# Constrained Device Application (Connected Devices)

## Lab Module 02

Be sure to implement all the PIOT-CDA-\* issues (requirements) listed at [PIOT-INF-02-001 - Lab Module 02](https://github.com/orgs/programming-the-iot/projects/1#column-9974938).

### Description

NOTE: Include two full paragraphs describing your implementation approach by answering the questions listed below.

What does your implementation do?

The implementation creates a Python application named ConstrainedDeviceApp, designed to monitor system performance metrics, specifically CPU and memory utilization. It utilizes a structured approach by defining several classes within designated modules, promoting modularity and ease of maintenance. Key components include:

ConstrainedDeviceApp: This is the main application that integrates various system performance tasks.

SystemPerformanceManager: This class manages system performance tasks and invokes them at regular intervals.

BaseSystemUtilTask: This serves as a base class for system utilization tasks, providing common functionality.

SystemCpuUtilTask: A subclass of BaseSystemUtilTask, this task focuses on retrieving CPU utilization metrics.

SystemMemUtilTask: Another subclass of BaseSystemUtilTask, this task monitors memory utilization.

The SystemPerformanceManager is responsible for creating instances of the CPU and memory tasks and scheduling them using the APScheduler library.

How does your implementation work?

Application Structure:

The application is organized into a package structure under ./programmingtheiot/cda/app for the main application and ./programmingtheiot/cda/system for system management modules.

Integration of SystemPerformanceManager:

Within ConstrainedDeviceApp, an instance of SystemPerformanceManager is created. This instance is responsible for starting and stopping the monitoring of system metrics.

Task Management:

The SystemPerformanceManager holds instances of SystemCpuUtilTask and SystemMemUtilTask. It utilizes the APScheduler to run these tasks at defined intervals, allowing for continuous monitoring without blocking the main application flow.

Scheduled Tasks:

Each task (CPU and memory) retrieves the respective system metrics when triggered by the scheduler. The results can be logged or processed as needed.

Base Class for Common Functionality:

BaseSystemUtilTask provides shared methods and properties that both CPU and memory tasks can inherit, ensuring consistency and reducing code duplication.

Start/Stop Methods:

The ConstrainedDeviceApp controls the lifecycle of the SystemPerformanceManager, invoking its start method to begin monitoring and stop to terminate it gracefully.

Overall, the implementation provides a robust and scalable solution for monitoring system performance metrics in a constrained device environment, leveraging existing libraries and a clear modular architecture.

### Code Repository and Branch

NOTE: Be sure to include the branch (e.g. https://github.com/programming-the-iot/python-components/tree/alpha001).

URL: https://github.com/zo1235/connectedDevicesPy/tree/lab02

### UML Design Diagram(s)

A screenshot of a computer

Description automatically generated

### Unit Tests Executed

NOTE: TA's will execute your unit tests. You only need to list each test case below

(e.g. ConfigUtilTest, DataUtilTest, etc). Be sure to include all previous tests, too,

since you need to ensure you haven't introduced regressions.

- ConfigUtiltest

- SystemCpuUtiltask

- SystemMemUtiltask

### Integration Tests Executed

NOTE: TA's will execute most of your integration tests using their own environment, with

some exceptions (such as your cloud connectivity tests). In such cases, they'll review

your code to ensure it's correct. As for the tests you execute, you only need to list each

test case below (e.g. SensorSimAdapterManagerTest, DeviceDataManagerTest, etc.)

- ConstrainedDeviceApp

- SystemPerformanceManager

-

EOF.