Bibliotecas
MQTT Client SL
JSON Utilities SL

PLC500, PLC500ED, PLC500MC
PLC410

Nota de Aplicação

Nota de Aplicação

PLC410, PLC500, PLC500ED, PLC500MC

Documento: 10012146401

Revisão: 00

Data de publicação: 08/2024

SUMÁRIO DAS REVISÕES

As informações abaixo descrevem as revisões ocorridas neste manual.

Versão	Revisão	Descrição
1.3.0	R00	Primeira edição.

SUMÁRIO

1 INTRODUÇÃO

Esta Nota de Aplicação destina-se a auxiliar na implementação do protocolo **MQTT** nos PLCs da WEG, modelos PLC410, PLC500, PLC500ED e PLC500MC. Salienta-se que os dados fornecidos podem mudar ligeiramente por conta do contínuo desenvolvimento e atualização dos produtos e bibliotecas.

São apresentadas as bibliotecas **MQTT Client SL** e **JSON Utilities SL**, integrantes da **CODESYS IIoT Libraries SL**, onde destacam-se os principais blocos de funções e métodos que contribuem no desenvolvimento de aplicações IoT. Além disso, são fornecidos exemplos práticos que demonstram a capacidade dessas bibliotecas.

Para mais informações a respeito do hardware, interfaces e protocolos de comunicação, consulte o **Manual do Usuário** do respectivo produto, disponível em www.weg.net. Para mais detalhes sobre as bibliotecas, blocos de função e métodos, consulte o **Codesys Online Help**, disponível em help.codesys.com.

ATENÇÃO!

Esta nota de aplicação é direcionada para profissionais treinados em redes industriais. A instalação e configuração dos dispositivos deve ser feita de acordo com o manual do fabricante.

1.1 ABREVIAÇÕES E DEFINIÇÕES

Broker: Servidor que gerencia o recebimento de mensagens enviadas pelos clientes publicadores (publishers), as enviando para os clientes subscritos (subscribers), através do protocolo MQTT.

Cliente MQTT: Refere-se a qualquer dispositivo ou aplicativo que usa o protocolo MQTT para se conectar a um broker MQTT. Existem dois tipos principais de clientes MQTT: publishers e subscribers.

Codesys: Plataforma de programação que permite desenvolver, configurar e monitorar soluções para automação industrial e integração de sistemas.

FB: Bloco de função (*Function Block*). Encapsula uma função específica ou um conjunto de funções relacionadas, com capacidade para incorporar dados e outros elementos.

IoT: Sigla que refere-se às tecnologias que facilitam a comunicação e a troca de dados entre dispositivos e a nuvem, bem como entre os próprios dispositivos (*Internet of Things*).

IIoT: Sigla que refere-se à aplicação de tecnologias IoT no contexto industrial. A IIoT envolve a interconexão de dispositivos, máquinas e sistemas industriais através da Internet para coleta, troca e análise de dados, com o objetivo de melhorar a eficiência, a automação e o monitoramento nas operações industriais.

IIoT Libraries SL: Conjunto de bibliotecas do Codesys desenvolvidas para aplicações IoT. O IIoT envolve o uso de tecnologias IoT em ambientes industriais para aprimorar automação, monitoramento e troca de dados.

JSON: O JSON (*JavaScript Object Notation*) é um formato leve de troca de dados que é fácil para os humanos lerem e escreverem, e fácil para as máquinas interpretarem e gerarem. Sua estrutura básica é composta por pares de chave e valor.

JSON Utilities SL: Biblioteca do Codesys específica para leitura e escrita de mensagens do tipo JSON, bem como busca por atributos, valores, e elemntos filhos e pais.

MQTT: Protocolo de transporte que utiliza a topologia publicação/subscrição para transferência de mensagens leves entre dispositivos.

MQTT Client SL: Biblioteca do Codesys específica para tratar do envio e recebimento de mensagens via MQTT.

Payload: Conteúdo da mensagem MQTT. Pode conter diversos objetos, atributos, valores, etc.

Publisher: Dispositivo ou aplicativo que envia mensagens para um tópico específico no broker MQTT.

QoS: Parâmetro utilizado para determinar o nível de qualidade de serviço em troca de mensagens utilizando o protocolo MQTT (*Quality of Service*).

ST: Texto estruturado (*Structured text*). Linguagem de programação definida pela IEC 61131-3. Fornece uma abordagem estruturada e procedural para desenvolver lógica de controle em sistemas industriais.

Subscriber: Dispositivo ou aplicativo que se conecta ao broker e expressa interesse em receber mensagens de um determinado tópico.

TLS: O TLS (*Transport Layer Security*) é um protocolo de segurança que garante a privacidade e a integridade dos dados transmitidos pela internet. Ele é comumente usado para criptografar a comunicação entre um cliente e um servidor para proteger contra ataques de interceptação e manipulação de dados.

Tópico: Forma de categorizar as mensagens no MQTT. Os publicadores enviam mensagens para tópicos específicos, e os assinantes recebem as mensagens dos tópicos aos quais estão subscritos.

1.2 AVISO IMPORTANTE SOBRE SEGURANÇA CIBERNÉTICA E COMUNICAÇÕES

Os PLCs da WEG, modelos PLC410, PLC500, PLC500ED e PLC500MC, possuem a capacidade de se conectar e trocar informações por meio de redes e protocolos de comunicação. Embora tenham sido projetados e submetidos a testes para garantir o correto funcionamento com outros sistemas de automação utilizando os protocolos mencionados neste manual, é fundamental que o cliente compreenda as responsabilidades associadas à segurança da informação e cibernética ao utilizar este equipamento.

Assim, é dever único e exclusivo do cliente adotar estratégias de defesa em profundidade e implementar políticas e medidas a fim de garantir a segurança do sistema como um todo, inclusive com relação às comunicações enviadas e recebidas pelo equipamento. Entre estas medidas podemos destacar a instalação de firewalls, programas de antivírus e malwares, criptografia de dados, controle de autenticação e acesso físico de usuários.

A WEG e suas afiliadas não se responsabilizam por danos ou perdas decorrentes de violações de segurança cibernética, incluindo, mas não se limitando a, acesso não autorizado, intrusão, vazamento e/ou roubo de dados ou informações, negação de serviço ou qualquer outra forma de violação de segurança. A utilização deste produto em condições para as quais não foi especificamente projetado não é recomendada e pode acarretar danos ao produto, à rede e ao sistema de automação. Neste sentido, é imprescindível que o cliente compreenda que a intervenção externa por programas de terceiros, a exemplo dos sniffers ou programas com ações semelhantes, possui o potencial de ocasionar interrupções ou restrições na funcionalidade do equipamento.

1.3 RECOMENDAÇÕES

No Codesys, recomendam-se os seguintes cuidados quando implementando aplicações IoT:

- Utilização de tarefas específicas para publicação e subscrição de payloads MQTT, e com baixa prioridade (20 - 31), de forma a evitar interferências em tarefas críticas. Obs: a tarefa de maior prioridade é a 0 (zero).
- Evite a operação de cargas críticas por meio de plataformas IoT devido à possibilidade de apresentarem latência elevada, variação no atraso e dependência de uma conexão estável com a internet.
- A frequência máxima de publicações de payloads deve ser baseada na necessidade da aplicação e na capacidade do servidor MQTT.

1.4 INSTALAÇÃO DAS BIBLIOTECAS

As bibliotecas MQTT Client SL e JSON Utilities SL, integrantes da **CODESYS IIoT Libraries SL**, vem instaladas por padrão no **WEG package** a partir da versão **1.3.x**. Contudo, caso seja necessário, seguem os passos para a instalação destas bibliotecas.

Para a instalação da biblioteca IIoT Libraries SL, abra o Codesys e acesse **Tools > CODESYS Installer**. Na nova janela aberta, clique em **Browse** e busque por "**IIoT**". Selecione a biblioteca **IIoT Libraries SL** e clique em **Install**. Ao instalar novas bibliotecas por meio do Codesys Installer, é necessário que o software Codesys seja previamente fechado.

A Figura 1.1 mostra a janela do **CODESYS installer**, a partir de onde a biblioteca CODESYS IIoT Libraries SL pode ser instalada.

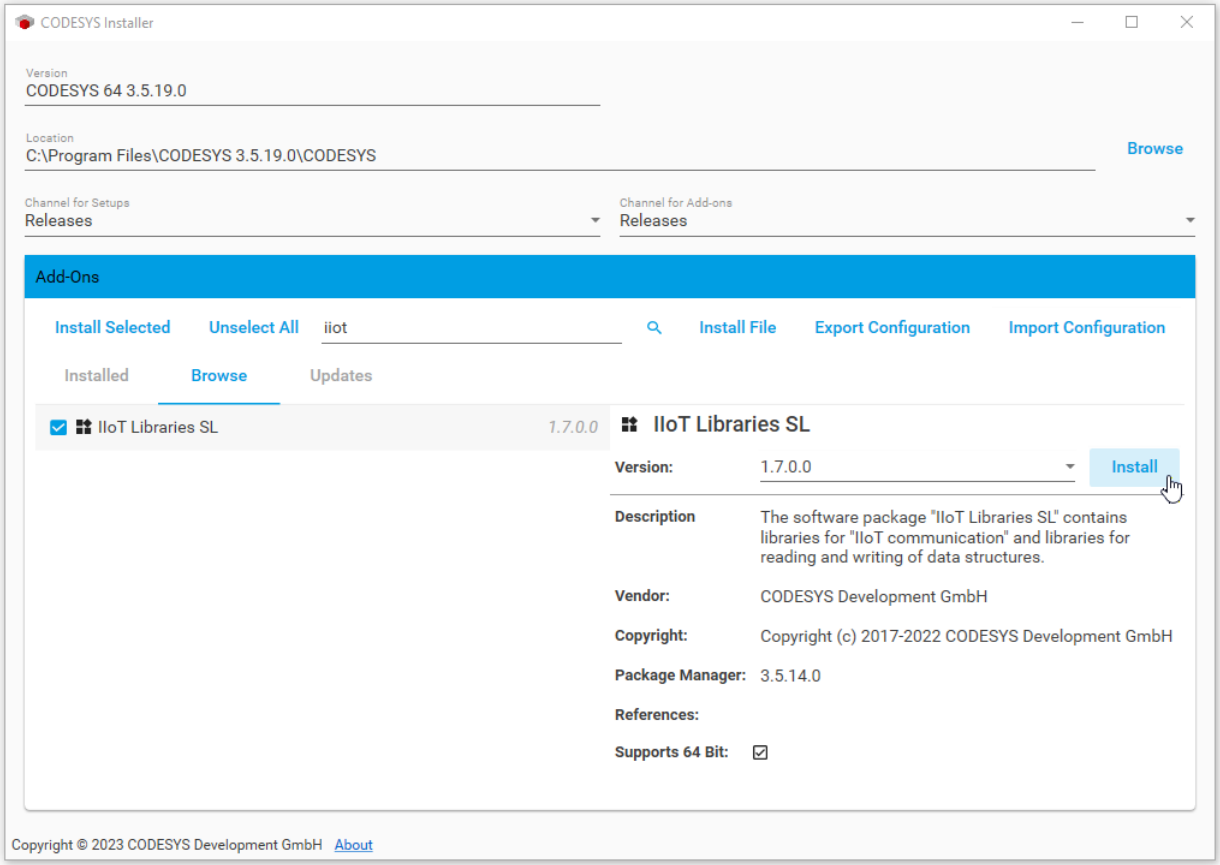


Figura 1.1: *Instalação da biblioteca CODESYS IIoT Libraries SL.*

Após a instalação, será possível adicionar as bibliotecas **MQTT Client SL** e **JSON Utilities SL**, bem como as demais bibliotecas presentes na IIoT Libraries SL, às aplicações do Codesys.

1.5 O PROTOCOLO MQTT

O MQTT (*Message Queuing Telemetry Transport*) é um protocolo de comunicação leve e eficiente, amplamente utilizado em aplicações IoT (*Internet of Things*). Possui como característica o padrão de troca de mensagens do tipo *publisher*/*subscriber*.

