Otto-von-Guericke-University Magdeburg
Faculty of Electrical Engineering and Information Technology
Institute for Automation Engineering
Chair for Automation/Modeling

Automation Lab

Temperature Control Lab 3
(TCL 3)
Experiment Report

submitted: January 19, 2025

by: Jigarkumar Ratilal Panchal
Siddharth Kamal Menon

Contents

1 Introduction

This report focuses on enhancing skills in information modeling and gaining practical experience with OPC UA (Open Platform Communications Unified Architecture), a framework for interoperability in automation systems. The document outlines the extension of a classical control loop with monitoring and optimization functionalities through direct access to control data. By leveraging OPC UA servers and clients, the report explores the design and implementation of an information model for a temperature control lab setup. Additionally, it provides practical insights into configuring automation systems, integrating tools like MATLAB and Node-RED, and implementing advanced features such as unit conversion in monitoring applications.

2 OPC UA

The experiment is structured into two distinct components: the theoretical component and the practical component.

2.1 Theoretical Preparation

The theoretical component focuses on establishing foundational knowledge by exploring the principles of OPC UA-based systems, including information modeling, data communication, and system architecture. This phase emphasizes understanding key concepts such as node structures, attributes, and the integration of base information models.

2.1.1 Documentation Point 1

Understanding the OPC UA Information Model

The OPC UA information model provides a hierarchical structure for a temperature control system. It organizes nodes under key components such as **Device**, **Nameplate**, **DynamicVariables**, and **Parameters** [1]. Each node is defined with attributes like *BrowseName*, *NodeClass*, *DataType*, and relationships such as *HasTypeDefinition* and *HasComponent* [2]. Below are key insights of theoretical preparation.

Device Node

- Represents the root of the system, encompassing subcomponents.
- Organized using references like *HasComponent* to link objects such as **Nameplate**, **DynamicVariables**, and **Parameters**.

Nameplate

- Represent Nameplate properties such as **ManufacturerName**, **ManufacturerProductDesig nation**, **ManufacturerProductFamil**, **SerialNumber**, **YearOfConstruction** and **Address**.
- Each node has a data type (**String**) and provides information about product using the *HasTypeDefinition* relationship.

DynamicVariables

- Represent dynamic system properties such as **Input: Temperature_1**, **ControllerError_1**, **ManipulatedVariable_1**, **Output:actualTemperature_1** and **LED** with unit °C , °C, % , °C and na respectively.
- Each node has a data type (**Double**) where as node **LED** has a data type (**Boolean**) and provides functionality like real-time updates using the *HasTypeDefinition* relationship.

Parameters

- Include static configuration data like **Proportional: Kp**, **Integral: Ki**, and **Temperature_1_High_Limit**.
- These variables allow system tuning and constraints, critical for the PID controller.

Controller Integration

- Includes nodes such as **ControlError**, **LED**, and **ManipulatedVariable**, which are integral for feedback mechanisms.
- The system supports object-oriented behavior, encapsulating attributes for modular design.

Custom Types and Specifications

- The **BaseDataVariableType** and references to custom data types. Allow specific node configurations. e.g. TCL_HardwareType.
- This design extends the flexibility of the OPC UA model for customized applications.

Real-world Example of Usage

- A controller managing temperature readings (**Input Temperature**) and issuing commands (**ManipulatedVariable**) is modeled with relationships and data flow in the information model.
- The OPC UA model facilitates efficient communication between nodes, like **Controller**, **Heater**, and **I/O Shield**.

