

## XGantt in projects for the target platform AnyCPU

### Preliminary remarks:

The .NET editions of the NETRONIC- X components exist for the x86 and x64 platforms.

Our X component assembly can only be integrated into the VisualStudio toolbox from the installation directory in the x86 variant. The x64 variant does not work here.

Mounting our X-component on the form (e.g. by dragging it) automatically inserts a reference to this 32bit variant of our assembly in the application.

When developing an application, the question arises for which target platform to develop. In principle, x86, x64 and AnyCPU are available for selection here.

Depending on the selected target platform, different scenarios have arisen so far:

x86: The application can be compiled and executed without any problems.

x64: The application compiles without problems, but crashes with a BadImageFormat exception when started via debugger. When started without the debugger, nothing happens.

AnyCPU: In the default setting the application can be compiled and executed without problems. However, this only works if the project property Prefer 32Bit is enabled, which ensures that a 32Bit process is started. In terms of content, this setting is equivalent to x86. If Prefer 32Bit is deactivated, compilation is also possible without any problems. But at startup a 64Bit process is started, so that the debugger again throws a BadImageFormat exception. Without debugger again nothing happens.

For the application to function correctly, it is necessary to load the correct variant of our X component assembly.

This can be done dynamically at runtime by interfering with the .NET assembly loading mechanism. At this point it is assumed that this loading mechanism is known (otherwise see <https://docs.microsoft.com/de-de/dotnet/framework/deployment/how-the-runtime-locates-assemblies>).

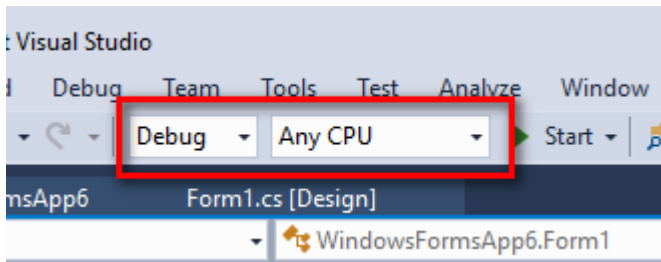
Basic procedure: It is ensured that our X component assembly is NOT found in the usual way including the global assembly cache. So it must neither be found via XCopy deployment nor installed and found via GAC. This activates an additional sub-item of the loading mechanism, with which it is possible to programmatically load an assembly and pass it into the still running loading mechanism. This then continues to work normally.

The following instructions describe this procedure using a new C# project as an example. Existing projects can be modified accordingly.

