Predix Modbus TCP Connectivity for NI Linux RT

User manual

Ovak Technologies
2017

Contents

1. Introduction

1.1. Definitions and Acronyms

Predix – is the Industrial Internet Platform that connects data from physical assets to powerful analytics. Predix can operate everywhere industry does, allowing quickly and securely connect their assets, collect data and run applications.

Predix Cloud – is a global secure cloud infrastructure that is optimized for industrial workloads and for meeting regulatory needs.

Predix Services – provides industrial services that developers can use to build, test and run Industrial Internet applications. It also provides a catalog services marketplace where developers can publish their own services as well as consume services from third parties.

Predix Machine – is the software layer responsible for communicating with the industrial asset and the Predix Cloud, as well as running local applications, like edge analytics. This component can be installed on gateways, industrial controllers and sensors.

Predix Connectivity – is for scenarios where a direct Internet connection is not readily available. The service enables machines to talk to the Predix Cloud via a virtual network comprised of cellular, fixed line and satellite technologies.

Data Node – machine data is received in Data Nodes.

Adapter – a component that acquires data from sensors through different protocols like Modbus.

Hoover Spillway – specifies which machine adapter data subscriptions are to be released.

Processor – if a customized logic needs to be injected or if the data needs to be aggregated or filtered, it can be done in processor.

Store and Forward – used to store data temporarily in FIFO order in case the destination is not reachable. The data is stored in an encrypted H2 file based database.

River – component to send data to a destination such as time series database in cloud or MQTT broker.

IIoT – Industrial Internet of Things

Java SE – Java Platform, Standard Edition

JDK – Java Development Kit

SDK – Software Development Kit

IDE – Integrated Development Environment

TCP – Transmission Control Protocol

RTU – Remote Terminal Unit

UAA – User Account and Authentication

1.2. Overview

The Predix Platform enables to create applications with an Industrial Internet focus and allows to manage and scale those applications as they are consumed by end users.

Predix provides Catalog Services that can be used as building blocks in the application. This will save a lot of time so you can avoid reinventing basic features that application requires. Services such as *Postgres*, *Redis cache*, *Blobstore*, and *Rabbit MQ* are the basics you will need when architecting and building your application. Other services such as *Predix Time Series*, *Predix Asset*, *Predix Analytics*, *Predix Machine* and *Predix Edge Manager* help realize true Industrial Internet applications that solve your IIoT use-case.

In addition, Predix allows you to monetize analytics, services, and applications. You can create applications and resell them to end customers. Or you might create building block Services or building block Analytics that are used inside other people's applications. Either way, Predix can help you create a revenue stream for your business.

For more information on how to get started with Predix visit www.predix.io.

1.3. Purpose

Predix Modbus TCP Connectivity for NI Linux RT toolkit is designed to allow users send and store sensors data to GE Predix or other cloud, using Modbus TCP protocol and GE Predix machine Modbus TCP container installed on NI Linux RT.

2. NI Linux RT Configuration

2.1. Java SE Installation

Java SE enables the user to develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers the rich user interface performance, versatility, portability, and security that today's applications require.

Follow this link to download and install JDK SE
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2.2. Configuration of Predix Machine, Modbus Registers and WebSocket

Predix Machine provides secure cloud connectivity to Industrial assets and manages them. Developers can create their own Predix Machine with the help of Predix Machine SDK. Predix Machine SDK can be added as a plugin in Eclipse IDE and can be generated by selecting individual bundles or certain feature groups where all necessary bundles required for that feature will be included automatically.

One of the main components of Predix Machine SDK is Machine Gateway Service which provides management of machine adapters in a centralized way. Its architecture is extensible so that developers can create their own custom adapters for real time data streaming. Predix Machine already has custom adapters for MQTT, Modbus, OPC-UA and health monitoring.

Let’s take an example – how Modbus data is send to cloud from sensors.

Modbus is a serial communication protocol used for transmitting information over serial lines between electronic devices. It’s an open protocol which means manufactures can build in into their equipment without having to pay royalties.

