EngWrapper API Reference

Create and modify solutions programmatically from .NET code.

Overview

The EngWrapper API automates solution engineering. From any .NET application you can create tags, templates, devices, alarms, historian items, scripts, displays, and runtime settings — the same objects you configure interactively in the engineering environment. Typical uses: bulk-loading configuration from spreadsheets or external systems, generating solutions from your own templates, and integrating solution builds into automated workflows.

Two classes provide the API, both in the `T.Eng.Wrapper` namespace:

Class	Purpose
`SolutionDB`	Opens an existing solution file (`.dbsln`) and adds, queries, or removes configuration objects. Also creates displays.
`InfoServer`	Solution-level operations: list templates and existing solutions, validate names, create new solutions.

Overview
How the API Works
Progress and logging
Opening a Solution
Committing Changes
Adding Objects
Tags
User Templates and Members
Enumerations
Asset Folders
Tag Providers
Device Channels
Device Nodes
Device Points
Device Access Types
Alarm Groups
Alarm Areas
Alarm Items
Historian Tables
Historian Items
Expressions
Script Classes
Script Tasks
Custom Profile Settings
Importing Objects from CSV Files
Importing Objects from JSON Files
Reading, Querying, and Deleting
Setting Per-Instance Tag Properties
Creating Displays
Lifecycle
InsertDisplay
Display utilities
Controlling the Runtime
Creating New Solutions
Enumerations Reference
Complete Example
Python Integration
Requirements
Loading the .NET Runtime
Run on an STA Thread
Adding Objects and Committing
Creating Displays from Python

Requirements

- The product installed on the machine (version 10.1 or newer).
- .NET Framework 4.8 project. Display creation additionally requires a WPF application (the API renders displays through a live WPF surface).
- References to `T.Eng.Wrapper.dll` and `T.Library.dll` from the product installation folder.
- Close the solution in the engineering environment before editing it through the API, or open it as remote engineering.

How the API Works

The workflow is: **open** **add** **commit** **close**.

1. Open a solution by constructing a `SolutionDB`.
2. Call `Add...` methods. Each call **queues** the new object; nothing is written yet.
3. Call `Commit()` to write all queued objects to the solution database in dependency order.
4. Dispose the `SolutionDB` (a `using` block, or `CloseSolutionDB()`).

Two groups of methods are **not** queued and execute immediately: display methods (`InsertDisplay`, `RemoveAllSymbols`, ...) and per-instance tag property overrides (`SetTagEnumeration`, `SetTagProperty`, ...), which resolve a live tag/member token and must be called after `Commit()`.

`Add...` methods and `Commit()` return `object`: the value is `true` on success or the caught `Exception` on failure.

Progress and logging

`SolutionDB` exposes a logger callback you can replace:

```
project.Logger = (message, progress) => Console.WriteLine($"{progress:P0} {message}");
```

Pass `enableLogger: true` in the constructor to activate it (default target is the console).

Opening a Solution

```
public SolutionDB(string solutionFilename, string server = null,
                  bool enableLogger = false, bool fullPropertiesLogger = false)
```

Parameter	Description
`solutionFilename`	Full path of the solution file (`.dbsln`).
`server`	Web services host for remote engineering. Omit (or pass `null`) for a local connection.
`enableLogger`	Enables the progress/error logger.
`fullPropertiesLogger`	Also logs every property written.

Example — create a working file from the blank template and open it:

```
using T.Eng.Wrapper;

File.Copy(@"C:\Users\Public\Documents\FrameworX\Templates\blank.dbsln",
          TDirectory.MyDocuments + @"FrameworX\Solutions\Test.dbsln", true);

using (SolutionDB project = new SolutionDB(
           TDirectory.MyDocuments + @"FrameworX\Solutions\Test.dbsln", "", true))
{
    // ... Add methods ...
    project.Commit();
    project.CloseSolutionDB();
}
```

Committing Changes

```
public object Commit()
```

Writes every queued object to the solution database, table by table, in the order required by object dependencies (templates before tags, channels before nodes, and so on). Call it once after all `Add...` calls. Returns `true` or the `Exception` that interrupted the write.

Adding Objects

All methods below queue rows for `Commit()`. String parameters that reference other objects accept plain names — the API adds the namespace prefix automatically (`Channel.`, `Node.`, `Alarm.Group.`, `Alarm.Area.`, `Historian.Table.`, `Tag.`).

