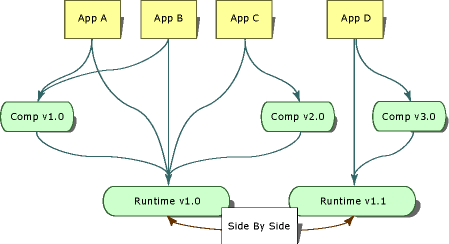
NET Framework 4

**Side-by-Side Execution**

Side-by-side execution is the ability to run multiple versions of an application or component on the same computer. You can have multiple versions of the common language runtime, and multiple versions of applications and components that use a version of the runtime, on the same computer at the same time.

The following illustration shows several applications using two different versions of the runtime on the same computer. Applications A, B, and C use runtime version 1.0, while application D uses runtime version 1.1.

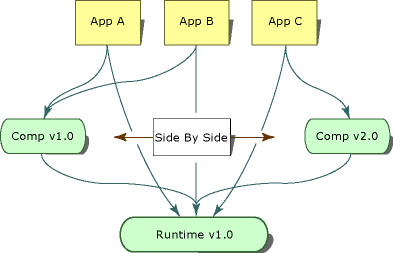
**Side-by-side execution of two versions of the runtime**



The .NET Framework consists of the common language runtime and approximately two dozen assemblies that contain the API types. The runtime and the .NET Framework assemblies are versioned separately. For example, version 1.0 of the runtime is actually version 1.0.3705.0, while version 1.0 of the .NET Framework assemblies is version 1.0.3300.0.

The following illustration shows several applications using two different versions of a component on the same computer. Application A and B use version 1.0 of the component while Application C uses version 2.0 of the same component.

**Side-by-side execution of two versions of a component**



Side-by-side execution gives you more control over which versions of a component an application binds to, and more control over which version of the runtime an application uses.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifBenefits of Side-by-Side Execution

Prior to Windows XP and the .NET Framework, DLL conflicts occurred because applications were unable to distinguish between incompatible versions of the same code. Type information contained in a DLL was bound only to a file name. An application had no way of knowing if the types contained in a DLL were the same types that the application was built with. As a result, a new version of a component could overwrite an older version and break applications.

Side-by-side execution and the .NET Framework provide the following features to eliminate DLL conflicts:

* Strong-named assemblies.

Side-by-side execution uses strong-named assemblies to bind type information to a specific version of an assembly. This prevents an application or component from binding to an invalid version of an assembly. Strong-named assemblies also allow multiple versions of a file to exist on the same computer and to be used by applications. For more information, see [Strong-Named Assemblies](http://msdn.microsoft.com/en-us/library/wd40t7ad.aspx).

* Version-aware code storage.

The .NET Framework provides version-aware code storage in the global assembly cache. The global assembly cache is a computer-wide code cache present on all computers with the .NET Framework installed. It stores assemblies based on version, culture, and publisher information, and supports multiple versions of components and applications. For more information, see [Global Assembly Cache](http://msdn.microsoft.com/en-us/library/yf1d93sz.aspx).

* Isolation.

Using the .NET Framework, you can create applications and components that execute in isolation. Isolation is an essential component of side-by-side execution. It involves being aware of the resources you are using and sharing resources with confidence among multiple versions of an application or component. Isolation also includes storing files in a version-specific way. For more information about isolation, see [Guidelines for Creating Applications and Components for Side-by-Side Execution](http://msdn.microsoft.com/en-us/library/z5e12zb4.aspx).

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifVersion Compatibility

Versions 1.0 and 1.1 of the .NET Framework are designed to be compatible with one another. An application built with the .NET Framework version 1.0 should run on version 1.1, and an application built with the .NET Framework version 1.1 should run on version 1.0. Note, however, that API features added in version 1.1 of the .NET Framework will not work with version 1.0 of the .NET Framework. Applications created with version 2.0 will run on version 2.0 only. Version 2.0 applications will not run on version 1.1 or earlier.

Versions of the .NET Framework are treated as a single unit consisting of the runtime and its associated .NET Framework assemblies (a concept referred to as assembly unification). You can redirect assembly binding to include other versions of the .NET Framework assemblies, but overriding the default assembly binding can be risky and must be rigorously tested before deployment.