As mensagens, comumente chamadas de *payloads*, são publicadas com base em tópicos. Os tópicos correspondem a *strings* que representam caminhos com hierárquias. Por exemplo, em factory1/lab2/temperature, factory1 é o nível superior, lab2 é um subnível e monitoring é a característica que descreve a última parte do tópico. Sempre que um dispositivo está subscrito em um tópico, receberá os payloads publicados neste mesmo tópico.

A comunicação entre os dispositivos ocorre por meio de um broker MQTT, o qual é responsável por gerir as publicações e subscrições (por exemplo, o Mosquitto).

A Figura 1.2 apresenta um exemplo de aplicação IoT utilizando o protocolo MQTT.

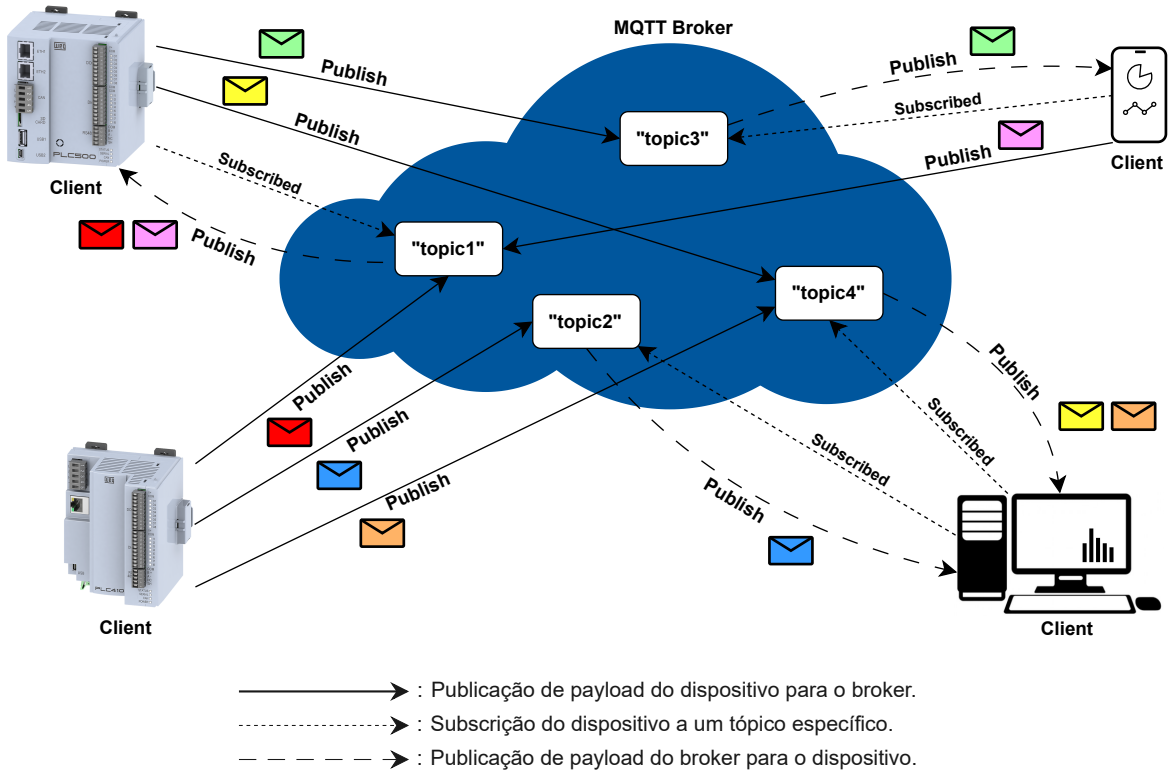


Figura 1.2: *Esquema de uma aplicação IoT utilizando o protocolo MQTT.*

Na Figura 1.2 existem quatro tópicos sendo utilizados para publicação, denominados *topic1*, *topic2*, *topic3* e *topic4*. A rede é composta por um PLC500, um PLC410, um computador e um smartphone. O PLC500 publica payloads nos tópicos 3 e 4, e se subscreve no tópico 1. O PLC410 publica payloads nos tópicos 1, 2 e 4. O computador se subscreve nos tópicos 2 e 4. O smartphone publica no tópico 1 e se subscreve no tópico 3.

Os dados publicados em mensagens MQTT podem variar amplamente, dependendo da aplicação e do contexto específico de uso. No entanto, algumas categorias gerais de dados comumente publicados incluem: leituras de sensores, estados de dispositivos, parâmetros de produção, eventos, alertas, notificações, controle de processos, etc.

NOTA!
Em muitos casos, o payload MQTT é formatado usando JSON (*JavaScript Object Notation*). A sintaxe JSON é leve e fácil de ler, facilitando o transporte de dados estruturados.

2 MQTT CLIENT SL

A biblioteca **MQTT Client SL** permite a conexão de um controlador Codesys a um broker MQTT. Em seguida, mensagens podem ser publicadas e subscritas com base em tópicos.

O formato da mensagem não é fixo, o que significa que uma string JSON ou qualquer estrutura de dados pode ser transmitida. A biblioteca MQTT Client SL possui os seguintes blocos de função:

- **MQTTClient**: Estabelece o link MQTT com o broker. Toda a aplicação utilizando a biblioteca MQTT Client SL necessita de pelo menos um FB MQTTClient. Utiliza-se apenas uma instância deste bloco para se conectar a um único broker.
- **MQTTPublish**: Utilizado para publicar mensagens em tópicos do broker. Pode ser utilizada uma instância para cada tópico a ser publicado.
- **MQTTSubscribe**: Utilizado para se subscrever a tópicos do broker. Pode ser utilizada uma instância para cada tópico a ser subscrito.

A Figura 2.1 mostra a biblioteca MQTT Client SL no Codesys.

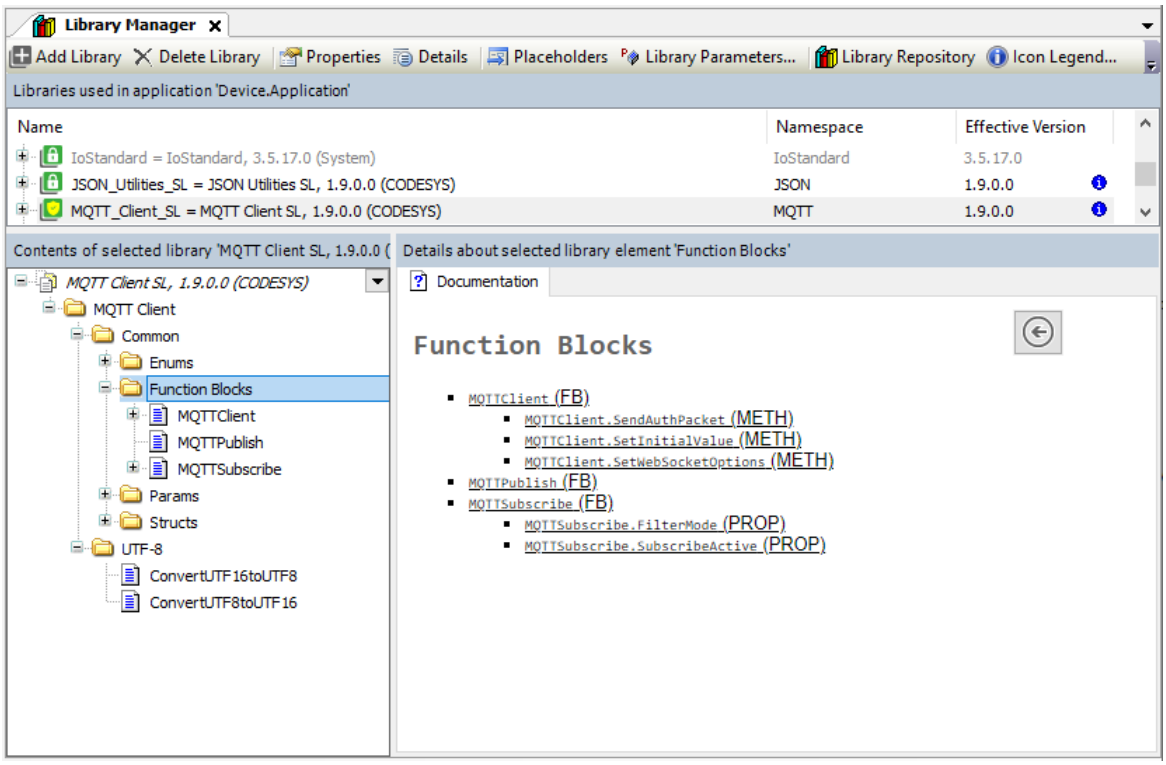


Figura 2.1: *Biblioteca MQTT Client SL no Codesys.*

Funções suportadas:

- Publicação e subscrição de mensagens com base no MQTT V3.1.1 e MQTT V5.0
- Criptografia TLS
- Certificados do cliente
- Qualidade de Serviço: 0, 1 e 2 (QoS0, QoS1, QoS2)
- O tamanho máximo do pacote e o tamanho da carga útil podem ser configurados por meio de uma lista de parâmetros
- Suporte a multitarefa (MQTTClient, MQTTPublish e MQTTSubscribe podem ser executados em tarefas diferentes)
- Last will messages (QoS0, QoS1, QoS2)
- Wildcards: Caracteres utilizados para subscrever em múltiplos tópicos simultaneamente

NOTA!

Os níveis de QoS determinam o grau de garantia na entrega de mensagens no MQTT. No QoS 0, a entrega ocorre no máximo uma vez, sem confirmação. Já no QoS 1, há garantia de pelo menos uma entrega, com confirmação do servidor. Por fim, o QoS 2 assegura a entrega exatamente uma vez, através de um processo de *handshake* mais complexo entre cliente e servidor.

A escolha entre esses níveis depende da necessidade de confiabilidade na entrega em relação à eficiência da comunicação.

2.1 WILDCARDS: #, +

- # (multi-níveis): Utilizado apenas ao final de tópicos. Permite que sejam recebidas todas as mensagens de tópicos que comecem com o padrão anterior ao wildcard. Exemplo: **factory1/lab1/#**
  - factory1/lab1/temperature ✓
  - factory1/lab1/humidity ✓
  - factory1/lab2/temperature ×
- + (único-nível): Permite que sejam recebidas todas as mensagens de tópicos que contenham qualquer string no lugar do wildcard. Exemplo: **factory1/lab1/+/temperature**
  - factory1/lab1/room1/temperature ✓
  - factory1/lab1/room1/humidity ×
  - factory1/lab2/room1/temperature ×
  - factory1/lab1/room2/temperature ✓

2.2 MQTTCLIENT

Seguem abaixo as principais variáveis do FB MQTTClient:

- **Entradas**:
  - **xEnable**: *True* = Ativa o bloco de função. *False* = Aborta/reseta a operação do bloco.
  - **uiPort**: Porta utilizada pelo broker MQTT. Usualmente a porta padrão para comunicações não encriptadas é a **1883**, enquanto a porta **8883** é geralmente utilizada para encriptação. Contudo, deve-se consultar as características do broker específico.
  - **sHostname**: Endereço do broker MQTT.
  - **xUseTLS**: *True* = Conexão encriptada utilizando TLS. *False* = Conexão não encriptada.
  - **xCleanSession**: Bit de sinalização usado para controlar o ciclo de vida do estado da sessão. *True* = significa que uma nova sessão será criada na conexão e a sessão será automaticamente descartada quando o cliente se desconectar. Se for *False*, significa que ele tentará reutilizar a sessão anterior ao se conectar. Se não houver uma sessão correspondente, uma nova sessão será criada, que sempre existirá após o cliente se desconectar.
  - **uiKeepAlive**: Tempo, em segundos, que o cliente e o servidor concordam em manter a conexão aberta sem que ocorra qualquer troca de mensagens. Valor inicial: 5 segundos. (opcional)
  - **wsUsername**: Nome de usuário (opcional).
  - **wsPassword**: Senha do usuário (opcional).
  - **sClientId**: Código identificador do usuário (opcional).

- **Saídas**:
  - **xDone**: *True* = Operação finalizada.
  - **xBusy**: *True* = Operação rodando.
  - **xError**: *True* = Erro na operação.
  - **eMQTTError**: Tipo do erro MQTT.
  - **xConnectedToBroker**: *True* = Conectado ao broker.