Tags

```
public object AddTag(string tagName, string tagDisplayText = "", object tagType = null,
    string tagUnits = "", string tagStartValue = "", string tagMin = "", string tagMax = "",
    string tagArray = "", string tagFormat = "", string tagParameters = "",
    string tagAssetPathName = null, eRetentive tagRetentive = eRetentive.None,
    eDomain tagDomain = eDomain.Server, eVisibility tagVisibility = eVisibility.Protected,
    string tagDescription = null, string tagEnumeration = null)
```

`tagType` accepts a `SolutionDB.eTagType` value **or** a user template name (string) to instantiate that template. `tagAssetPathName` places the tag under an asset folder. `tagEnumeration` links an enumeration set.

```
project.AddTag("Tag1", "Comments of Tag1", SolutionDB.eTagType.Double,
               "V", "25", "0", "800", "", "", tagDescription: "Desc Tag1");
project.AddTag("TagTemplate1", tagType: "PID", tagDescription: "PID instance");
project.AddTag("TagEnum1", "State tag", SolutionDB.eTagType.Digital, tagEnumeration: "Enum1");
```

User Templates and Members

```
public object AddUserTemplate(string templateName)

public object AddMemberName(string templateName, string memberName, string memberDisplayText = "",
    object memberType = null, string memberUnits = "", string memberStartValue = "",
    string memberMin = "", string memberMax = "", string memberArray = "", string memberFormat = "",
    string memberParameters = "", eRetentive memberRetentive = eRetentive.None,
    eDomain memberDomain = eDomain.Server, eVisibility memberVisibility = eVisibility.Protected,
    string memberDescription = null, string memberEnumeration = null)
```

```
project.AddUserTemplate("PID");
project.AddMemberName("PID", "sp", memberType: SolutionDB.eTagType.Double, memberDescription: "Setpoint");
project.AddMemberName("PID", "pv", memberType: SolutionDB.eTagType.Double, memberDescription: "Process
value");
```

Enumerations

```
public object AddEnumeration(string enumerationName)
public object AddEnumerationValue(string enumerationName, string valueName, string value,
                                  string displayText = "", string description = "")
```

Create the enumeration with `AddEnumeration`, then add each value with `AddEnumerationValue`: `value` is the numeric value and `displayText` is the label shown for it. This writes the same columns the Designer enumeration editor uses. (The lower-level `AddMemberName` with `memberStartValue` still works, but `AddEnumerationValue` is the recommended, clearer call.)

```
project.AddEnumeration("Enum1");
project.AddEnumerationValue("Enum1", "Open",  "0");
project.AddEnumerationValue("Enum1", "Close", "1");
```

Asset Folders

```
public object AddAssetFolder(string newName, string parentAssetFolder = null)

public object InsertAssetName(string assetName, string parentAssetFolder, string alias)
```

`AddAssetFolder` creates a folder in the asset tree (`parentAssetFolder` null or empty for root). `InsertAssetName` inserts an existing asset under a folder with an alias.

Tag Providers

```
public object AddTagProvider(string name, string protocol, string protocolOptions,
    string primaryStation = "", string backupStation = "", string _interface = "TCPIP",
    string separators = "BranchSeparator=/;AttributeSeparator=/", int serviceType = 0,
    int access = 0, int readTime = 500, int writeTime = 500,
    string serverIP = "", string description = "")
```

```
project.AddTagProvider("TcpDataAccess1", "TcpDataAccess", "", "Ip1", "Ip2", "Custom");
```

Device Channels

```
public object AddDeviceChannel(string channelName, string channelProtocol, string channelInterface,
    string channelDescription = "", string channelSettings = "", string channelTimeout = "",
    eInitialState channelInitialState = eInitialState.Enabled, string channelProtocolOptions = "")
```

The helper `PrepareChannelTimeout` builds the timeout string:

```
public static string PrepareChannelTimeout(int tx, int rxStart, int rxFinish, int nextByte, int retry)
```

```
project.AddDeviceChannel("Channel1", "Modbus", "TCPIP", "Channel1", "Settings",
    SolutionDB.PrepareChannelTimeout(1000, 2000, 3000, 250, 5),
    SolutionDB.eInitialState.Enabled,
    "BlockSize=50;Encoding=RTU TCP;SingleWrite=Use single write;SlaveID=2;CoilWriteValue=FF;Offset
address=True");
```

Device Nodes

```
public object AddDeviceNode(string nodeName, string nodeChannelName, string nodePrimaryConnection,
    string nodeSecondaryConnection = "", string nodeDescription = "",
    string syncSettings = "", string syncStation = "", string syncDate = "")
```

```
project.AddDeviceNode("Node1", "Channel1", "192.168.10.19;502;1", "192.168.10.19;503;1", "Node For
Channel1");
```

Device Points

```
public object AddDevicePoint(string pointName, string pointLabel = "", string pointNode = "",
    string pointAddress = "", eDataType pointDataType = eDataType.Native, string pointModifiers = "",
    string accessTypeName = "ReadWrite", string pointScaling = "")
```

`pointName` is the tag (or template member) the point maps to. The helper `PrepareScaling` builds the scaling string:

```
public static string PrepareScaling(float rawMin, float rawMax, float engMin, float engMax)
```

```
project.AddDevicePoint("Tag1", "Label for Tag1", "Node1", "40001",
    SolutionDB.eDataType.Native, "", "ReadWrite", SolutionDB.PrepareScaling(0, 65535, 0, 800));
project.AddDevicePoint("TagTemplate1.sp", "", "Node1", "40003",
    SolutionDB.eDataType.Native, "", "ReadWrite", SolutionDB.PrepareScaling(0, 65535, 0, 200));
```