## Instruction:

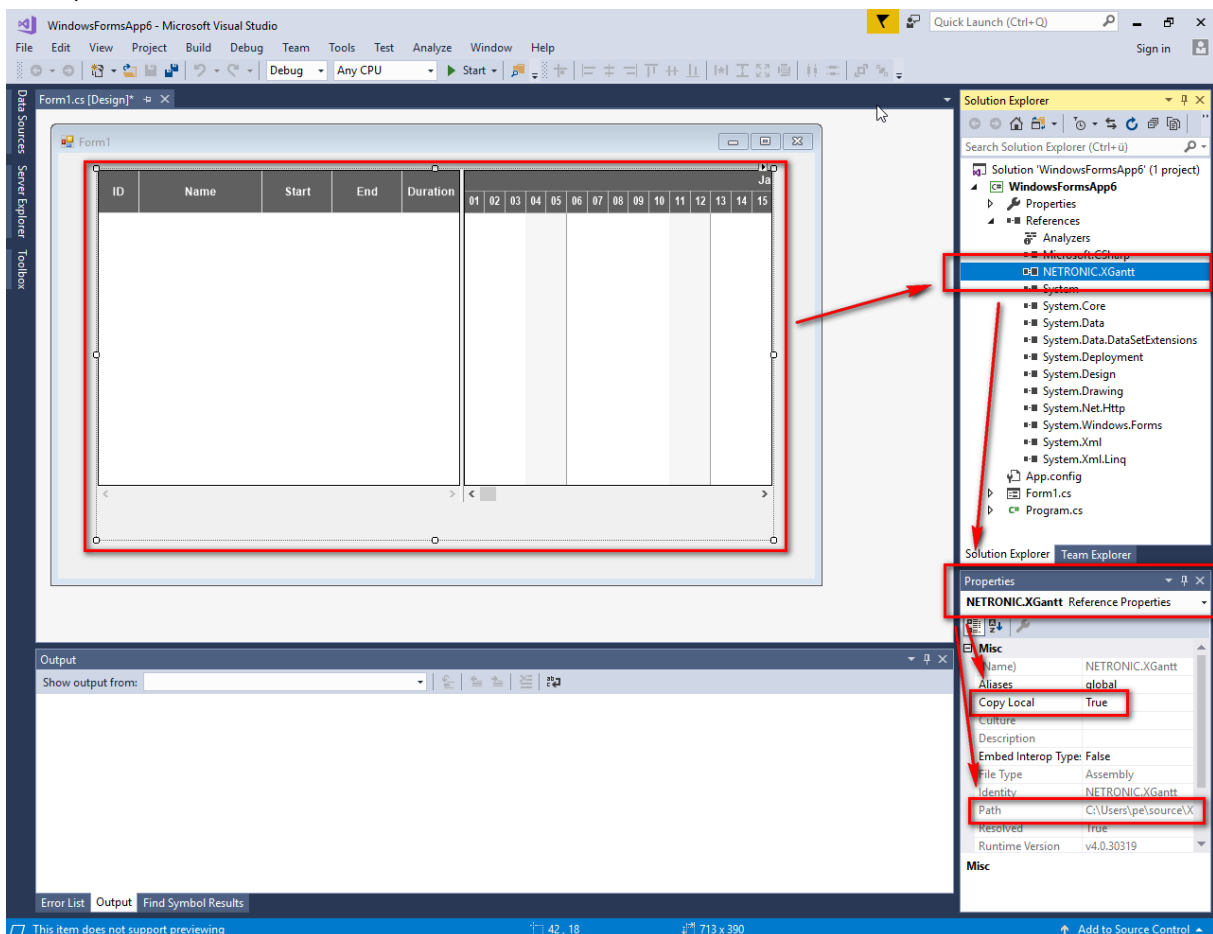
Start VisualStudio and create a new C# project (Windows Forms App) .

By default, the configuration is Debug and the target platform is AnyCPU.



If you have not already done so, paste the 32bit variant of XGantt from the installation directory into the toolbox.

