# Sorba AI Analytics Connector

Run SORBA SDE machine learning models on Unified Namespace data over MQTT.

- Name: **SorbaAI**
- Version: 1.0.0.0
- Protocol: MQTT
- Interface: TCP/IP
- Runtime: .NET 2.0 (Multiplatform)
- Configuration:
  - UNS / TagProviders

Download SorbaAI solution template: SORBA.dbsln (version 10.1.6)


Overview
Architecture
When to use Sorba AI, ML.NET or Python
Prerequisites
Setup
Station Configuration
Algorithms
Model Structure
Training a Model
Outputs
Solution Template
Multiple Stations
Monitoring Traffic
Troubleshooting


---

## Overview

The SorbaAI connector links the Unified Namespace to SORBA SDE, an industrial machine learning platform. It exposes a SORBA model as a TagProvider namespace: live tag values go to SORBA for training and inference, and SORBA sends back predictions, model-quality metrics and anomaly indicators as tags you can use in displays, alarms and scripts.

Supported model types: Regression, Classification, Clustering, Forecasting, Digital Twin and Optimization.

## Architecture

```
Input tags (UNS) --+
                   +--> SorbaAI TagProvider (Runtime) <--> MQTT broker <--> SORBA SDE
Historian ---------+
```

- The MQTT broker is external to both products: usually the broker that comes with SORBA SDE, but any MQTT 3.1.1 broker reachable from both sides works.
- The connector publishes the model structure and the live input values. SORBA trains the model and publishes predictions and diagnostics back.
- When SORBA asks for training history, the connector reads it from the Historian module, whatever storage location the solution uses. The connector keeps no history of its own.
- The station's **Instance ID** identifies the gateway to SORBA and must match an Integration record in SORBA SDE.

### Designer and Runtime

Both the Designer and the Runtime load the station, but only the **Runtime** acts as the SORBA gateway: it publishes the structure and values, and answers SORBA's history requests.

The Designer connects as a read-only mirror. In **Unified Namespace > Asset Tree** it shows the model tree, the SORBA outputs, `output_message` and the values the Runtime sends, but the SorbaAI tags cannot be edited there: the Designer refuses every write to them and never sends anything to SORBA. With only the Designer open, SORBA receives nothing from the gateway.

Set the model fields and run training from the Runtime, for example from a display (see *Training a Model*).

### MQTT topics

All topics are under `sorba_framewor_x/<Instance ID>/`.

| Topic | Direction | Content |
|---|---|---|
| `tags/structures` | To SORBA | Model structure, every 10 s. |
| `udt_definitions` | To SORBA | UDT definitions, every 10 s. |
| `tags/inputs/realtime` | To SORBA | Current values of all model tags except OUTPUTS, every 1 s. |
| `tags/inputs/historical/request` | From SORBA | Request for training history. |
| `tags/inputs/historical` | To SORBA | Historian samples of each input for the requested window. |

| | | |
|---|---|---|
| `tags/outputs/realtime` | From SORBA | Predictions and diagnostics. |
| `tags/outputs/structure` | From SORBA | Structure of the OUTPUTS folder. |

## When to use Sorba AI, ML.NET or Python

FrameworX offers more than one way to add machine learning to a solution. Use this as a guide:

| Option | Best for | What you need |
|---|---|---|
| **Sorba AI** | Users who want AI results without building models. Training, model management and predictions are handled by SORBA SDE, and the results appear as tags in the UNS. | SORBA SDE and its license. No ML coding. |
| **ML.NET (built-in)** | Integrators who want models running inside FrameworX, with no external process. Good for anomaly detection, forecasting, regression and classification on tag data. | C# Script Classes. See Skill ML.NET and AI ML Integration Connector. |
| **Python scripts** | Teams that already have models or libraries in Python and only need an engine to run them with live data. | Python 3.7 libraries called from FrameworX scripts. See Python and .NET Integration Reference. |

Rule of thumb: if you want to *consume* AI, use Sorba AI. If you want to *build* the model yourself, use ML.NET or Python.

## Prerequisites

- SORBA SDE running and reachable on the network.
- An MQTT broker reachable from SORBA SDE and from the Runtime computer: the broker included with SORBA SDE (default port 1883), the built-in MQTT broker, or any MQTT 3.1.1 broker.
- The Runtime running. Only the Runtime exchanges data with SORBA.
- For training: active historization of every input tag in the Historian module, with data covering the training window.

