Debugging Scripts Code with Microsoft Visual Studio

Debug and troubleshoot your scripts.

This document explains how to debug script code using the Microsoft Visual Studio environment instead of our internal debugging tools.

From FrameworX 10.1.6, the built-in script debugger in the Designer attaches to Script Tasks and Classes of a Multiplatform (.NET 10) solution when the Designer and the 64-bit runtime run on the same Windows x64 computer. See Code Editor Reference. On other computers, for example Windows on ARM64, the Designer does not offer the built-in debugger for Tasks and Classes of a Multiplatform solution. Use the procedure on this page when the built-in debugger is not available, or when you prefer Visual Studio.

In 10.1.5, the built-in debugger does not attach to Tasks and Classes of a Multiplatform (.NET 10) solution, and the procedure on this page is the recommended method for them.

However, the same process can also be used for Windows-only solutions.

Requirements
Debug Information
Procedure

Requirements

- Microsoft Visual Studio
- .NET 10 for a Multiplatform solution in FrameworX 10.1.5 and 10.1.6, or .NET Framework 4.8 for a Windows-only solution.

Debug Information

Before attaching Visual Studio, ensure the following conditions are met for each display or script you want to debug:

1. Debug Info checkbox: The Debug Info checkbox must be enabled in the display/script settings. This causes the output to be compiled as Debug. If it is not enabled, the output will be Release and debugging will not be possible.

2. User access permission: The currently logged-in engineering user must have access permission to the display or script. Even if the Debug Info checkbox is enabled, compilation will produce Release output if the user lacks the required permission.

3. Content Hidden plugin setting: If **Content Hidden** is checked on the plugin, compilation is always Release, regardless of the Debug Info checkbox or user permissions.

4. Attach Requirement: To attach an external Visual Studio debugger, ALL scripts and/or displays must be compiled as Debug, no exceptions.

- **Server-side debugging (TServer / TRunModule):** Every ScriptTask and ScriptClass with Domain = Server must be compiled as Debug.
- **Client-side debugging (TRichClient):** Every ScriptTask, ScriptClass, and Display must be compiled as Debug.

> **Scripts and Displays**
>
> A single script or display compiled as Release, even if not currently in use, will prevent the VS debugger from attaching. The debugger will be blocked or the RichClient will crash on attach attempt.

Procedure

1 - Run your solution

2 - Open **Microsoft Visual Studio**.

3 - Go to **Debug > Attach to Process...**.

4 - In the list, select the relevant **Process** (see guidance below).

5 - Click **Attach**. Visual Studio is now attached to the running process.

6 - In **Solution Explorer**, you will see a tree of multiple components, for example:

If you do not see the script you need, you must first trigger/run that script in the application. Once it is executed, it will appear in the list.

```
Solution
+-- External Sources
    +-- Modules without sources
        |-- Anonymously Hosted DynamicMethod
        |-- Class_0_638948088211382802_0
        |-- Class_2_639003032851421397_0
        |-- Flee.dll
        |-- MessagePack.Annotations.dll
        +-- MessagePack.dll
```

In this example, we are debugging a **Class**, so you can see `Class_0` and `Class_2`.
The number corresponds to the Class ID. To confirm the mapping, go to **Scripts > Class** and check the **ID** column.

7 - Right-click the target class (e.g. `Class_2_639003032851421397_0`) and select **Decompile Source to Symbol File**.

8 - The class will now be moved under **External Sources**, and the tree will look similar to:

```
Solution
+-- External Sources
    |-- Class_2_639003032851421397_0
    |   +-- Scripts
    |       +-- Class_Class1.cs
    +-- Modules without sources
        |-- Anonymously Hosted DynamicMethod
        |-- Class_0_638948088211382802_0
        |-- Flee.dll
        |-- MessagePack.Annotations.dll
        +-- MessagePack.dll
```

9 - You can now open the decompiled .cs file, set breakpoints, and debug the code in real time.

Process

- If your solution is **multiplatform**, you must attach to the `dotnet.exe` process. You will likely see multiple processes with this name, so use the **Command Line** column to identify which one corresponds to the script you want to debug.
- If your solution is **Windows-only**, you must attach to the `TRunModule.exe` process. Again, there may be multiple instances, so check the **Command Line** to determine which one is associated with the script.

If the script module is running inside the **TServer** process, you must attach to `TServer.exe` instead of the processes mentioned above. Check that in Solution > Settings.

- If the script is running on the **client side**, you must attach to the `TRichClient.exe` process.

Process example for a Multiplatform solution. In FrameworX 10.1.5 and 10.1.6, the Multiplatform runtime files are in the `net10.0` folder:

```
"<processName>" "<ProductPath>\fx-10\net10.0\TRunModule.dll" /module:t.modules.Script /ip1:127.0.0.1 /port1:
3101 /connectiontimeout:60 /profile:0
```

In this section...