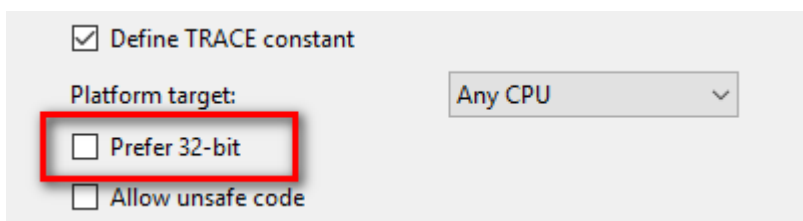
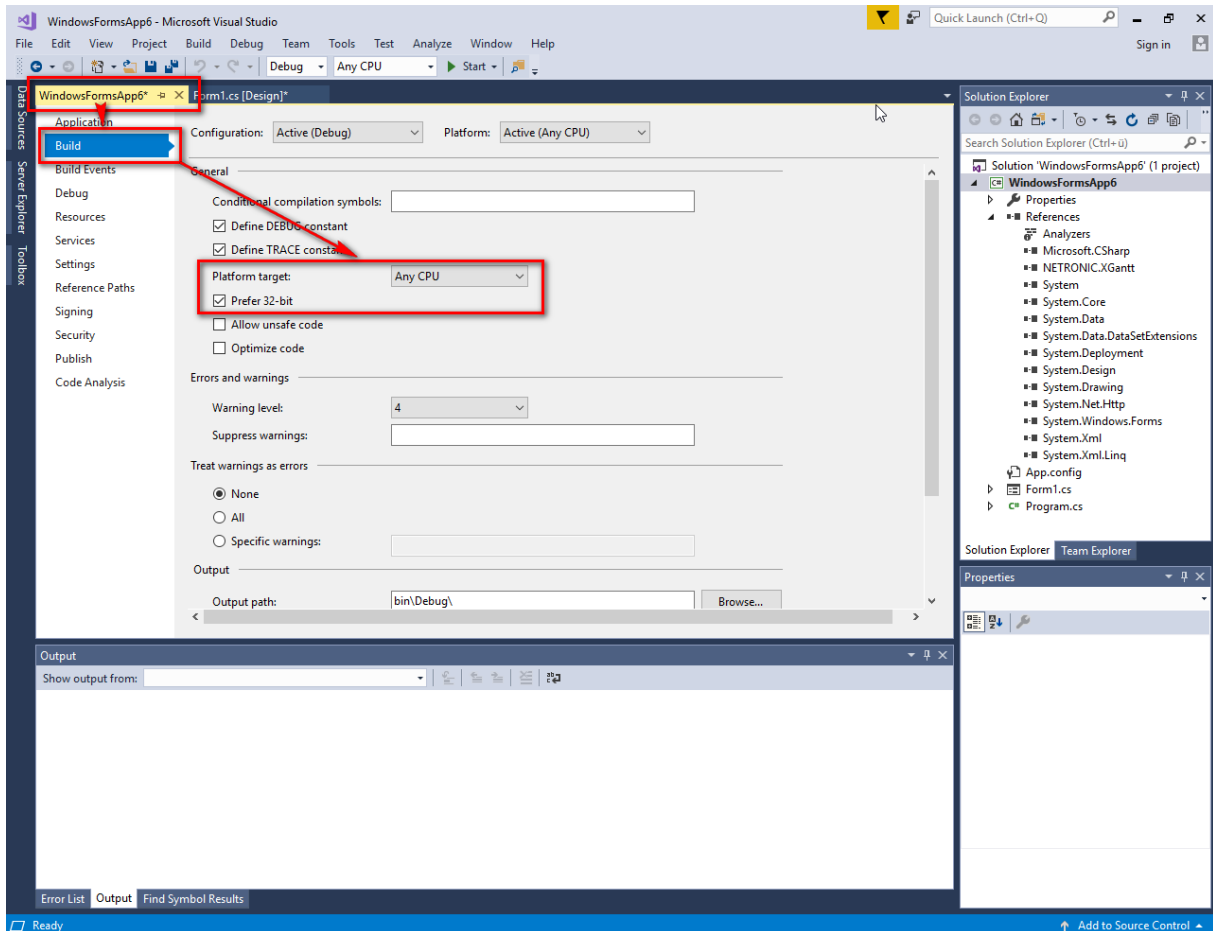
The X component can now be assembled from the toolbox, e.g. by dragging it onto the mold. In doing so, a reference to the assembly (i.e. to the 32bit variant) is automatically inserted into the application. The properties of the reference are displayed in the