NOTA!

A principal diferença entre as portas encriptadas (TLS) e não encriptadas é a segurança dos dados transmitidos. Quando o MQTT opera em uma porta não encriptada, os dados são transmitidos em texto simples, o que significa que podem ser lidos por qualquer pessoa com acesso à rede entre o cliente e o servidor MQTT. Isso representa um risco de segurança, especialmente quando informações sensíveis estão sendo transmitidas.

2.3 MQTTPUBLISH

Seguem abaixo as principais variáveis do FB MQTTPublish:

- **Entradas**:
  - **xExecute**: *True* = Habilita o bloco de função.
  - **udiTimeOut**: Tempo máximo para ser executado. Gera erro após este tempo.
  - **eQoS**: Nível QoS da mensagem para publicação.
  - **pbPayload**: Ponteiro para a string que contém o payload a ser publicado. Pode-se utilizar os FBs da biblioteca **JSON Utilities** para a criação de um payload no formato JSON.
  - **udiPayloadSize**: Tamanho do payload a ser publicado.
  - **mqttClient**: Instância do FB MQTTClient.
  - **wsTopicName**: Nome do tópico para publicação.
- **Saídas**:
  - **xDone**: *True* = Operação finalizada.
  - **xBusy**: *True* = Operação rodando.
  - **xError**: *True* = Erro na operação.
  - **eMQTTError**: Tipo do erro MQTT.

2.4 MQTTSUBSCRIBE

Seguem abaixo as principais variáveis do FB MQTTSubscribe:

- **Entradas**:
  - **xEnable**: *True* = Habilita o bloco de função.
  - **eSubscribeQoS**: Nível QoS da mensagem para subscrição.
  - **pbPayload**: Ponteiro para a string que armazenará o payload recebido.
  - **udiMaxPayloadSize**: Tamanho máximo da string recebida.

- **wsTopicFilter**: Filtro de tópicos. Pode ser utilizado um tópico fixo ou em conjunto com os caracteres # e +.
- **mqttClient**: Instância do FB MQTTClient.
- **udiTimeOut**: Tempo máximo para ser executado. Gera erro após este tempo.

- **Saídas**:
  - **xDone**: *True* = Operação finalizada.
  - **xBusy**: *True* = Operação rodando.
  - **xError**: *True* = Erro na operação.
  - **eMQTTError**: Tipo do erro MQTT.
  - **xReceived**: *True* = Mensagem recebida.
  - **xSubscribeActive**: *True* = Subscribe está aguardando um payload no tópico especificado.

2.5 PUBLICAÇÃO E SUBSCRIÇÃO SEM ENCRIPTAÇÃO

Esta aplicação, denominada **MQTT_simple**, é um exemplo simples de utilização dos blocos de função da biblioteca MQTT. Ela realiza a conexão com o broker MQTT, a publicação de uma string contendo o valor de uma variável que representa a temperatura de um dispositivo, e enquanto isso, se subscreve no mesmo tópico, ou seja, receberá o payload assim que publicado. A porta de comunicação utilizada é a **1883**, sem encriptação e sem autenticação.

De forma detalhada, o FB **MQTTClient** conecta o dispositivo ao broker test.mosquitto.org, o qual é um servidor público utilizado para testes e experimentação com o protocolo MQTT. Os FBs **RS** e **TON** são utilizados para, em caso de erro do MQTTClient, desabilitar o FB, aguardar um segundo e então habilitá-lo novamente.

Para a publicação de payloads utiliza-se o FB **MQTTPublish** dentro de uma máquina de estados simples. No estado 0 um contador é incrementado a cada ciclo, e após 50 ciclos, passa para o estado 1. No estado 1, a variável **uiTemperature**, do tipo inteira sem sinal, é incrementada em uma unidade, e então passa para o estado 2. No estado 2, é criado o payload para publicação. O payload é formado pela string `'simple MQTT message: temperature = '` adicionando-se o valor da variável uiTemperature.

No estado 3 é realizada a publicação propriamente dita. O FB MQTTPublish é habilitado apenas se o MQTTClient estiver com a saída **xConnectedToBroker** em estado alto. O tópico definido para publicação é o "**weg/drivesbt/plc**". Nota-se que a entrada do payload é a **pbPayload**, ou seja, é um **ponteiro** para a string desejada. Também deve ser definida na entrada **mqttClient** a instância do FB MQTTClient utilizado, que no caso é o **MQTTClient_0**.

Assim que uma das saídas **xDonePub** ou **xErrorPub** estiver em estado alto, o FB MQTTPublish_0 é desabilitado, e a máquina de estados volta ao estado 0, reiniciando o processo de publicação.

Fora da máquina de estados, temos o FB **MQTTSubscribe**. Ele também é dependente da conexão do FB MQTTClient através da variável xConnectedToBroker. Quando habilitado, Este FB fica aguardando a publicação de payloads em qualquer subtópico de "**weg/drivesbt/**", devido à utilização do caractere # ao final do tópico. Enquanto o FB estiver ativo, a variável **xSubscribeActive** estará em estado alto. Quando um payload é recebido, a variável **xReceived** adquire o valor alto, e nesse momento a string **sPayloadSub** pode ser manipulada de acordo com a aplicação.

NOTA!

Nexte exemplo, todo o código do **MQTTClient**, **MQTTPublisher** e **MQTTSubscribe** é realizado em uma só tarefa, de forma a facilitar a reprodução pelo usuário. Contudo, conforme salientado nas recomendações desta nota, deve-se procurar utilizar tarefas específicas e com prioridades compatíveis para cada bloco de funções, de forma a não interferir no programa principal.

NOTA!

Para a utilização dos blocos de função **MQTTClient**, **MQTTPublisher** e **MQTTSubscribe**, não esqueça de adicionar a biblioteca **MQTT Client SL** na aplicação do Codesys.

A Figura 2.2 mostra a declaração de variáveis e a Figura 2.3 mostra o código em ST para a aplicação MQTT_simple.

MQTT_simple - Declaration

```
PROGRAM MQTT_simple
VAR
  // Functions Blocks
  MQTTClient_0: MQTT.MQTTClient;
  MQTTPublish_0: MQTT.MQTTPublish;
  MQTTSubscribe_0: MQTT.MQTTSubscribe;

  // MQTTClient
  sHostName : STRING := 'test.mosquitto.org';
  uiPort : UINT := 1883;
  xConnectedToBroker, xDoneClient, xErrorClient : BOOL;
  eMQTTErrorClient : MQTT.MQTT_ERROR;
  uiQoS : MQTT.MQTT_QOS := 1;

  // MQTTPublish
  sPayloadPub : STRING(255);
  pbPayloadPub : POINTER TO BYTE := ADR(sPayloadPub);
  wsTopicPub : WSTRING(1024) := "weg/drivesbt/plc";
  udiPayloadSizePub : UDINT;
  xDonePub, xErrorPub : BOOL;
  eMQTTErrorPub : MQTT.MQTT_ERROR;

  // MQTTSubscribe
  sPayloadSub : STRING(255);
  pbPayloadSub : POINTER TO BYTE := ADR(sPayloadSub);
  wsTopicSub: WSTRING(1024) := "weg/drivesbt/#";
  udiMaxPayloadSize : UDINT := 1024;
  udiPayloadSizeSub : UDINT;
  xDoneSub, xErrorSub, xReceived, xSubscribeActive : BOOL;
  eMQTTErrorSub : MQTT.MQTT_ERROR;

  // PRG variables
  uiTemperature : UINT := 0;
  STATE, uiCount : UINT := 0;
  RS_0 : RS;
  TON_0 : TON;
END_VAR
```

Figura 2.2: *Declaração de variáveis MQTT_simple.*

MQTT_simple - Structured text (ST)

```
// Check MQTTClient error
RS_0(SET := NOT xErrorClient, RESET1 := xErrorClient); // Set Reset
TON_0(IN := RS_0.Q1, PT := T#1S); // Timer

// FB MQTTClient
MQTTClient_0(
   xEnable := TON_0.Q, sHostname := sHostName, uiPort := uiPort, uiKeepAlive := 60,
   xError => xErrorClient, eMQTTError => eMQTTErrorClient, xConnectedToBroker => xConnectedToBroker, xDone => xDoneClient);

// Publish State Machine
CASE STATE OF

0: // Increment uiCount
   uiCount := uiCount + 1;
   IF uiCount > 50 THEN uiCount := 0; STATE := 1; END_IF;

1: // Increment of uiTemperature variable
   uiTemperature := uiTemperature + 1;
   IF uiTemperature > 1000 THEN uiTemperature := 0; END_IF;
   STATE := 2;

2: // Creation of the Payload and size
   sPayloadPub := CONCAT('simple MQTT message: temperature = ',UINT_TO_STRING(uiTemperature));
   udiPayloadSizePub := DINT_TO_UDINT(StrLenA(pbPayloadPub));
   STATE := 3;

3: // Publish
   MQTTPublish_0(
      xExecute := ((NOT xErrorPub) AND (NOT xDonePub) AND (xConnectedToBroker)),
      wsTopicName := wsTopicPub, eQoS := uiQoS, mqttClient := MQTTClient_0,
      pbPayload := pbPayloadPub, udiPayloadSize := udiPayloadSizePub,
      xDone => xDonePub, xError => xErrorPub, eMQTTError => eMQTTErrorPub);

   // Reset publisher
   IF xDonePub = 1 OR xErrorPub = 1 THEN MQTTPublish_0.xExecute := 0; STATE := 0; END_IF

END_CASE;

// FB MQTTSubscribe
MQTTSubscribe_0(
   xEnable := ((NOT xErrorSub) AND (NOT xDoneSub) AND (xConnectedToBroker)),
   wsTopicFilter := wsTopicSub, eSubscribeQoS := uiQoS, mqttClient := MQTTClient_0,
   pbPayload := pbPayloadSub, udiMaxPayloadSize := udiMaxPayloadSize,
   xDone => xDoneSub, xError => xErrorSub, eMQTTError => eMQTTErrorSub,
   xReceived => xReceived, udiPayloadSize => udiPayloadSizeSub, xSubscribeActive => xSubscribeActive);
```

Figura 2.3: *Programa MQTT_simple em texto estruturado.*

2.6 PUBLICAÇÃO E SUBSCRIÇÃO COM ENCRIPTAÇÃO

Essa seção apresenta o exemplo **MQTT_encryption**, no qual a aplicação anterior é modificada para que a comunicação entre os clientes e o broker MQTT seja criptografada, garantindo que os dados envolvidos não possam ser interceptados ou manipulados por terceiros não autorizados. Essa comunicação utiliza o protocolo **TLS** (*Transport Layer Security*), o qual envolve o uso de certificados digitais para autenticar as partes envolvidas na comunicação e criptografar os dados transmitidos.

Da mesma forma que a aplicação anterior, neste exemplo o FB **MQTTClient** conecta o dispositivo ao broker test.mosquitto.org e realiza a publicação de uma string contendo o valor da variável uiTemperature, e enquanto isso, se subscreve no mesmo tópico. Neste caso, é utilizada a porta **8883**, a qual é a porta padrão para comunicações MQTT seguras. Além disso, a entrada **xUseTLS** do FB MQTTClient deve ser configurada com o valor booleano **True**. O restante do código é exatamente igual ao exemplo anterior.

NOTA!

Usualmente a porta padrão para comunicações não encriptadas é a **1883**, enquanto a porta **8883** é geralmente utilizada para encriptação. Contudo, deve-se consultar as características e as portas disponíveis no broker MQTT específico.

A Figura 2.4 mostra a declaração de variáveis e a Figura 2.5 mostra o código em ST para a aplicação MQTT_encryption.

