SolarPanels Geolocation Example

Explore a distributed solar panel demo that simulates data across multiple sources using a real-time architecture.

> The SolarPanel Demo showcases a distributed application example by simulating the data collection, publishing, and viewing processes of a solar panel solution. This demo uses a distributed software architecture to handle and process real-time data from multiple sources.
>
> There are two versions of the SolarPanel demo.

The first version, *SolarPanels Geolocation Example.dbsln*, included by default with the software installation, requires only one solution file, with the data defined in tags within the solution itself and the simulated communication driver populating the values.

Download the solution file: SolarPanels Geolocation Example.dbsln

---

The second version, *fxEdgeCollector* and *fxAssetsMonitor and* , uses the internal simulator to act as multiple devices, publishing data to an edge collector application. This application transforms the data and sends it to a centralized broker and the Canary Historian. The software platform serves as the visualization tool, displaying data from the centralized MQTT broker and the Canary Historian, enabling monitoring of the entire system.

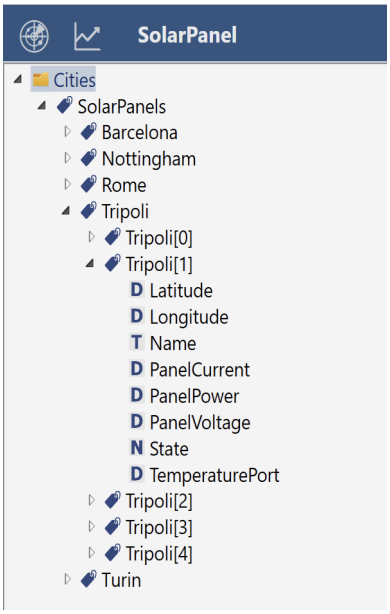
Download the solution files:

- Edge: fxEdgeCollector.dbsln
- Monitor: fxAssetsMonitor.dbsln

Technical Information

Asset Selection

The key display in that demo is the left panel, with the TAssetTree Control.



That control allows the operator to select that area he wants to monitor, and that information is saved on the built-in variable **Client.Context**. The key properties of the Client.Context are listed in the following table.

Client.Context Properties Updated by the TAssetTree UI control	
Property	**Description**
AssetName	The complete path for the asset. Example: \SolarPanels\Tripoli\Tripoli[1].PanelVoltage.
AssetPath	Path of Parent Node (excluding final attribute)
AssetNodeName	Name of parent folder. Example: Tripoli
AssetNodeDepth	The Depth of the node in the tree. Example: 2

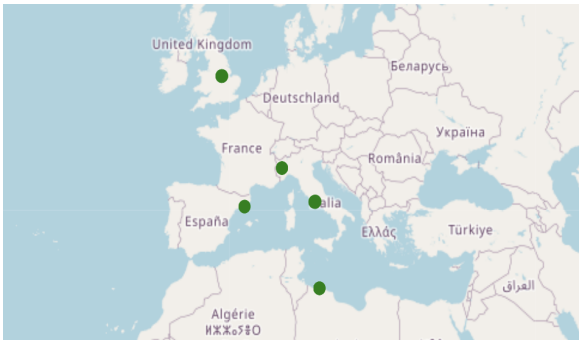
If you add Client.Context.AssetName to dynamic UI properties in the display, it will show that string value, like: "SolarPanels\Tripoli\Tripoli[1].PanelVoltage"

To show the VALUE of attribute use the syntax Asset(<string with asset name>). Example: Asset(Client.Context.AssetName)

For advanced applications instead of using the Client.Context.AssetName variable, you can create any expression that results the string with the AssetName you want to monitor.

The properties presented in this table are automatically populated when a TAssetTree control is loaded in any display. In order to use that features, you must design your Operator UI having only AssetTree active at any given time.

Map Update with Geo Location



The display showing the map uses the TMapsOSM control, which is available for both WPF and HTML5 pages.

When running only on Windows, the TMapsESRI control and the TMapsGMap control are also available.

The *DisplayIsOpen()* task monitors the *Client.Context.AssetName*, and when it changes, it uses the map control API to zoom to the coordinates of the selected city.

This demo was created using the *StretchFill* option for Displays, which resizes to fill the entire client area but will distort the image if resized to other aspect ratios.

You can also setup the system to various other **OnResize** options: StretchUniform, NoAction, Responsive, and ResizeChidren. For more information go to Display Settings page.