Properties window. Here you can see the installation path. You can also see the default activation of XCopy deployment ("Copy local: True").



This must be deactivated. To do this, set the Copy Local setting in the Properties window to False.

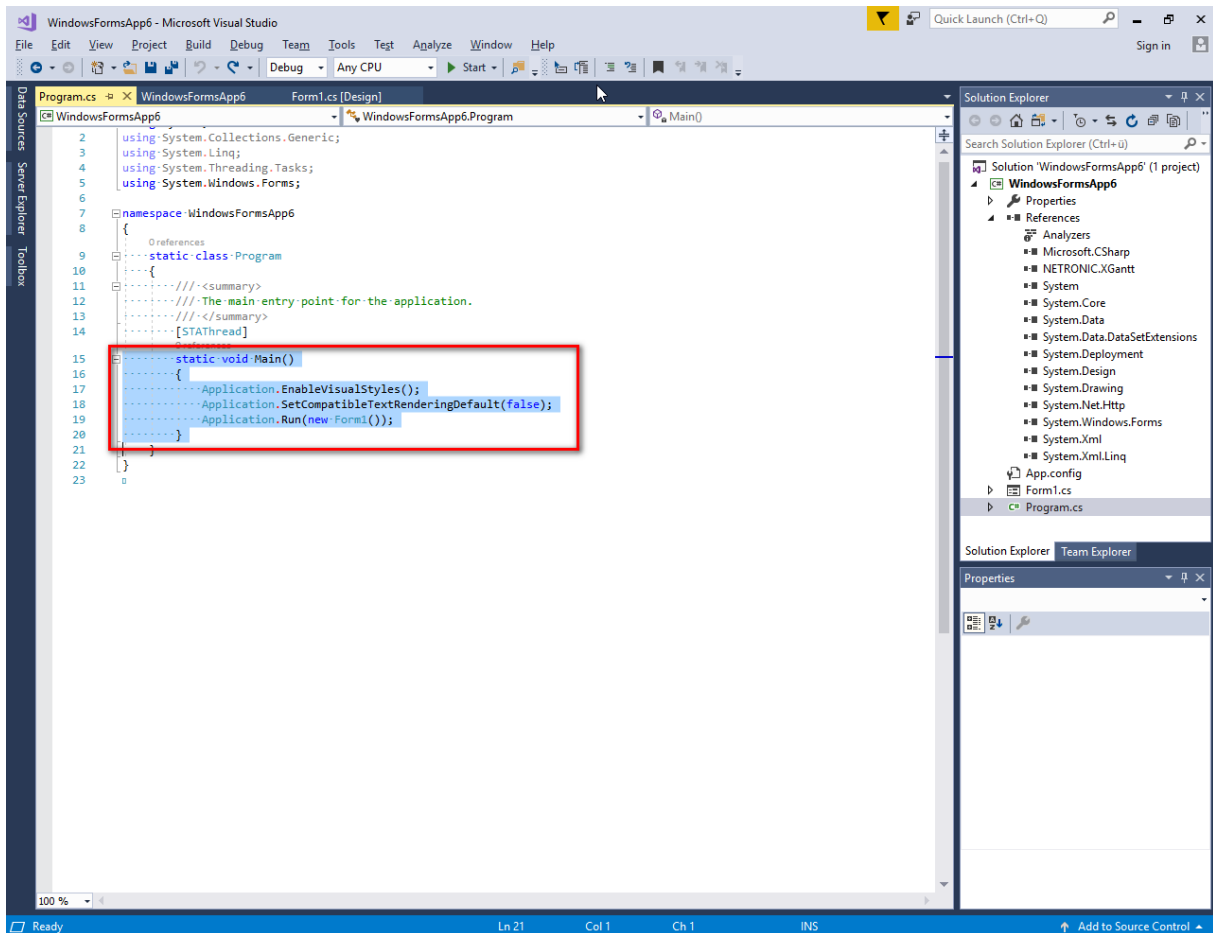
Now the properties of the application have to be changed.

