Scripts References Reference

Review object references from scripts.

Script References (Reference) enables adding external DLLs (Dynamic Link Libraries) to extend the capabilities of scripts and display codes in your solution.

Framework Compatibility:

- **Windows/WPF**: .NET 4.8 or .NET Standard 2.0
- **MultiPlatform/Web**: .NET 10 (FrameworX 10.1.5 and later; .NET 8 on 10.1.4 and earlier) or .NET Standard 2.0

Script References allow you to:

- Add external .NET assemblies to your solution
- Extend functionality beyond built-in namespaces
- Use third-party libraries in Scripts and Displays
- Share custom components across solutions

Configuring References
Using Referenced Assemblies
Namespace Declarations Dialog
Best Practices
Troubleshooting

Configuring References

FrameworX includes many .NET namespaces by default. Script References are needed only when you require additional external assemblies not included in the platform.

Assemblies Referenced by Default

The assemblies available to scripts without a Script Reference depend on the script target:

Script target	Referenced by default
All targets	FrameworX libraries (T.Library, T.Kernel, T.Modules, T.Toolkit), Newtonsoft.Json, Humanizer
Server, .NET Framework 4.8 (Windows solutions)	.NET Framework core assemblies (System, System.Core, System.Data, System.Xml, System.Drawing, System.DirectoryServices, System.ServiceModel, System.Web, and others), plus System.ServiceProcess, System.Management, MailKit, MimeKit, BouncyCastle, and Google APIs
Server, .NET 10 (Multi-Platform solutions)	The .NET Standard 2.0 API set, plus MailKit, MimeKit, BouncyCastle, Google APIs, and the MQTT Sparkplug B and InfluxDB utility libraries. Windows-only assemblies, such as System.Management, are not included.
Client, Rich Client and Smart Client	.NET Framework core assemblies and WPF assemblies (PresentationCore, PresentationFramework, WindowsBase, System.Windows.Forms, and others)
Client, HTML5	The .NET Standard 2.0 API set and the OpenSilver libraries

Any assembly not referenced by default for the script target must be added as a Script Reference, including assemblies that are part of .NET. For example, `System.Management` (WMI) is available by default only to Server scripts on .NET Framework 4.8. To use it in Server scripts on .NET 10 or in Client scripts, add `_ProductPath_System.Management.dll` as a Script Reference with the matching **Target Domain**. System.Management works only on Windows: in a Multi-Platform solution running on Linux, it compiles but fails at runtime.

To add a Script Reference:

1. Navigate to **Scripts** **References**
2. Click the **plus icon**
3. Browse and select the DLL file
4. Configure:
    - **Name**: Reference identifier
    - **Description**: Purpose of the library
    - **Target Domain**: Server, Client, ServerAndClient, or Disabled
5. Click **OK**

The new reference appears in the References table.

Configuration Properties

Property	Description

Target Domain	Where the DLL executes: **Server** (server-side scripts), **Client** (client-side scripts and Displays), **ServerAndClient** (both), or **Disabled** (not used in builds and not loaded)
NetAssemblyName	Reference name in the solution
DefaultNamespace	Primary namespace of the DLL
Resolved	Green check = the file was found and loaded. Red X = the file could not be loaded (hover for details). The value is saved with the solution and is refreshed only when the row is added or edited, or when you click **Verify References**. It is not rechecked at build time.
Portable	Indicates cross-platform compatibility
AssemblyPathRuntime	Runtime path where assembly loads from

Assembly Path Macros

Use these macros instead of absolute paths for portability:

Macro	Description	Typical Use
`_ThirdParty_`	MyDocuments/[ProductName]/ThirdParty	Server-side DLLs
`_WpfControls_`	[InstallPath]/WpfControls	Client-side controls
`_ProductPath_`	The version folder that contains `Designer.exe`. Default installation: `C:\Program Files\Tatsoft\FrameworX\fx-10\` (not `C:\Program Files\Tatsoft\` or `C:\Program Files\Tatsoft\FrameworX\`). If FrameworX was installed in another location, it is the `fx-10\` folder of that installation.	System components
`_SolutionPath_`	The folder that contains the `.dbsln` file. For `D:\Projects\Plant1\Plant1.dbsln`, it is `D:\Projects\Plant1\`.	Solution-specific DLLs
`_ExecutionPath_`	Working directory (usually solution folder)	Runtime files
`_GAC_`	Global Assembly Cache	Shared .NET assemblies