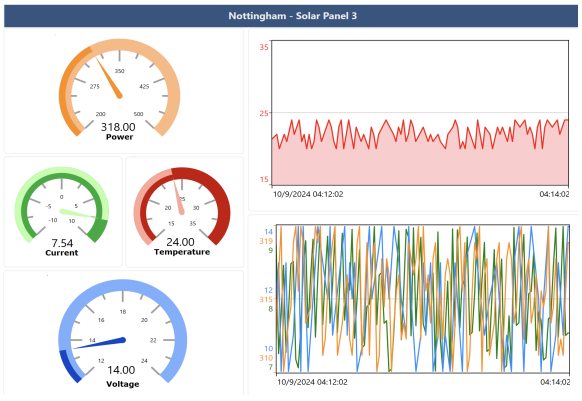
Asset Information Page

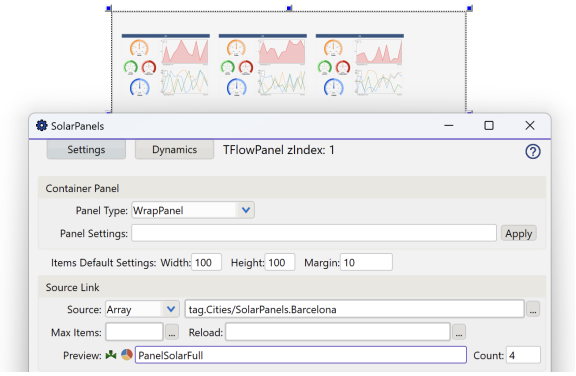
When one specific SolarPanel is selected a Process Screen shows the detailed information about that asset.

This diagram was created using a Smart Symbol.

To set the symbol parameters and bind the symbol to a specific city, there are two ways to execute the setup:

1. Using the TFlowPanel control
2. Using the *OnCreateDisplay* method





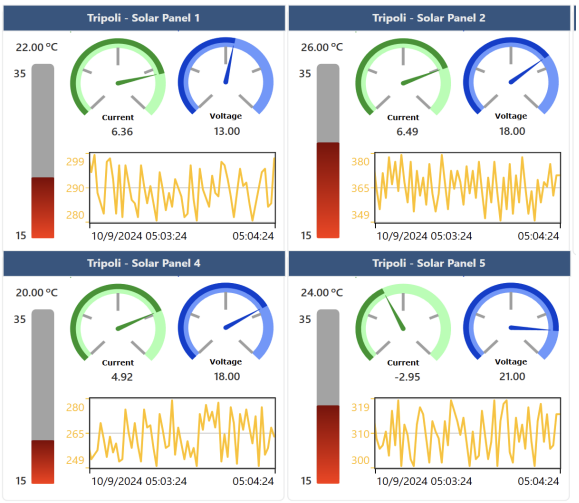
Configuration of the Symbol Parameters

The TFlowPanel is a UI control that allows you to set up multiple element types to add dynamics to containers like WrapPanel, Grid, or StackPanel. However, in this case, it would not be a typical use of that control, as it would be adding dynamics to just one object, rather than multiple, which is its main functionality. For more information see FlowPanel Control.

The second option, which involves display opening methods, was used in the demo. The *OnCreateDisplay(Canvas)* method is called by the platform when the display is created, allowing you to customize the screen by adding custom elements.

```
public void OnCreateDisplay(Canvas canvas)
{
        Dictionary<string, object> labels = new Dictionary<string, object>();
        string selectAsset = @Client.Context.AssetPath;
        labels["Name"] = selectAsset + "Name";
        labels["Current"] = selectAsset + "PanelCurrent";
        labels["Voltage"] = selectAsset + "PanelVoltage";
        labels["Power"] = selectAsset + "PanelPower";
        labels["Temperature"] = selectAsset + "";
        TSymbol.CreateSymbol("PanelSolarInfo", canvas, labels, 5, 5, false, false, false, false, null);
}
```

Multiple Assets Page



When a city is selected, a dynamic page is created to display multiple instances of the SolarPanelDashboard, each one mapped to a unit in the selected city.

Similar to the detailed page, there are two ways to set this up in the configuration: using the TFlowPanel control, as previously described, or by using the *OnCreateDisplay* internal method.