OPC UA Information Model

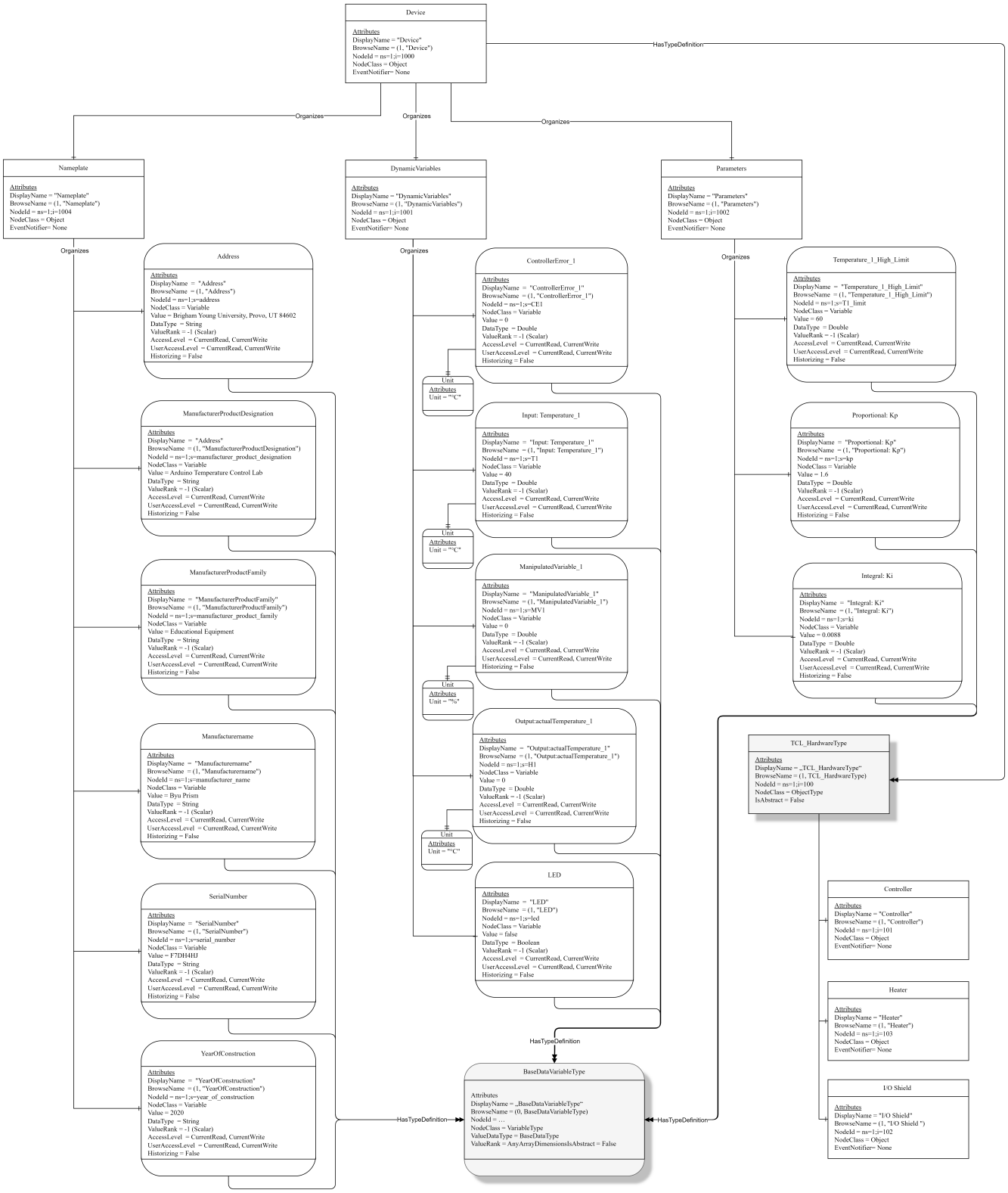

Figure 2.1: OPC UA information model.

2.1.2 Documentation Point 2

1. Overview of the Function

The function `constructAlarmAddressSpace` is used to configure the address space of an OPC UA server dynamically. The address space is a hierarchical structure where nodes represent various components and variables. The function uses the `node-opcua` library to create and organize these nodes.

2. Key Objects and Concepts

- **server**: Represents the OPC UA server instance.
- **addressSpace**: The hierarchical model of nodes in the server.
- **namespace**: A scoped container to define nodes uniquely.
- **coreServer.choreCompact.opcua**: Provides access to OPC UA core definitions like `DataType`, `LocalizedText`, and `Variant`.
- **Attributes in Node Configuration:**
  - `browseName`: A human-readable identifier for the node.
  - `nodeId`: A unique identifier for the node.
  - `dataType`: Defines the type of data the variable holds (e.g., `Double`, `Boolean`, `String`).
  - `value`: Contains the data value and its type.

3. Hierarchical Structure of the Server

The address space is divided into folders and variables:

Root and Main Folders

- **Root Folder**: The top-level node in the server's address space.
- **Device**: A folder that acts as the root for device-specific components.
  - Subfolders under **Device**:
    - **DynamicVariables**: Holds variables that represent system dynamics.
    - **Parameters**: Contains configuration parameters.
    - **Nameplate**: Represents metadata about the device, such as manufacturer details.

4. Explanation of Used Attributes

Variables in DynamicVariables

- **Input: Temperature_1:**
  - `browseName`: `"Input: Temperature_1"`.
  - `nodeId`: `"ns=1;s=T1"` (namespace index = 1, string identifier = "T1").
  - `dataType`: `Double` (floating-point number).
  - `value`: `40` (default temperature value).

5. Code Highlights

- **Node Creation**: Each node is added using methods like `addFolder` or `addVariable`.
- **Namespace Management**: `namespace.addVariable` ensures that variables are scoped to the appropriate namespace.
- **Type Safety**: Attributes like `dataType` ensure that variable values conform to specific types (e.g., `Double`, `Boolean`, `String`).

2.2 Practical part

The practical component, on the other hand, involves the application of these theoretical principles through hands-on implementation. Participants engage in tasks such as designing OPC UA node sets, configuring server-client interactions, and integrating real-time monitoring and optimization functionalities. This practical application reinforces theoretical insights and provides an experiential understanding of industrial automation systems.

2.2.1 Documentation Point 3

Screen shots with data and time at the right bottom of the screen

A. Explanation of the JavaScript Code

The JavaScript code adds a variable `tempAct` to an OPC UA server's address space using the `node-opcua` library. Here's a concise breakdown:

- **Namespace and Organization:** The variable is added to the server's namespace and organized under the parent node `dynVar`.
- **Attributes:**
  - **browseName:** `"Output: actualTemperature_1"`, a human-readable name.
  - **nodeId:** `"ns=1;s=H1"`, a unique identifier in namespace 1 with string ID `"H1"`.
  - **dataType:** `"Double"`, representing a 64-bit floating-point value.
- **Value:**
  - **dataType:** Ensures the value is a `Double`.
  - **value:** Initially set to `undefined`, to be updated dynamically.
- **Purpose:** Creates a dynamic output variable representing `actualTemperature_1` for use in control or monitoring systems.

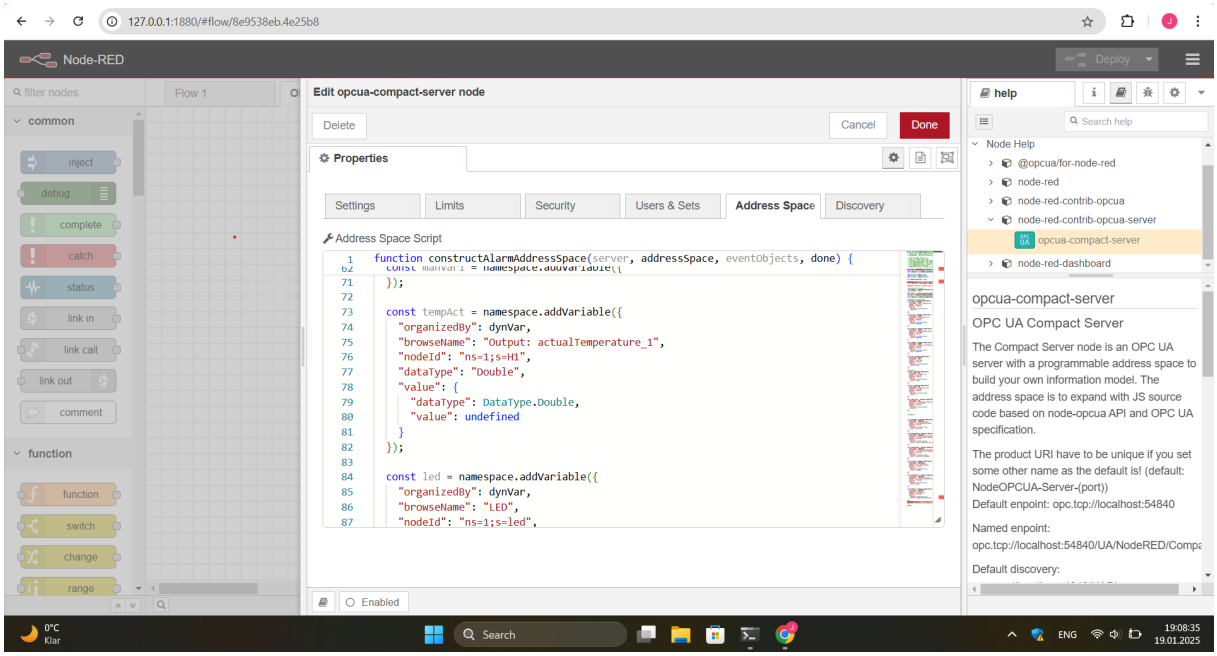

Figure 2.2: Output: actualTemperature_1 in the “Compact Context Server”.

B. The GUI root

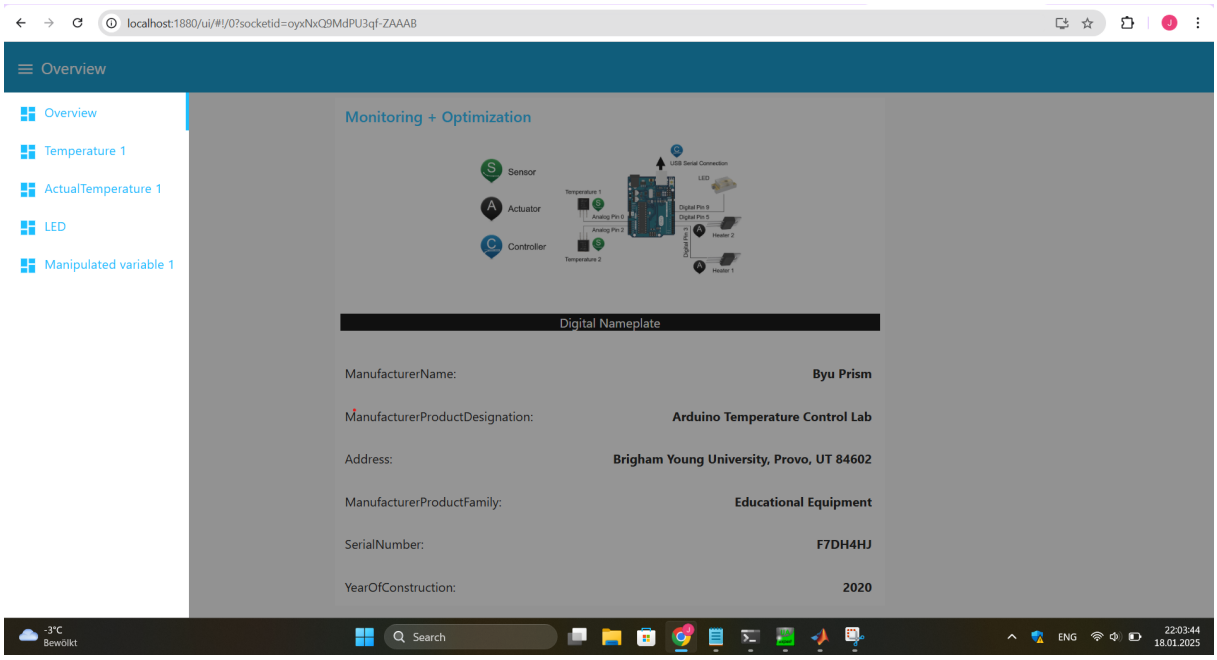

Figure 2.3: The GUI Root.