Folder macros already end with a path separator, so the rest of the path follows the macro directly, without a slash. Examples for a default installation and a solution saved as `D:\Projects\Plant1\Plant1.dbsln`:

AssemblyPathRuntime	Resolves to
`_ProductPath_References\MyLib.1.0.0\MyLib.dll`	`C:\Program Files\Tatsoft\FrameworX\fx-10\References\MyLib.1.0.0\MyLib.dll`
`_SolutionPath_References\MyLib.1.0.0\MyLib.dll`	`D:\Projects\Plant1\References\MyLib.1.0.0\MyLib.dll`

Everything after the macro is part of the path, including version folders. If the DLL is moved to a folder with a different name (for example, `MyLib.1.0.1` instead of `MyLib.1.0.0`), the reference no longer resolves. When you add a DLL with the **plus icon**, the Designer writes the macro automatically if the file is inside the installation folder or the solution folder, so re-adding the reference is the safest way to fix a wrong path.

Additional Macros

Macro	Description
`_ExecutionPathAndName_`	Working directory + solution name
`_ProgramFiles_`	Environment.SpecialFolder.ProgramFiles
`_ProgramFilesX86_`	Environment.SpecialFolder.ProgramFilesX86

`_SolutionName_`	Solution name without path/extension
`_SolutionPathAndName_`	Solution path + name (no extension)
`_Transfers_`	Default transfers folder

DLL Placement Best Practices

ThirdParty Folder

Location: `MyDocuments/[ProductName]/ThirdParty`

- Store server-side external DLLs
- Use macro: `_ThirdParty_`
- Domain: Server
- Usage: Script Tasks, Script Classes (Server)

Special Subfolder:

- `/SNMP` - Additional MIB files for SNMP driver

WpfControls Folder

Location: `[InstallPath]/WpfControls`

- Store client-side WPF control DLLs
- Use macro: `_WpfControls_`
- Domain: Client
- Usage: Displays, Code Behind

> **SmartClient Note**
>
> WPF controls require additional manifest configuration for SmartClient deployment. Controls created with T.Portable specification can run on Web Pages.

Multi-Engineering (Solution Server)
When editing a solution remotely via Solution Server, compilation runs on the client (Designer) machine, not on the server. The server only executes the compiled output. This means the DLL must be present on both machines: on the client so the Designer can compile, and on the server so the runtime can load it. Use path macros (such as _ThirdParty_ or _SolutionPath_) and place the DLL in the same relative folder on both machines. This ensures the reference resolves correctly in both environments.
The same applies to the .NET runtime: scripts compile to .NET Framework 4.8 in Windows solutions and to .NET 10 in Multi-Platform solutions (FrameworX 10.1.5 and later), so a Multi-Platform solution also needs the .NET 10 Desktop Runtime on the Designer machine. See Script Compilation Target.
Note: the Verify References check also runs on the client side and reflects client-side accessibility only.

Remote Clients

Each process resolves `AssemblyPathRuntime` against its own file system, and **Target Domain** decides which process loads the row: the server-side Script module skips `Client` rows, and the client-side Script module skips `Server` rows. A `Client` DLL must therefore be present on every machine that runs a Client, not only on the server.

Exception: assemblies that are part of .NET Framework and registered in the Windows Global Assembly Cache, such as `System.Management`, do not need to be copied to client machines. On Rich Client and Smart Client, .NET loads the version installed with Windows, whatever path is configured.

Rich Client has one fallback that reaches a local installation folder, and it is tied to **WpfControls**: when the resolved path is absolute, contains a `\WpfControls\` folder segment, and the file is not found there, the client retries the same file name in the WpfControls folder of its own installation. A path without that segment — a DLL kept in a subfolder of the solution, for example — gets no such fallback.

To load a Client DLL on Rich Clients running on other machines, place the DLL in the WpfControls folder of the FrameworX installation on each of those machines.

SmartClient resolves differently: it looks for the file name in its local WpfControls folder, and otherwise downloads it from `<SmartClient URL>/WPFControls/`.

An `AssemblyPathRuntime` that starts with `http://` or `https://` is downloaded before loading, into a `WPFControlsCache` folder created next to the running executable.

Using Referenced Assemblies

1. Add the Reference

Go to **Scripts References** and add the DLL location.

2. Add Namespace Declarations

Open Code Editor and click the **Namespace Declarations** button to add namespaces.