On the Build tab of the project properties, settings for the currently selected configuration are displayed (here Debug). AnyCPU is selected under the Platform target setting. Also a 32bit process is preferred (Prefer 32-bit). This setting must be deactivated.



Please note that this change must be made individually for each configuration.

Now the intervention in the loading mechanism must be made in the Program.cs file. For this purpose, the part marked in the following is changed.



The new code is:

```
static void Main()
{
    System.AppDomain.CurrentDomain.AssemblyResolve += Resolver;

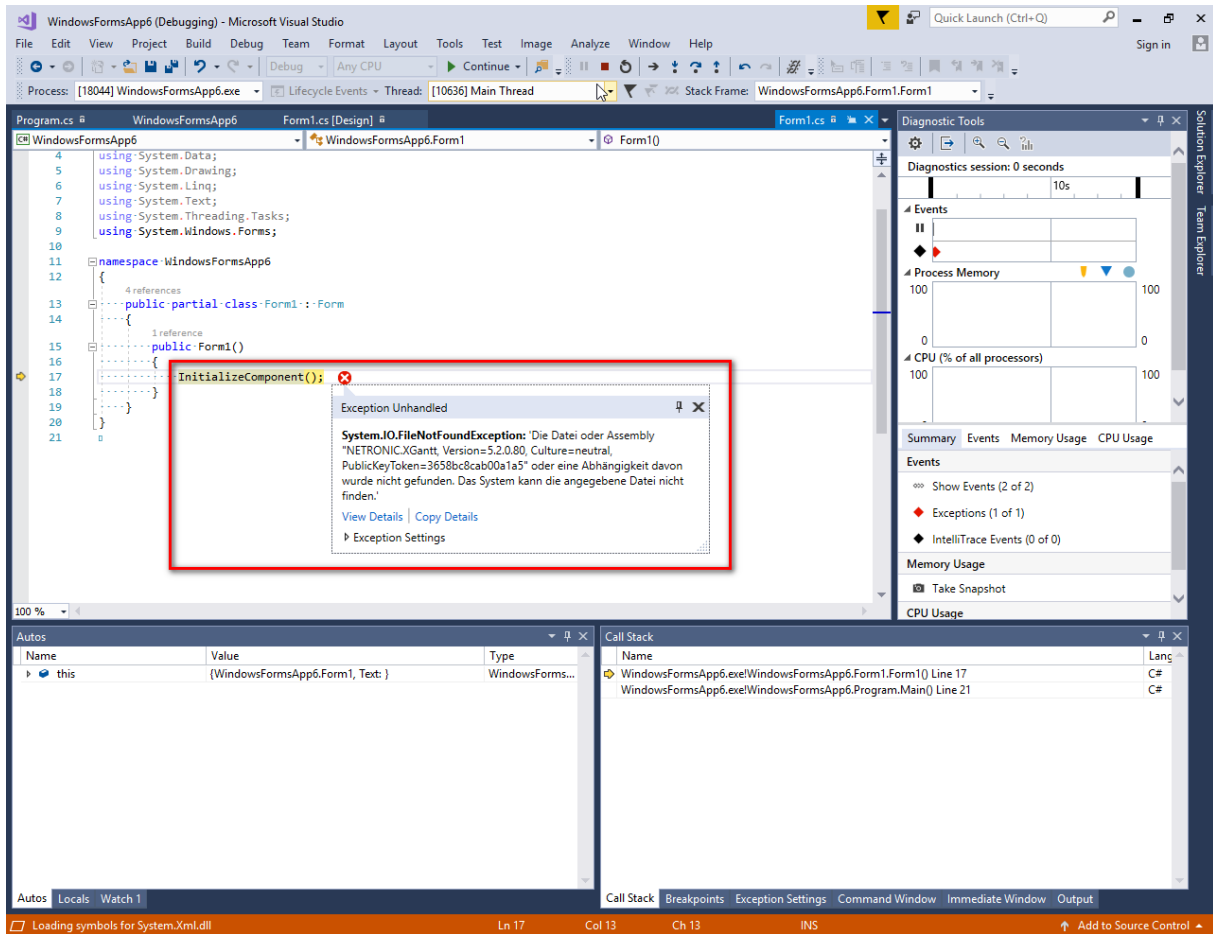
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}

/// Will attempt to load missing assembly from either x86 or x64 subdir
private static System.Reflection.Assembly Resolver(object sender, ResolveEventArgs args)
{
    if (args.Name.StartsWith("NETRONIC.X"))
    {
        string assemblyName = args.Name.Split(new[] { ',' }, 2)[0] + ".dll";
        string archSpecificPath = System.IO.Path.Combine(AppDomain.CurrentDomain.SetupInformation.ApplicationBase,
            Environment.Is64BitProcess ? "x64" : "x86",
            assemblyName);

        return System.IO.File.Exists(archSpecificPath)
            ? System.Reflection.Assembly.LoadFile(archSpecificPath)
            : null;
    }
    return null;
}
```



Now, when starting the application, a *System.IO.FileNotFoundException* occurs because our X-component assembly was not found by the loading mechanism and also the folders from which to load programmatically are missing.



The screenshot shows the Visual Studio IDE during a debugging session. The main window displays the source code for `Form1.cs` in the `WindowsFormsApp6` namespace. The code includes using statements for `System.Data`, `System.Drawing`, `System.Linq`, `System.Text`, `System.Threading.Tasks`, and `System.Windows.Forms`. A `public partial class Form1 : Form` is defined with an `InitializeComponent()` method. A red box highlights the `InitializeComponent()` method call in the code.

An "Exception Unhandled" dialog box is open, displaying the following error message:

```
System.IO.FileNotFoundException: 'Die Datei oder Assembly "NETRONIC.XGantt, Version=5.2.0.80, Culture=neutral, PublicKeyToken=3658bc8cab00a1a5" oder eine Abhängigkeit davon wurde nicht gefunden. Das System kann die angegebene Datei nicht finden.'
```