Modbus comprises from two categories – Modbus TCP and Modbus RTU. The difference between the two is that Modbus TCP runs on the Ethernet physical layer and Modbus RTU is a serial level protocol. Modbus operates on memory registers to configure, monitor and control device I/O. Modbus has a simple structure that differentiates between four basic data types:

- **Coil (Discrete Output)** – 1 bit registers, may be read or written.
- **Discrete Input** – 1 bit registers used as inputs and may only be read.
- **Input Register** – 16 bit registers used for input and may only be read.
- **Holding Register** – most universal 16 bit registers and may be read or written.

It’s the manufacturer choice to decide whether a specific device includes all these register types or not. Like most protocols, Modbus uses client/server type protocol. The client sends the request to server and server sends back response with requested data or acknowledgement. A Modbus message may also include a Unit ID, a number between 0 and 255, which is used to identify the server in networks.

Predix Machine is flexible enough for sending data by modifying only configuration files (no coding is required). There is an adapter available in Predix-Modbus Adapter, to read data from Modbus. This is added as a bundle in Predix Machine under machine/bundle folder. The configuration for Modbus Adapter can be found under configuration/machine. When Predix Machine starts, the bundle is activating, and starting to read **com.ge.dspmicro.machineadapter.modbus-0.config** file.

Here is *config file* content sample.

 # [Required] Configuration file to load that contains the information about the Modbus nodes.

com.ge.dspmicro.machineadapter.modbus.configFile="configuration/machine/com.ge.dspmicro.machineadapter.modbus-0.xml"

 # [Required] The logical name of this adapter.

com.ge.dspmicro.machineadapter.modbus.name="Modbus Machine Adapter"

 # [Optional] A description of this adapter.
com.ge.dspmicro.machineadapter.modbus.description="Supports basic read/write capability from Modbus nodes. Supports subscription to a group of Modbus nodes."

 # [Optional] Thread pool size for modbus to handle subscriptions.
com.ge.dspmicro.machineadapter.modbus.subscriptionThreadPoolSize=I"5"

 # [Optional] Maximum retry times to read/write device before throw an exception.
com.ge.dspmicro.machineadapter.modbus.nodeCommMaxTries=I"2"

In the config file, we can specify the XML file to be read for Modbus configuration, a logical name and description for the Adapter, thread pool size to handle subscriptions and the maximum retries for connecting to a node if it fails while connecting. Given below is the content of **com.ge.dspmicro.machineadapter.modbus-0.xml** file. This configuration is used to read data from Modbus TCP.

Below is *XML file* content sample.

```
<?xml version="1.0" encoding="UTF-8"?>
    <modbusMachineAdapterConfig>
      <name>Onsite monitor modbus nodes</name>
      <description>Onsite monitor modbus nodes</description>
      <dataNodeConfigs>
        <channel protocol="TCP_IP" tcpIpAddress="127.0.0.1" tcpIpPort="502">
          <unit id="1">
            <register name="Compressor-2017:CompressionRatio" dataType="SHORT" address="1" registerType="HOLDING" description="pressure ratio" />
            <register name="Compressor-2017:DischargePressure" dataType="SHORT" address="2" registerType="HOLDING" description="discharge pressure" />
          </unit>
        </channel>
      </dataNodeConfigs>
      <dataSubscriptionConfigs>
        <dataSubscriptionConfig name="ModBusSubscription"
          updateInterval="60" startPointUnit="MINUTES" startPointOffset="10">
          <nodeName>Compressor-2017:CompressionRatio</nodeName>
          <nodeName>Compressor-2017:DischargePressure</nodeName>
        </dataSubscriptionConfig>
      </dataSubscriptionConfigs>
    </modbusMachineAdapterConfig>
```

XML file is the configuration to read data from two registers (with address 1 and 2) from a Modbus TCP running in localhost (127.0.0.1), 502 port. Each register is represented as a Data Node. The data is holding register address 1 is of data type *SHORT* and it holds the *CompressionRatio* information. Likewise, all data nodes that need to be read can be configured in the *'dataNodeConfig'* section. The nodes that need to be subscribed can be given in *'dataSubscriptionConfig'* section. In the above sample nodes are given in *'ModbusSubscription'* and will start after 10 minutes from bundle start (*startPointOffset* is given as 10 minutes) and it will be read for every one minute (*updateInterval* is given as 60

seconds). This subscription needs to be specified in the spillway config file (*configuration/machine/com.ge.dspmicro.hoover.spillway-0.config*).

