MQTT Integration Reference

Overview

This document presents how to connect to MQTT brokers and integrate external UNS data into FrameworX solutions. Two approaches are available:

1. **TagProvider (UNS Module)** — Auto-discovery, direct namespace integration
2. **Device Module** — Explicit mapping with Points, traditional SCADA approach

Overview
Approach 1: TagProvider (Recommended for large datasets with frequent definition updates)
Approach 2: Device Module
AssetTree (UNS Folder Organization)
Demo Workflows
Comparison: TagProvider vs Device Module
TableType Reference (Updated)
MCP Context Hints for AI

Approach 1: TagProvider (Recommended for large datasets with frequent definition updates)

TagProvider creates a direct connection to an external data source and integrates it into the Unified Namespace. Its items are read through a folder of the Asset Tree linked to the connection, or through a tag's DataLink.

JSON Structure (UnsTagProviders)

json

```
{
  "ObjectIdentification": {
    "TableType": "UnsTagProviders"
  },
  "Data": [
    {
      "Name": "MQTT",
      "Protocol": "MQTT",
      "PrimaryStation": "localhost;1883;ClientID;;;;;None;False;;AtLeastOnce;10;False;False;",
      "BackupStation": "",
      "Interface": "Custom",
      "Separators": "BranchSeparator=/;AttributeSeparator=/;",
      "Access": 0,
      "ReadTime": 500,
      "WriteTime": 500,
      "Description": "MQTT Broker Connection"
    }
  ]
}
```

JSON Structure (UnsTagProviders) with mandatory fields only

son

```
{
  "ObjectIdentification": {
    "TableType": "UnsTagProviders"
  },
  "Data": [
    {
      "Name": "MQTT",
      "Protocol": "MQTT",
      "PrimaryStation": "localhost;1883;ClientID;;;;;None;False;;AtLeastOnce;10;False;False;",
    }
  ]
}
```

`PrimaryStation Format (MQTT)`

```
{Host};{Port};{ClientID};{Username};{Password};{CertFile};{KeyFile};{TLS};{CleanSession};{WillTopic};{QoS};
{KeepAlive};{RetainPublish};{UseWebSocket};
```

Position	Field	Example	Description
1	Host	`localhost`	Broker hostname or IP

2	Port	`1883`	Broker port (1883=TCP, 8883=TLS)
3	ClientID	`MyClient`	Unique client identifier
4	Username	(empty)	Authentication username
5	Password	(empty)	Authentication password
6	CertFile	(empty)	TLS certificate file path
7	KeyFile	(empty)	TLS key file path
8	TLS	`None`	TLS mode: None, TLS, mTLS
9	CleanSession	`False`	Clean session flag
10	WillTopic	(empty)	Last Will topic
11	QoS	`AtLeastOnce`	QoS: AtMostOnce, AtLeastOnce, ExactlyOnce
12	KeepAlive	`10`	Keep-alive interval (seconds)
13	RetainPublish	`False`	Retain published messages
14	UseWebSocket	`False`	Use WebSocket transport

Special Placeholders:

- `_SolutionName_` — Replaced with solution name at runtime
- `_Guid_` — Replaced with unique GUID at runtime

Separators Configuration

`BranchSeparator=/;AttributeSeparator=/;`

Maps MQTT topic hierarchy to UNS path:

- Topic `plant/line1/tank1/temperature` Tag path `plant/line1/tank1/temperature`

Accessing TagProvider Data

Read topics through a folder of the Asset Tree linked to the connection. In this table, MQTT is a folder under Plant, linked to the root of the connection named MQTT:

MQTT Topic	UNS Access
`plant/line1/temp`	`Asset("/Plant/MQTT/plant/line1/temp")`
`spBv1.0/Group/Node/temp`	`Asset("/Plant/MQTT/spBv1.0/Group/Node/temp")`

Note: Asset() is the form for displays and expressions; C# and VB.NET code reads the same path with `TK.Data.AssetValue`. A tag's DataLink names the connection and the topic: `MQTT.("plant/line1/temp")`.

Approach 2: Device Module

The Device Module provides explicit control over tag-to-topic mapping using Channels, Nodes, and Points.

Available MQTT Protocols

Protocol	Description	Use Case
`MQTT`	Standard MQTT 3.1.1/5.0	Generic MQTT brokers
`MQTTspB`	Sparkplug B specification	Industrial IoT, auto-discovery

Channel JSON (DevicesChannels)

Standard MQTT:

json

```
{
  "ObjectIdentification": {
    "TableType": "DevicesChannels"
  },
  "Data": [
    {
      "Name": "MQTT1",
      "Protocol": "MQTT",
      "ProtocolOptions": "TimePublishRate=500;StoreAndForward=false",
      "Interface": "Custom",
      "Description": "MQTT Client Connector"
    }
  ]
}
```