The connector is installed with the product. There is no separate installation step.

## Setup

The configuration has two sides that must match: an Integration record in SORBA SDE and a SorbaAI TagProvider in the solution. Without the matching Integration record, SORBA ignores the gateway's MQTT traffic.

### SORBA SDE side

In the SORBA SDE web UI, go to **Edit Device > Services > Integrations** and add an Integration with these values:

| Field | Value |
|---|---|
| Integration | `ExternalTool` |
| Name | The Instance ID you will use in the station, for example `frameworx_gateway` (the station default). |
| Server Name | MQTT broker host, for example `localhost` when using the SORBA SDE broker. |
| Port | MQTT broker port (default `1883`). |
| UserName | MQTT broker username. |
| Password | MQTT broker password. |
| Secure communication | Off. The connector does not use TLS. |

Save the record.

### TagProvider in the solution

1. In the Designer, go to **Unified Namespace > TagProvider Services** and create a new TagProvider.
2. In the dialog, set **Service** to **Extension Module**, set **Provider** to **SorbaAI Connector**, and enter a **Name**.
3. Fill in the station editor, including the **Input Sources** (see *Station Configuration*).
4. In **Unified Namespace > Asset Tree**, link the new TagProvider to a folder.
5. Save the solution and start the Runtime.

When the Runtime starts, the station connects to the broker and starts publishing to SORBA. The linked folder shows the model tree under the Model Name: `INPUTS`, `OUTPUTS`, `CONFIG` and, for most algorithms, `TARGET`.

⚠ The dialog has no Test button for this connector. To check the connection, watch the topics with an MQTT viewer (see *Monitoring Traffic*).

The connector does not reconnect to the MQTT broker automatically. Start the broker and SORBA SDE before the Runtime. If the broker restarts, restart the Runtime or the station.

## Station Configuration

The station editor stores the connection as a semicolon-separated string of 13 positional fields. The defaults work for a local SORBA SDE with its own broker.

| # | Field | Default | Description |
|---|---|---|---|
| 1 | Broker Host | `localhost` | Hostname or IP of the MQTT broker. |
| 2 | Port | `1883` | MQTT broker port. |
| 3 | Instance ID | `frameworx_gateway` | Identifies this gateway to SORBA and forms the topic root `sorba_framewor_x/<Instance ID>/`. Must match the Name of the Integration record in SORBA SDE. Characters other than letters, digits and underscore are replaced by _, and `sbr_` is added in front if the result is empty, a single character, or starts with a digit. Use letters, digits and underscores, starting with a letter, so the ID is used exactly as typed. |
| 4 | Algorithm | `Regression` | Model type: `Regression`, `Classification`, `Clustering`, `Forecasting`, `DigitalTwin` or `Optimization`. Defines the TARGET folder (see *Algorithms*). |
| 5 | Model Name | `MODEL` | Root of the model tree, for example `MODEL/CONFIG/...`. SORBA also uses it as the model name in its messages. |
| 6 | Username | empty | MQTT broker username. Leave empty for anonymous brokers. |
| 7 | Password | empty | MQTT broker password. |
| 8-10 | TLS CA PEM, TLS Client Cert PEM, TLS Client Key PEM | empty | Reserved, not used. Leave them empty. |
| 11 | Output TTL Seconds | `30` | Shown as **Output TTL (s)** in the editor. An output that SORBA stops refreshing for longer than this is removed from the model tree. |
| 12 | Reserved | empty | Leave empty. |
| 13 | Input Sources | empty | Comma-separated tag paths, for example `Tag.Machine.Power`. Managed with the tag picker in the station editor (see below). |

### Input Sources

Input Sources lists the tags streamed to SORBA as model inputs. Manage the list in the station editor: **Add Tag...** opens a tag picker and **Remove** deletes the selected rows. The list shows each **Source Tag** and the **Input Name** it gets in the model. Inputs are added only through the picker.

Each source becomes `<Model Name>/INPUTS/<Input Name>`. The Input Name is the tag path with the `Tag.` prefix removed and dots and other separators replaced by _. Duplicate names get a suffix: _2, _3 and so on.

| Source Tag | Input Name | Model tag |
|---|---|---|
| `Tag.Boiler1.Temperature` | `Boiler1_Temperature` | `<Model Name>/INPUTS/Boiler1_Temperature` |
| `Tag.Machine.Power` | `Machine_Power` | `<Model Name>/INPUTS/Machine_Power` |

Input values are read every second and sent to SORBA.

ⓘ Use the Input Name, not the source path, wherever SORBA expects an input, such as `TARGET/response_variable` and the optimization lists. For example, `Machine_Power`, not `Tag.Machine.Power`.

### Example

| Scenario | Configuration string |
|---|---|
| Local broker, Regression model with three inputs | `localhost;1883;frameworx_gateway;Regression;MODEL;;;;;;30;;Tag.Boiler1.Temperature, Tag.Boiler1.Pressure,Tag.Pump1.FlowRate` |

## Algorithms

The Algorithm setting selects the model type and the content of the TARGET folder.

| Algorithm | Use case | TARGET folder |
|---|---|---|
| Regression | Predict a continuous variable from other inputs, for example Temperature from Pressure and Flow. | `response_variable` (Text) |
| Classification | Predict a discrete class from inputs, for example normal or fault. | `response_variable` (Text) |
| Clustering | Group similar samples without labels. | No TARGET folder |
| Forecasting | Predict future values of a time series. | `response_variable` (TextArray), plus `lag_steps`, `future_steps` and `window_length` under `CONFIG/TRAINING` |
| DigitalTwin | Multi-output predictive model of a physical asset. | `response_variable` (TextArray) |
| Optimization | Recommend control-variable settings that minimize or maximize an objective. | `control_variables`, `independent_variables`, `optimization_goals`, `optimization_variables` (TextArray) |

## Model Structure

The model tree appears under the Model Name. Layout for a Regression model named `MODEL`:

```
MODEL/
+-- CONFIG/
|   +-- RUNTIME/
|   |   +-- autolearning_status
|   |   +-- autolearning_time_range(min)
|   |   +-- autolearning_trigger
|   +-- TRAINING/
|       +-- DATASET/
|       |   +-- discard_nulls
|       |   +-- replace_with_last_value
|       |   +-- replace_with_value
|       +-- SYNTHETIC_TAGS/
|       +-- algorithm_type
|       +-- end_date
|       +-- history_storage_provider
|       +-- initialize_training
|       +-- output_message
|       +-- start_date
|       +-- time_zone
+-- INPUTS/             <- one tag per Input Source
+-- OUTPUTS/            <- created as SORBA sends results
+-- TARGET/
    +-- response_variable
```

### TARGET

| Algorithm | Tag | Type | Value |
|---|---|---|---|
| Regression, Classification | `response_variable` | Text | Input Name of the variable to predict, for example `Machine_Power`. |
| Forecasting, DigitalTwin | `response_variable` | TextArray | Input Names of the variables to predict. Enter them comma-separated, for example `Temperature, Pressure`. |
| Optimization | `optimization_variables` | TextArray | Variables SORBA tries to minimize or maximize. |
| Optimization | `optimization_goals` | TextArray | Optimization direction: `minimum` or `maximum`. |
| Optimization | `control_variables` | TextArray | Variables SORBA may recommend changes to. |
| Optimization | `independent_variables` | TextArray | Context variables SORBA reads but does not adjust. |

For Optimization, use each input in only one of `optimization_variables`, `control_variables` and `independent_variables`.

### CONFIG/TRAINING

| Tag | Type | Default | Description |
|---|---|---|---|
| `time_zone` | Text | `UTC` | Time zone SORBA assumes for the gateway. Keep `UTC`: the connector sends dates and timestamps in UTC. |
| `algorithm_type` | Text | From Algorithm | Set by the connector from the station's Algorithm. |
| `output_message` | Text | empty | Status and error messages from SORBA (see *Training a Model*). |
| `initialize_training` | Digital | `false` | Set to `true`, after the other fields, to start training. |
| `start_date` | DateTime | empty | Start of the training window, in local time. |
| `end_date` | DateTime | empty | End of the training window, in local time. |
| `history_storage_provider` | Text | `[default]` | Reserved. Keep `[default]`. |
| `lag_steps`, `future_steps`, `window_length` | Integer | `6`, `3`, `18` | Forecasting only: number of past samples used for each prediction, number of steps ahead to predict, and length of the analysis window. |

### CONFIG/TRAINING/DATASET

How SORBA handles null samples during training.

| Tag | Type | Default | Description |
|---|---|---|---|
| `discard_nulls` | Digital | `false` | Discard samples with null values. |
| `replace_with_last_value` | Digital | `false` | Replace null samples with the last good value of the same tag. |
| `replace_with_value` | Float | `0.0` | Constant used to replace null samples. |

### CONFIG/TRAINING/SYNTHETIC_TAGS

Folder for derived features. Not covered by this page; contact Tatsoft Support before using it.

### CONFIG/RUNTIME

| Tag | Type | Default | Description |
|---|---|---|---|
| `autolearning_status` | Text | `Standby` | Autolearning state. |
| `autolearning_trigger` | Digital | `true` | Autolearning trigger. |
| `autolearning_time_range(min)` | Integer | `10` | Autolearning time range, in minutes. |

These settings are sent to SORBA with the rest of the model.

## Training a Model



All training is done from the Runtime. The SorbaAI tags in the Designer Asset Tree are read-only: write the training fields from a Runtime display or script.

1. Check that the Historian has data for every input tag over the window you want to use.
2. Set `TARGET/response_variable` to the Input Name of one of the model inputs, for example `Machine_Power`. For other algorithms, fill the TARGET tags listed in *Model Structure*.
3. Set `CONFIG/TRAINING/start_date` and `end_date` in local time, as date values (see *Dates and times*). The connector sends them to SORBA in UTC. Keep `CONFIG/TRAINING/time_zone` = `UTC`.
4. Optionally, set the `CONFIG/TRAINING/DATASET` options.
5. Set `CONFIG/TRAINING/initialize_training` = `true`. Do this last.

SORBA validates the fields and reports progress in `CONFIG/TRAINING/output_message`. After `initialize_training` is set, SORBA requests the training history within a couple of seconds, the Runtime answers with the Historian samples of each input for the window, and the outputs appear under `OUTPUTS` after a few minutes.

### Status messages

Messages seen in `output_message` while setting up a training run, in the order they usually appear:

| Message | Meaning and action |
|---|---|
| `Target response variable "" was not found in the inputs in instance <InstanceId> and asset <Model>` | `response_variable` is empty or is not an Input Name. Set it to one of the inputs. |
| `Training start date is not in the correct format in instance <InstanceId> and asset <Model>` | The training dates are not set or not valid. Set `start_date` and `end_date`. |
| `Waiting for training confirmation in instance <InstanceId> and asset <Model>` | The fields are valid. Set `initialize_training` = `true`. |
| `No data found for training ...` | The Historian returned no samples for the training window. Check the values in `start_date` and `end_date`, including day and month, and that every input tag is historized over that window. |
| `Model <Model> is already trained` | SORBA already trained this model and does nothing. See *Training again*. |

### Dates and times

`start_date` and `end_date` take local time and reach SORBA in UTC. For example, 15 May 2026 16:30 on a computer at UTC-3 reaches SORBA as `May 15, 2026, 7:30:00 PM`.

- In a display, bind a DateTimePicker to each tag and set the time as well as the date. A date set without a time is sent as 00:00.
- In code, assign `DateTime` values, not text. Text is converted to a date before it reaches the connector, and the day and month order may not be the one you expect: `05/10/2026` can arrive as May 10.

Example, in the code-behind of a display button: the last 12 hours as the training window. `/Folder1/Alias/MODEL` is the path the template uses: the folder Folder1, then Alias, the folder inside it linked to the station, then the Model Name.

```
DateTime end = DateTime.Now;
TK.Asset("/Folder1/Alias/MODEL/CONFIG/TRAINING/start_date") = end.AddHours(-12);
TK.Asset("/Folder1/Alias/MODEL/CONFIG/TRAINING/end_date") = end;
```

### Training again

SORBA trains a model once. If a trained model receives `initialize_training` again, `output_message` shows `Model <Model> is already trained` and nothing happens. To train again, set a new Model Name in the station and update the displays bound to the old model paths.

To avoid editing each binding, build the paths from a tag that holds the Model Name, for example `TK.Asset("/Folder1/Alias/" + @Tag.ModelName + "/OUTPUTS/...")`.

## Outputs

The `OUTPUTS` folder is empty until SORBA sends results. Each output is created the first time SORBA sends it, as a numeric (Double) or Digital tag, and is removed if SORBA stops refreshing it for longer than **Output TTL Seconds** (default 30).

Predictions update about every 10 seconds and are timestamped when they arrive. In a trend, the prediction curve therefore appears shifted to the right (later) relative to the inputs. To compare prediction and actual value, use a trend window of several minutes, or inputs that vary slowly.