C. The Signal Curve

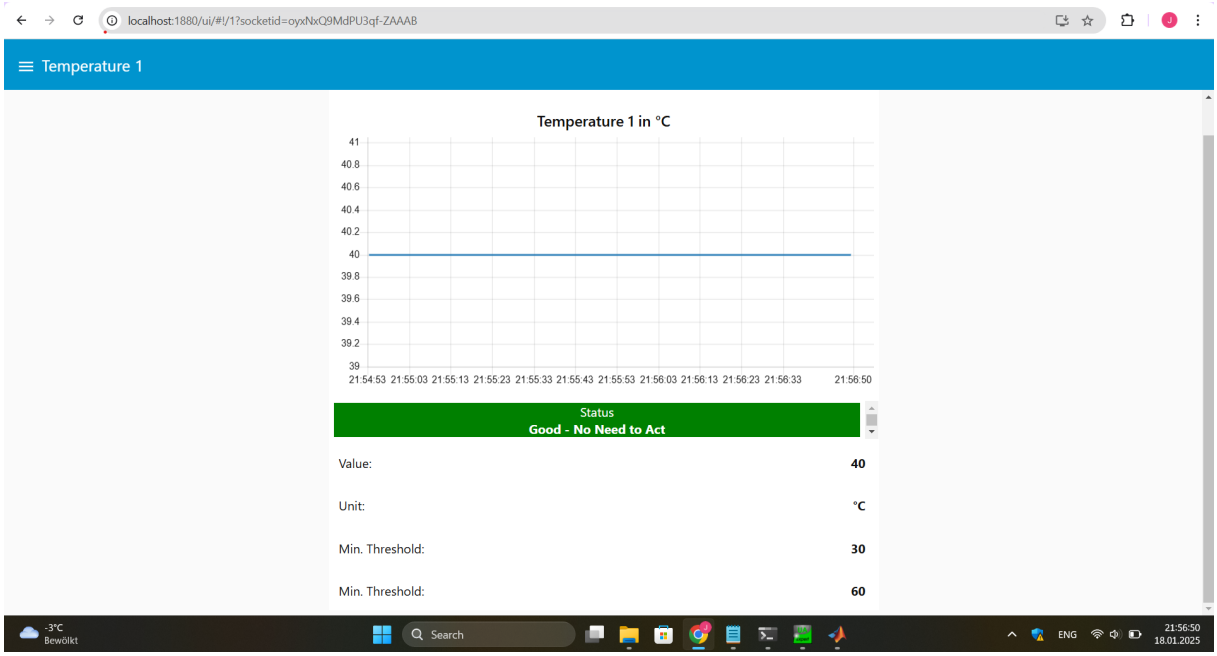

Figure 2.4: The Signal Curve of Input: Temperature_1.