Device Access Types

```
public object AddDeviceAccessType(string accessTypeName, bool readOnStartup = false,
    int readPolling = 1, int readPollingRate = 500, string readTrigger = "",
    string readStatus = "", string readCompleted = "", bool writeEventEnabled = false,
    int writeEvent = 0, string writeTrigger = "", string writeStatus = "", string writeCompleted = "",
    bool acceptUnsolicited = false, string blockCommand = "", bool isDeviceControl = false,
    string acessTypeDescription = "")
```

`readPolling`: 0 = Never, 1 = Always, 2 = OnDisplayOrServer. `writeEvent`: 0 = Changed, 1 = ChangedUp, 2 = ChangedDown.

```
project.AddDeviceAccessType("AccessType1");
```

Alarm Groups

```
public object AddAlarmGroup(string alarmGroupName, eAlarmAck alarmGroupAck = eAlarmAck.Ack,
    eAlarmSound alarmGroupSound = eAlarmSound.None, int alarmShow = 1,
    eAlarmLog alarmLog = eAlarmLog.ActiveNormAck, string ackTimeout = "", string autoAckTimeout = "",
    string alarmGroupActiveTextColor = "", string alarmGroupActiveBackgroundColor = "",
    string alarmGroupAckedTextColor = "", string alarmGroupAckedBackgroundColor = "",
    string alarmGroupNormalTextColor = "", string alarmGroupNormalBackgroundColor = "",
    int activeTimeDeadband = 0)
```

```
project.AddAlarmGroup("LogOnly", SolutionDB.eAlarmAck.Ack, SolutionDB.eAlarmSound.None,
                      0, SolutionDB.eAlarmLog.ActiveNormAck, "10000", "20000");
```

Alarm Areas

```
public object AddAlarmArea(string alarmArea, string alarmAreaParent = "")
```

```
project.AddAlarmArea("Level1");
project.AddAlarmArea("Level2", "Level1");
```

Alarm Items

```
public object AddAlarmItem(string alarmName, string alarmGroupName = "", string alarmMessage = "",
    Int32 alarmPriority = 0, string alarmArea = "",
    eCondition alarmCondition = eCondition.GreatherEqual, string alarmValue = "1",
    string alarmAuxValue = "", string alarmSetPoint = "", string alarmSetPointDeadband = "",
    string alarmComment = "", string alarmAuxValue2 = "", string alarmAuxValue3 = "",
    bool alarmDisable = false)
```

```
project.AddAlarmItem("Tag1", "LogOnly", "Tag1 Failure", 100, "Level2", SolutionDB.eCondition.Hi, "650");
```

Historian Tables

```
public object AddHistoricalTable(string tableName, int autoCreate = 1, int saveOnChange = 1,
    int timeDeadBand = 0, int lifeTime = 0, string trigger = "", int saveQuality = 0)
```

```
project.AddHistoricalTable("Log1", 1, 1, 1000, 10, "Tag1");
```

Historian Items

```
public object AddHistoricalItem(string pointName, string pointHistoricalTable = "",
    double deadBand = 0, double deviation = 0, double rateOfChange = 0,
    int deviationDeadBandType = 1, double deviationDeadBandLimit = 0.0)
```

deviationDeadBandType: 0 = Absolute, 1 = Percentage.

```
project.AddHistoricalItem("Tag1", "Log1", 10);
```

Expressions

```
public object AddExpression(string tagName, string expression,
    eExecution execution = eExecution.Trigger_OnChange, string disableCondition = "")
```

```
project.AddExpression("Tag1", "Tag.Tag1 = 0", SolutionDB.eExecution.Trigger_OnChange);
```

Script Classes

```
public object AddScriptClass(string name, eScriptCode codeType, eDomain domain, string code)
```

```
string script = "public int Add(int a)\n{\n\treturn a + 1;\n}\n";
project.AddScriptClass("Class1", SolutionDB.eScriptCode.CSharp, SolutionDB.eDomain.Server, script);
```

Script Tasks

```
public object AddScriptTask(string name, eScriptCode codeType, string trigger, TimeSpan period,
    eDomain domain, string code, int initialState = 0)
```

`trigger` is an object/event name (for example `"Server.Second"`); `period` is an execution interval (use `TimeSpan.Zero` with a trigger). `initialState`: 0 = Enabled, 1 = Disabled.

```
string script = "@Tag.Tag2[1] = @Script.Class.Class1.Add(TK.To<int>(@Tag.Tag2[1]));\n";
project.AddScriptTask("Task1", SolutionDB.eScriptCode.CSharp, "Server.Second",
                      TimeSpan.Zero, SolutionDB.eDomain.Server, script);
```

Custom Profile Settings

Overrides connection settings per execution profile (database connections, device nodes, tag providers).

```
public object AddCustomProfileSettings(string replaceObject, int tableType = 0, string provider = "",
    string database = "", string connectionString = "", string logonPassword = "", string serverIP = "",
    string protocol = "", string protocolOptions = "", string primaryStation = "")
```

`tableType`: 0 = Dataset DB, 1 = Device Node, 2 = Tag Provider.

```
project.AddCustomProfileSettings("Dataset.DB.AlarmHistorian", 0, "SQLite", "SQLite2",
                                 "Conn1", "1234", "servip1", "prot1", "protop1", "123.123.4.6");
```

Importing Objects from CSV Files

Every object type also has a bulk method that reads a CSV file: the first row holds column names, each following row creates one object. Columns map by name to the parameters of the corresponding `Add...` method; missing columns take the same defaults. Rows still go through the queue — call `Commit()` afterwards.