Here is *spillway config* sample.

 # [Required] A friendly and unique name of the spillway.

com.ge.dspmicro.hoover.spillway.name="ModbusSpillway"

 # [Optional] A brief description of the spillway.
com.ge.dspmicro.hoover.spillway.description="Simple modbus and opcua spillway"

 # [Required] An array of data subscriptions where the data will come from
*com.ge.dspmicro.hoover.spillway.dataSubscriptions=[*
*"ModBusSubscription", *
]

 # [Required] Destination Data River name to where the data will be sent.
 # Change to the Data River by replacing the value with: Sender Service
com.ge.dspmicro.hoover.spillway.destination="ModbWSSenderService"

 # [Optional] Type name of data processing logic defined in Processor.
com.ge.dspmicro.hoover.spillway.processType=""

 # [Optional] Type name of StoreForward associated with this spillway.
com.ge.dspmicro.hoover.spillway.storeforward="ModbusStoreForward"

In the config file, you can specify a name for spillway, the data subscriptions to be considered, the name of destination river – where the data needs to be send and the name of the store forward associated with this spillway. You can add a processor if you need to manipulate data in between and the processor can be specified in spillway config.

The store forward name given in the spillway config needs to be specified in the store forward config file (*configuration/machine/ com.ge.dspmicro.storeforward-0.config*).

Below is *store forward config file* sample.

 # [Required] A friendly and unique name of the storeforward.

com.ge.dspmicro.storeforward.name="ModbusStoreForward"

The store forward config files also hold information like username and password for H2 (used by Predix internally to temporarily store data) and more. This will be added by Predix internally without assistance. The spillway destination specified in spillway config needs to be specified in WebSocket river config (configuration/machine/com.ge.dspmicro.websocketriver.send-0.config).

Here is *WebSocket river config* sample.

 # [Required] A friendly and unique name of the WebSocket River.

com.ge.dspmicro.websocketriver.send.river.name="ModbWSSenderService"

 # [Required] The URL of the WebSocket server to send to. Must begin with ws:// or wss://
com.ge.dspmicro.websocketriver.send.destination.url="wss://aghazaryan85-websocket-server.run.aws-usw02-pr.ice.predix.io/livestream/messages"

 # [Required] The name of the header where the zone ID will be inserted
com.ge.dspmicro.websocketriver.send.header.zone.name="Predix-Zone-Id"

 # [Required] The zone ID for the TimeSeries service instance
com.ge.dspmicro.websocketriver.send.header.zone.value="213e310a-2289-4535-8fd4-8e95f3c196f0"

 # [Required] The timeout in milliseconds to wait for a response before assuming a transfer has failed
com.ge.dspmicro.websocketriver.send.timeout=I"10000"

In the WebSocket config file, you should also specify the url of time series (time series is a database that is optimized for handling time series data), Predix zone ID and its value (you will get these values from the free time series service provided by GE in the Predix cloud. An account needs to be created in www.predix.io and subscribed to the time series service). You should be authenticated in Predix Cloud and the username and password need to be given in the identity config file (*configuration/machine/ com.ge.dspmicro.predixcloud.identity.config*). We will get the UAA url from Predix cloud UAA service.

Below is *identity config* sample.