MQTT_encryption - Declaration

```
PROGRAM MQTT_encryption
VAR
  // Functions Blocks
  MQTTClient_0: MQTT.MQTTClient;
  MQTTPublish_0: MQTT.MQTTPublish;
  MQTTSubscribe_0: MQTT.MQTTSubscribe;

  // MQTTClient
  sHostName : STRING := 'test.mosquitto.org';
  uiPort : UINT := 8883; // Encripted port
  xUseTLS : BOOL := 1; // Using TLS
  xConnectedToBroker, xDoneClient, xErrorClient : BOOL;
  eMQTTErrorClient : MQTT.MQTT_ERROR;
  uiQoS : MQTT.MQTT_QOS := 1;

  // MQTTPublish
  sPayloadPub : STRING(255);
  pbPayloadPub : POINTER TO BYTE := ADR(sPayloadPub);
  wsTopicPub : WSTRING(1024) := "weg/drivesbt/plc";
  udiPayloadSizePub : UDINT;
  xDonePub, xErrorPub : BOOL;
  eMQTTErrorPub : MQTT.MQTT_ERROR;

  // MQTTSubscribe
  sPayloadSub : STRING(255);
  pbPayloadSub : POINTER TO BYTE := ADR(sPayloadSub);
  wsTopicSub: WSTRING(1024) := "weg/drivesbt/#";
  udiMaxPayloadSize : UDINT := 1024;
  udiPayloadSizeSub : UDINT;
  xDoneSub, xErrorSub, xReceived, xSubscribeActive : BOOL;
  eMQTTErrorSub : MQTT.MQTT_ERROR;

  // PRG variables
  STATE, uiCount, uiTemperature : UINT := 0;
  RS_0 : RS;
  TON_0 : TON;
END_VAR
```

Figura 2.4: *Declaração de variáveis MQTT_encryption.*

MQTT_encryption - Structured text (ST)

```
// Check MQTTClient error
RS_0(SET := NOT xErrorClient, RESET1 := xErrorClient); // Set Reset
TON_0(IN := RS_0.Q1, PT := T#1S); // Timer

// FB MQTTClient
MQTTClient_0(
   xEnable := TON_0.Q, sHostname := sHostName, uiPort := uiPort, uiKeepAlive := 60, xUseTLS := xUseTLS,
   xError => xErrorClient, eMQTTError => eMQTTErrorClient, xConnectedToBroker => xConnectedToBroker, xDone => xDoneClient);

// Publish State Machine
CASE STATE OF

0: // Increment uiCount
   uiCount := uiCount + 1;
   IF uiCount > 50 THEN uiCount := 0; STATE := 1; END_IF;

1: // Increment of uiTemperature variable
   uiTemperature := uiTemperature + 1;
   IF uiTemperature > 1000 THEN uiTemperature := 0; END_IF;
   STATE := 2;

2: // Creation of the Payload and size
   sPayloadPub := CONCAT('simple MQTT message: temperature = ',UINT_TO_STRING(uiTemperature));
   udiPayloadSizePub := DINT_TO_UDINT(StrLenA(pbPayloadPub));
   STATE := 3;

3: // Publish
   MQTTPublish_0(
      xExecute := ((NOT xErrorPub) AND (NOT xDonePub) AND (xConnectedToBroker)),
      wsTopicName := wsTopicPub, eQoS := uiQoS, mqttClient := MQTTClient_0,
      pbPayload := pbPayloadPub, udiPayloadSize := udiPayloadSizePub,
      xDone => xDonePub, xError => xErrorPub, eMQTTError => eMQTTErrorPub);

   // Reset publisher
   IF xDonePub = 1 OR xErrorPub = 1 THEN MQTTPublish_0.xExecute := 0; STATE := 0; END_IF

END_CASE;

// FB MQTTSubscribe
MQTTSubscribe_0(
   xEnable := ((NOT xErrorSub) AND (NOT xDoneSub) AND (xConnectedToBroker)),
   wsTopicFilter := wsTopicSub, eSubscribeQoS := uiQoS, mqttClient := MQTTClient_0,
   pbPayload := pbPayloadSub, udiMaxPayloadSize := udiMaxPayloadSize,
   xDone => xDoneSub, xError => xErrorSub, eMQTTError => eMQTTErrorSub,
   xReceived => xReceived, udiPayloadSize => udiPayloadSizeSub, xSubscribeActive => xSubscribeActive);
```

Figura 2.5: *Programa MQTT_encryption em texto estruturado.*

Antes da comunicação MQTT ser iniciada, um *handshake* TLS ocorre entre o cliente e o servidor. Durante esse processo, as duas partes negociam os parâmetros de segurança, como o algoritmo de criptografia e a troca de chaves. Após, o servidor envia um certificado digital para autenticar sua identidade para o cliente. Toda a comunicação subsequente é criptografada usando as chaves compartilhadas estabelecidas durante o *handshake*. O certificado do servidor pode ser encontrado no **Security Screen** do Codesys, sendo classificado inicialmente como **Quarantined Certificates**. É possível arrastar o certificado para a pasta **Trusted Certificates**, de forma a garantir para o Codesys que a fonte é confiável.

A Figura 2.6 mostra a *Security Screen* do Codesys e o certificado digital pertencente ao broker *Mosquitto*.

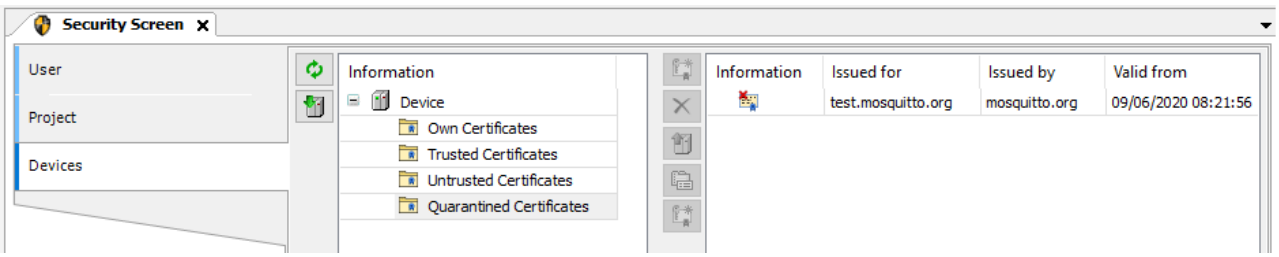


Figura 2.6: *Security Screen do Codesys.*

3 JSON UTILITIES SL

O formato JSON é estruturado como uma coleção de pares chave-valor, onde uma chave é uma string e o valor pode ser um número, uma string, um booleano, um objeto, um array, ou nulo. Esses pares chave-valor são agrupados em objetos, que por sua vez podem ser aninhados em outros objetos ou arrays. Essa estrutura hierárquica do JSON permite representar dados de maneira organizada, facilitando a leitura e a manipulação por máquinas e humanos.

A biblioteca **JSON Utilities SL**, por sua vez, é utilizada para a leitura e escrita de dados no formato JSON a partir de controladores utilizando Codesys. Nela, os dados são armazenados em uma matriz de estruturas do tipo **JSONData**, onde os elementos são armazenados na forma de **JSONElements**.

Um JSONElement possui a seguinte estrutura básica:

- **Value**: O valor do dado é retornado como wsValue (WSTRING), xValue (BOOL), lrValue (LREAL) ou liValue (LINT).
- **eType**: Tipo do elemento. Pode retornar os seguintes valores:

Nome	Valor	Descrição
KEY	0	O elemento é uma chave armazenada como WSTRING.
WSTRING_VALUE	1	Elemento é um valor WSTRING.
LINT_VALUE	2	Elemento é um valor LINT.
LREAL_VALUE	3	Elemento é um valor LREAL.
BOOL_VALUE	4	Elemento é um valor BOOL.
NULL	5	Elemento é NULL.
JSON_ARRAY	6	Elemento é um ARRAY.
JSON_OBJECT	7	Elemento é um objeto.
NONE		Sem tipo JSON.

- **diIndex**: Índice do elemento na matriz de dados. O tipo é DINT.
- **diParentIndex**: Índice do elemento que está a um nível hierárquico acima do elemento em questão. O tipo é DINT.

A biblioteca JSON Utilities possui os seguintes blocos de função:

- **FindFirstValueByKey**: Encontra o valor correspondente atrelado a uma dada chave.
- **JSONBuilder**: Criação simples e rápida de um arquivo JSON.
- **JSONByteArrayReader**: Lê dados de uma WSTRING no formato JSON e salva no formato JSONData.
- **JSONByteArrayWriter**: Escreve em uma WSTRING o conteúdo de um JSONData.
- **JSONData**: Principal bloco de funções da biblioteca. Este FB contém os dados do JSON propriamente dito e possui métodos para acessar, buscar e configurar os dados.
- **JSONDataFactory**: Necessário para alocar memória para a criação de um JSONData.
- **JSONFileReader**: Lê dados de um arquivo no formato JSON e salva no formato JSONData.
- **JSONFileWriter**: Escreve em um arquivo o conteúdo de um JSONData.

A Figura 3.1 mostra a biblioteca JSON Utilities SL no Codesys.

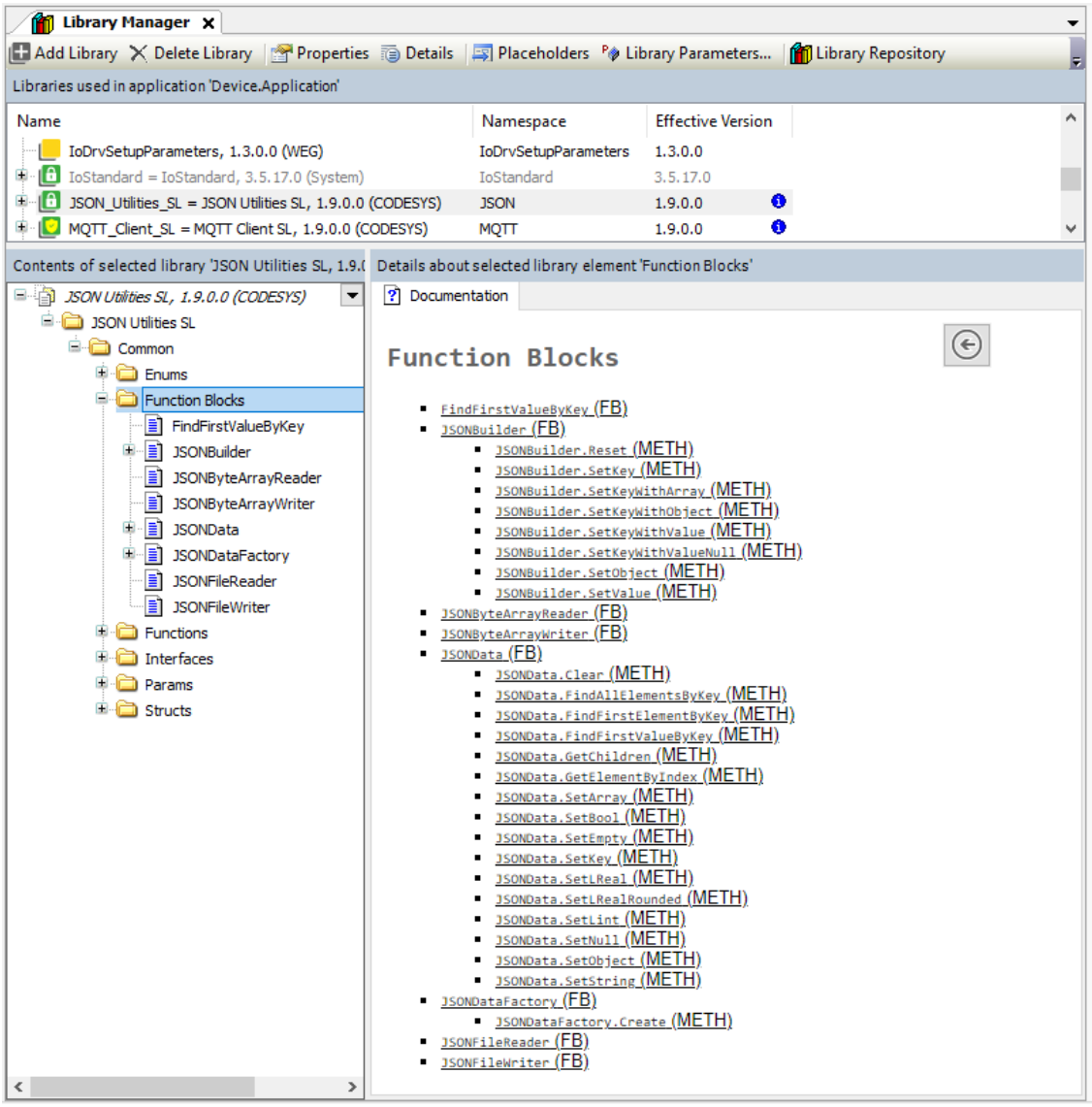


Figura 3.1: *Biblioteca JSON Utilities SL no Codesys.*

3.1 INDEX DOS ELEMENTOS

No Codesys, é necessário entender a sintaxe de como os elementos do JSON são organizados, de forma a manipulá-los da melhor forma. A Figura 3.2 mostra um exemplo de JSON com os respectivos Index, Parent Index, Tipo e Valor de cada elemento.

```
{(0)
   "device1"(1): {(2)
       "temperature"(3): 39.2(4),
       "humidity"(5) : 25(6)
    },
   "device2"(7): {(8)
       "motor1"(9): {(10)
          "rotation"(11):"clockwise"  (12),
          "fault"(13): 0(14)
          }
    }
  "time"(15): 1705424300(16)
}
```