Method	Required columns	Optional columns
`AddTags(csv)`	`Name`, `Type`	`DisplayText`, `Units`, `StartValue`, `Min`, `Max`, `Array`, `Format`, `Parameters`, `Path`, `Retentive`, `Domain`, `Visibility`, `Description`, `Enumeration`
`AddUserTemplates(csv)`	`Name`	
`AddMemberNames(template, csv)`	`Name`, `Type`	`DisplayText`, `Units`, `StartValue`, `Min`, `Max`, `Format`, `Parameters`, `Array`, `Retentive`, `Domain`, `Visibility`, `Description`, `Enumeration`
`AddEnumerations(csv)`	`Name`	

AddTagProviders(csv)	Name, Protocol	ProtocolOptions, PrimaryStation, BackupStation, Interface, Separators, ServiceType, Access, ReadTime, WriteTime, ServerIP, Description
AddDeviceChannels(csv)	Name, Protocol	Interface, Description, Settings, Timeout, InitialState, ProtocolOptions
AddDeviceNodes(csv)	Name, Channel	PrimaryStation, BackupStation, Description, SyncSettings, SyncStation, SyncDate
AddDevicePoints(csv)	TagName, Node	Label, Address, DataType, Modifiers, AccessType, Scaling
AddDeviceAccessTypes(csv)	Name	ReadOnStartup, ReadPolling, ReadPollingRate, ReadTrigger, ReadStatus, ReadCompleted, WriteEventEnabled, WriteEvent, WriteTrigger, WriteStatus, WriteCompleted, AcceptUnsolicited, BlockCommand, IsDeviceControl, Description
AddAlarmGroups(csv)	Name	AckRequired, Sound, Show, LogEvents, Colors, AckTimeout, AutoAckTime, ActiveTimeDeadband
AddAlarmAreas(csv)	Name	Parent
AddAlarmItems(csv)	TagName	Group, Message, Priority, Area, Condition, Limit, AuxValue, SetPoint, SetpointDeadband, Comment, AuxValue2, AuxValue3, Disable
AddHistoricalTables(csv)	Name	AutoCreate, SaveOnChange, TimeDeadband, LifeTime, Trigger, SaveQuality
AddHistoricalItems(csv)	TagName	HistorianTable, DeadBand, Deviation, RateOfChange, DeviationDeadBandType, DeviationDeadBandLimit
AddExpressions(csv)		ObjectName, Expression, Execution, DisableCondition
AddScriptClasses(csv)	Name	Code, Domain, Contents
AddScriptTasks(csv)	Name	Code, Trigger, Period, Domain, Contents, InitialState

Example — a tags CSV:

```
Name,Type,Units,Min,Max,Description
Temperature1,Double,degC,0,150,Reactor temperature
Pressure1,Double,bar,0,16,Line pressure
Running1,Digital,,,,Motor running state
```

```
project.AddTags(@"C:\Import\Tags.csv");
project.Commit();
```

When several files reference each other, import in dependency order: templates and members first, then tags, access types, channels, nodes, points, tag providers.

Importing Objects from JSON Files

Imports JSON object files — the same format the engineering environment imports and exports — including displays, scripts, datasets, and any other document object.

```
public ImportResult   ImportJsonFile(string filePath)
public ImportResult[] ImportJsonFiles(string[] fileNames)
```

Files are processed in dependency order with upsert-by-name semantics (existing objects are updated, new ones created). Each `ImportResult` reports `Created`, `Modified`, `Skipped`, `Errors`, and `Messages`.

If the JSON files include displays or symbols, initialize the display container first (see Creating Displays):

```
project.IntializeCreateDisplays(frame);
ImportResult[] results = project.ImportJsonFiles(jsonFiles);
project.FinalizeCreateDisplays(frame);
```

Reading, Querying, and Deleting

```
public string[] GetUserTemplateNames()
public string[] GetUserTemplateMembers(string templateName)
public bool     DeleteUserTemplate(string templateName)
public bool     DeleteUserTemplateMember(string templateName, string memberName)
public bool     DeleteTag(string tagName)
```

Delete methods return `true` when the object no longer exists.

For tables without a dedicated method, generic access is available — `GetTable(tableName)` and `GetRows(tableName)` read configuration tables, `AddNewRow(tableName, row)` and `AddGeneric(table, properties)` queue arbitrary rows. Use these only when no typed method covers the object; they require knowledge of the underlying table schema.

