STORIA DELLE VERSIONI

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rev** | **Date** | **Document** | **Issued by** | **Checked by** | **Approved by** |
| 1 | 14/06/2023 | File Exporter | Ugur Akcelik |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

INDICES

[1. Scope 2](#__RefHeading___Toc451794242)

[2. Campo di applicazione 2](#__RefHeading___Toc451794243)

[3. Definizioni ed abbreviazioni 2](#__RefHeading___Toc451794244)

[4. Riferimenti applicabili 2](#__RefHeading___Toc451794245)

[5. Responsabilità 2](#__RefHeading___Toc451794246)

[6. Modalità operative 2](#__RefHeading___Toc451794247)

[7. Documenti generati 2](#__RefHeading___Toc451794248)

[8. Allegati 2](#__RefHeading___Toc451794249)

# Scope

The scope of this Developer Manual document is to provide comprehensive information and guidance to other developers regarding the functionality and inner workings of the codebase. It aims to enable developers to understand and work effectively with the existing codebase, ensuring smooth collaboration and efficient development processes.

When working on the codebase, it is **highly recommended** to follow the guideline provided by ThingsBoard for rule node development. The guideline can be found at the following [URL](https://thingsboard.io/docs/user-guide/contribution/rule-node-development/): ThingsBoard's rule node development

Please refer to the guideline for best practices, coding standards, and development practices specific to ThingsBoard's guidelines. It will provide valuable insights and recommendations for contributing to the codebase effectively and ensuring the code quality aligns with the platform's standards.

# Overview of the Codebase

The codebase consists of two classes: **TbFtpNode** and **TbFtpNodeConfiguration**. These classes are part of a larger project related to ThingsBoard. Let's take a closer look at each class and their purpose within the codebase:

## TbFtpNode

The TbFtpNode class represents a specific type of node in the ThingsBoard platform that interacts with an FTP server.

It implements the necessary logic and functionality for handling FTP-related operations.

This class likely contains methods for establishing a connection to the FTP server, uploading and downloading files, and managing files and directories on the server.

It may also include additional methods for handling various FTP-related tasks and functionalities.

## TbFtpNodeConfiguration

The TbFtpNodeConfiguration class is a configuration class used by the TbFtpNode.

It implements the NodeConfiguration interface, which suggests that it provides a set of configurable properties specific to the FTP node.

The class includes various properties such as serverUrl, folder, username, password, port, tries, scheduleMethods, which define the FTP server details, connection parameters, and scheduling information.

It also overrides the defaultConfiguration() method from the interface, which returns a default configuration object with pre-defined values for the properties.

Together, these two classes contribute to the overall functionality of the FTP node within the ThingsBoard platform.

The TbFtpNode class encapsulates the logic for interacting with the FTP server, while the TbFtpNodeConfiguration class provides the necessary configuration options for the node.

# Code Documentation

## TbFtpNode Class

**The global class variables** listed above provide essential information and hold important data for the functionality of the FTP node. Here is a brief description of each variable:

A screen shot of a computer program

Description automatically generated with low confidence

**config**: Represents the configuration object that holds the FTP node's configuration settings, including server URL, username, password, port, scheduling parameters, etc.

**scheduledFuture**: Represents a scheduled future object that manages and controls the scheduling of FTP operations. It is used to schedule and cancel tasks associated with the FTP node.

**scheduler**: Represents a scheduled executor service, which provides a thread pool for executing scheduled FTP operations. It handles the execution of tasks based on the configured schedule.

**folderPath**: Specifies the folder path where the FTP node will perform file operations. By default, it is set to "D:\logs". This variable determines the directory on the local machine where files will be transferred to or from the FTP server.

**deviceList**: Holds the list of devices associated with the FTP node. It stores the devices for which the FTP node will perform file transfer operations. The list allows the node to track and manage the devices involved in FTP operations.

**ctx**: Represents the ThingsBoard context object, which provides access to the node's runtime environment and resources. It allows the FTP node to interact with the ThingsBoard platform, access its services, and perform various operations.

**customerId**: Identifies the customer associated with the FTP node. It represents the customer ID for which the FTP node is operating. This variable allows the node to handle operations specific to a particular customer.

**Inherited Methods**

By implementing methods from the TbNode interface, the TbFtpNode class becomes a functional node in the ThingsBoard system, capable of handling initialization and processing of messages. These methods play a crucial role in defining the behavior of the node and enabling seamless integration with the ThingsBoard platform:

**init(TbContext ctx, TbNodeConfiguration configuration)**

This method is responsible for initializing the TbFtpNode instance. It is called during the node initialization process and receives the TbContext object and a TbNodeConfiguration object as parameters. The method sets up the necessary resources and configurations for the node to operate correctly.

**onMsg(TbContext ctx, TbMsg msg)**

This method is invoked whenever a message is received by the TbFtpNode. It handles the incoming message and performs the necessary processing or actions based on the message content. The TbContext object provides access to the execution context, while the TbMsg object represents the received message. This method is where the core functionality and logic of the node are implemented.

**destroy()**

The destroy() method is responsible for cleaning up and releasing any resources held by the TbFtpNode instance. It is called during the node's termination or when it is being removed from the ThingsBoard platform. This method is invoked to shut down the node and perform any necessary cleanup tasks, such as closing connections, releasing file handles, or stopping background processes.

The destroy() method ensures that the TbFtpNode instance is properly terminated and all associated resources are freed, preventing any potential memory leaks or resource conflicts. It is an essential part of the node lifecycle and ensures the stability and efficiency of the system by correctly handling the termination of nodes when they are no longer needed.

**Implemented Methods**

These methods provide essential functionalities for scheduling FTP uploads, managing the FTP connection, converting data to CSV format, and performing necessary file operations:

**scheduleUploadFtp**

This method is responsible for scheduling the FTP upload task based on the configured schedule. It calculates the initial delay until the scheduled time, creates a scheduled executor service, and schedules the task to run periodically at the specified interval. The task retrieves device data, using the convertDataToCsv method to convert it to CSV format, and then uploads the CSV file to the FTP server.

**cancelScheduleUploadFtp**

This method cancels the scheduled FTP upload task and shuts down the scheduler. It cancels the scheduled future if it exists and terminates the scheduler, ensuring that any running or pending tasks are completed before shutdown.

**uploadFtp() throws FileNotFoundException**

This method performs the FTP upload of the generated CSV files. It establishes a connection to the FTP server, sets the necessary parameters, iterates through the files in the specified folder, and uploads each file to the configured FTP directory. If the specified folder is not found, a FileNotFoundException is thrown.

**testFtpConnection**

This method tests the FTP connection by attempting to establish a connection with the FTP server using the provided URL, username, password, and port. It returns a boolean value indicating the success or failure of the connection.

**convertDataToCSV**

This method converts the retrieved time-series data into a CSV format for further processing or analysis. It takes the result of the time-series query, the list of timeseries keys, and the device associated with the data. It creates the necessary folder structure, opens or appends to a CSV file, writes the headers if the file is new, processes the time-series data, and writes the data rows to the CSV file in a sorted manner.

# Critical Parts

## Message Processing

The **onMsg** method processes incoming messages in the FTP node. It checks the type of the message and performs specific actions if the message type is "TB\_MSG\_TEST".

First, it retrieves the necessary parameters from the message metadata, such as the FTP server URL, username, password, port, schedule method. These parameters are used to configure the FTP node if they are different from the current configuration.

If the FTP node needs to be reconfigured, the method updates the configuration with the new parameters and proceeds to test the FTP connection using the testFtpConnection method. If the connection is successful, it retrieves the device ID from the message metadata.

If the device ID is null or empty, it retrieves a list of devices associated with the current tenant and customer and populates the deviceList with the retrieved devices. Otherwise, it retrieves the device with the specified device ID and sets it as the only device in the deviceList.

The method then sets the customer ID from the message metadata and cancels any previously scheduled FTP upload tasks using the cancelScheduleUploadFtp method. Finally, it schedules the FTP upload task based on the configured schedule using the scheduleUploadFtp method.

Overall, the onMsg method handles the processing of test messages, configuration updates, FTP connection testing, device retrieval, and scheduling of FTP upload tasks based on the incoming messages.

## Scheduling Events

The **scheduleUploadFtp** method is responsible for scheduling events to fetch timeseries data from the database and try to upload the created files to the FTP server based on the configured schedule.