 # [Required] The Predix cloud URL of an OAuth2 authorization endpoint. This is the UAA URL for
 # the technician to log into the cloud.
#
com.ge.dspmicro.predixcloud.identity.oauth.authorize.url="https://71d11fe1-ea4c-4e73-969a-e1a1b8abeee7.predix-uaa.run.aws-usw02-pr.ice.predix.io"

#
 # [Required] Predix Cloud enrollment endpoint url
#
com.ge.dspmicro.predixcloud.identity.uaa.enroll.url=""
#
 # [Required] Predix Cloud UAA token endpoint
#
com.ge.dspmicro.predixcloud.identity.uaa.token.url="https://86fc3eb6-3061-43e3-873d-4f29847b2815.predix-uaa.run.aws-usw02-pr.ice.predix.io/oauth/token"

#
 # Predix Cloud UAA client credentials
#

com.ge.dspmicro.predixcloud.identity.uaa.clientid="app_client_id"
com.ge.dspmicro.predixcloud.identity.uaa.clientsecret=""
com.ge.dspmicro.predixcloud.identity.uaa.clientsecret.encrypted=""

You can query the timeseries data using Time Series Query in Predix Toolkit (https://predix-toolkit.run.aws-usw02-pr.ice.predix.io/).

Given below is the pictorial representation depicting how data is sent to Predix time series in cloud.

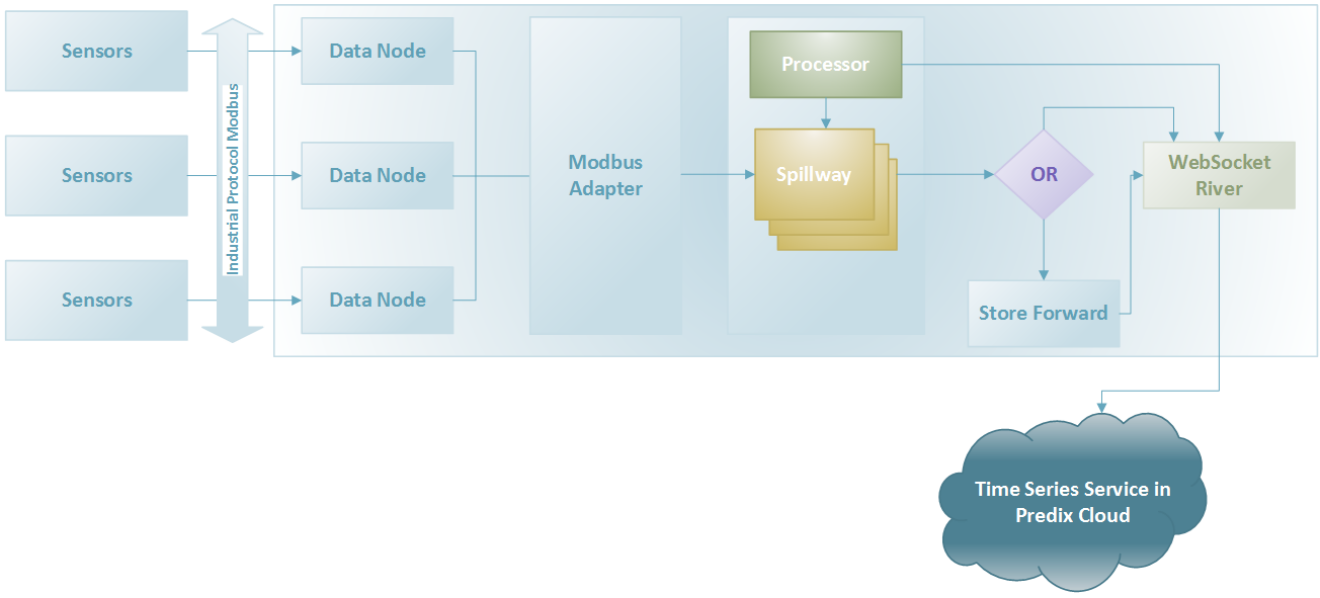

Figure 1. Data Transfer from Sensors to Predix Time Series

In short, data will be read from various Modbus sensors using Modbus Adapter. It will be sent to the Predix Time Series through Spillway and WebSocket River. If data needs to be stored temporarily, a store forward can be configured. Processor, an optional component, can be added for data aggregation or manipulation. You will be able to send data to Predix Time Series only when it is authenticated successfully through UAA.