Setting Per-Instance Tag Properties



Available from **FrameworX 10.1.5 Update 5d**. These methods are not present in earlier builds.

When two tags are built from the same user template, each instance carries its own value on the same member, set programmatically. This is the same per-instance override the Tag Properties dialog applies in the engineering environment. From FrameworX 10.1.6, six properties are supported: StartValue, Min, Max, Disable, DisplayText, and Enumeration. In 10.1.5, the list also includes Description. Passing a `null`/empty value clears the override for this one property and reverts the member to the user-template default. Other overrides on the same member are preserved.

Removed in FrameworX 10.1.6: the `Description` member of `eTagProperty` and the `SetTagDescription` method. A member's Description is defined once on the user template and shared by every instance of the member. The Tag Properties dialog shows it read-only on member instances, the same way it shows the member Type. From FrameworX 10.1.6, a script calling `SetTagDescription` or passing `eTagProperty.Description` no longer compiles. The remaining `eTagProperty` members keep their numeric values, so `Enumeration` stays 6. An extension compiled against 10.1.5 and passing the former Description value (5) to `SetTagProperty` receives an `ArgumentOutOfRangeException`. To annotate an individual instance, set the Description on the instance tag itself.

These methods are **not** queued — they resolve a live tag/member token and execute immediately, so call them **after** `Commit()`, when the tag and its member already exist. A valid target is a basic-typed member instance, array element, or parameterized reference; a plain top-level tag (whose properties are set through `AddTag`) or a user-template/container member is rejected.

```
// From FrameworX 10.1.6 (Description removed, numeric values unchanged)
public enum eTagProperty { StartValue = 0, Min = 1, Max = 2, Disable = 3, DisplayText = 4, Enumeration = 6 }

public object SetTagProperty(string tagName, string memberName, eTagProperty property, string value)

public object SetTagEnumeration(string tagName, string memberName, string enumerationName)
public object SetTagStartValue(string tagName, string memberName, string startValue)
public object SetTagMin(string tagName, string memberName, string min)
public object SetTagMax(string tagName, string memberName, string max)
public object SetTagDisplayText(string tagName, string memberName, string displayText)
// SetTagDescription(string tagName, string memberName, string description): removed in FrameworX 10.1.6
public object SetTagDisable(string tagName, string memberName, bool disable)
```

```
project.AddUserTemplate("Machine");
project.AddMemberName("Machine", "State", memberType: SolutionDB.eTagType.Integer);
project.AddTag("M1", tagType: "Machine");
project.AddTag("M2", tagType: "Machine");
project.Commit();   // tags and members must exist before overrides can resolve

// Same member, a different enumeration per instance:
project.SetTagEnumeration("M1", "State", "EnumB");
project.SetTagEnumeration("M2", "State", "EnumC");

// Other per-instance overrides on M1.State:
project.SetTagStartValue("M1", "State", "5");
project.SetTagMin("M1", "State", "0");
project.SetTagMax("M1", "State", "100");

// Clear one override - reverts that member to the user-template default:
project.SetTagEnumeration("M1", "State", "");
```

Creating Displays

Display methods render through a live WPF surface, so they require a WPF application and run immediately (no `Commit()` needed).

Lifecycle

```
public void IntializeCreateDisplays(System.Windows.Controls.Frame frame)
public void FinalizeCreateDisplays(System.Windows.Controls.Frame frame)
```

Call `IntializeCreateDisplays` once with a WPF `Frame` before any display operation, and `FinalizeCreateDisplays` when done.

InsertDisplay

```
public void InsertDisplay(string displayName, string listCSV)

public void InsertDisplay(string displayName, int engine, int panelType, int mode,
                          string listCSV, char sep = ',')
```

Creates the display if it does not exist, then places the elements described in `listCSV`. The first overload creates a WPF, canvas-panel, Page-mode display; the second selects the rendering engine, panel type, and display mode (`eDisplayMode` value) explicitly.

The CSV columns:

Column	Meaning
`Symbol`	Symbol to place. Solution symbols by name; library symbols by path (for example `HMI/Tanks/VerticalTank_Scale`) — imported into the solution automatically.
`TagName`	When `Symbol` is empty: places the default representation of this tag.
`Top`, `Left`	Position (required).
`Width`, `Height`	Optional size.
`Uid`	Element identifier.
`Labels`	`Property=Value` pairs separated by `;` — binds symbol properties and labels to tags or values (for example `Level=Tag1;MinScale=0;MaxScale=100`).
`Display`	Optional. When present, only rows matching the target display name are used — one CSV can describe several displays.

A row with `Symbol` = `CurrentDisplay` configures the display itself instead of placing an element. Its `Labels` accept: `Mode`, `Title`, `TitleBackground`, `TitleForeground`, `TitleFontFamily`, `TitleFontSize`, `TitleMargin`, `TitleFontAlignment`, `Border`, `Placement`, `Target`, `CloseButton`, `DialogButtons` (string, for example `OKCancel`), `PopupStaysOpen`, `PopupStaysOpenLostFocus`, `Navigation`, `LockedWidth`, `LockedHeight`, plus any other display property by name (for example `Background`). `Width` and `Height` come from their own columns.

