Proper .NET DLL structure for app

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 2k times

1

...if there is such a thing. Here's an image of two approachs for structuring DLLs/references in a .NET application: http://www.experts-exchange.com/images/t80668/compArch.png. The app can be a website (it is in this case) or a winform. Each box represents a DLL. For the winform app, just replace "webcontrols" with "winformcomponents".

The first (top) image is what I like. You might want to extend "some" of the base web controls and directly use others. The 2nd image makes you extend any web controls via interface. To me that seems overkill since you may want to simply use what is already there without modification. Which is better and what are the advantages/disadvantages?

The first image puts the lowest common constructs(exceptions, fileIO, constants, etc) into a common.dll. The 2nd image puts app business logic and common into one DLL. Which is better and what are the advantages/disadvantages of each apporach?

c# .net asp.net architecture

Share

Improve this question

Follow

asked Nov 23, 2008 at 18:21

4thSpace

44.3k ● 100 ● 305 ● 491

2 Answers

Sorted by: Highest score (default)

1

Having lots of references is usually bad because loading DLL's has a non-negligble cost. It's not as elegant perhaps, but having fewer modules improves your performance. As so often in our craft, you have to find the balance between elegance of total modularization and the harsh reality of performance. And as usual in our craft, you won't know what balance is until you start profiling to measure the performance of your application.

Share Improve this answer

Follow

answered Nov 23, 2008 at 19:19

John Källén

7,923 ● 35 ● 67

0

I think this is a really down to programmer preference.

It all boils down to dependencies really. More things in one DLL means it will naturally create many more dependtdents on that DLL.

I personally tend to follow along similar lines to the MS structure, for these reasons:

- It makes it easier for newcomers to the custom "framework" to find what they want (e.g.

`CompName.Web.UI` and `CompName.Data`.

- It helps reduce the dependencies to "obvious" choices. I am not too keen on `CompName.Common` type DLL's because it does not clearly indicate possible dependents, whereas `CompName.Web.UI` suggests that it is likely to be used by any web apps.
- Obvious size reduction, since DLL content will be more "relevant".

DLL's for tiers within an app make sense, the types within should only be those types required by the business model, other objects (such as utility, data access etc.) should be in their own libraries.

answered Nov 23, 2008 at 19:06

Rob Cooper

28.9k ● 26 ● 105 ● 142