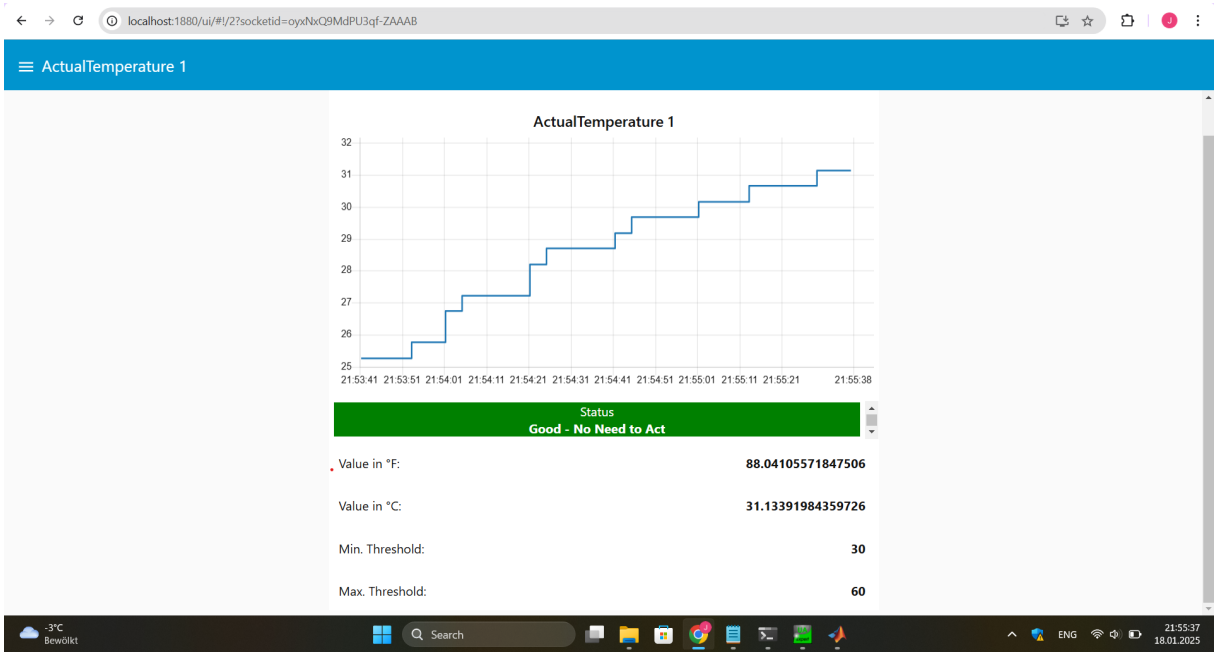

Figure 2.5: The Signal Curve of Output: actualTemperature_1.

2.2.2 Documentation Point 4

The Unit Conversion

Explanation of the Code

The following code implements a temperature conversion from Celsius (°C) to Fahrenheit (°F) in a Node-RED function node. Below is an explanation of its components:

- **Input Temperature:** The input temperature is read from `msg.payload`.
- **Conversion Logic:** The formula $(\text{temperature} \times 9/5) + 32$ converts the temperature from Celsius (°C) to Fahrenheit (°F).
- **Set Converted Unit:** The converted temperature and the unit (`"°F"`) are assigned to `msg.payload` and `msg.unit`, respectively.
- **Optional Reference:** If needed, the original temperature and unit can be added to the message using `msg.originalTemperature` and `msg.originalUnit`.

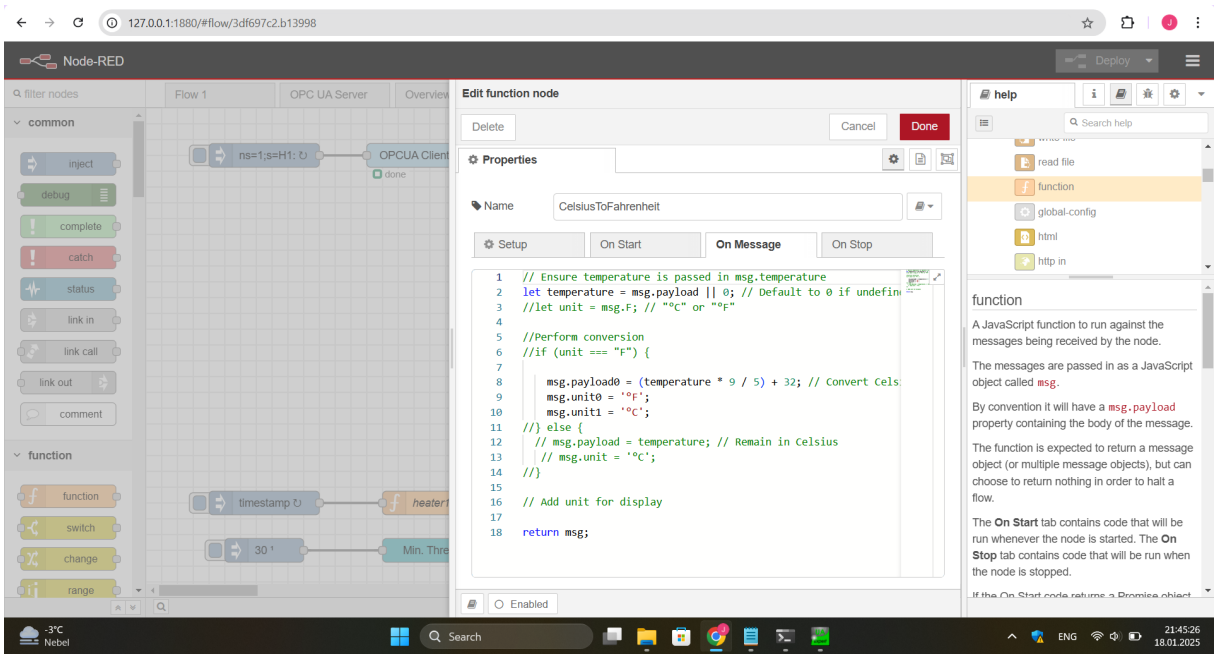

Figure 2.6: Celsius to Fahrenheit Conversion Code.