First, the method retrieves the current date and time in the system's default time zone using ZonedDateTime.now(ZoneId.systemDefault()). It also retrieves the desired scheduling method from the configuration.

Next, it constructs the desired time for the event based on the configured schedule hour and minute using LocalTime.of(config.getScheduleHour(), config.getScheduleMinute()). The current date and the desired time are combined to create a ZonedDateTime representing the scheduled time.

If the scheduled time is already in the past, it adjusts the scheduled time to the next day, week, or month based on the selected scheduling method (DAILY, WEEKLY, or MONTHLY, respectively).

The initial delay until the scheduled time is calculated using now.until(scheduledTime, ChronoUnit.MILLIS). This delay determines the time between the current time and the scheduled time.

The scheduleUploadFtp method creates a ScheduledExecutorService with a thread pool size of 1. This means that **only one** task can be scheduled and executed at a time. This design choice ensures that the execution of scheduled events is serialized and avoids potential conflicts or race conditions that could arise from concurrent execution.

Additionally, it's important to note that this implementation assumes that each customer can have only one scheduled event at a time. If a new scheduling request is received for the same customer, the existing scheduled event will be canceled and replaced with the new one.

The task to be executed at the scheduled time is defined as a Runnable and is assigned to the scheduledTask variable. This task involves fetching timeseries data from the database and trying to upload the created files to the FTP server.

Inside the task, it retrieves the necessary timestamps (nowTime, startTs, endTs) for fetching the timeseries data. It iterates over the devices in the deviceList and retrieves the timeseries data for each device using ctx.getTimeseriesService().findAll().

The retrieved timeseries data is then passed to the convertDataToCSV method along with the device information to convert it into CSV format.

After processing the timeseries data, the method calls the uploadFtp method to attempt uploading the created files to the FTP server.

Depending on the selected scheduling method (DAILY, WEEKLY, or MONTHLY), the scheduledTask is scheduled to run at fixed intervals using the scheduler.scheduleAtFixedRate method. The initial delay, interval, and time units are set accordingly.

Finally, the method prints a "Scheduled" message to indicate that the scheduling has been successfully set up.

In summary, the scheduleUploadFtp method schedules events to fetch timeseries data, convert it to CSV format, and try to upload the files to the FTP server based on the configured schedule.

# Console Logs (Testing phase v.0.0)

The console logs in the provided code serve as informative messages to track the execution and progress of the testing phase of the application. Here is a description of each console log statement:

"**Scheduling upload**": This log indicates that the upload process is being scheduled.

"**Scheduled task**": This log signifies that the scheduled task is being executed.

"**nowTime =** ": This log displays the value of the current time in milliseconds.

"**startTs**": This log shows the value of the start timestamp for fetching time series data. It is calculated based on the current time and the scheduled time.

"**endTs**": This log presents the value of the end timestamp for fetching time series data. It represents the current time.

"**Scheduled**": This log indicates that the scheduling process has been completed successfully.

"**onMsg FTP node TB\_MSG\_TEST**": This log is displayed when the message type received type is TB\_MSG\_TEST.

"**msg =** ": This log prints the details of the received message.

"**url** ": This log shows the value of the url metadata field from the received message.

"**username**": This log displays the value of the username metadata field from the received message.

"**password**": This log indicates the value of the password metadata field from the received message.

"**port**": This log presents the value of the port metadata field from the received message.

"**scheduleMethod**": This log shows the value of the scheduleMethod metadata field from the received message.

"**scheduleHour**": This log displays the value of the scheduleHour metadata field from the received message.

"**scheduleMinute**": This log indicates the value of the scheduleMinute metadata field from the received message.

"**Configuring FTP node**": This log indicates that the FTP node is being configured based on the received message.

"**Ftp connection is successful**": This log signifies that the FTP connection is successful after configuration.

"**deviceId**": This log displays the value of the deviceId metadata field from the received message.

"**deviceId is empty**": This log indicates that the deviceId is empty.

"**deviceId is not empty**": This log indicates that the deviceId is not empty.

"**deviceList**": This log shows the list of devices being processed.

"**test schedule cancel**": This log indicates the start of the cancellation process for the schedule upload.

"**test schedule cancel end**": This log indicates the end of the cancellation process for the schedule upload.

These console logs provide valuable information for debugging, monitoring, and understanding the execution flow of the application on testing phase.