Example — update the main page and create a popup:

```
project.IntializeCreateDisplays(this.frame);

StringBuilder sb = new StringBuilder();
sb.AppendLine("Symbol,Top,Left,Labels,Uid");
sb.AppendLine("CurrentDisplay,0,0,Background=#FF0000FF");
sb.AppendLine("HMI/Tanks/VerticalTank_Scale,52,63,Level=Tag1;Minscale=-10;Maxscale=110,VerticalTankScale1");
sb.AppendLine("HMI/Buttons/SelectorSwitch,52,605,Control=Client.SimulationDigital,SelectorSwitch1");
project.InsertDisplay("MainPage", sb.ToString());

sb = new StringBuilder();
sb.AppendLine("Symbol,TagName,Top,Left,Labels,Uid");
sb.AppendLine("CurrentDisplay,,0,0,Background=#FFFF0000;Width=600;Height=600;" +
    "Mode=" + SolutionDB.eDisplayMode.Popup + ";" +
    "CloseButton=1;" +
    "Title=New Title;" +
    "TitleBackground=#FFFF0000;" +
    "Border=" + SolutionDB.eDisplayBorder.Thin + ";" +
    "Placement=" + SolutionDB.eDisplayPlacement.BottomLeft + ";" +
    "Target=" + SolutionDB.eDisplayTarget.Mouse + ";" +
    "DialogButtons=OKCancel");
sb.AppendLine("HMI/Tanks/VerticalTank_Scale,,52,63,Level=Tag3,VerticalTankScale3");
project.InsertDisplay("Test2", sb.ToString());

project.FinalizeCreateDisplays(this.frame);
```

Display utilities

```
public string[] GetDisplays()
public string   GetDisplaySymbols(string displayName, char sep = ',')
public void     RemoveAllSymbols(string displayName)
```

`GetDisplaySymbols` returns the same CSV shape `InsertDisplay` accepts — useful to read a display, modify it, and write it back.

Controlling the Runtime

The runtime-control methods are not available. They stay in the API, and a call returns without an error, but it has no effect.

```
public void RunStartup(string userName, string password, bool connect, int timeoutSecWaitStarting = 60)
public void ShutdownRuntime()
public void HotReload(string userName, string password)
public void ConnectToRuntime(int timeoutSecWaitStarting = 60)
public bool IsConnectedToRuntime()
public void DisconnectFromRuntime()
```

Method	Description
`RunStartup`	Not available. The call has no effect.
`ShutdownRuntime`	Not available. The call has no effect.
`HotReload`	Not available. The call has no effect. The misspelled `HotRealod` stays as a deprecated `[Obsolete]` alias and has no effect either.
`ConnectToRuntime` / `DisconnectFromRuntime` / `IsConnectedToRuntime`	Not available. `ConnectToRuntime` and `DisconnectFromRuntime` have no effect, and `IsConnectedToRuntime` always returns `false`.
`IsRuntimeRunning`	Not available. The call always returns `false`, also while a runtime is running.

To start, stop or reload a runtime, use the supported paths instead:

- Start: use **Run Startup** in the Designer or **Run in Profile** in Solution Center. From a script or another application, run `TStartup.exe` with the `/solution:` parameter.
- Stop: use **Stop Runtime** in the Designer or **Stop** in Solution Center. A runtime installed as a Windows Service stops with `net stop "TStartup-<SolutionName>"`.
- Apply configuration changes to a running runtime: use **Hot Reload** in the Designer.

The steps are in Runtime How-to.

Creating New Solutions

The `InfoServer` static class provides solution-level operations. Call `Initialize()` once before use and `Uninitialize()` when done.

```
public static void      Initialize()
public static void      Uninitialize()
public static DataTable GetTemplatesList(int targetOS, int family)
public static DataTable GetSolutionsList()
public static string    GetDefaultNewSolutionName(string location, bool newFolder)
public static bool      IsSolutionNameInUse(string solutionName, string location)
public static DataTable CreateNewSolution(string[] array)
```

`CreateNewSolution` takes a string array indexed by the `eNewSolutionArguments` enumeration — `Name`, `Description`, `Location`, `TargetFramework` (`".net 4.8"` for Windows or the .NET version for multiplatform), `ProductFamily`, `ProductModel`, `SolutionTemplate` (template file path), `SolutionType`:

```
InfoServer.Initialize();

string[] args = new string[(int)eNewSolutionArguments.NumArgs];
args[(int)eNewSolutionArguments.Name]            = "MySolution";
args[(int)eNewSolutionArguments.Description]     = "Created by code";
args[(int)eNewSolutionArguments.Location]        = @"C:\Solutions\MySolution\";
args[(int)eNewSolutionArguments.TargetFramework] = ".net 4.8";
args[(int)eNewSolutionArguments.ProductFamily]   = "FrameworX";
args[(int)eNewSolutionArguments.ProductModel]    = "Unlimited";
args[(int)eNewSolutionArguments.SolutionTemplate] = "";
args[(int)eNewSolutionArguments.SolutionType]    = "Solution";

InfoServer.CreateNewSolution(args);
InfoServer.Uninitialize();
```

A simpler alternative when templates and product models are not needed: copy the blank solution template (`Templates\blank.dbsln` in the public documents folder) to the target path and open it with `SolutionDB`.

