's iOS devices (iPhone and iPad), **jsp** web pages, **Flash/Flexx** applications, and **Ruby-on-Rails** applications all belong to this camp.

On these platforms, data from the server are reshaped on the client to suit whatever the technology prefers; they might appear as objects or data records or XML. [OData](#), as implemented with [WCF Data Services](#), is the primary vehicle for exchanging data with the Application Server. The application may benefit from using a WCF Data Services programming model generated for its client environment if such a model exists (it does for iOS).

Windows Phone 7 (WP7), although a Silverlight.NET platform, is in the same boat at the present time. WP7 is missing a few of the Silverlight 4 types that are essential to a DevForce client. The absence of *IQueryable* is perhaps the most critical. Meanwhile, some of our customers have released WP7 applications (and iOS) applications written using the OData approach.