Index	Parent Index	Tipo	Valor
0	-1	Objeto	
1	0	Chave	device1
2	1	Objeto	
3	2	Chave	temperature
4	3	LREAL	39.2
5	2	Chave	humidity
6	5	LINT	25
7	0	Chave	device2
8	7	Objeto	
9	8	Chave	motor1
10	9	Objeto	
11	10	Chave	rotation
12	11	STRING	clockwise
13	10	Chave	fault
14	13	BOOL	0
15	0	Chave	time
16	15	LINT	1705424300

Figura 3.2: *Exemplo de JSON com os respectivos índices dos elementos.*

O JSON acima ilustra dados sendo publicados de um equipamento que está adquirindo informações de dois dispositivos distintos, *device1* e *device2*. O *device1* possui duas chaves, *temperature* e *humidity*, que contém os valores 39.2 e 25, uma do tipo LREAL e a outra do tipo LINT, respectivamente. O *device2* por sua vez, contém informações relativas à chave *motor1*, a qual possui mais duas chaves internas, *rotation* e *fault*, representando o sentido de rotação e o estado de falha deste motor. Essas variáveis são do tipo STRING e BOOL, respectivamente. Por fim, tem-se a chave *time*, contendo o tempo unix em que o dado foi publicado.

O Index de cada elemento é mostrado em superescrito, entre parênteses, sendo incrementado a cada novo elemento criado. Considera-se como Index inicial, ou raíz, o número -1.

Todo elemento possui um Parent Index, que é o índice do elemento que está a um nível hierárquico acima do elemento em questão. O Parent Index de um valor é sempre a sua respectiva chave.

Na próxima seção é mostrado o código em ST para criar o JSON apresentado, detalhando os blocos de função e os principais métodos utilizados.

3.2 CRIAÇÃO E ESCRITA DE UM JSON

Para criar o JSON apresentado na seção anterior, são utilizados os blocos de função **JSONDataFactory**, **JSONData** e **JSONByteArrayWriter**. O JSONDataFactory é utilizado apenas na declaração de variáveis, de forma a alocar a memória necessária para a criação efetiva da estrutura do tipo JSONData. Sendo assim, para cada JSON utilizado deve-se criar um ponteiro específico do tipo **POINTER TO JSONData**. Para a checagem de erro, é preciso adicionar a biblioteca **CAA FB Factory**, uma vez que a saída do método **Create** do JSONDataFactory é do tipo **FBF.ERROR**.

O FB JSONData é o responsável pela criação dos objetos, chaves e valores presentes no JSON. Existem métodos específicos desse FB para a criação de BOOL, LINT, LREAL, STRING, ARRAY, etc. Além disso, existem métodos para a busca de valores, chaves e elementos, os quais são abordados na seção 3.4.

O FB JSONByteArrayWriter é utilizado ao final do código para escrever o conteúdo do JSONData criado em uma variável WSTRING. Nesse caso, o conteúdo do ponteiro pJsonData será escrito na string **wsJsonData**.

A maioria das variáveis utilizadas neste exemplo foram declaradas com valores constantes. Contudo, em aplicações reais, estes dados são obtidos de sensores, equipamentos e outros dispositivos industriais, conforme a necessidade do projetista.

Apesar de terem sido utilizadas constantes na maior parte do código, a variável **liDateTime** representa o tempo atual da criação do payload em tempo unix. Ela é calculada a partir da utilização da função **SysTimeRtcHighResGet**, presente na biblioteca **SysTimeRtc**.

A variável **xStartWrite** inicia com valor alto, e garante que o programa seja executado até que a variável **xDoneWrite** adquira também o valor alto, o que acontece quando a escrita da string wsJsonDataWrite é realizada com sucesso pelo FB jsonArrayWriter.

> **NOTA!**
>
> Para a utilização dos blocos de função **JSONDataFactory**, **JSONData** e **JSONByteArrayWriter**, não esqueça de adicionar a biblioteca **JSON Utilities SL** na aplicação do Codesys.

A Figura 3.3 mostra a declaração de variáveis e a Figura 3.4 mostra o código em ST para a criação do JSON apresentado na seção anterior.

WriteJsonData - Declaration

```
PROGRAM WriteJsonData
VAR
  // Creating of the JSONData
  factory : JSON.JSONDataFactory; // Allocate memory to the JSONData
  eDataFactoryError : FBF.ERROR; // Error code: 0: OK, 30103: no more memory
  pJsonDataWrite : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);

  // Function Blocks
  jsonArrayWriter : JSON.JSONByteArrayWriter;

  // PRG variables
  wsJsonDataWrite : WSTRING(1024);
  xStartWrite : BOOL := 1; xDoneWrite : BOOL := 0;
  diCounter, diObject1, diObject2, diObject3 : DINT;
  bFault : BOOL := 0;
  liDateTime : LINT := 0;
  lrTemperature : LREAL := 39.2;
  liHumidity : LINT := 25;
  Rtc : ULINT;

END_VAR
```

Figura 3.3: *Declaração de variáveis do programa WriteJsonData.*

WriteJsonData - Structured text (ST)

```
IF xStartWrite = TRUE AND xDoneWrite = FALSE THEN

  // First element (object) - Index of the root object is -1.
  diCounter := 0;
  pJsonDataWrite^.SetObject(diIndex := diCounter, diParentIndex := -1);

  // Key inside an object - The parent index of the key is the index of the parent object.
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "device1", diParentIndex := 0, diIndex := diCounter);

  // Object inside "device1" - The parent index of the value is the index of corresponding key.
  diCounter := diCounter + 1; diObject1 := diCounter;
  pJsonDataWrite^.SetObject(diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - temperature (LREAL)
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "temperature", diParentIndex := diObject1, diIndex := diCounter);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetLReal(lrValue := lrTemperature, diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - humidity (LINT)
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "humidity", diParentIndex := diObject1, diIndex := diCounter);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetLInt(liValue := liHumidity, diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - device2
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "device2", diParentIndex := 0, diIndex := diCounter);

  // Object inside device2
  diCounter := diCounter + 1; diObject2 := diCounter;
  pJsonDataWrite^.SetObject(diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - motor1
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "motor1", diParentIndex := diObject2, diIndex := diCounter);

  // Object inside motor1
  diCounter := diCounter + 1; diObject3 := diCounter;
  pJsonDataWrite^.SetObject(diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - rotation (STRING)
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "rotation", diParentIndex := diObject3, diIndex := diCounter);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetString(wsValue := "clockwise", diIndex := diCounter, diParentIndex := diCounter - 1);

  // Key inside an object - fault (BOOL)
  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "fault", diParentIndex := diObject3, diIndex := diCounter);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetBool(bValue := bFault, diIndex := diCounter, diParentIndex := diCounter - 1);

  // Timestamp (use the library: SysTimeRtc)
  SysTimeRtcHighResGet(pTimestamp := Rtc);
  liDateTime := ULINT_TO_LINT(Rtc);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetKey(wsKey := "time", diParentIndex := 0, diIndex := diCounter);

  diCounter := diCounter + 1;
  pJsonDataWrite^.SetLInt(liValue := liDateTime, diIndex := diCounter, diParentIndex := diCounter - 1);

  // Write the JSON String. The value will be written to the variable wsJsonDataWrite
  jsonArrayWriter(xExecute := xStartWrite, pwData := ADR(wsJsonDataWrite), udiSize := SIZEOF(wsJsonDataWrite),
     jsonData := pJsonDataWrite^, xDone => xDoneWrite);

  IF xDoneWrite = TRUE THEN xStartWrite := FALSE; END_IF

END_IF
```

Figura 3.4: *Programa WriteJsonData em texto estruturado.*

Com o programa rodando, é possível conferir o JSONData formado, analisando o index, parent index, tipo e valor de cada elemento. A Figura 3.5 mostra a tela de declaração de variáveis do programa WriteJsonData em modo *online*, onde podem ser conferidos os valores atuais das variáveis.

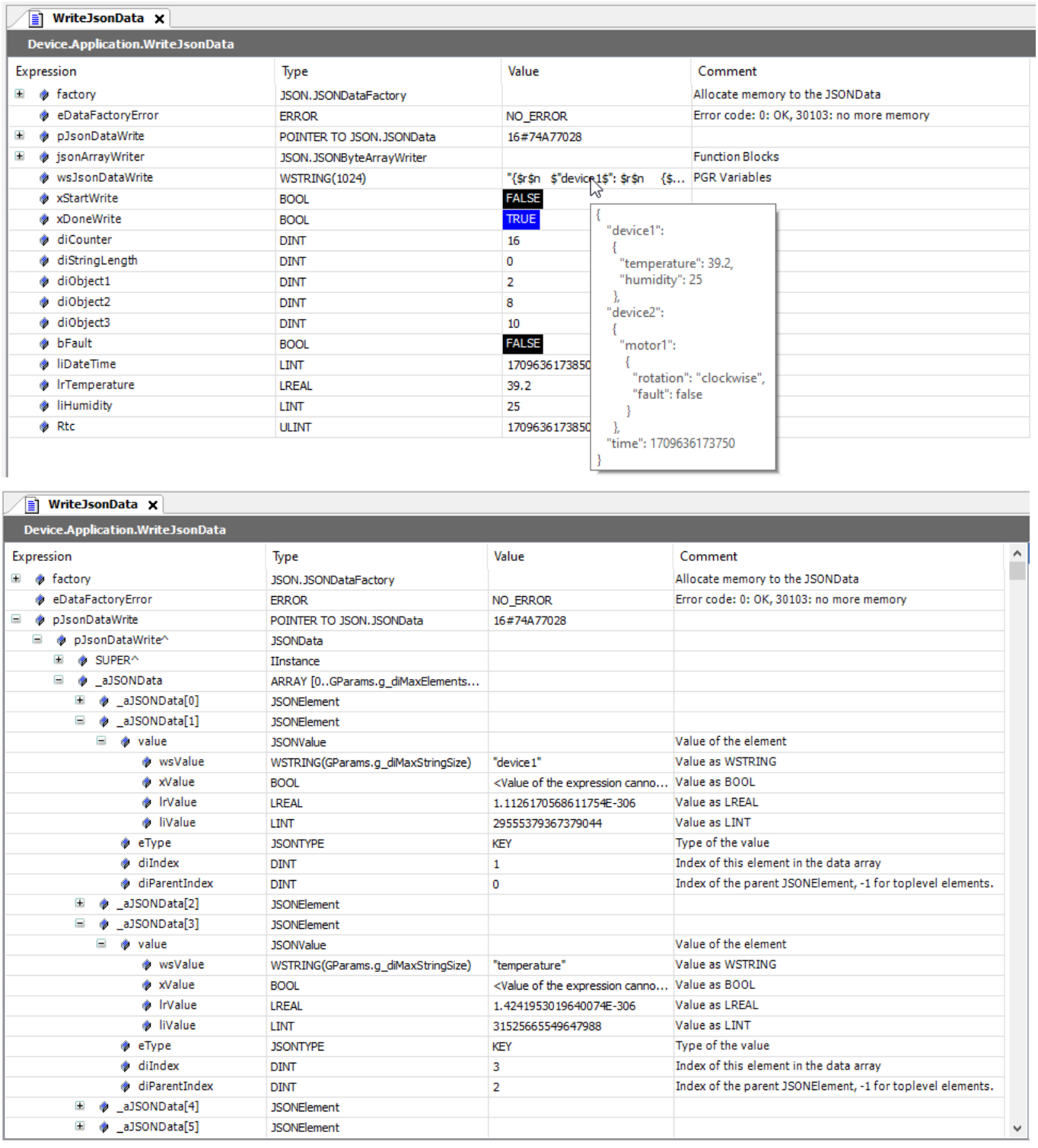


Figura 3.5: *Tela de declaração de variáveis do programa WriteJsonData em modo online.*

3.3 LEITURA DE UM JSON

Neste exemplo, considera-se a existência do programa **WriteJsonData**, criado na seção anterior. O WriteJsonData é executado como uma função no presente exemplo, de forma a garantir a criação da string **wsJsonDataWrite**, que armazena o conteúdo do ponteiro **pJsonDataWrite**.