Enumerations Reference

All enumerations are nested in `SolutionDB`.

Enumeration	Values
`eTagType`	`Digital`, `Integer`, `Double`, `Decimal`, `Text`, `DateTime`, `Counter`, `Timer`, `Reference`, `TimeSpan`, `Long`, `Image`, `Guid`, `DataTable`, `Json`
`eDataType`	`Native`, `Bit`, `Byte`, `Char`, `Word`, `Short`, `Dword`, `Integer`, `Long`, `ULong`, `BCD`, `LBCD`, `Single`, `Real`, `ASCII`, `Unicode`, `OPCDateTime`, `Timer`, `Counter`, `Control`, `SBCD`, `LSBCD`
`eRetentive`	`None`, `ValueOnly`, `PropertiesAndValue`, `PropertiesExceptValue`
`eDomain`	`Server`, `Client`
`eVisibility`	`Private`, `Protected`, `Public`
`eCondition`	`Hi`, `HiHi`, `Lo`, `LoLo`, `RateOfChange`, `DeviationMinor`, `DeviationMajor`, `Equal`, `GreaterThan`, `GreatherEqual`, `LessThan`, `LessEqual`, `Changed`, `ChangedUp`, `ChangedDown`, `NotEqual`
`eInitialState`	`Enabled`, `Disabled`, `Remote`, `Reserved`
`eAlarmAck`	`No`, `Ack`, `Comment`
`eAlarmSound`	`None`, `Beep`, `Asterisk`, `Exclamation`, `Hand`
`eAlarmLog`	`None`, `Active`, `ActiveAck`, `ActiveNorm`, `ActiveNormAck`
`eExecution`	`Trigger_OnChange`, `Trigger_OnTimeInterval`, `Trigger_OnTimeOfDay`, `Trigger_OnChange_Startup`

eScriptCode	CSharp, VBdotNet
eDisplayMode	Page, Popup, Dialog, PopupWindow
eDisplayBorder	None, Thin, Double, Flat3D, Resize
eDisplayPlacement	Center, TopLeft, TopRight, BottomLeft, BottomRight
eDisplayTarget	Window, Mouse

Complete Example

```
using T.Eng.Wrapper;

File.Copy(@"C:\Users\Public\Documents\FrameworX\Templates\blank.dbsln",
          TDirectory.MyDocuments + @"FrameworX\Solutions\Test.dbsln", true);

using (SolutionDB project = new SolutionDB(
           TDirectory.MyDocuments + @"FrameworX\Solutions\Test.dbsln", "", true))
{
    // Devices
    project.AddDeviceChannel("Channel1", "Modbus", "TCPIP", "Channel1", "Settings",
        SolutionDB.PrepareChannelTimeout(1000, 2000, 3000, 250, 5),
        SolutionDB.eInitialState.Enabled,
        "BlockSize=50;Encoding=RTU TCP;SingleWrite=Use single write;SlaveID=2;CoilWriteValue=FF;Offset
address=True");
    project.AddDeviceNode("Node1", "Channel1", "192.168.10.19;502;1");
    project.AddDeviceAccessType("AccessType1");

    // Templates, enumerations, tags
    project.AddUserTemplate("PID");
    project.AddMemberName("PID", "sp", memberType: SolutionDB.eTagType.Double);
    project.AddMemberName("PID", "pv", memberType: SolutionDB.eTagType.Double);
    project.AddTag("Tag1", "Comments of Tag1", SolutionDB.eTagType.Double, "V", "25", "0", "800");
    project.AddTag("TagTemplate1", tagType: "PID");

    // Points
    project.AddDevicePoint("Tag1", "", "Node1", "40001",
        SolutionDB.eDataType.Native, "", "ReadWrite", SolutionDB.PrepareScaling(0, 65535, 0, 800));

    // Alarms
    project.AddAlarmArea("Level1");
    project.AddAlarmGroup("LogOnly");
    project.AddAlarmItem("Tag1", "LogOnly", "Tag1 Failure", 100, "Level1", SolutionDB.eCondition.Hi, "650");

    // Historian
    project.AddHistoricalTable("Log1");
    project.AddHistoricalItem("Tag1", "Log1", 10);

    // Scripts
    project.AddScriptClass("Class1", SolutionDB.eScriptCode.CSharp, SolutionDB.eDomain.Server,
        "public int Add(int a)\n{\n\treturn a + 1;\n}\n");

    // Write everything
    project.Commit();

    // Displays
    project.IntializeCreateDisplays(this.frame);
    StringBuilder sb = new StringBuilder();
    sb.AppendLine("Symbol,Top,Left,Labels,Uid");
    sb.AppendLine("HMI/Tanks/VerticalTank_Scale,52,63,Level=Tag1,Tank1");
    project.InsertDisplay("MainPage", sb.ToString());
    project.FinalizeCreateDisplays(this.frame);

    project.CloseSolutionDB();
}
```

Python Integration

The API is a .NET library, so it is not limited to C# — any language with a .NET bridge can drive it. This section shows the supported pattern for Python using pythonnet, running in-process against the installed product. No compilation or IDE is required.

