Devices Import and Export Example

Import/export device configurations.

Download the solution Devices Import and Export Example.dbsln.

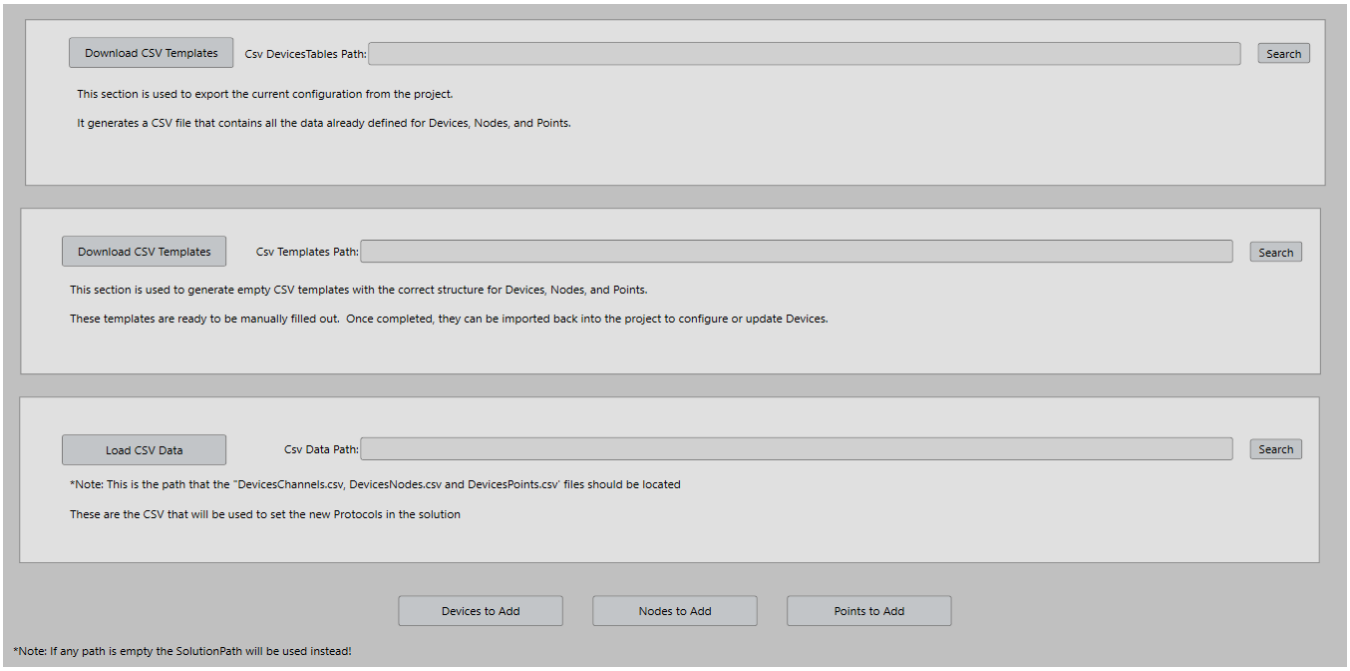
This solutions demonstrate how solutions can be dynamically configured or updated using external CSV files. It supports export, template generation, and import of the devices tables.

Summary

This feature allows projects to be dynamically configured or updated using external CSV files. It supports export, template generation, and import of the following configuration tables:

- DevicesChannels
- DevicesNodes
- DevicesPoints

These CSV operations are handled through script tasks and a utility class (`AutoDevices_DeviceManipulation`), which offer full control over structured device data via code.



Exporting Current Device Configuration

You can export the current project's configuration to CSV using the **Download CSV Templates** button.

Internally Executed Task: `AutoDevices_DownloadCsv`

This task performs the following:

```
// Gets the export path from a project Tag or defaults to the SolutionPath
string path = @Info.Solution.SolutionPath;
if (!string.IsNullOrEmpty(@Tag.AutoDevices_AutoDevices.DownloadPath)) {
    path = @Tag.AutoDevices_AutoDevices.DownloadPath;
}

// Exports each table as CSV
@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableAsCSV(
    TK.ProjectDB.GetDataTable("DevicesChannels"), path);
@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableAsCSV(
    TK.ProjectDB.GetDataTable("DevicesNodes"), path);
@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableAsCSV(
    TK.ProjectDB.GetDataTable("DevicesPoints"), path);
```

Generating Empty CSV Templates

To create empty templates (with only headers), click the **Download CSV Templates** button in the second panel.