Utilizando o FB **JSONByteArrayReader**, pode-se ler o conteúdo da string wsJsonDataWrite e atribuir esse valor a uma estrutura do tipo JSONData. Nota-se que, inicialmente, é realizada a declaração do ponteiro pJsonDataRead a partir do FB JSONDataFactory, garantindo assim que exista memória suficiente para a criação da estrutura JSONData.

A variável **xStartRead** inicia com valor alto, e garante que o programa seja executado até que a variável **xDoneRead** adquira também o valor alto, o que acontece quando a leitura da string wsJsonDataWrite é realizada com sucesso pelo FB jsonArrayReader, armazenando seu conteúdo no ponteiro **pJsonDataRead**.

NOTA!

Para a utilização dos blocos de função **JSONDataFactory**, **JSONData** e **JSONByteArrayReader**, não esqueça de adicionar a biblioteca **JSON Utilities SL** na aplicação do Codesys.

A biblioteca **StringUtils** pode ser adicionada para permitir a utilização da função **StrLenW**, o qual calcula o comprimento de uma WSTRING.

ReadJsonData - Declaration

```
PROGRAM ReadJsonData
VAR
  // Creating of the JSONData
  factory : JSON.JSONDataFactory; // Allocate memory to the JSONData
  eDataFactoryError : FBF.ERROR; // Error code: 0: OK, 30103: no more memory
  pJsonDataRead : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);

  // Function Blocks
  jsonArrayReader: JSON.JSONByteArrayReader;

  // PRG variables
  diStringSize : DINT;
  xStartRead  : BOOL := 1;
  xDoneRead : BOOL := 0;

END_VAR
```

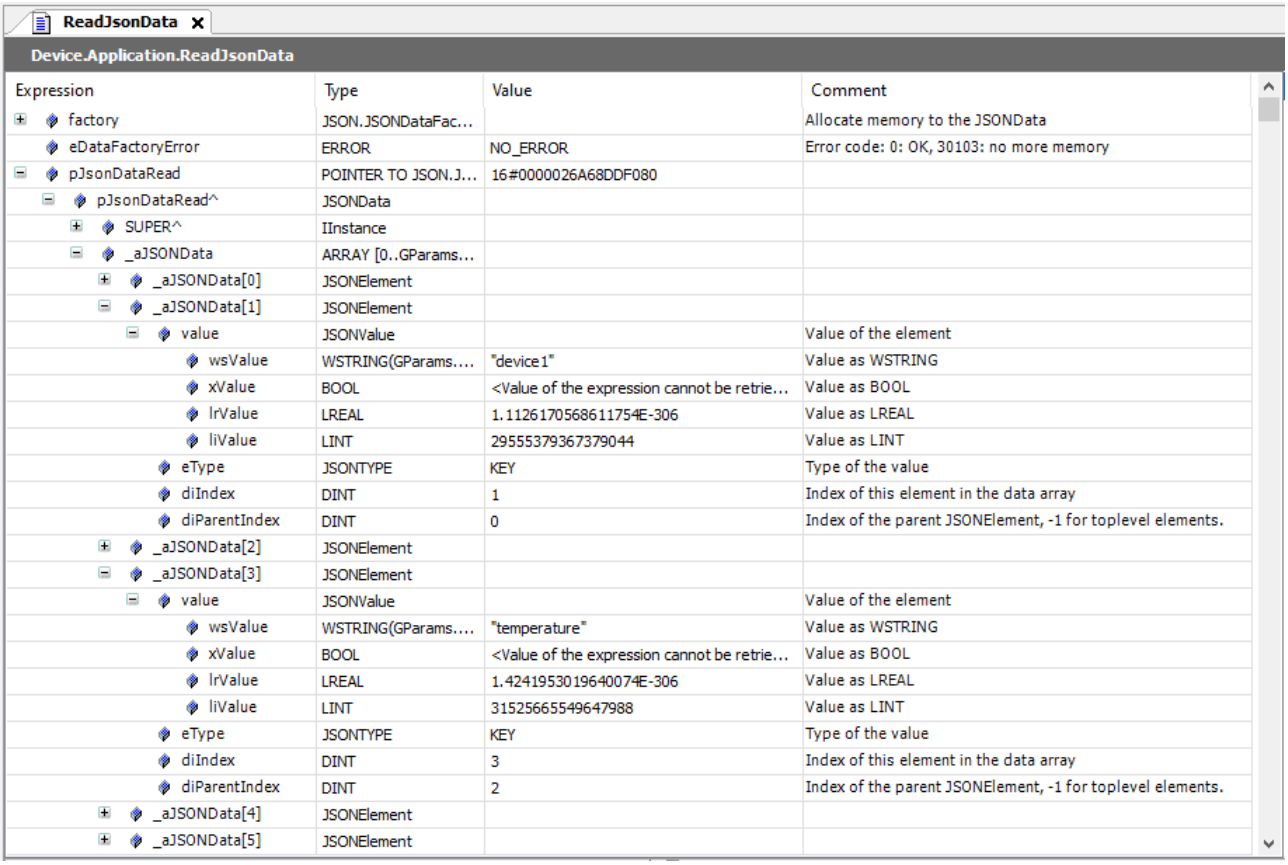
Figura 3.6: *Declaração de variáveis do programa ReadJsonData.*

ReadJsonData - Structured text (ST)

```
// Execute the program only once
IF xStartRead = TRUE AND xDoneRead = FALSE THEN

  // Execute the WriteJsonData function
  WriteJsonData();

  // Read the JSON String. The value will be written to the pointer to JSONData called pJsonDataRead
  jsonArrayReader(xExecute := xStartRead, pwData := ADR(WriteJsonData.wsJsonDataWrite), jsonData := pJsonDataRead^,
    xDone => xDoneRead);

  // Calculate the length of a WSTRING
  diStringSize := StrLenW(ADR(WriteJsonData.wsJsonDataWrite));

  IF xDoneRead = TRUE THEN xStartRead := FALSE; END_IF

END_IF;
```

Figura 3.7: *Programa ReadJsonData em texto estruturado.*

Com o programa rodando, é possível conferir o JSONData formado, analisando o index, parent index, tipo e valor de cada elemento. A Figura 3.8 mostra a tela de declaração de variáveis do programa ReadJsonData em modo *online*, onde podem ser conferidos os valores atuais das variáveis.



ReadJsonData

Device.Application.ReadJsonData

Expression	Type	Value	Comment
factory	JSON.JSONDataFac...		Allocate memory to the JSONData
eDataFactoryError	ERROR	NO_ERROR	Error code: 0: OK, 30103: no more memory
pJsonDataRead	POINTER TO JSON.J...	16#0000026A68DDF080	
pJsonDataRead^	JSONData		
SUPER^	IInstance		
_aJSONData	ARRAY [0..GParams...		
_aJSONData[0]	JSONElement		
_aJSONData[1]	JSONElement		
value	JSONValue		Value of the element
wsValue	WSTRING(GParams....	"device1"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot be retrie...	Value as BOOL
lrValue	LREAL	1.1126170568611754E-306	Value as LREAL
liValue	LINT	29555379367379044	Value as LINT
eType	JSONTYPE	KEY	Type of the value
diIndex	DINT	1	Index of this element in the data array
diParentIndex	DINT	0	Index of the parent JSONElement, -1 for toplevel elements.
_aJSONData[2]	JSONElement		
_aJSONData[3]	JSONElement		
value	JSONValue		Value of the element
wsValue	WSTRING(GParams....	"temperature"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot be retrie...	Value as BOOL
lrValue	LREAL	1.4241953019640074E-306	Value as LREAL
liValue	LINT	31525665549647988	Value as LINT
eType	JSONTYPE	KEY	Type of the value
diIndex	DINT	3	Index of this element in the data array
diParentIndex	DINT	2	Index of the parent JSONElement, -1 for toplevel elements.
_aJSONData[4]	JSONElement		
_aJSONData[5]	JSONElement		

Figura 3.8: *Tela de declaração de variáveis do programa ReadJsonData em modo online.*

3.4 BUSCA DE ELEMENTOS, CHAVES E VALORES

A biblioteca JSON Utilities SL também fornece métodos para a busca de valores, chaves e elementos, contidos no FB JSONData. Destacam-se os seguintes métodos:

- **FindFirstElementByKey**: Busca o primeiro elemento na matriz de dados com base em uma chave específica.
- **FindFirstValueByKey**: Busca o primeiro valor na matriz de dados com base em uma chave específica.
- **GetElementByIndex**: Obtém o elemento com base no índice específico.
- **JSONElementToString**: Função que permite transformar o valor de um JSONElement em uma string.

A aplicação **FindGetJson** implementa os métodos citados acima de forma a obter elementos, chaves e valores contidos no ponteiro **pJsonDataWrite**. Para isso, considera-se a existência do programa **WriteJsonData**, criado anteriormente. O WriteJsonData é executado como uma função, de forma a garantir a criação do ponteiro pJsonDataWrite, o qual contém uma estrutura do tipo JSONData.

A Figura 3.9 apresenta a declaração de variáveis e a Figura 3.10 o código em ST da aplicação FindGetJson.

FindGetJson - Declaration

```
PROGRAM FindGetJson
VAR

// jsonElements
   jsonElement1, jsonElement2, jsonElement3, jsonElement4 : JSON.JSONElement;

// jsonElement1
   diElement1Type, diElement1Index, diElement1ParIndex : DINT;
   wsElement1Value : WSTRING(JSON.GParams.g_diMaxStringSize);

// jsonElement2
   diElement2Type, diElement2Index, diElement2ParIndex : DINT;
   lrElement2Value : LREAL;

// jsonElement3
   wsElement3Value : WSTRING(JSON.GParams.g_diMaxStringSize);

// jsonElement4
   wsElement4Value : WSTRING(JSON.GParams.g_diMaxStringSize);

// PRG Variables
   xDoneFindGet : BOOL := 0;

END_VAR
```

Figura 3.9: *Declaração de variáveis do programa FindGetJson.*

FindGetJson - Structured text (ST)

```
IF xDoneFindGet = FALSE THEN

  // Execute the WriteJsonData function
  WriteJsonData();

  // Search for the first element by Key -> device1
  WriteJsonData.pJsonDataWrite^.FindFirstElementByKey(wsKey := "device1", diStartIndex := 0, jsonElement => jsonElement1);

  // Search for the first value by Key -> temperature
  WriteJsonData.pJsonDataWrite^.FindFirstValueByKey(wsKey := "temperature", diStartIndex := 0, jsonElement => jsonElement2);

  // Get the Element by Index
  WriteJsonData.pJsonDataWrite^.GetElementByIndex(diIndex := 15, jsonElement => jsonElement3);
  WriteJsonData.pJsonDataWrite^.GetElementByIndex(diIndex := 16, jsonElement => jsonElement4);

  // jsonElement1 = element of the key
  diElement1Type := jsonElement1.eType;
  wsElement1Value := jsonElement1.value.wsValue;
  diElement1Index := jsonElement1.diIndex;
  diElement1ParIndex := jsonElement1.diParentIndex;

  // jsonElement2 = value of the key
  diElement2Type := jsonElement2.eType;
  lrElement2Value := jsonElement2.value.lrValue;
  diElement2Index := jsonElement2.diIndex;
  diElement2ParIndex := jsonElement2.diParentIndex;

  // jsonElement3
  JSON.JSONElementToString(element := jsonElement3, wsResult := wsElement3Value);

  // jsonElement4
  JSON.JSONElementToString(element := jsonElement4, wsResult := wsElement4Value);

END_IF
```

Figura 3.10: *Programa FindGetJson em texto estruturado.*

De forma semelhante ao mostrado na seção anterior, com o programa rodando é possível conferir o conteúdo dos elementos obtidos a partir do uso dos métodos presentes no FB JSONData. A Figura 3.11 mostra a tela de declaração de variáveis do programa FindGetJson em modo *online*, e a Figura 3.12 mostra os valores atribuídos às variáveis.

