Extension Module API Reference

Build a third-party .NET module that appears as a native TagProvider in the Unified Namespace.

What is an Extension Module

An **Extension Module** is a third-party .NET module that plugs into the Unified Namespace (UNS) as a TagProvider. Once installed, it shows up in Designer next to the built-in TagProviders, and its data appears in the Asset Tree like any other part of the UNS.

The difference from a regular TagProvider is that an Extension Module is **active**: it has its own .NET code running inside its branch of the UNS. It does not only move data in and out; it decides what to expose, reacts to events, and can create and update properties on its own.

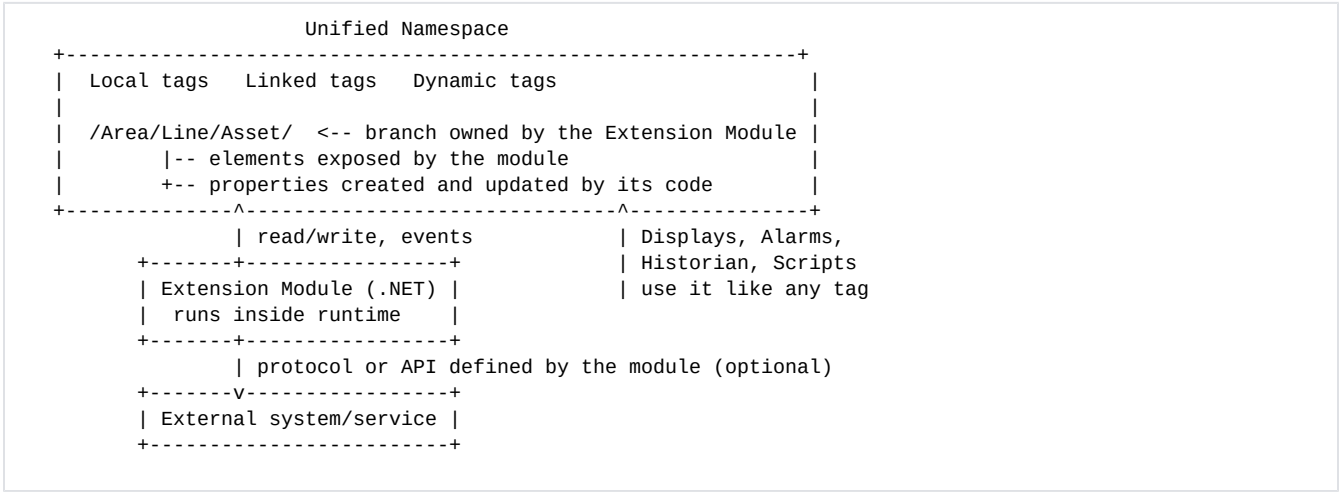
Typical uses are AI/ML engines, analytics platforms, cloud services and custom business logic. **Sorba AI** is the first Extension Module available for FrameworX (see Sorba AI Analytics Connector), and any FrameworX user or partner can build their own with the API described on this page.

Architecture: how an Extension Module differs from other connections

FrameworX has three levels of connectivity. Each one adds something on top of the previous one.

Level	Example	What it does	Code running behind it
Device connector (protocol driver)	Modbus	Reads and writes addresses you configure manually. No discovery, no data model.	None. Only the protocol.
TagProvider: Tag Discovery	MQTT Client, OPC UA Client, SQL, PI	Browses the external system and exposes its tree and data model in the UNS. Read/write goes through the protocol.	None. The tree is mirrored as it exists in the source.
TagProvider: Extension Module	Any third-party system or service, packaged as a .NET module	Exposes its own data model in the UNS **and** runs .NET code that monitors events, creates properties and acts on that part of the namespace.	Yes. A .NET assembly loaded by the runtime.

In short: with Tag Discovery, FrameworX *mirrors* an external namespace. With an Extension Module, a piece of the UNS is *owned and driven by code*. For example, when an input value changes, the module's code can react, read history from the Historian, call an external service, and publish the results back as new properties, all without user scripts.



How an Extension Module extends the UNS

1. **Install.** The partner's .NET assembly and its XML descriptor are placed in the FrameworX extensions folder.
2. **Create the connection.** In Designer, go to **Unified Namespace** **TagProvider Services**, create a new item and select **Extension Module**. The installed modules are listed there. Fill in the module's connection parameters.
3. **Link it to the Asset Tree.** In the Asset Tree, link the connection to a folder. It does not need to be a root folder; any level of the hierarchy works.
4. **Use it.** The module creates its elements under that folder automatically. They behave like any other UNS data: you can browse them in Designer, bind them in displays, use them in scripts, alarms and the Historian, and reach them at runtime with `Asset("/Path/...")`.

Extension Modules share the standard TagProvider runtime behavior: on-demand activation, automatic cleanup of dynamic tags, and RTDB integration.

Overview

You can build your own connector for FrameworX. The Extension Module API enables the development of custom .NET modules that integrate directly into the FrameworX Unified Namespace as native TagProviders. It lets third-party systems, such as analytics platforms, AI and machine learning engines, cloud services, or custom business logic, expose their data inside FrameworX (browsable and read/write) with no external gateway or middleware. This extends the platform's connectivity to virtually any system, surfacing external data and results directly where displays, scripts, and alarms already consume them, and giving customers a single namespace that unifies plant data with third-party intelligence.

If you need to access this API, contact us through support@tatsoft.com.