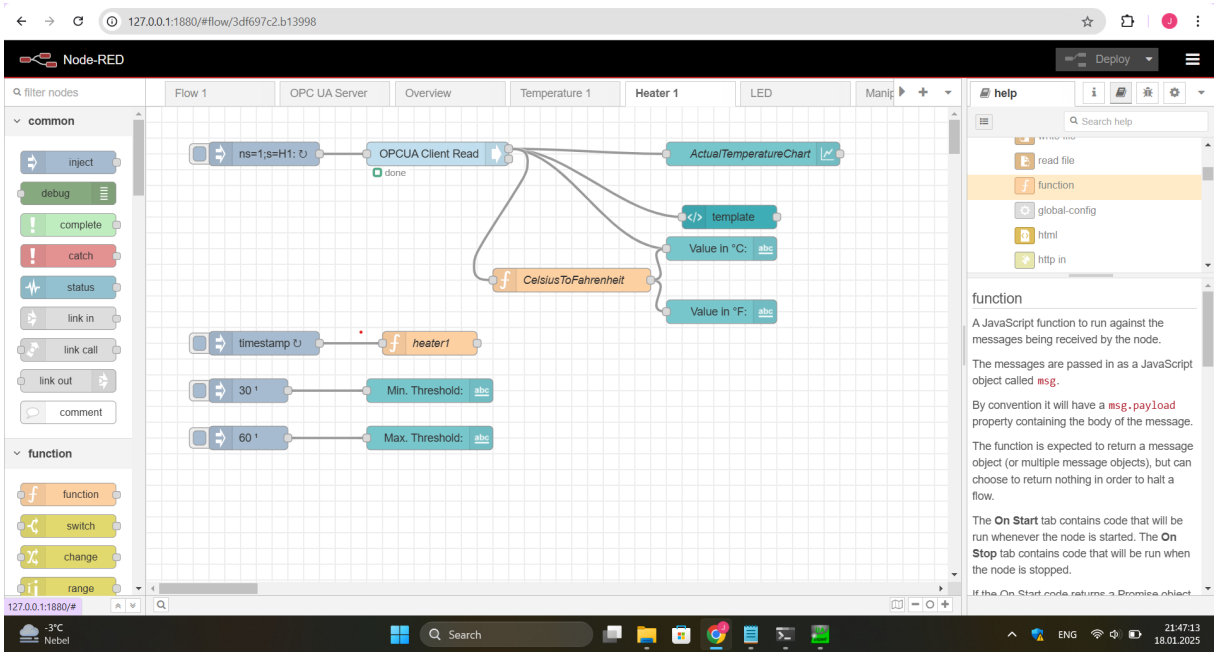

Figure 2.7: Celsius to Fahrenheit Conversion Connection.

3 Conclusion

The practical implementation of an OPC UA-based system, as outlined in this lab, demonstrates the critical role of standardized information modeling and secure, interoperable communication in automation engineering. By extending the classical control loop with monitoring and optimization functionalities, this lab integrates modern concepts of industrial automation, including edge computing and advanced GUI interfaces. The designed OPC UA node set effectively maps system components and parameters, ensuring scalability and adherence to best practices in information modeling. This experiment highlights the efficiency of combining Matlab and Node-RED in facilitating data exchange and real-time monitoring, offering valuable insights into modern automation systems' design and application. Future advancements could explore further optimizations and integrations to enhance robustness and functionality.

Bibliography

[1] "Material TCL 3," 2024, course: [Automation Lab], University: [Institute for Automation Engineering OVGU].

[2] M. D. Dr. Wolfgang Mahnke, Stefan-Helmut Leitner, *OPC Unified Architecture*. Berlin, Heidelberg: Springer-Verlag, 2009.