Skill MQTT Device Module

Create local UNS tags from MQTT topics via the Device Module pipeline (Channel, Node, Points). Covers flat MQTT (one tag per topic) and SparkplugB (UserTypes required). Use when you need alarms, historian, or scripting on MQTT data.

Prerequisite: Read Skill MQTT Integration first to confirm Device Module is the right path. If you only need monitoring/visualization without alarms, use Skill MQTT TagProvider instead.

When to Use This Skill

Use when:

- The user needs **alarms** on MQTT data points
- The user needs **historian logging** of MQTT values
- The user needs **scripts** that reference MQTT data by tag name
- The user wants **structured UserTypes** (UDTs) for grouped device data
- The application requires **explicit control** over which topics become tags
- The broker publishes **SparkplugB** data (Device Module with UserTypes is the only correct path for SpB device mapping)

Do NOT use when:

- The user wants dynamic namespace browsing with AssetTree navigation (use Skill MQTT TagProvider)
- The data structure is unknown and the user wants to explore first — use Designer's DataExplorer MQTT Tools to browse, then return to this skill
- The user only needs monitoring dashboards without alarms or historian

Prerequisites

- Solution must be open
- MQTT broker accessible (local or remote)
- Know the topic structure (or browse it first with DataExplorer MQTT Tools)
- For SparkplugB: know the GroupID/NodeID/DeviceID hierarchy and variable list

Decision: MQTT Flat vs SparkplugB

Aspect	MQTT (flat)	MQTTspB (SparkplugB)
Protocol name	`MQTT`	`MQTTspB`
Interface	`Custom`	`TCPIP`
When to use	Generic MQTT brokers, custom topic hierarchies	SparkplugB-compliant edge nodes
Tag mapping	One simple tag per topic	**REQUIRES UserTypes** — simple tags NOT supported
PrimaryStation	9 semicolon-separated fields	15 semicolon-separated fields
Protocol Options	`TimePublishRate;StoreAndForward`	`Type;TagProperties;TimePublishRate;StoreAndForward;PublishDBIRTHAfterNBIRTH;UseParenthesesForArray;PublishUDTsAsSimpleTags`
Address format	MQTT topic path	SparkplugB path: `GroupId/NodeId/DeviceId`

Critical rule: Always call `list_protocols('mqtt')` or `list_protocols('MQTTspB')` before writing DevicesChannels or DevicesNodes. Never guess PrimaryStation format.

Path A: MQTT Flat — One Tag Per Topic

Use when connecting to a standard MQTT broker publishing flat topics (not SparkplugB).

Step 1: Get protocol schemas

```
list_protocols('mqtt')
get_table_schema('DevicesChannels,DevicesNodes,DevicesPoints')
```

Step 2: Create Channel + Node

MQTT flat PrimaryStation — 9 fields: `BrokerURL;Port;ClientID;UserName;Password;NetworkSecurity;X509Certificate;QoS;KeepAlive`

```
{
  "tables": [
    {
      "table_type": "DevicesChannels",
      "data": [{
        "Name": "MQTT_Channel",
        "Protocol": "MQTT",
        "Interface": "Custom",
        "ProtocolOptions": "500;false",
        "Description": "MQTT broker connection"
      }]
    },
    {
      "table_type": "DevicesNodes",
      "data": [{
        "Name": "Broker1",
        "Channel": "MQTT_Channel",
        "PrimaryStation": "localhost;1883;_SolutionName_-_Guid_;;;None;;AtLeastOnce;10",
        "Description": "Local MQTT broker"
      }]
    }
  ]
}
```

Common PrimaryStation examples:

Scenario	PrimaryStation
Local broker, anonymous	`localhost;1883;_SolutionName_-_Guid_;;;None;;AtLeastOnce;10`
Remote broker	`mqtt.factory.local;1883;_SolutionName_-_Guid_;;;None;;AtLeastOnce;10`
With authentication	`broker.cloud.io;8883;MyClient;admin;pass123;TLSv.1.2;;AtLeastOnce;30`

Step 3: Create UNS tags

Tags must exist before creating DevicesPoints:

```
{
  "table_type": "UnsTags",
  "data": [
    {"Name": "Plant/Line1/Temperature", "Type": "Double", "Description": "Line 1 temperature"},
    {"Name": "Plant/Line1/Pressure", "Type": "Double", "Description": "Line 1 pressure"},
    {"Name": "Plant/Line1/Running", "Type": "Digital", "Description": "Line 1 run status"}
  ]
}
```

Step 4: Map tags to topics via DevicesPoints

Address = the MQTT topic path. The driver subscribes and updates tags automatically at runtime.

```
{
  "table_type": "DevicesPoints",
  "data": [
    {"TagName": "Plant/Line1/Temperature", "Node": "Broker1", "Address": "plant/line1/temperature",
"AccessType": "ReadOnly"},
    {"TagName": "Plant/Line1/Pressure", "Node": "Broker1", "Address": "plant/line1/pressure", "AccessType":
"ReadOnly"},
    {"TagName": "Plant/Line1/Running", "Node": "Broker1", "Address": "plant/line1/running", "AccessType":
"ReadWrite"}
  ]
}
```

After this, configure alarms and historian on these tags using the standard workflow (AlarmsItems, HistorianTags).

