# AquaTwin API User Guide

*Water and sewer model automation, results, and CSV export*

**Developer Reference Guide**

Based on the AquaTwin APIExample project

This guide explains how to use the AquaTwin APIs from a .NET application. The examples cover water and sewer model execution, model editing, result access, CSV export, valve criticality, and asset-management results.

## Contents and Guide

## 1. Requirements

- Windows
- A valid AquaTwin installation and license
- An ArcGIS Pro project (`.aprx`) configured as an AquaTwin water or sewer project
- The model input and engine files associated with that project

When using compiled assemblies instead of project references, add references to the AquaTwin assemblies from the AquaTwin `bin` folder. The example project uses references to:

- `AquaTwinAPI.dll`
- `GNext.Application.dll`
- `GNext.Core.dll`
- `GNext.Editor.dll`
- `GNextML.Model.dll`
- `GNextML.License.dll`

Depending on the features used and the AquaTwin build, additional companion assemblies may be required. Keep the referenced AquaTwin assemblies and their native dependencies together.

Add the main API namespace:

```
using GNext.API;
```

## 2. Standard API lifecycle

Use AquaTwin APIs in this order:

1. Create an instance of either `AquaTwinWaterAPI` or `AquaTwinSewerAPI`.
2. Call `InitApplication()`.
3. Call `InitProject(projectPath, loadScenario)`.
4. Read or modify model data.
5. Run the scenario or load existing output.
6. Read or export results.
7. Call `ResetProject()` and `ResetApplication()`.

Always clean up in a `finally` block so resources are released when an operation fails.

```
var api = new AquaTwinWaterAPI();

try
{
    if (!api.InitApplication())
        throw new InvalidOperationException("AquaTwin initialization failed.");

    if (!api.InitProject(projectPath, true))
        throw new InvalidOperationException("The AquaTwin project could not be opened.");

    // Work with the project here.
}
finally
{
    api.ResetProject();
    api.ResetApplication();
}
```

The second argument of `InitProject` controls scenario loading:

- `true` loads the active scenario. If an active scenario is unavailable, AquaTwin loads the Base scenario.
- `false` opens the project without loading a hydraulic or hydrologic scenario.

To change scenarios after initialization, call `OpenScenario`. An empty scenario name represents the Base scenario.

```
api.OpenScenario("Base");
```

## 3. Running a water model

Create an instance of `AquaTwinWaterAPI`, initialize the project, retrieve its `WaterModelInfo`, make any required changes, and run the scenario.

```
using GNext.API;
using GNext.Core.Common;

public static void RunWaterModel(string projectPath, string csvFolder)
{
    var api = new AquaTwinWaterAPI();

    try
    {
        if (!api.InitApplication())
            throw new InvalidOperationException("AquaTwin initialization failed.");

        if (!api.InitProject(projectPath, true))
            throw new InvalidOperationException("The water project could not be opened.");

        var model = api.GetModelInfo()
            ?? throw new InvalidOperationException("Water model data is unavailable.");

        var junction = model.Junctions?.FirstOrDefault(x => x.AssetId == "219");
        if (junction != null)
            junction.BaseDemand *= 1.1f;

        var pipe = model.Pipes?.FirstOrDefault(x => x.AssetId == "153");
        if (pipe != null)
            pipe.InitialStatus = LinkStatus.Closed;

        string message = string.Empty;
        if (!api.RunScenario(ref message))
            throw new InvalidOperationException($"Water simulation failed: {message}");

        if (!api.RefreshOutput(6))
            throw new InvalidOperationException("Output time step 6 could not be loaded.");

        if (!api.ExportOutputToCSV(csvFolder, false))
            throw new IOException("Water output could not be exported.");
    }
    finally
    {
        api.ResetProject();
        api.ResetApplication();
    }
}
```

### Saving model changes

`RunScenario` saves model-data changes before running. If the model is changed without calling `RunScenario`, call `SaveScenarioData()` explicitly.

Changes are made to the scenario loaded by the API. Work on a suitable scenario when the original model must remain unchanged.