To use the *OnCreateDisplay()* method, use the following code:

```
public void OnCreateDisplay(Canvas canvas)
{
        double leftPosition = 5;
        double topPosition = 5;
        int len = TK.GetTagChildren(@Client.Context.AssetPath).Length;
        string assetPath = @Client.Context.AssetPath;
        for(int i = 0; i < len; i++) {
                Dictionary<string, object> labels = new Dictionary<string, object>();
                labels["Name"] = assetPath + "[" + i + "].Name";
                labels["Current"] = assetPath + "[" + i + "].PanelCurrent";
                labels["Voltage"] = assetPath + "[" + i + "].PanelVoltage";
                labels["Power"] = assetPath + "[" + i + "].PanelPower";
                labels["Temperature"] = assetPath + "[" + i + "].TemperaturePort";
                if((leftPosition + symbolWidth) >= display.Width) {
                        topPosition = topPosition + symbolHeight;
                        leftPosition = 5;
                }
                symbol = TSymbol.CreateSymbol("SimplePanel_WPF", canvas, labels, leftPosition, topPosition,
false, false, false, false, null);
                leftPosition =  leftPosition + symbolWidth;
        }
```

Using Controls or Code

Using built-in controls, like FlowPanel, offers the advantage of simplicity and requires no coding knowledge. On the other hand, the code-based option provides full control and advanced customization options.

AssetsMonitor Demo

blocked URL

Overview

The intent of this variation of the *SolarPanel* demo is to showcase a more comprehensive scenario, utilizing external Historian tools —in this case, Canary— alongside a distributed architecture. In this setup, independent Edge solutions publish data to a centralized Historian (or MQTT broker), while another solution monitors the aggregated data.

Our platform is used on both sides of this architecture. The EDGE version of the product is used as the data collector and publisher, while the unlimited product version is used to monitor the entire system.

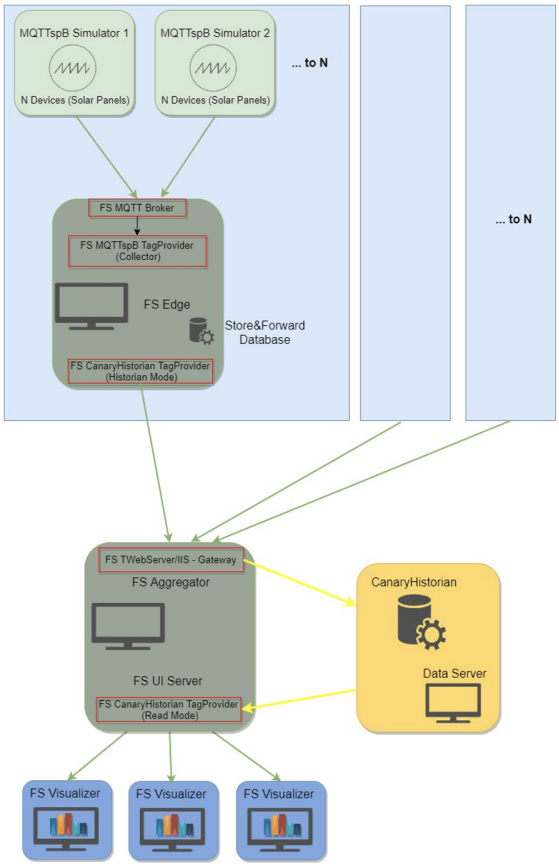
Solution Files

Download the solution files:

- Edge: fxEdgeCollector.dbsln
- Monitor: fxAssetsMonitor.dbsln

Architecture Diagram

The following image details the components integrated in this example:



MQTT SparkplugB simulator

We used this solution to publish data to the MQTT broker. The edge software can collect data using multiple protocols, run scripts to normalize the data, and publish it to the centralized MQTT broker or directly to the Canary Historian (or other systems) using Store and Forward protection.

The product's built-in MQTT SparkplugB Simulator was used to simulate multiple devices publishing data to the EdgeCollector.

MQTT Broker

For the Broker, we used the native MQTT Broker that is included with the framework. See more at Built-In MQTT Broker page.

Any other commercial MQTT broker would fit seamlessly into this solution. For cloud deployments, we recommend the HiveMQ broker. See more at HiveMQ Broker page.

Secure Web Gateway

This application also showcases the Secure Web Gateway product included in the product software, which allows safe routing of data packets across network security zones. See more at Secure Multi-Port Gateway page.

Canary Historian

The collected data is archived in the Canary Historian, using the TagProvider connection that the software product has with Historian systems.

Any other historian could be used, but Canary offers benefits due to its partnership with Tatsoft, which includes free Historian tags up to a certain quantity, easy integration setup, and upgrade options. See more at the Canary Labs page.

Data Visualization

Software Product's Unlimited was used for data visualization and aggregation.

In this section...