Asset Monitor Demo

Installation note. Starting in FrameworX 10.1.5, the `Asset Monitor Demo.dbsln` is no longer included in the product installer. On first install of an upgraded machine, Solution Center renames any prior `Asset Monitor Demo.dbsln` to `__backup__` and does not copy a fresh one — existing installations may continue to use the backed-up copy. **The demo's design and documentation remain published here for reference**, and the `.dbsln` is still downloadable from the attachments below for any customer who wants to re-deploy it.

For replacements, see the active demos: BottlingLine ML Demo, SolarPanels MCP Demo, and LocalAI KnowledgeGraph Demo.

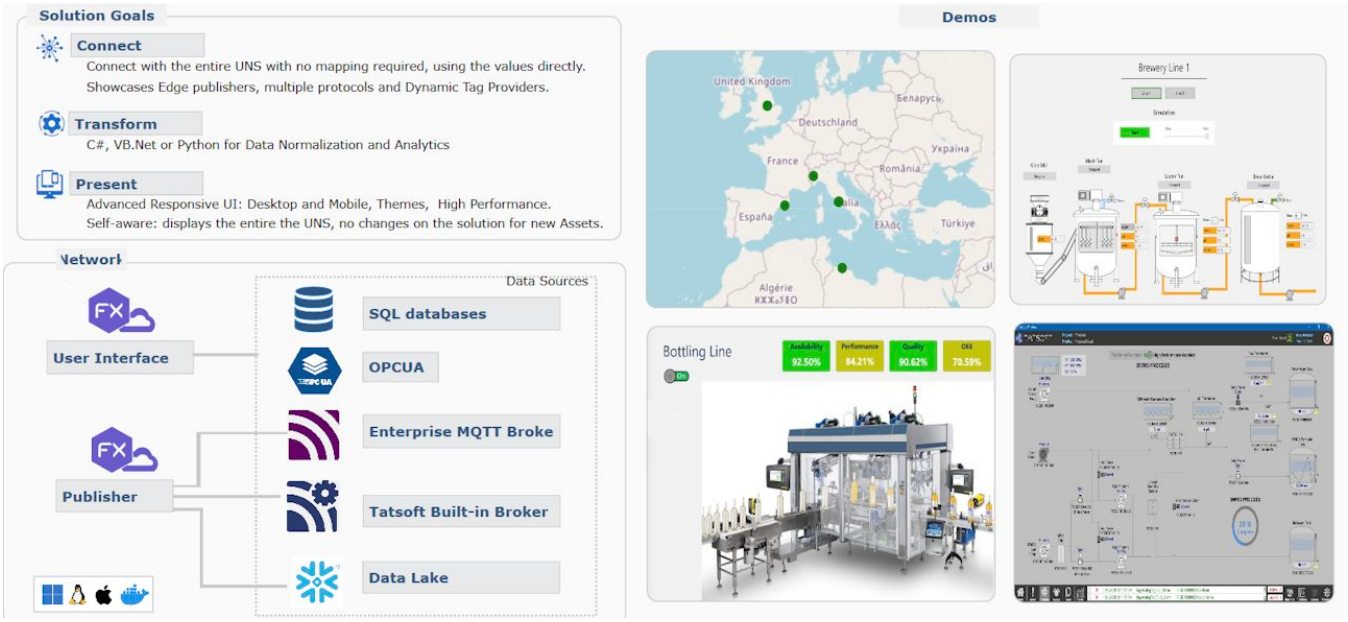
This demo showcases many features through an Asset Monitor solution.

Download the Demo Solution: Asset Monitor Demo.dbsln

This demo showcases, among other features, the following:

- Create a local UNS with tags and templates.
- Map an external TagProvider (MQTT in this case) for dynamic assets.
- Organize displays for Desktop, Tablet, and Mobile while sharing the same layouts.
- Manage display navigation based on the selected asset and user context.
- Integrate dashboard pages with Canvas (process diagram pages) for a fully responsive user experience.
- Create an Area display that automatically expands to show all lines within that area (UNS children).
- Build a display using Reference Tags to automatically map to the selected local tag.
- Create a display using the `Asset( )` syntax to dynamically map to an external asset.
- Enable connections from Windows Native Graphics (WPF), HTTP, and HTTPS on the same server.
- Leverage Execution Profiles to allow the same application to run on both a cloud server and locally with the appropriate settings.
- Utilize client-side tags and scripts for advanced UI—managing Solar Panel and Production assets while simplifying UI development.
- Implement server-side scripts for data calculations (in this case, simulations, though the same concept applies to analytics).