Sparkplug B:

json

```
{
  "ObjectIdentification": {
    "TableType": "DevicesChannels"
  },
  "Data": [
    {
      "Name": "SparkplugCollector",
      "Protocol": "MQTTspB",
      "ProtocolOptions": "Type=Collector;TimePublishRate=500;StoreAndForward=false;
PublishDBIRTHAfterNBIRTH=true;UseParenthesesForArray=false;PublishUDTsAsSimpleTags=false",
      "Interface": "TCPIP",
      "Description": "Sparkplug B Collector"
    }
  ]
}
```

ProtocolOptions Reference

MQTT Protocol:

Option	Values	Default	Description
`TimePublishRate`	ms	500	Publish interval
`StoreAndForward`	true/false	false	Buffer on disconnect

MQTTspB Protocol:

Option	Values	Default	Description
`Type`	Collector/Edge	Collector	Node type
`TimePublishRate`	ms	500	Publish interval
`StoreAndForward`	true/false	false	Buffer on disconnect
`PublishDBIRTHAfterNBIRTH`	true/false	true	Birth certificate behavior
`UseParenthesesForArray`	true/false	false	Array notation style
`PublishUDTsAsSimpleTags`	true/false	false	UDT handling

Node JSON (DevicesNodes)

json

```json
{
  "ObjectIdentification": {
    "TableType": "DevicesNodes"
  },
  "Data": [
    {
      "Name": "MQTTBroker1",
      "Channel": "Channel.MQTT1",
      "PrimaryStation": "broker.example.com;1883;MyClient;;;None;;AtLeastOnce;10;",
      "BackupStation": "",
      "Description": "Production MQTT Broker"
    }
  ]
}
```

Node PrimaryStation Format

Same format as TagProvider PrimaryStation (see above).

Point Address Format

For Device Module Points, the Address field contains the MQTT topic:

json

```json
{
  "ObjectIdentification": {
    "TableType": "DevicesPoints"
  },
  "Data": [
    {
      "TagName": "Tag.Line1.Temperature",
      "Node": "Node.MQTTBroker1",
      "Address": "plant/line1/temperature",
      "AccessType": "Read"
    }
  ]
}
```

AssetTree (UNS Folder Organization)

The AssetTree defines the hierarchical folder structure in the Unified Namespace.

JSON Structure (UnsAssetTree)

json

```json
{
  "ObjectIdentification": {
    "TableType": "UnsAssetTree"
  },
  "Data": [
    {
      "Name": "Plant",
      "Parent": -1,
      "Children": "",
      "Description": "Root folder"
    },
    {
      "Name": "Line1",
      "Parent": 0,
      "Children": "",
      "Description": "Production Line 1"
    },
    {
      "Name": "Tank1",
      "Parent": 1,
      "Children": "",
      "Description": "Fermentation Tank"
    }
  ]
}
```

Fields

Field	Type	Description
`Name`	string	Folder name
`Parent`	int	Parent folder ID (-1 for root)
`Children`	string	Child mappings (TagProvider topics)
`Description`	string	Optional description

Mapping TagProvider Topics to Folders

The `Children` field can map TagProvider topics into the folder:

`@ProviderName.("topic/path")\t\tDisplayName\tProtected\r\n`

Example:

`@MQTT.("spBv1.0/Cities")\t\tCities\tProtected`

This maps the MQTT topic `spBv1.0/Cities` into the folder with display name "Cities".

Demo Workflows

Demo 1: TagProvider Approach (Simplest)

Scenario: Connect to MQTT broker and auto-discover all topics.

Steps:

1. Create TagProvider with MQTT connection
2. Create an Asset Tree folder (for example Plant)
3. Select it and click **Insert data from TagProvider** on the Asset Tree toolbar: the folder linked to the TagProvider (for example ProductionMQTT, linked to the broker root) is created inside it
4. Read topics through the linked folder, with the topic path after it
5. Add historian logging for selected tags
6. Create alarms for critical values

JSON to Import:

json

```
{
  "ObjectIdentification": {"TableType": "UnsTagProviders"},
  "Data": [{
    "Name": "ProductionMQTT",
    "Protocol": "MQTT",
    "PrimaryStation": "mqtt.factory.local;1883;FrameworX-001;;;;;None;False;;AtLeastOnce;10;False;False;",
    "Separators": "BranchSeparator=/;AttributeSeparator=/;",
    "Description": "Production floor MQTT broker"
  }]
}
```

Access pattern: `Asset("/Plant/ProductionMQTT/line1/tank1/temperature")` in displays and expressions; `TK.Data.AssetValue` with the same path in C# or VB.NET code.

Demo 2: Device Module Approach (Explicit Mapping)

Scenario: Map specific MQTT topics to local Tags with full control.