FindGetJson

Device.Application.FindGetJson

Expression	Type	Value	Comment
jsonElement1	JSON.JSONElement		jsonElements
value	JSONValue		Value of the element
wsValue	WSTRING(GParams.g_diMaxStringSize)	"device1"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot ...	Value as BOOL
lrValue	LREAL	1.1126170568611754E-306	Value as LREAL
liValue	LINT	29555379367379044	Value as LINT
eType	JSONTYPE	KEY	Type of the value
diIndex	DINT	1	Index of this element in the data array
diParentIndex	DINT	0	Index of the parent JSONElement, -1 for toplevel elements.
jsonElement2	JSON.JSONElement		jsonElements
value	JSONValue		Value of the element
wsValue	WSTRING(GParams.g_diMaxStringSize)	"豹香香睐"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot ...	Value as BOOL
lrValue	LREAL	39.2	Value as LREAL
liValue	LINT	4630713726853028250	Value as LINT
eType	JSONTYPE	LREAL_VALUE	Type of the value
diIndex	DINT	4	Index of this element in the data array
diParentIndex	DINT	3	Index of the parent JSONElement, -1 for toplevel elements.
jsonElement3	JSON.JSONElement		jsonElements
value	JSONValue		Value of the element
wsValue	WSTRING(GParams.g_diMaxStringSize)	"time"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot ...	Value as BOOL
lrValue	LREAL	9.34605036874169O8E-307	Value as LREAL
liValue	LINT	28429440806092916	Value as LINT
eType	JSONTYPE	KEY	Type of the value
diIndex	DINT	15	Index of this element in the data array
diParentIndex	DINT	0	Index of the parent JSONElement, -1 for toplevel elements.
jsonElement4	JSON.JSONElement		jsonElements
value	JSONValue		Value of the element
wsValue	WSTRING(GParams.g_diMaxStringSize)	"□□□"	Value as WSTRING
xValue	BOOL	<Value of the expression cannot ...	Value as BOOL
lrValue	LREAL	8.446743499649 81E-312	Value as LREAL
liValue	LINT	1709639917438	Value as LINT
eType	JSONTYPE	LINT_VALUE	Type of the value
diIndex	DINT	16	Index of this element in the data array
diParentIndex	DINT	15	Index of the parent JSONElement, -1 for toplevel elements.
diElement1Type	DINT	0	jsonElement1

Figura 3.11: *Tela de declaração de variáveis do programa FindGetJson em modo online.*

```
// jsonElement1 = element of the key
diElement1Type 0      := jsonElement1.eType KEY;
wsElement1Value "device1"    := jsonElement1.value.wsValue "device1";
diElement1Index 1     := jsonElement1.diIndex 1;
diElement1ParIndex 0  := jsonElement1.diParentIndex 0;

// jsonElement2 = value of the key
diElement2Type 3      := jsonElement2.eType LREAL_VALU ▸;
lrElement2Value 39.2  := jsonElement2.value.lrValue 39.2;
diElement2Index 4     := jsonElement2.diIndex 4;
diElement2ParIndex 3  := jsonElement2.diParentIndex 3;

// jsonElement3
JSON.JSONElementToString(element := jsonElement3, wsResult := wsElement3Value "time");

// jsonElement4
JSON.JSONElementToString(element := jsonElement4, wsResult := wsElement4Value "1705682686 ▸");
```

Figura 3.12: *Valores atribuídos às variáveis do programa FindGetJson em modo online.*

4 MQTT CLIENT + JSON UTILITIES

De forma a demonstrar a utilização das bibliotecas **MQTT Client SL** e **JSON Utilities SL** em conjunto, são criadas aplicações que mostram a criação e publicação de um JSON, bem como a subscrição e consequente manipulação do JSON recebido.

4.1 CRIAÇÃO E PUBLICAÇÃO DE UM JSON

Na aplicação **MQTT_JSON_Pub** são criados dois JSONs, um de estado, contendo o valor atual das duas primeiras saídas digitais do PLC, e o outro de comando, o qual é utilizado para alterar o estado destas saídas digitais. Os tópicos de publicação são o "**weg/drivesbt/state**" e o "**weg/drivesbt/command**", respectivamente. São utilizados dois blocos de função do tipo **MQTTPublish**, cada um responsável pela publicação de um JSON. Emprega-se uma máquina de estados para realizar o gerenciamento destas operações.

A Figura 4.1 mostra a declaração de variáveis e a Figura 4.2 mostra o código em ST para a aplicação MQTT_JSON_Pub.

MQTT_JSON_Pub - Declaration

```
PROGRAM MQTT_JSON_Pub
VAR
  // Functions Blocks MQTT
  MQTTClient_0: MQTT.MQTTClient;
  MQTTPublish_1, MQTTPublish_2: MQTT.MQTTPublish;

  // Creating of the JSONDatas
  factory : JSON.JSONDataFactory; // Allocate memory to the JSONData
  eDataFactoryError : FBF.ERROR; // Error code: 0: OK, 30103: no more memory
  pJsonData1 : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);
  pJsonData2 : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);

  // JSON
  jsonArrayWriter : JSON.JSONByteArrayWriter;
  wsJsonData1, wsJsonData2 : WSTRING(JSON.GParams.g_diMaxStringSize);
  xDoneWrite1, xDoneWrite2, xErrorWrite1, xErrorWrite2 : BOOL := 0;

  // MQTTClient
  sHostName : STRING := 'test.mosquitto.org';
  uiPort : UINT := 1883;
  xConnectedToBroker, xDoneClient, xErrorClient : BOOL;
  eMQTTErrorClient : MQTT.MQTT_ERROR;
  uiQoS : MQTT.MQTT_QOS := 1;

  // MQTTPublish
  sPayloadPub1, sPayloadPub2 : STRING(JSON.GParams.g_diMaxStringSize);
  pbPayloadPub1 : POINTER TO BYTE := ADR(sPayloadPub1);
  pbPayloadPub2 : POINTER TO BYTE := ADR(sPayloadPub2);
  wsTopicPub1 : WSTRING(1024) := "weg/drivesbt/state";
  wsTopicPub2 : WSTRING(1024) := "weg/drivesbt/command" ;
  udiPayloadSizePub1, udiPayloadSizePub2 : UDINT;
  xDonePub1, xErrorPub1, xDonePub2, xErrorPub2 : BOOL;
  eMQTTErrorPub1, eMQTTErrorPub2 : MQTT.MQTT_ERROR;

  // PRG variables
  diCounter, diObject1, diObject2 : DINT;
  liDateTime : LINT := 0;
  Rtc : ULINT;
  STATE, uiCount : UINT := 0;
  RS_0 : RS;
  TON_0 : TON;
END_VAR
```

Figura 4.1: *Declaração de variáveis da aplicação MQTT_JSON_Pub.*

MQTT_JSON_Pub - Structured text (ST) - Part 1

```
// Check MQTTClient error
RS_0(SET := NOT xErrorClient, RESET1 := xErrorClient); // Set Reset
TON_0(IN := RS_0.Q1, PT := T#1S); // Timer

// FB MQTTClient
MQTTClient_0(
   xEnable := TON_0.Q, sHostname := sHostName, uiPort := uiPort, uiKeepAlive := 60,
   xError => xErrorClient, eMQTTError => eMQTTErrorClient, xConnectedToBroker => xConnectedToBroker, xDone => xDoneClient);

// Publish State Machine
CASE STATE OF

0: // Increment uiCount
   uiCount := uiCount + 1;
   IF uiCount > 50 THEN uiCount := 0; STATE := 1; END_IF;

1: // Variables of the payload

   // Timestamp (use the library: SysTimeRtc)
   SysTimeRtcHighResGet(pTimestamp := Rtc);
   liDateTime := ULINT_TO_LINT(Rtc);

   // Switching DO states
   IF DO1 = 1 THEN DO1 := 0; ELSE DO1 := 1; END_IF;
   IF DO2 = 1 THEN DO2 := 0; ELSE DO2 := 1; END_IF;

STATE := 2;

2: // Creation of the first JSON (state)
   diCounter := 0;
   pJsonData1^.SetObject(diIndex := diCounter, diParentIndex := -1);
   diCounter := diCounter + 1;
   pJsonData1^.SetKey(wsKey := "state", diParentIndex := 0, diIndex := diCounter);
   diCounter := diCounter + 1; diObject1 := diCounter;
   pJsonData1^.SetObject(diIndex := diCounter, diParentIndex := diCounter - 1);

   diCounter := diCounter + 1;
   pJsonData1^.SetKey(wsKey := "DO1", diParentIndex := diObject1, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData1^.SetBool(bValue := DO1, diIndex := diCounter, diParentIndex := diCounter - 1);
   diCounter := diCounter + 1;
   pJsonData1^.SetKey(wsKey := "DO2", diParentIndex := diObject1, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData1^.SetBool(bValue := DO2, diIndex := diCounter, diParentIndex := diCounter - 1);

   diCounter := diCounter + 1;
   pJsonData1^.SetKey(wsKey := "time", diParentIndex := 0, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData1^.SetLInt(liValue := liDateTime, diIndex := diCounter, diParentIndex := diCounter - 1);

   STATE := 3;

3: // Creation of the second JSON (command)
   diCounter := 0;
   pJsonData2^.SetObject(diIndex := diCounter, diParentIndex := -1);
   diCounter := diCounter + 1;
   pJsonData2^.SetKey(wsKey := "command", diParentIndex := 0, diIndex := diCounter);
   diCounter := diCounter + 1; diObject2 := diCounter;
   pJsonData2^.SetObject(diIndex := diCounter, diParentIndex := diCounter - 1);

   diCounter := diCounter + 1;
   pJsonData2^.SetKey(wsKey := "DO1", diParentIndex := diObject2, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData2^.SetBool(bValue := DO1, diIndex := diCounter, diParentIndex := diCounter - 1);
   diCounter := diCounter + 1;
   pJsonData2^.SetKey(wsKey := "DO2", diParentIndex := diObject2, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData2^.SetBool(bValue := DO2, diIndex := diCounter, diParentIndex := diCounter - 1);

   diCounter := diCounter + 1;
   pJsonData2^.SetKey(wsKey := "time", diParentIndex := 0, diIndex := diCounter);
   diCounter := diCounter + 1;
   pJsonData2^.SetLInt(liValue := liDateTime, diIndex := diCounter, diParentIndex := diCounter - 1);

   STATE := 4;
```