Summary



Technical Information

This demo showcases, among other features, the following:

- Create a local UNS with tags and templates.
- Map an external TagProvider (MQTT in this case) for dynamic assets.
- Organize displays for Desktop, Tablet, and Mobile while sharing the same layouts.
- Manage display navigation based on the selected asset and user context.
- Integrate dashboard pages with Canvas (process diagram pages) for a fully responsive user experience.
- Create an Area display that automatically expands to show all lines within that area (UNS children).
- Build a display using Reference Tags to automatically map to the selected local tag.
- Create a display using the `Asset()` syntax to dynamically map to an external asset.
- Enable connections from Windows Native Graphics (WPF), HTTP, and HTTPS on the same server.
- Leverage Execution Profiles to allow the same application to run on both a cloud server and locally with the appropriate settings.
- Utilize client-side tags and scripts for advanced UI—managing Solar Panel and Production assets while simplifying UI development.
- Implement server-side scripts for data calculations (in this case, simulations, though the same concept applies to analytics).

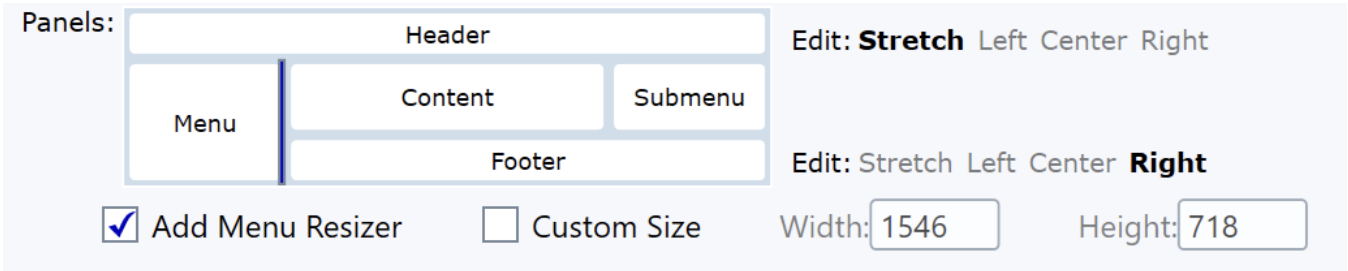
Organizing Displays Layouts

A good starting point for solutions deploying user interfaces is to organize the main layouts.

The layout used at startup is defined in **Displays-ClientSettings**, and for this solution, we named it **"Startup."**

We configured the Startup Layout with a Header, a left-side Menu, and a main content area**.**

- The **Header Edit** was set to **Stretch**, ensuring it spans the entire width of the display.
- The **left-side Menu** is positioned below the Header and is **resizable.**
- These settings are configured under **DisplaysLayouts** within the **Panels** section.



Next, we defined which pages would be used for the Header, Menu, and initial content.

The Layout feature can automatically detect whether the client is opening on a portrait or landscape mobile device, allowing customized page selection for each context. If no customization is specified, the display listed in the PAGE column will be used.

To disable a panel, add an "_" character in the corresponding cell. In this solution, we applied this to remove the left-side Menu when opening in Portrait Mobile mode.

Region	Page	Mobile	MobileLandsca...	Description
Header	Header	HeaderMobile	HeaderPad	Optional display for the Header panel
Footer				Optional display for the Footer panel
Menu	MenuTree	_		Optional display for the left side Menu panel
Submenu				Optional display for the right side Submenu panel
Content	Home			Page display with the main contents

When starting a RichClient, SmartClient, or WebPage on a desktop computer, the "Page" column is used.

When a client opens a web connection to the server, replacements are applied based on whether the device is in Mobile Portrait Mode or Mobile Landscape Mode (Tablet or Pad).

DisplaysLayouts

Only one layout is needed to define the initial page for Desktop, Tablets, and Mobile, whether using Windows Native (WPF) or HTML5. There is no need to create a separate layout for each device type.

In this demo, we created three additional layouts—Desktop, Pad, and Mobile—to allow the desktop to emulate other devices and for instructional purposes.

In this solution, all displays are shared across all devices. The only customization was applied to the header pages to enhance the navigation experience.

MainPage stays put. The landing content here is a `Home` page wired into the Startup layout's Content region; the predefined `MainPage` (Display ID 0) remains as the Solution Center preview anchor and is never renamed or deleted. Keeping a wired `Home` alongside an unused-looking `MainPage` is by design — do not "consolidate" them by renaming.