Path B: MQTTspB (SparkplugB) — UserTypes Required

Use when connecting to a SparkplugB-compliant broker. **The MQTTspB protocol does NOT support simple tags in DevicesPoints.** You MUST create a UserType first, then create tags typed to that UserType, then map the main tags (not individual members) as DevicesPoints. Members auto-expand at runtime.

This is a platform constraint from the MQTTspB Collector Connector, not optional intelligence.

Step 1: Get protocol schemas

```
list_protocols('MQTTspB')
get_table_schema('UnsUserTypes,UnsTags,DevicesChannels,DevicesNodes,DevicesPoints')
```

Step 2: Create UserType matching SparkplugB variables

Analyze the SparkplugB device structure. Map SparkplugB types to FrameworX types: double to Double, boolean to Digital, string to Text, integer to Integer.

Example using the built-in simulator (Cities/Barcelona/Panel_1 has 8 variables):

```
{
  "table_type": "UnsUserTypes",
  "data": [{
    "Name": "SolarPanel",
    "Description": "SparkplugB solar panel device model",
    "Members": [
      {"Name": "Current", "Type": "Double"},
      {"Name": "Latitude", "Type": "Double"},
      {"Name": "Longitude", "Type": "Double"},
      {"Name": "Name", "Type": "Text"},
      {"Name": "Power", "Type": "Double"},
      {"Name": "State", "Type": "Digital"},
      {"Name": "Temperature", "Type": "Integer"},
      {"Name": "Voltage", "Type": "Double"}
    ]
  }]
}
```

Step 3: Create tag instances from UserType

One tag per SparkplugB DeviceID. Each tag automatically gets all 8 members:

```
{
  "table_type": "UnsTags",
  "data": [
    {"Name": "SolarFarm/Barcelona/Panel_1", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Barcelona/Panel_2", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Barcelona/Panel_3", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Barcelona/Panel_4", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Bilbao/Panel_1", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Bilbao/Panel_2", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Bilbao/Panel_3", "Type": "SolarPanel"},
    {"Name": "SolarFarm/Bilbao/Panel_4", "Type": "SolarPanel"}
  ]
}
```

Step 4: Create Channel + Node

MQTTspB ProtocolOptions — 7 fields:`Type;TagProperties;TimePublishRate;StoreAndForward;PublishDBIRTHAfterNBIRTH;UseParenthesesForArray;PublishUDTsAsSimpleTags`

MQTTspB PrimaryStation — 15 fields:`BrokerURL;Port;ClientID;UserName;Password;X509Certificate;CertificatePsw;NetworkSecurity;WebSocket;CollectorID;QoS;KeepAlive;ReportOutOfRangeByException;DisableRebirthRequest;PublishSTATE`

```
{
  "tables": [
    {
      "table_type": "DevicesChannels",
      "data": [{
        "Name": "SpB_Channel",
        "Protocol": "MQTTspB",
        "Interface": "TCPIP",
        "ProtocolOptions": "Collector;;500;false;true;false;false",
        "Description": "SparkplugB Collector"
      }]
    },
    {
      "table_type": "DevicesNodes",
      "data": [{
        "Name": "SpB_Broker",
        "Channel": "SpB_Channel",
        "PrimaryStation": "localhost;1883;_SolutionName_-_Guid_;;;;;None;false;;AtLeastOnce;10;false;false;
true",
        "Description": "Built-in broker with SparkplugB simulator"
      }]
    }
  ]
}
```

Step 5: Map main tags as DevicesPoints

For MQTTspB, map only the **main tag** (the UserType instance). Members auto-expand at runtime. Address = `GroupId/NodeId/DeviceId`:

```
{
  "table_type": "DevicesPoints",
  "data": [
    {"TagName": "SolarFarm/Barcelona/Panel_1", "Node": "SpB_Broker", "Address": "Cities/Barcelona/Panel_1",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Barcelona/Panel_2", "Node": "SpB_Broker", "Address": "Cities/Barcelona/Panel_2",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Barcelona/Panel_3", "Node": "SpB_Broker", "Address": "Cities/Barcelona/Panel_3",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Barcelona/Panel_4", "Node": "SpB_Broker", "Address": "Cities/Barcelona/Panel_4",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Bilbao/Panel_1", "Node": "SpB_Broker", "Address": "Cities/Bilbao/Panel_1",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Bilbao/Panel_2", "Node": "SpB_Broker", "Address": "Cities/Bilbao/Panel_2",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Bilbao/Panel_3", "Node": "SpB_Broker", "Address": "Cities/Bilbao/Panel_3",
"AccessType": "ReadOnly"},
    {"TagName": "SolarFarm/Bilbao/Panel_4", "Node": "SpB_Broker", "Address": "Cities/Bilbao/Panel_4",
"AccessType": "ReadOnly"}
  ]
}
```