## 4. Reading water results

Call `RefreshOutput(timeStep)` before reading the output result properties for a particular output time step.

```
if (!api.RefreshOutput(6))
    throw new InvalidOperationException("The requested output could not be loaded.");

var model = api.GetModelInfo();
var junction = model?.Junctions?.FirstOrDefault(x => x.AssetId == "219");

if (junction != null)
{
    // Formatted value for the currently selected time step.
    string pressure = junction.Pressure;

    // Raw numeric value. Check the collection and key before reading.
    if (junction.Output != null &&
        junction.Output.Count > 0 &&
        junction.Output[0].TryGetValue("Pressure", out float rawPressure))
    {
        // Use rawPressure in calculations.
    }
}
```

Formatted properties such as `Pressure` and `Flow` use the asset's current time-step setting with numeric output values and unit labels. Raw `Output` dictionaries provide numeric values for calculation and automation.

## 5. Running a sewer model

The sewer workflow uses `AquaTwinSewerAPI` and returns `SWMMModelInfo`.

```
using GNext.API;
using GNext.Core.SWMM;

public static void RunSewerModel(string projectPath, string csvFolder)
{
    var api = new AquaTwinSewerAPI();

    try
    {
        if (!api.InitApplication())
            throw new InvalidOperationException("AquaTwin initialization failed.");

        if (!api.InitProject(projectPath, true))
            throw new InvalidOperationException("The sewer project could not be opened.");

        var model = api.GetModelInfo()
            ?? throw new InvalidOperationException("Sewer model data is unavailable.");

        var junction = model.Junctions?.FirstOrDefault(x => x.AssetId == "KRO1012");
        var dryWeatherFlows = junction?.Inflows?.DWFs;
        if (dryWeatherFlows != null && dryWeatherFlows.Count > 0)
            dryWeatherFlows[0].Base *= 1.1;

        var pump = model.Pumps?.FirstOrDefault(x => x.AssetId == "PUMP1");
        if (pump != null)
            pump.Status = PumpStatus.OFF;

        string message = string.Empty;
        if (!api.RunScenario(ref message))
            throw new InvalidOperationException($"Sewer simulation failed: {message}");

        if (!api.RefreshOutput(6))
            throw new InvalidOperationException("Output time step 6 could not be loaded.");

        if (!api.ExportOutputToCSV(csvFolder, true))
            throw new IOException("Sewer output could not be exported.");
    }
    finally
    {
        api.ResetProject();
        api.ResetApplication();
    }
}
```

After refreshing output, formatted properties (numeric values + unit labels) and raw output dictionaries are available in the same way as for water models:

```
var junction = model.Junctions?.FirstOrDefault(x => x.AssetId == "KRO1012");
string depth = junction?.Depth;

if (junction?.Output != null &&
    junction.Output.Count > 0 &&
    junction.Output[0].TryGetValue("Depth", out float rawDepth))
{
    // Use rawDepth in calculations.
}

var conduit = model.Conduits?
    .FirstOrDefault(x => x.AssetId == "KRO3001-KRO3002");
string maximumDepthRatio = conduit?.MaxDD;
```

# 6. Exporting results to CSV

Both water and sewer APIs provide two CSV export overloads.

## Export the currently loaded time step

Call `RefreshOutput` first, and then export:

```
api.RefreshOutput(6);
bool exported = api.ExportOutputToCSV(@"C:\AquaTwinExports", false);
```

## Select and export a time step in one call

```
bool exported = api.ExportOutputToCSV(
    7,
    @"C:\AquaTwinExports",
    true);
```

The arguments are:

- `timeStep`: the output time-step index to load.
- `sPath`: destination folder. If it is empty, AquaTwin uses the project's database folder.
- `bIncludeOutputSummary`: `true` includes Summary properties; `false` exports only Output properties.

Each asset type is written to a separate CSV file. All files include the asset ID, CSV values are escaped when necessary, and exported result values do not include display-unit suffixes.