3. Architecture

The important part of library is Predix Modbus TCP Connectivity for NI Linux RT which can open from tools menu.

The tool is intended to upload Predix Machine container and all required files to Controller, then run and stop container.

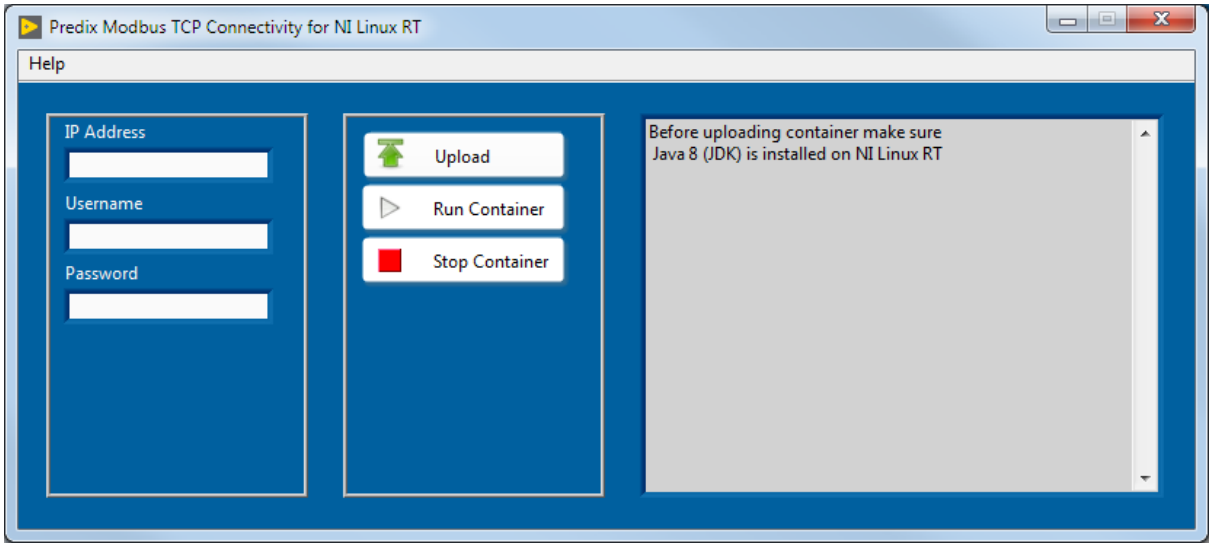

Figure 2. Predix Modbus TCP Connectivity for NI Linux RT

The user should fill the following information: Address, Username, Password of NI Linux RT controller. Once the required data is entered the container should be uploaded.

The following buttons can be used to operate NI Linux RT.

- **Upload** – is designed to upload Predix Machine container and all required files to NI Linux RT. Before uploading user can configure Modbus and Web socket parts in configuration files, for uploading used secure copy (scp).
- **Run Container** – runs Predix Machine container which will read registers from Modbus TCP slave and send registers data to Predix cloud with timestamp (timeseries) using Web socket. If there is a network problem container store data in database until network recovery.
- **Stop Container** – stops Predix Machine container. Close Web socket, Modbus master and all related references.

4. LabVIEW Features and Concepts Used

- Case structures
- Events
- Arrays
- While Loops
- For Loops
- Shift Registers
- String Functions
- Waveforms

- Modbus TCP slave.lvclass

5. System Requirements

- LabVIEW Full Development System 2015 or later
- LabVIEW Real-Time Module
- NI Linux Real-Time
- Java 8 (JDK) installed in NI Linux RT

6. Support Information

For technical support, please, contact Ovak Technologies at:

Phone: +1.281.506.0020

Email: support@ovaktechnologies.com

Web: www.ovaktechnologies.com