From FrameworX 10.1.6, to make a namespace available to every C# and VB.NET script of the solution at once, enter it in **Global Script Namespace Declarations** on **Solution Settings**. See Solution Settings Reference.

> **Important**
>
> Never put `using` (C#) or `Import` (VB.NET) statements directly in code. Always use the Namespace Declarations dialog. Direct statements will cause compilation errors.

3. Use in Code

```
// After adding reference and namespace declaration
// You can use types from the external assembly
var processor = new ExternalLibrary.DataProcessor();
var result = processor.Process(data);
```

DLL Loading Behavior

Runtime Loading

- DLLs load when Script Module starts
- To update DLL:
    1. Stop Script Module
    2. Close Designer
    3. Replace DLL file
    4. Restart Script Module

Designer Loading

- DLLs load when Scripts page first opens
- To update DLL after initial load:
    1. Close Designer completely
    2. Replace DLL file
    3. Reopen Designer
    4. Navigate to Scripts

> The platform caches loaded assemblies for performance. Complete Designer restart is required to reload updated DLLs.

Namespace Declarations Dialog

Access via Code Editor toolbar button:

From FrameworX 10.1.6, the predefined namespaces shown in this dialog include the Global Script Namespace Declarations of the solution.

This dialog allows you to:

- View default included namespaces
- Add namespaces from referenced assemblies
- Add system namespaces not included by default
- Manage both C# `using` and VB.NET `Import` statements

The declarations apply to:

- Current Script Class or Task (when editing scripts)
- Current Display (when editing Code Behind)

Best Practices

- [ ] **Always use path macros** - Never hardcode absolute paths
- [ ] **Match target frameworks** - Ensure DLL compatibility
- [ ] **Organize by domain** - Server DLLs in ThirdParty, Client in WpfControls
- [ ] **Document dependencies** - Note required DLLs in solution documentation
- [ ] **Version contro**l - Include referenced DLLs in source control
- [ ] **Test after updates** - Verify functionality when updating DLLs
- [ ] **Use Namespace Declarations** - Never add using/import statements directly in code

Troubleshooting

Reference Not Resolved (Red X)

- Hover over row to see error details
- Verify DLL targets correct .NET version
- Check assembly dependencies are available
- Ensure path macro is valid

Types Not Found in Code

- Confirm reference shows Resolved (green check)
- Add namespace via Namespace Declarations
- A namespace declaration alone does not add the assembly to the build. If the assembly is not referenced by default for the script target (see Assemblies Referenced by Default), add it in **Scripts References**
- Verify Target Domain matches usage location
- Check DLL compatibility with solution framework

Build Errors Even Though the Reference Looks Resolved

Symptoms: the script fails with errors such as `CS0103` (the name does not exist), `CS0246` (type or namespace not found) or `CS0122` (inaccessible due to its protection level) on types from the referenced DLL, even when using the full namespace.

- The build includes a reference only when its **Target Domain** matches the domain of the Script Class or Task being compiled. A reference set to Server is not available to a Client class, and the reverse.
- If the resolved path does not exist, the Designer looks for the same file name directly in the `_ProductPath_` folder (for example, `C:\Program Files\Tatsoft\FrameworX\fx-10\MyLib.dll`) and directly in the `_SolutionPath_` folder (for example, `D:\Projects\Plant1\MyLib.dll`), without subfolders. If the file is not found there either, the reference is left out of the build without a separate message.
- A green check in **Resolved** may be outdated (for example, after copying the solution to another computer or moving the DLL). Click **Verify References** to check the path again on the current computer.

DLL Updates Not Reflected

- Close Designer completely
- Replace DLL file
- Reopen Designer and Scripts page
- For runtime: Stop/restart Script Module

Client DLL Loads on the Server but Not on Remote Clients

- The Script module writes `Build References :: Failed openning '<path>': <error>` to the trace of the process that tried to load it — the path shown is the fully resolved path on that machine, after macro expansion
- Verify that path exists on the client machine, not only on the server
- Only a path containing a `\WpfControls\` segment falls back to the client's local installation folder — see Remote Clients above
- A reference downloaded over http/https logs `Build References :: Failed downloading '<url>': <error>` instead

Path Issues When Moving Solutions

- Use macros instead of absolute paths

- Ensure DLLs exist in standard folders
- Copy ThirdParty folder with solution

In this section...