Defining the UNS (Unified Namespace)

As a general rule, defining the UNS is the best way to start a solution. This definition includes:

- Determining which data sources will be used.
- Specifying which tags will be created locally.
- Defining which assets will use external dynamic TagProviders.

For this demo, we defined two sets of data:

- Local Tags for Production Lines
- External TagProvider for Solar Panels

Local Tags and UserTypes

To streamline the creation of a data model, we defined the UserTypes:

- OEE – containing OEE information.
- Operation – storing the operation status for a production line.
- LINE UserType – including both OEE and Operation members.

This approach allows for the creation of a single tag—for example, `Press103` of type `LINE`—which will automatically include all attributes used in this demo.

Path	Name	Type	Array	Parameters	Domain
/Sites/Dallas/Slit	Src200	Line			
/Sites/Dallas/Press	Press106	Line			
/Sites/Dallas/Press	Press105	Line			
/Sites/Dallas/Press	Press104	Line			
/Sites/Dallas/Press	Press103	Line			
/Sites/Dallas/Lam	Lam502	Line			
/Sites/Dallas/Lam	Lam501	Line			
/Sites/Dallas/Bag	Elba7	Line			
/Sites/Dallas/Bag	Elba1	Line			
	MyLine	Reference		Line;	
	UI	UI			Client
	Placeholder	Placeholder			Client

The MyLine tag is used in the OEEDashboard and LineDashboard displays, pointing to the selected asset.

The UI tag is a common best practice when creating solutions, as it holds all internal variables used to manage the User Interface. The PlaceHolder is a predefined client tag useful for temporary data and testing.

Dynamic External TagProviders

This solution also uses assets, whose definition and count are dynamically queried from an external data source. In this case, we use an MQTT broker.

When the application starts (at Script ServerStartup), our built-in MQTT broker and the built-in MQTTspB simulator are executed.

The solution connects to the Broker, and all discovered assets are placed under the SolarPanels node in the UNS.

Data Transformation using Scripts

> **.NET and Python Scripts**
>
> Everything that can be accomplished using **Microsoft Visual Studio** and the **.NET programming languages** can also be done within **FramworX**.

In fact, it's even easier, as you can also add Python code. The platform natively supports server-side scripts with multi-tasking and client-side scripts that activate based on the open displays.

Server-side scripts are defined in Scripts-Tasks. They use only tags and server-domain objects and are available for all remote clients.

In this solution, the Server Tags represent the Production Line assets (e.g., `Press103`), and server-side scripts run a simulation on that data.

Client-side scripts are embedded in the open displays. In addition to the display loaded in the Layout, displays can have ChildWindows. All code from all loaded displays executes on the client side, whether in Windows or a browser.

In this solution, the CodeBehind of the `MenuTree` display handles navigation automation based on the selected asset.

The SITES display consists of two ChildDisplays:

- SitesMap
- Another display that dynamically changes based on the selected site.

When switching Displays and ChildDisplays, you are dynamically loading the scripts that should be executed on the client side.

Excellence in User Interface

This solution demonstrates features that deliver the best possible experience for users and operators.

Fully Responsive, Including Canvas

It is common for dashboards to be responsive. In a dashboard, the display layout consists of cells, which automatically adjust when resizing the client area or opening on a different device to best fit the available space.

FrameworX not only enables the creation of responsive dashboards but takes it a step further by allowing you to build responsive Canvas-based process diagrams, enabling advanced user interfaces with both dashboards and drawing containers.

Here are some examples of responsive displays in this solution, created using Responsive Canvas—something that would be time-consuming to achieve with dashboards alone, and wouldn't replicate the same user experience:

- SiteDallas, SiteMaps, Network, and Features

Even some displays that resemble dashboards are sometimes easier to create using freeform drawing tools. For example, the OEEDashboard page consists of two Canvas drawings:

- OEELeft
- OEERight

An example of a traditional cell-based dashboard can be found on the LineDashboard page.

Themes, with No Extra Programming

By keeping your solution consistent with default colors for graphical elements, and selecting new ones from the Themes palette, you can make your solution theme-ready with zero programming required.

Performance with Milliseconds Update.

The combination of FrameworX's proprietary optimizations, WebAssembly technology, and .NET for CodeBehind delivers the best possible performance.

For example, the Home Page of this solution includes two animations using the Client.Millisecond property to showcase real-time updates.

In this section...