The set of outputs depends on the algorithm. For Regression, expect the outputs below, where `<Target>` is the value of `TARGET` `/response_variable` and `<Input>` stands for each of the other inputs.

| Output | Meaning |
|---|---|
| `<Target>_Prediction` | Predicted value of the response variable. |
| `<Target>_Prediction_Error` | Difference between predicted and actual value. |
| `<Target>_Prediction_Upper_Band`, `<Target>_Prediction_Lower_Band` | Upper and lower bounds of the confidence band. |
| `<Target>_Prediction_Drift`, `<Target>_Prediction_Drift_Score` | Prediction drift flag and score. |
| `<Target>_Target_Drift`, `<Target>_Target_Drift_Score` | Drift flag and score of the response variable. |
| `<Input>_Tag_Ranking_Percent` | Contribution of the input to the prediction (feature importance). |
| `<Input>_Tag_Ranking_p_value` | Statistical significance of that contribution. |
| `<Input>_outlier` | Set when the input is currently an outlier. |

| | |
|---|---|
| `<Input>_Data_Drift`, `<Input>_Data_Drift_Score` | Drift flag and score of the input. |
| `Dataset_Drift` | Drift flag for the whole dataset. |
| `Best_Score_Training`, `Best_Score_Realtime` | Model quality score on training data and on live data. |
| `Model_Status_1_ok_2_error_3_warning` | Model status: 1 OK, 2 error, 3 warning. |
| `Data_Status_1_ok_2x_error_3x_warning` | Data status: 1 OK, 2x error, 3x warning. |
| `Anomaly_Score_Percent` | Current anomaly score, in percent. |
| `Anomaly_Detection_Analog` | Analog anomaly signal. |
| `Anomaly_Detection_Boolean` | Anomaly flag. |
| `Anomaly_Detection_High_Limit_Threshold`, `Anomaly_Detection_Low_Limit_Threshold` | Thresholds that set the anomaly flag. |
| `Anomaly_Detection_Notification_Warning`, `Anomaly_Detection_Notification_Critical` | Warning and critical notification flags. |

Start with `<Target>_Prediction` (the predicted value), `<Target>_Prediction_Error` (how far it is from the actual value) and `Model_Status_1_ok_2_error_3_warning` (model health). Plot the prediction together with the Upper and Lower Band outputs to show the confidence band.

## Solution Template

The `SORBA.dbsln` template at the top of this page is a working Regression example. The `SorbaRegression` display shows the prediction, the model status and the drift indicators, and has the buttons that start a training run. The `SorbaOutputs` popup opens from it and lists every output of the model. The SorbaAI TagProvider in the template is set up as described in *Setup*: point the station to your broker and Instance ID before you start the Runtime.

### Binding elements to the model

The displays use no local tags for model values. Each element binds straight to the asset in the TagProvider through its full path, made of three parts:

| Part | In the template |
|---|---|
| Asset Tree folder linked to the station | `/Folder1/Alias` |
| Model Name | The value of `Tag.SorbaModelName` |
| Path in the model tree (see *Model Structure*) | For example `OUTPUTS/Machine_Power_Prediction` |

For example, `/Folder1/Alias/MODEL/OUTPUTS/Machine_Power_Prediction`.

The same path reads and writes. A text or a gauge reads it, and a TextBox, a DateTimePicker or a button writes the TARGET and CONFIG fields, as the training buttons in the template do. Outputs are read-only: only SORBA writes them, and a write to an `OUTPUTS` path is refused.

In the template, elements bind with `TK.Asset("<path>")`, and the DataGrid in `SorbaOutputs` with `Asset("<path>")`. The paths are built from `Tag.SorbaModelName`, so a new Model Name needs no binding changes.

In a DataGrid source, convert the tag to text when you build the path: `Asset("/Folder1/Alias/" + Tag.SorbaModelName.ToString() + "/OUTPUTS")`. Without `.ToString()` the grid stays empty.

### Finding paths with the Object Browser

1. Open the Object Browser (**Tools > Object Browser**).
2. Expand the Asset Tree folder linked to the station, then the Model Name.
3. Select the asset, for example **OUTPUTS > Machine_Power_Prediction**, and copy its path with the button at the top.
4. Paste the path into the element binding.

Output names depend on the inputs and the algorithm, for example `Machine_Power_Prediction` or `Machine_Speed_Data_Drift`. The `OUTPUTS` folder fills after the model is trained, so train it once before you bind the outputs.

### Adapting the template

| Change | What to update |
|---|---|

| | |
|---|---|
| Asset Tree folder | If the station is linked to a folder other than `/Folder1/Alias`, replace that prefix in the bindings of both displays. |
| Model Name | Set `Tag.SorbaModelName` to the station's Model Name. |
| Inputs | The template streams `Tag.Machine.Power`, `Tag.Machine.Current` and `Tag.Machine.Speed`, simulated and historized in the solution. With other inputs the output names change: rebind those elements with the Object Browser and set `TARGET /response_variable` to one of your Input Names. |
| Trend | The pens show the template's inputs and prediction. Replace them with your own. |
| Algorithm | The template is for Regression. Other algorithms have different TARGET fields and outputs (see *Algorithms* and *Outputs*): keep the layout and rebind the elements. |

## Multiple Stations

A solution can have several SorbaAI stations, each managing its own SORBA model.

| Setting | Between stations |
|---|---|
| Broker Host, Port | Same broker or different brokers. |
| Instance ID | **Must be unique.** It defines the MQTT topic namespace of the station. |
| Algorithm | Independent per station. |

Each station needs its own Integration record in SORBA SDE, with the station's Instance ID as Name.

## Monitoring Traffic

Keep an MQTT viewer connected to the same broker as the station, during setup and in operation. It shows the exchange between the gateway and SORBA as it happens.

- MQTT Explorer: free, cross-platform, tree view.
- `mosquitto_sub`, the command-line client that ships with Mosquitto: `mosquitto_sub -h <broker host> -t "sorba_framewor_x/<Instance ID>/#" -v`

Subscribe to `sorba_framewor_x/<Instance ID>/#` to see every topic listed under *MQTT topics*. The viewer tells you whether SORBA is answering at all, which separates a SORBA-side problem from a gateway-side one.

## Troubleshooting

| Symptom | Likely cause | Fix |
|---|---|---|
| The model tree appears, but SORBA receives nothing. The trace shows `MQTT broker not reachable`. | Wrong Broker Host or Port, broker not running, or a firewall blocking the port. | Check host and port, start the broker, and test it with `mosquitto_sub` from the Runtime computer. Then restart the Runtime or the station. |
| Only the Designer is open and SORBA receives nothing. | Only the Runtime acts as the gateway. | Start the Runtime. |
| Editing a SorbaAI tag in the Designer Asset Tree fails with `SorbaAI tags are read-only in the Designer`. | The Designer is a read-only mirror. | Write the value from the Runtime, for example from a display. |
| Traffic stopped after the broker restarted. | The connector does not reconnect automatically. | Restart the Runtime or the station. |
| The station runs but SORBA never responds. | No matching Integration record in SORBA SDE. | Add an Integration whose Name equals the station's Instance ID. |
| `output_message`: `Target response variable "" was not found ...` | `response_variable` is empty or is not an Input Name. | Set it to an Input Name such as `Boiler1_Temperature`, not `Tag.Boiler1.Temperature`. |
| `output_message`: `Training start date is not in the correct format ...` | `start_date` or `end_date` not set or not valid. | Set both dates. See *Dates and times*. |
| `output_message`: `Model <Model> is already trained` | SORBA trains a model once. | Use a new Model Name. See *Training again*. |

| | | |
|---|---|---|
| `output_message`: `No data found for training ...` | No Historian samples in the training window: wrong dates (for example, day and month swapped) or inputs not historized. | Check the dates in `start_date` and `end_date` and the Historian data of each input. See *Dates and times*. |
| Training was started but no outputs appear. | Training still running, or no Historian data for one or more inputs in the window. | Allow a few minutes. Check that every input tag is historized and that the Historian covers the window. Check `output_message` and the MQTT viewer. |
| Outputs appear, then disappear after about 30 s. | SORBA stopped refreshing them and the Output TTL expired. | Check why SORBA stopped publishing, or raise **Output TTL Seconds**. |
| In a trend, the prediction lags the actual value. | Predictions arrive about every 10 s and are timestamped on arrival. | Use a trend window of several minutes, or slowly varying inputs. |
| An OUTPUTS value read with `TK.Asset()` in the Runtime returns empty. | No client uses the tag yet. | Use the tag in a display, script or alarm. Values flow after the first subscription. |

For more detail, the driver writes a diagnostic trace to `C:\Temp\SorbaAI_Trace.log`.

---

### In this section...

---