Water exports can include:

- `Junction.csv`
- `Reservoir.csv`
- `Tank.csv`
- `Pipe.csv`
- `Pump.csv`
- `Valve.csv`

Sewer exports can include:

- `Subcatchment.csv`
- `Junction.csv`
- `Outfall.csv`
- `Divider.csv`
- `Storage.csv`
- `Conduit.csv`
- `Pump.csv`
- `Orifice.csv`

- `Weir.csv`
- `Outlet.csv`

The destination directory is created when it does not exist. Existing files with the same names are replaced. Check the returned Boolean and ensure the application can write to the selected folder.

## 7. Adding water controls

Controls can be added through `WaterModelInfo.Controls`. The following example opens a link at time 1 and closes it at time 15:

```
using GNext.Core.Water;

model.Controls.Add(new Control
{
    AssetId = "10",
    Setting = 1,
    ControlType = ControlType.Timer,
    Time = 1
});

model.Controls.Add(new Control
{
    AssetId = "10",
    Setting = 0,
    ControlType = ControlType.Timer,
    Time = 15
});
```

For a value-based control, identify the controlling node and threshold:

```
model.Controls.Add(new Control
{
    AssetId = "335",
    Setting = 1,
    ControlType = ControlType.Below,
    NodeId = "1",
    NodeSetting = 17.1f
});
```

The controlled `AssetId` must identify a link such as a pipe, pump, or valve. A setting of `0` means closed and `1` means open; other settings can represent pump or valve settings where supported.

## 8. Reading valve criticality information

Valve criticality data is available through the shared `AssetService` after a water project is initialized.

```
using GNext.API;
using GNext.Application;

var api = new AquaTwinWaterAPI();

try
{
    if (!api.InitApplication() || !api.InitProject(projectPath, true))
        return;

    var assetService = AssetService.GetInstance();
    var criticality = assetService?.GetValveCriticalityInfo();

    if (criticality?.PoundInfos == null || criticality.PoundInfos.Count == 0)
        return;

    string poundId = "1";
    var isolatedCustomers = criticality.GetCategorizedCustomers(poundId);
    var lowPressureCustomers = criticality.GetLowPressureCustomers(poundId);
}
finally
{
    api.ResetProject();
    api.ResetApplication();
}
```

A `null` result normally means that the valve-criticality file is missing or invalid. Use a valid pound ID from the loaded criticality data.

## 9. Reading asset-management results

Initialize either a water or sewer project, then use `AssetService.GetInstance()` to access project asset data.

```
var activeRiskScenario = AssetService.GetInstance()?.GetRiskInfo();
var riskResults = activeRiskScenario?.RiskResults;

var failurePredictions = AssetService.GetInstance()?.GetMLPredictionData();
```

Other asset datasets, including inspection information, are also exposed through `AssetService`. Availability depends on the files and modules configured for the project.

## 10. Error handling and troubleshooting

### Initialization fails

- Confirm that AquaTwin is installed and licensed.
- Confirm that all required managed and native assemblies are available.
- Ensure that the application architecture matches the AquaTwin installation.

### Project initialization fails

- Confirm that the `.aprx` path exists.
- Confirm that the project is configured for the selected API type: water or sewer.
- Verify that its AquaTwin asset and model folders are present.

### A scenario does not run

- Read the message returned through `RunScenario(ref message)`.
- Confirm that the project has an associated model and valid active scenario.
- For sewer models, confirm that the SWMM engine library is available.

### Output cannot be refreshed

- Run the scenario first or confirm that a compatible output file already exists.
- Use a time-step index that exists in the result file.
- Do not read formatted result properties until output has been refreshed.

### CSV export fails

- Check the Boolean return value.
- Confirm that the destination is a folder, not a CSV filename.
- Confirm that the folder is writable and exported files are not locked by another application.
- If the path is empty, confirm that the project asset-data folder is available.

## 11. Recommended practices

- Use asset IDs rather than collection positions to find model elements.
- Check collections, assets, output lists, and dictionary keys before reading them.
- Preserve and report the message returned by `RunScenario`.
- Use raw output values for calculations and formatted properties for display.
- Always reset the project and application in a `finally` block.

For complete source examples, see `APIExample.cs` in the `APIExample` project.