Internally Executed Task: `AutoDevices_DownloadCsvTemplate`

This task saves header-only templates based on the columns currently supported:

```
string path = @Info.Solution.SolutionPath;
if (!string.IsNullOrEmpty(@Tag.AutoDevices_AutoDevices.DownloadPathTemplates)) {
    path = @Tag.AutoDevices_AutoDevices.DownloadPathTemplates;
}

@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableHeaderAsCSV(
    TK.ProjectDB.GetDataTable("DevicesChannels"), path);
@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableHeaderAsCSV(
    TK.ProjectDB.GetDataTable("DevicesNodes"), path);
@Script.Class.AutoDevices_DeviceManipulation.PrintDataTableHeaderAsCSV(
    TK.ProjectDB.GetDataTable("DevicesPoints"), path);
```

Importing CSV Data into the Project

To apply CSV configurations into the project, use the **Load CSV Data** option.

Internally Executed Task: `AutoDevices_LoadDataFromCsv:`

To apply CSV configurations into the project, use the **Load CSV Data** option available in the interface.

This will read external `.csv` files for `DevicesChannels`, `DevicesNodes`, and `DevicesPoints`, and populate the corresponding configuration tables in the current solution.

Example Usage

You can generate the CSV Empty templates with this solution and manually fill them with:

- **A Device** (e.g., `Device1`)
- **A Node** under this device (e.g., `Node1`)
- **Tags A, B, and C** assigned to the node, representing example data points

Once filled, you can import the CSVs back into the project to immediately see how the configuration behaves in runtime.

> Tags A, B, and C are already pre-created in the example project for you to use and test with!

Resetting the Configuration

If you want to reset the imported configuration:

- Close and Re-open the project in Engineering mode
- Manually delete the Devices, Nodes, and Points created via CSV
- You can then repeat the process with different CSV inputs or regenerate new templates



This makes the test project reusable for future import/export operations.

```
// Path setup
string basePath = @Info.Solution.SolutionPath;
if (!string.IsNullOrEmpty(@Tag.AutoDevices_AutoDevices.LoadDataPath)) {
    basePath = @Tag.AutoDevices_AutoDevices.LoadDataPath;
}

// Get destination tables from project
DataTable channels = TK.ProjectDB.GetDataTable("DevicesChannels");
DataTable nodes    = TK.ProjectDB.GetDataTable("DevicesNodes");
DataTable points   = TK.ProjectDB.GetDataTable("DevicesPoints");

// Load CSV into temporary tables
DataTable newChannels = @Script.Class.AutoDevices_DeviceManipulation.getDataTableFromCsv(Path.Combine
(basePath, "DevicesChannels.csv"));
DataTable newNodes    = @Script.Class.AutoDevices_DeviceManipulation.getDataTableFromCsv(Path.Combine
(basePath, "DevicesNodes.csv"));
DataTable newPoints   = @Script.Class.AutoDevices_DeviceManipulation.getDataTableFromCsv(Path.Combine
(basePath, "DevicesPoints.csv"));

// Apply the new data to the project tables
@Script.Class.AutoDevices_DeviceManipulation.LoadData(newChannels, channels);
@Script.Class.AutoDevices_DeviceManipulation.LoadData(newNodes, nodes);
@Script.Class.AutoDevices_DeviceManipulation.LoadData(newPoints, points);

// Apply and restart
TK.ProjectDB.ApplySolutionChanges();
@Info.Module.Device.Stop(0);
@Info.Module.Device.Start(2);
```

How It Works

- `PrintDataTableAsCSV()`
  Exports a full `DataTable` with headers and content to `\DataTables`.
- `PrintDataTableHeaderAsCSV()`
  Generates only the headers of necessaire columns into `\CSV Templates`.
- `getDataTableFromCsv(path)`
  Parses the CSV file and converts it into a `DataTable`. Logs any parsing inconsistencies.
- `LoadData(fromTable, toTable)`
  Copies values from the CSV `DataTable` into the project's internal tables, matching only accepted columns. Then performs a database update.



All of these methods are implemented in the script class `AutoDevices_DeviceManipulation`, available as part of the provided solution example. This class can be reused or customized as needed to support other configuration workflows.

In this section...