At runtime, `Tag.SolarFarm/Barcelona/Panel_1.Current` receives the live Current value from the SparkplugB metric. All 8 members of each tag update automatically.

Hybrid Pattern: TagProvider + Device Module

A common production pattern combines both approaches:

1. **TagProvider** (Skill MQTT TagProvider) for full dynamic namespace browsing in the AssetTree
2. **Device Module** (this skill) for the specific tags that need alarms, historian, or script logic

The two coexist. TagProvider shows everything dynamically. Device Module tags provide the persistent infrastructure for operational logic. Example:

```
{
  "tables": [
    {
      "table_type": "UnsTagProviders",
      "data": [{
        "Name": "SolarBroker",
        "Protocol": "MQTTspB",
        "PrimaryStation": "localhost;1883;_SolutionName_-_Guid_;;;;;None;false;;AtLeastOnce;10;false;false;
true",
        "Separators": "BranchSeparator=/;AttributeSeparator=/;",
        "Description": "Dynamic browsing of all solar data"
      }]
    },
    {
      "table_type": "UnsAssetTree",
      "data": [
        {"Name": "SolarFarm"},
        {
          "Name": "AllData",
          "Parent": "SolarFarm",
          "TagProviderLink": "SolarBroker.(\"spBv1.0/Cities\")",
          "Description": "Full broker namespace for navigation"
        }
      ]
    }
  ]
}
```

Then add Device Module tags only for the critical points where you configure alarms (e.g., high temperature) and historian (e.g., power trending).

Common Pitfalls

Mistake	Why it happens	How to avoid
Using simple tags with MQTTspB DevicesPoints	Not knowing the platform constraint	MQTTspB REQUIRES UserTypes. Create UserType first, then use typed tags in Points.
Mapping individual UserType members as Points	Expecting per-member mapping like MQTT flat	For MQTTspB, map only the main tag. Members auto-expand at runtime.
Wrong Interface for MQTTspB	Assuming Custom like MQTT flat	MQTTspB uses Interface `TCPIP`. MQTT flat uses `Custom`. Check list_protocols.
PrimaryStation field count mismatch	MQTT has 9 fields, MQTTspB has 15	Call list_protocols for the exact format.
Guessing Protocol name	Writing "SparkplugB" or "MQTT_SpB"	Exact names: `MQTT` and `MQTTspB`. Call list_protocols.
Creating DevicesPoints before tags	Tags are a dependency of Points	Create UnsTags first, then DevicesPoints. Multi-table write handles order.
Forgetting ProtocolOptions on Channel	Defaults may not match your needs	MQTTspB default: `Collector;;500;false;true;false;false`. MQTT default: `500;false`.

Quick Reference — MCP Tool Calls

Task	Tool call
Get MQTT flat protocol	`list_protocols('mqtt')`
Get SparkplugB protocol	`list_protocols('MQTTspB')`
Get device table schemas	`get_table_schema('DevicesChannels,DevicesNodes,DevicesPoints')`
Get UserType schema	`get_table_schema('UnsUserTypes')`
Create Channel + Node	`write_objects(data='{"tables":[...]}')`
Create UserType	`write_objects('UnsUserTypes', data=[...])`
Create Tags	`write_objects('UnsTags', data=[...])`
Create Points	`write_objects('DevicesPoints', data=[...])`
Start built-in broker	`designer_action('builtin_broker', 'start')`
Start simulator	`designer_action('simulator', 'start')`

See also

- Skill MQTT Integration. Router: choose TagProvider vs Device Module before loading detailed steps.
- Skill MQTT TagProvider. Dynamic discovery path: TagProvider + AssetTree + Asset() UI.
- Skill Discovery Services. Conceptual foundation: three connection patterns.
- MQTT Client Connector. MQTT flat protocol, station format, address syntax.
- MQTT SparkplugB Collector Connector. MQTTspB protocol, UserType requirement, tag mapping.
- MQTT SpB Publisher Simulator. Built-in simulator configuration.
- MQTT Tools Reference. Data Explorer MQTT browsing and export.
- Devices Module Reference. Channel/Node/Points architecture.