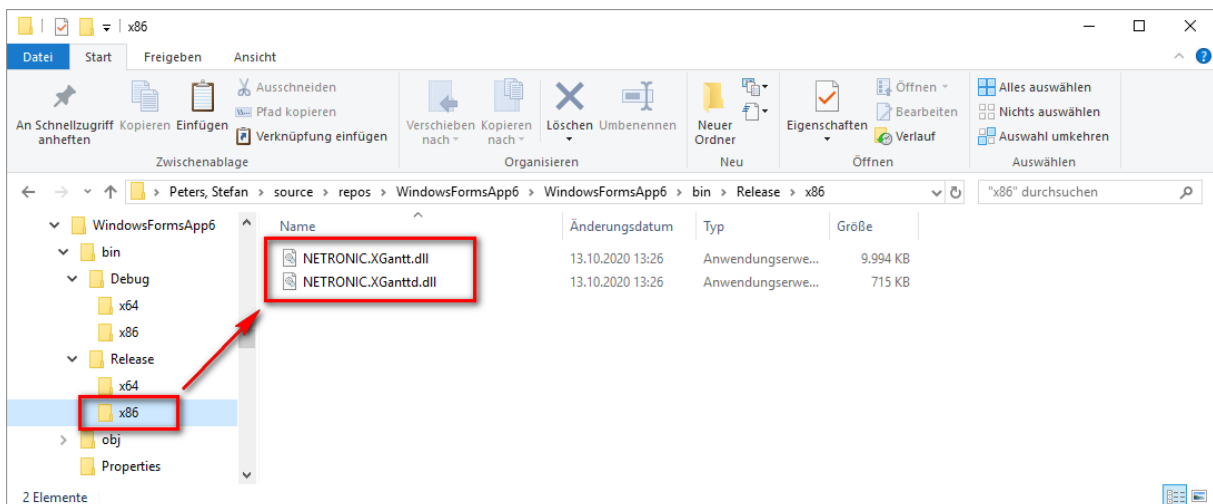
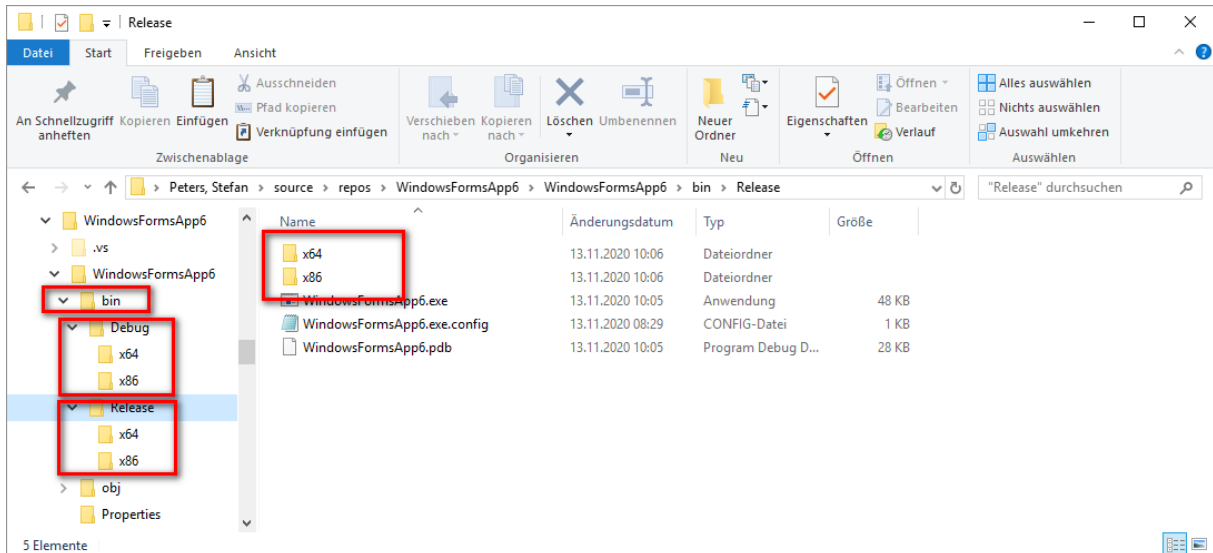
The "Autos" window shows the current object `this` of type `WindowsFormsApp6.Form1, Text`. The "Call Stack" window shows the call stack with the following entries:

Name	Value	Type
this	{WindowsFormsApp6.Form1, Text}	WindowsForms...

Name	Language
WindowsFormsApp6.exe!WindowsFormsApp6.Form1.InitializeComponent() Line 17	C#
WindowsFormsApp6.exe!WindowsFormsApp6.Program.Main() Line 21	C#

The "Diagnostics Tools" window on the right shows a summary of the diagnostic session, including events, process memory, and CPU usage. The "Events" section shows one exception.

These folders are now created under the application directory (for debug and/or release) and filled with our X component assemblies of the correct variants. As an example, the x86 and x64 directories were named. Note the corresponding code in the modified Program.cs file. Here naturally also another directory structure is conceivable.



Now that the directories have been created and filled, the application can be executed. Depending on the architecture of the started process (x86 or x64), the appropriate X component assembly is loaded automatically.

### Hint for .NET Core

If you have set .NET Core 3.1 as target framework, the folders x86 and x64 are not to be created under Debug and Release, respectively, but there is an intermediate level netcoreapp3.1, under which the folders are to be created. This then looks like this:

