PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Relatório de Projeto Final de Programação – SimpleEHF

Projeto Final de Programação – INF2102

Aluno: João Pedro Coutinho – 2212628
Orientador: Markus Endler

TURMA 3WA

Prof. Clarisse Sieckenius

1. Software Description and Purpose 4
2. Requirements Specification 5
2.1. Functional Requirements: 5
2.2. Non-functional Requirements: 5
3. Software Architecture Description / Model 6
3.1. Scheduler Module: 6
3.2. Energy Management Module: 6
3.3. Hardware Abstraction Module: 6
4. Software Functional Description 9
4.1. Energy management: 9
4.2. Task scheduling: 9
4.3. Hardware access: 9
5. User Manual 11
5.1. Getting Started 11
5.1.1. Installation 11
5.1.2. Framework Initialization 12
5.2. Task operations 12
5.2.1. Creating a Task 12
5.2.2. Executing Tasks 12
5.2.3. Removing Tasks 13
5.3. Sensor, Actuator, and Radio Integration 13
5.3.1. Sensor Integration 13
5.3.2. Actuator Integration 13
5.3.3. Radio Integration 13
5.4. Energy Manager 14
5.5. Best Practices 14
6. Scenaries 15
6.1. Examples of Scenarios in which Users Successfully Utilize the Software. 15
Scenario 1: 15
Scenario 2: 16
6.2. Examples of Scenarios in Which Users Encounter Difficulties Using the Software 18
Scenario 1: 18
Scenario 2: 19

Abstract:

The rapid growth of IoT devices necessitates the exploration of alternative energy sources, particularly intermittent options. In response, the Simple Energy Harvesting Framework (SimpleEHF) has been developed to streamline the creation of intermittent computing projects, catering to users with limited electronics knowledge. SimpleEHF offers user-friendly energy monitoring interfaces and a non-preemptive priority-based scheduler, ensuring efficient energy management and minimizing conflicts among tasks. Built with ease of use in mind, SimpleEHF is implemented in C++ and designed to be accessible to a wide range of users.

Additionally, SimpleEHF supports extensibility by seamlessly integrating commonly used hardware components found in energy harvesting devices. This flexibility allows researchers and developers to leverage existing technologies and further enhance the capabilities of their projects. By simplifying the development process, SimpleEHF expedites the creation of intermittent computing applications, empowering users to harness the potential of alternative energy sources.

Overall, SimpleEHF plays a vital role in improving energy efficiency in IoT applications. By enabling the effective utilization of intermittent energy sources, SimpleEHF contributes to the sustainable growth of the IoT ecosystem. Its user-friendly features and extensibility make it an invaluable tool for researchers, developers, students and hobbyists aiming to innovate in the field of energy-efficient IoT solutions.

1. Software Description and Purpose

The rapid proliferation of IoT devices in the coming years necessitates exploring alternative energy sources to power these wireless devices. While batteries are commonly used, they are not always practical for long-term operation. Harvesting energy from the surrounding environment, such as radiofrequency, light, noise, thermal, and motion, provides a promising solution. However, the intermittent nature of the collected energy poses challenges for ensuring continuous device functionality.

To address these challenges, we propose a framework for Arduino called Simple Energy Harvesting Framework (SimpleEHF). SimpleEHF is designed to simplify the development of intermittent computing projects with energy harvesting for researchers with limited electronics knowledge. It offers interfaces for monitoring energy levels in IoT devices and includes a scheduler for task execution. By abstracting hardware concerns, SimpleEHF enables faster development of intermittent computing applications in the field.

The framework is written in C++ and emphasizes ease of use. It incorporates a non-preemptive priority-based scheduler, ideal for energy harvesting applications that require task-based execution. This approach ensures that concurrent tasks, which may deplete available energy, are avoided. Instead, tasks are executed sequentially, enabling better control and energy management.

Furthermore, SimpleEHF is designed to be extensible, offering extension points for common hardware components found in energy harvesting devices. This allows researchers and developers to easily integrate their specific hardware into the framework, enhancing its versatility.

The expected lifespan of the SimpleEHF program is manly tied to the longevity of energy harvesting research in the GIST laboratory. As a foundational framework for intermittent computing applications, it aims to support long-term development and advancements of projects in the field.

Overall, SimpleEHF presents a promising solution for researchers, students, developers and hobbyists working on intermittent computing with energy harvesting. Its user-friendly nature, combined with a focus on energy management and extensibility, makes it a valuable tool for advancing the field of energy-efficient IoT applications.

2. Requirements Specification

2.1. Functional Requirements:

Id	Short Description	Long Description
FR1	Scheduler	The framework should allow the creation of a scheduler.
FR2	Create task	The framework should allow the creation of tasks.
FR3	Remove task	The framework should allow the deletion of tasks.
FR4	Priority	The framework should allow tasks to have execution priorities.
FR5	Priority order	The framework's scheduler should execute tasks in descending order of priority.
FR6	Single task	The framework's scheduler should execute only one task at a time.
FR7	Energy measurement	The framework should allow the measurement of stored energy in a device.