Steps:

1. Create local Tags for data points
2. Create MQTT Channel
3. Create Node with broker connection
4. Create Points mapping Tags to topics
5. Add historian and alarms

JSON to Import (in order):

1. Tags:

json

```
{
  "ObjectIdentification": {"TableType": "UnsTags"},
  "Data": [
    {"Name": "Temperature", "Type": 153, "Path": "\\Line1\\Tank1", "Description": "Tank temperature", "Units":
"°C"},
    {"Name": "Pressure", "Type": 153, "Path": "\\Line1\\Tank1", "Description": "Tank pressure", "Units": "bar"},

    {"Name": "Level", "Type": 153, "Path": "\\Line1\\Tank1", "Description": "Tank level", "Units": "%"}
  ]
}
```

2. Channel:

json

```
{
  "ObjectIdentification": {"TableType": "DevicesChannels"},
  "Data": [{
    "Name": "MQTTBroker",
    "Protocol": "MQTT",
    "ProtocolOptions": "TimePublishRate=500;StoreAndForward=false",
    "Description": "Factory MQTT connection"
  }]
}
```

3. Node:

json

```
{
  "ObjectIdentification": {"TableType": "DevicesNodes"},
  "Data": [{
    "Name": "FactoryBroker",
    "Channel": "Channel.MQTTBroker",
    "PrimaryStation": "mqtt.factory.local;1883;FrameworX-001;;;None;;AtLeastOnce;10;",
    "Description": "Production MQTT broker"
  }]
}
```

4. Points:

json

```
{
  "ObjectIdentification": {"TableType": "DevicesPoints"},
  "Data": [
    {"TagName": "Tag.Line1.Tank1.Temperature", "Node": "Node.FactoryBroker", "Address": "line1/tank1
/temperature", "AccessType": "Read"},
    {"TagName": "Tag.Line1.Tank1.Pressure", "Node": "Node.FactoryBroker", "Address": "line1/tank1/pressure", "A
ccessType": "Read"},
    {"TagName": "Tag.Line1.Tank1.Level", "Node": "Node.FactoryBroker", "Address": "line1/tank1/level", "AccessT
ype": "Read"}
  ]
}
```

Demo 3: Sparkplug B Collector

Scenario: Auto-discover Sparkplug B edge nodes and metrics.

Steps:

1. Create MQTTspB Channel (Collector mode)
2. Create Node with broker connection
3. Sparkplug metrics auto-discovered as Tags
4. Configure historian for discovered tags

JSON to Import:

json

```
{
  "ObjectIdentification": {"TableType": "DevicesChannels"},
  "Data": [{
    "Name": "SparkplugCollector",
    "Protocol": "MQTTspB",
    "ProtocolOptions": "Type=Collector;TimePublishRate=500;PublishDBIRTHAfterNBIRTH=true",
    "Description": "Sparkplug B edge node collector"
  }]
}
```

json

```
{
  "ObjectIdentification": {"TableType": "DevicesNodes"},
  "Data": [{
    "Name": "SparkplugBroker",
    "Channel": "Channel.SparkplugCollector",
    "PrimaryStation": "mqtt.factory.local;1883;FrameworX-Collector;;;;;None;False;;AtLeastOnce;10;False;",
    "Description": "Sparkplug B broker connection"
  }]
}
```

Comparison: TagProvider vs Device Module

Aspect	TagProvider	Device Module
Setup	Single object	Channel + Node + Points
Discovery	Automatic	Manual mapping
Tag naming	Provider namespace	Local tags
Control	Less (auto)	Full (explicit)
Best for	UNS integration, browsing	SCADA, specific tags
Access pattern	`Asset("/<folder>/<linked folder>/topic/path")`	`Tag.Path.Name`

TableType Reference (Updated)

TableType	Object Type	Module
`UnsTagProviders`	TagProvider	UNS
`UnsAssetTree`	AssetTree folder	UNS
`UnsTags`	Tag	UNS
`DevicesChannels`	Channel	Devices
`DevicesNodes`	Node	Devices
`DevicesPoints`	Point	Devices

MCP Context Hints for AI

When given MQTT topic documentation:

1. **Analyze topic structure**
   - Identify hierarchy pattern (e.g., `site/area/equipment/metric`)
   - Note data types if documented
   - Identify which topics are critical (need alarms)
2. **Choose approach**
   - **TagProvider** if: Auto-discovery wanted, UNS integration focus
   - **Device Module** if: Specific tags needed, traditional SCADA
3. **Create Tag hierarchy** (Device Module approach)
   - Map topic hierarchy to folder path: `site/area/equipment \Site\Area\Equipment`
   - Create Tags matching metrics
4. **Configure connection**
   - Use broker address, port, credentials from documentation
   - Set appropriate QoS level
5. **Add historian logging**
   - Use `add_tags_to_historian` for tags requiring time series storage. Confirm specification with user.
   - Default to Table1 unless specific retention customization is needed
   - Present Alarm Items created on Designer UI for user approval.
6. **Add alarms**
   - Use `add_tags_to_alarms` for critical measurements

- Choose appropriate condition (HiHi, LoLo, etc.)
- Set setpoints from process documentation
- Present Alarm Items created on Designer UI for user approval.