> Download python code sample: EngWrapper-Python-Sample.zip

Requirements

- Windows with the product installed (version 10.1 or newer).
- 64-bit Python 3.x.
- pythonnet 3.0.5 or newer (`pip install pythonnet`).

Loading the .NET Runtime

Three rules apply before any API call:

1. Select the .NET Framework runtime **before** importing `clr`. The assemblies target .NET Framework 4.8; pythonnet defaults to .NET (Core) and cannot switch after `clr` is imported.
2. Make the installation folder resolvable for managed assemblies, native libraries, and late-loaded dependencies.
3. Initialize the SQLite library once per process, before opening any solution.

```
INSTALL = r"C:\Program Files\Tatsoft\FrameworX\fx-10"

from pythonnet import load
load("netfx")                     # must run before importing clr
import clr

import os, sys
sys.path.append(INSTALL)          # managed assembly resolution
os.add_dll_directory(INSTALL)     # native DLL resolution (SQLite)
os.chdir(INSTALL)                 # probing base for late-loaded dependencies

clr.AddReference("T.Library")
from T.Library import AdoAPI
AdoAPI.InitializeSQLiteLibrary()  # once per process

clr.AddReference("T.Eng.Wrapper")
from T.Eng.Wrapper import SolutionDB
```

Run on an STA Thread

The engineering engine initializes UI components internally, so every call must run on a single-threaded-apartment (STA) thread. Python threads are MTA. Create a .NET thread, mark it STA, and do all the work there:

```
from System.Threading import Thread, ThreadStart, ApartmentState

def work():
    project = SolutionDB(r"C:\path\to\MySolution.dbsln", "", True)
    # ... Add methods, Commit, displays ...
    project.CloseSolutionDB()
    project.Dispose()

thread = Thread(ThreadStart(work))
thread.SetApartmentState(ApartmentState.STA)
thread.Start()
thread.Join()
```

Adding Objects and Committing

The workflow is the same as in C# — open, add, commit, close. Optional parameters map to Python keyword arguments, and enumerations are reached through the class:

```
project.AddTag("Temperature1", "Reactor temperature",
               SolutionDB.eTagType.Double, "degC", "20", "0", "150",
               tagDescription="Created from Python")
result = project.Commit()   # True on success, the caught Exception on failure
```

Creating Displays from Python

Display creation renders through a live WPF surface, which a Python process does not have by default. Create both objects below on the STA worker thread; the `Application` must exist **before** `IntializeCreateDisplays`. No window is required.

```
clr.AddReference("PresentationFramework")
from System.Windows import Application, ShutdownMode
from System.Windows.Controls import Frame

app = Application.Current
if app is None:
    app = Application()
    app.ShutdownMode = ShutdownMode.OnExplicitShutdown
frame = Frame()
```

Build the element list as a CSV string. Python has no `StringBuilder.AppendLine`, so join the rows explicitly with `\r\n`.

> ⚠ Rows must be separated by CRLF (`\r\n`). With plain `\n` the call succeeds but inserts nothing — the whole string is parsed as a single header line.

Convert enumeration values with `str()` when composing labels. Create displays **after** `Commit()`, so the tags the symbols bind to already exist; display methods write immediately and need no second commit.

```
project.IntializeCreateDisplays(frame)

rows = [
    "Symbol,Top,Left,Labels,Uid",
    "CurrentDisplay,0,0,Background=#FF0000FF",
    "HMI/Tanks/VerticalTank_Scale,52,63,"
    "Level=Temperature1;Minscale=0;Maxscale=150,Tank1",
]
project.InsertDisplay("MainPage", "\r\n".join(rows) + "\r\n")

mode = str(SolutionDB.eDisplayMode.Popup)
rows = [
    "Symbol,TagName,Top,Left,Labels,Uid",
    "CurrentDisplay,,0,0,Width=600;Height=400;"
    "Mode=%s;CloseButton=1;Title=Detail" % mode,
    "HMI/Tanks/VerticalTank_Scale,,52,63,Level=Pressure1,Tank2",
]
project.InsertDisplay("PressurePopup", "\r\n".join(rows) + "\r\n")

project.FinalizeCreateDisplays(frame)
```

Each `InsertDisplay` call takes several seconds: the engine renders the display and waits for the visual tree to settle before saving. Use `GetDisplays()` to confirm the result.

In this section...