MQTT_JSON_Pub - Structured text (ST) - Part 2

```
4: // Generate the wsJsonData1 JSON
   jsonArrayWriter(xExecute := TRUE, pwData := ADR(wsJsonData1), udiSize := SIZEOF(wsJsonData1), jsonData := pJsonData1^,
      xError => xErrorWrite1, xDone => xDoneWrite1);

   IF xDoneWrite1 = 1 OR xErrorWrite1 = 1 THEN
      jsonArrayWriter(xExecute := FALSE, pwData := ADR(wsJsonData1), udiSize := SIZEOF(wsJsonData1), jsonData := pJsonData1^,
         xError => xErrorWrite1, xDone => xDoneWrite1);
      STATE := 5;
   END_IF

5: // Generate the wsJsonData2 JSON
   jsonArrayWriter(xExecute := TRUE, pwData := ADR(wsJsonData2), udiSize := SIZEOF(wsJsonData2), jsonData := pJsonData2^,
      xError => xErrorWrite2, xDone => xDoneWrite2);

   IF xDoneWrite2 = 1 OR xErrorWrite2 = 1 THEN
      jsonArrayWriter(xExecute := FALSE, pwData := ADR(wsJsonData2), udiSize := SIZEOF(wsJsonData2), jsonData := pJsonData2^,
      xError => xErrorWrite2, xDone => xDoneWrite2);
      STATE := 6;
   END_IF

6: // Creation of the payloads and calculate their sizes

   // Calculates the WSTRING sizes
   udiPayloadSizePub1 := DINT_TO_UINT(StrLenW(pstData := ADR(wsJsonData1)));
   udiPayloadSizePub2 := DINT_TO_UINT(StrLenW(pstData := ADR(wsJsonData2)));

   // Convertion from UTF16 to UTF8 (WSTRING to STRING)
   ConvertUTF16toUTF8(sourceStart := ADR(wsJsonData1), targetStart := ADR(sPayloadPub1),
      dwTargetBufferSize := udiPayloadSizePub1, bStrictConversion := 0);

   ConvertUTF16toUTF8(sourceStart := ADR(wsJsonData2), targetStart := ADR(sPayloadPub2),
      dwTargetBufferSize := udiPayloadSizePub2, bStrictConversion := 0);

   // Calculates the STRING sizes
   udiPayloadSizePub1 := DINT_TO_UINT(StrLenA(pstData := pbPayloadPub1));
   udiPayloadSizePub2 := DINT_TO_UINT(StrLenA(pstData := pbPayloadPub2));

   STATE := 7;

7: // Publisher 1
   MQTTPublish_1(
      xExecute := ((NOT xErrorPub1) AND (NOT xDonePub1) AND (xConnectedToBroker)),wsTopicName := wsTopicPub1,
      eQoS := uiQoS, mqttClient := MQTTClient_0, pbPayload := pbPayloadPub1, udiPayloadSize := udiPayloadSizePub1,
      xDone => xDonePub1, xError => xErrorPub1, eMQTTError => eMQTTErrorPub1);

   // Reset publisher 1
   IF xDonePub1 = 1 OR xErrorPub1 = 1 THEN MQTTPublish_1.xExecute := 0; STATE := 8; END_IF

8: // Publisher 2
   MQTTPublish_2(
      xExecute := ((NOT xErrorPub2) AND (NOT xDonePub2) AND (xConnectedToBroker)), wsTopicName := wsTopicPub2,
      eQoS := uiQoS, mqttClient := MQTTClient_0, pbPayload := pbPayloadPub2, udiPayloadSize := udiPayloadSizePub2,
      xDone => xDonePub2, xError => xErrorPub2, eMQTTError => eMQTTErrorPub2);

   // Reset publisher 2
   IF xDonePub2 = 1 OR xErrorPub2 = 1 THEN MQTTPublish_2.xExecute := 0; STATE := 0; END_IF

END_CASE;
```

Figura 4.2: *Programa MQTT_JSON_Pub em texto estruturado.*

4.2 SUBSCRIÇÃO E MANIPULAÇÃO DE UM JSON

Neste exemplo, o FB **MQTTSubscribe_1** aguarda por payloads publicados no tópico "**weg/drivesbt/state**", e o **MQTTSubscribe_2** espera por publicações em "**weg/drivesbt/command**". Quando um payload é recebido, a variável **xReceived** do respectivo bloco de função vai para o estado alto, e o conteúdo do ponteiro recebido é transformado em um JSON, o qual pode ser facilmente manipulado para a aquisição das chaves e valores de interesse.

Os valores recebidos no tópico de estado são utilizados apenas para visualização do estado atual das duas primeiras saídas digitais do produto. Contudo, os valores recebidos no tópico de comando são utilizados para definir o estado destas duas saídas digitais.

A Figura 4.3 mostra a declaração de variáveis e a Figura 4.4 mostra o código em ST para a aplicação MQTT_JSON_Sub.

MQTT_JSON_Sub - Declaration

```
PROGRAM MQTT_JSON_Sub
VAR
  // Functions Blocks MQTT
  MQTTClient_0: MQTT.MQTTClient;
  MQTTSubscribe_1, MQTTSubscribe_2 : MQTT.MQTTSubscribe;

  // Creating of the JSONDatas
  factory : JSON.JSONDataFactory; // Allocate memory to the JSONData
  eDataFactoryError : FBF.ERROR; // Error code: 0: OK, 30103: no more memory
  pJsonDataState : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);
  pJsonDataCommand : POINTER TO JSON.JSONData := factory.Create(eError => eDataFactoryError);

  // JSON
  jsonArrayReaderSub1, jsonArrayReaderSub2 : JSON.JSONByteArrayReader;
  wsJsonDataState, wsJsonDataCommand : WSTRING(JSON.GParams.g_diMaxStringSize);
  jsonElement1, jsonElement2, jsonElement3, jsonElement4 : JSON.JSONElement;

  // MQTTClient
  sHostName : STRING := 'test.mosquitto.org';
  uiPort : UINT := 1883;
  xConnectedToBroker, xDoneClient, xErrorClient : BOOL;
  eMQTTErrorClient : MQTT.MQTT_ERROR;
  uiQoS : MQTT.MQTT_QOS := 1;

  // MQTTSubscribe
  sPayloadSubState, sPayloadSubCommand : STRING(JSON.GParams.g_diMaxStringSize);
  pbPayloadSubState : POINTER TO BYTE := ADR(sPayloadSubState);
  pbPayloadSubCommand : POINTER TO BYTE := ADR(sPayloadSubCommand);
  wsTopicSubState: WSTRING(1024) := "weg/drivesbt/state" ;
  wsTopicSubCommand: WSTRING(1024) := "weg/drivesbt/command" ;
  udiMaxPayloadSize : UDINT := DINT_TO_UDINT(JSON.GParams.g_diMaxStringSize);
  udiPayloadSizeSubState, udiPayloadSizeSubCommand : UDINT := DINT_TO_UDINT(JSON.GParams.g_diMaxStringSize);
  xDoneSub1, xErrorSub1, xReceived1, xSubscribeActive1 : BOOL;
  xDoneSub2, xErrorSub2, xReceived2, xSubscribeActive2 : BOOL;
  eMQTTErrorSub1, eMQTTErrorSub2: MQTT.MQTT_ERROR;

  // PRG variables
  liDateTime : LINT := 0;
  Rtc : ULINT;
  RS_0 : RS;
  TON_0 : TON;
END_VAR
```

Figura 4.3: *Declaração de variáveis MQTT_JSON_Sub.*

MQTT_JSON_Sub - Structured text (ST)

```
// Check MQTTClient error
RS_0(SET := NOT xErrorClient, RESET1 := xErrorClient); // Set Reset
TON_0(IN := RS_0.Q1, PT := T#1S); // Timer

// FB MQTTClient
MQTTClient_0(
   xEnable := TON_0.Q, sHostname := sHostName, uiPort := uiPort, uiKeepAlive := 60,
   xError => xErrorClient, eMQTTError => eMQTTErrorClient, xConnectedToBroker => xConnectedToBroker, xDone => xDoneClient);

// FB MQTTSubscribe 1
MQTTSubscribe_1(
   xEnable := ((NOT xErrorSub1) AND (NOT xDoneSub1) AND (xConnectedToBroker)), wsTopicFilter := wsTopicSubState,
   eSubscribeQoS := uiQoS, mqttClient := MQTTClient_0, pbPayload := pbPayloadSubState,
   udiMaxPayloadSize := udiMaxPayloadSize, xDone => xDoneSub1, xError => xErrorSub1, eMQTTError => eMQTTErrorSub1,
   xReceived => xReceived1, udiPayloadSize => udiPayloadSizeSubState, xSubscribeActive => xSubscribeActive1);

// FB MQTTSubscribe 2
MQTTSubscribe_2(
   xEnable := ((NOT xErrorSub2) AND (NOT xDoneSub2) AND (xConnectedToBroker)), wsTopicFilter := wsTopicSubCommand,
   eSubscribeQoS := uiQoS, mqttClient := MQTTClient_0, pbPayload := pbPayloadSubCommand,
   udiMaxPayloadSize := udiMaxPayloadSize, xDone => xDoneSub2, xError => xErrorSub2, eMQTTError => eMQTTErrorSub2,
   xReceived => xReceived2, udiPayloadSize => udiPayloadSizeSubCommand, xSubscribeActive => xSubscribeActive2);

// Payload received from "weg/drivesbt/state"
IF xReceived1 = 1 THEN

   // Treating the pointer to byte
   ConvertUTF8toUTF16(sourceStart := pbPayloadSubState, targetStart := ADR(wsJsonDataState),
      dwTargetBufferSize := udiMaxPayloadSize, bStrictConversion := 0);

   // Calculates the STRING size
   udiPayloadSizeSubState := DINT_TO_UINT(StrLenW(pstData := ADR(wsJsonDataState)));

   // Read the WSTRING and trasform it to JSON
   jsonArrayReaderSub1(xExecute := TRUE, pwData := ADR(wsJsonDataState), jsonData := pJsonDataState^);
   IF jsonArrayReaderSub1.xError = 1 OR jsonArrayReaderSub1.xDone = 1 THEN
      jsonArrayReaderSub1(xExecute := FALSE, pwData := ADR(wsJsonDataState), jsonData := pJsonDataState^);
   END_IF

   // Find the values of the keys DO1 and DO2
   pJsonDataState^.FindFirstValueByKey(wsKey := "DO1", diStartIndex := 1, jsonElement => jsonElement1);
   pJsonDataState^.FindFirstValueByKey(wsKey := "DO2", diStartIndex := 1, jsonElement => jsonElement2);

END_IF

// Payload received from "weg/drivesbt/command"
IF xReceived2 = 1 THEN

   // Treating the pointer to byte
   ConvertUTF8toUTF16(sourceStart := pbPayloadSubCommand, targetStart := ADR(wsJsonDataCommand),
      dwTargetBufferSize := udiMaxPayloadSize, bStrictConversion := 0);

   // Calculates the STRING size
   udiPayloadSizeSubCommand := DINT_TO_UINT(StrLenW(pstData := ADR(wsJsonDataCommand)));

   // Read the WSTRING and trasform it to JSON
   jsonArrayReaderSub2(xExecute := TRUE, pwData := ADR(wsJsonDataCommand), jsonData := pJsonDataCommand^);
   IF jsonArrayReaderSub2.xError = 1 OR jsonArrayReaderSub2.xDone = 1 THEN
      jsonArrayReaderSub2(xExecute := FALSE, pwData := ADR(wsJsonDataCommand), jsonData := pJsonDataCommand^);
   END_IF

   // Find the values of the keys and add it to the DO1 and DO2 variables of the PLCx
   pJsonDataCommand^.FindFirstValueByKey(wsKey := "DO1", diStartIndex := 1, jsonElement => jsonElement3);
   pJsonDataCommand^.FindFirstValueByKey(wsKey := "DO2", diStartIndex := 1, jsonElement => jsonElement4);

   DO1 := jsonElement3.value.xValue;
   DO2 := jsonElement4.value.xValue;

END_IF
```

Figura 4.4: *Programa MQTT_JSON_Sub em texto estruturado.*

4.3 PUBLICAÇÃO E SUBSCRIÇÃO

As aplicações 4.1 e 4.2 foram criados para possibilitar a sua integração, permitindo a publicação e a subscrição nos mesmos tópicos, podendo assim ser aplicados tanto em um único PLC quanto em PLCs distintos. Utilize ambos os exemplos simultaneamente para um processo completo de publicação e subscrição.

WEG Drives & Controls - Automação LTDA.
Jaraguá do Sul - SC - Brasil
Fone 55 (47) 3276-4000 - Fax 55 (47) 3276-4020
São Paulo - SP - Brasil
Fone 55 (11) 5053-2300 - Fax 55 (11) 5052-4212
automacao@weg.net
www.weg.net