2.2. Non-functional Requirements:

Id	Short Description	Long Description
NFR1	C++ version	The framework should be implemented in C++ 14 or earlier versions
NFR2	Electronics Knowledge	Users of the framework should be able to enjoy its functionalities without requiring knowledge in electronics.
NFR3	Basic OOP	Users of the framework should be able to enjoy its functionalities with only basic knowledge in object-oriented programming.
NFR4	Arduino	The framework should be compatible with the Arduino prototyping environment.
NFR5	8-bit	The framework should be executable on 8-bit microcontrollers.

3. Software Architecture Description / Model

The Simple Energy Harvesting Framework (SimpleEHF) follows a modular architecture that allows for flexible and scalable development of projects involving intermittent computing with energy harvesting on the Arduino platform. The framework is organized into four modules, each responsible for specific functionality and interactions. The modular architecture of SimpleEHF promotes code reusability, maintainability, and extensibility. The following modules can be identified in the architecture of SimpleEHF:

3.1. Scheduler Module:

This module enables the integration of a scheduler SimpleEHF to the Arduino environment. It provides the necessary resources and services for the creation and execution of the scheduler. This module also handles the creation, deletion, and prioritization of tasks. It provides an interface for developers to define tasks and assign priorities to them. The module ensures efficient task scheduling and execution based on the availability of energy and priority.

3.2. Energy Management Module:

This module focuses on energy monitoring, optimization, and coordination of task execution based on energy availability. It provides functions for monitoring the energy level in the energy buffer and implements energy optimization strategies to maximize the device's operational time. The module communicates with the Scheduler Module to coordinate task execution based on the available energy.

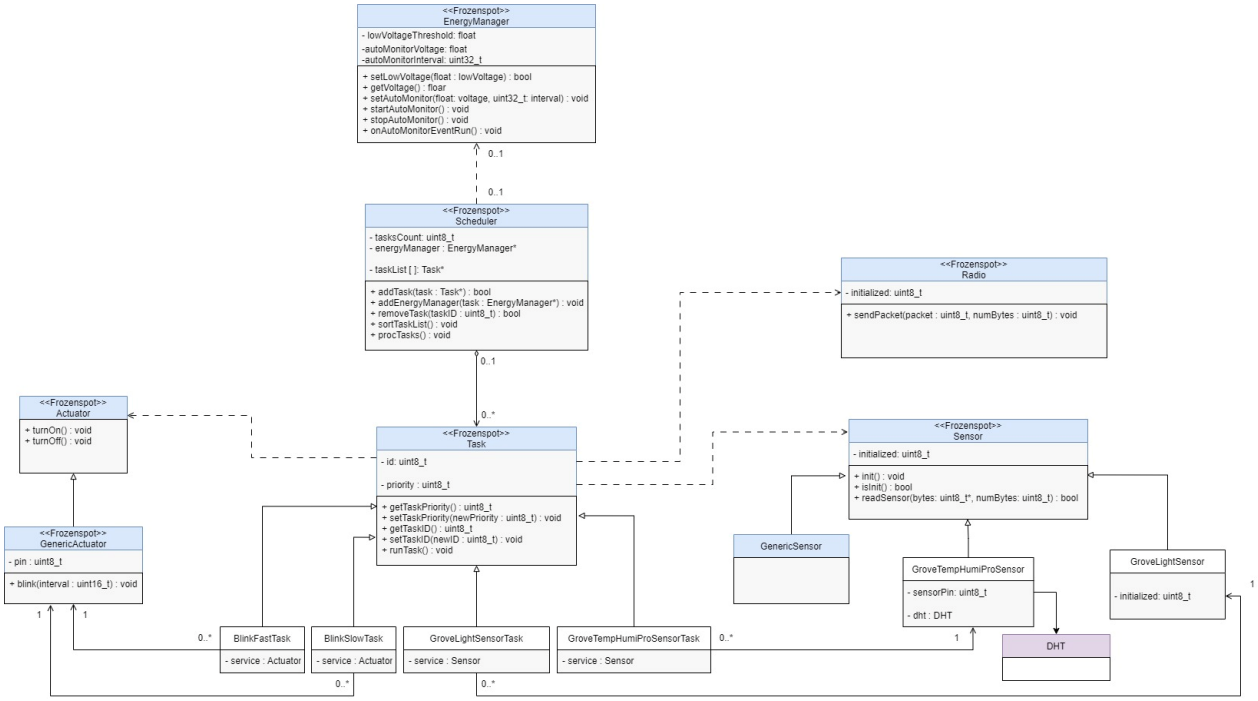
3.3. Hardware Abstraction Module:

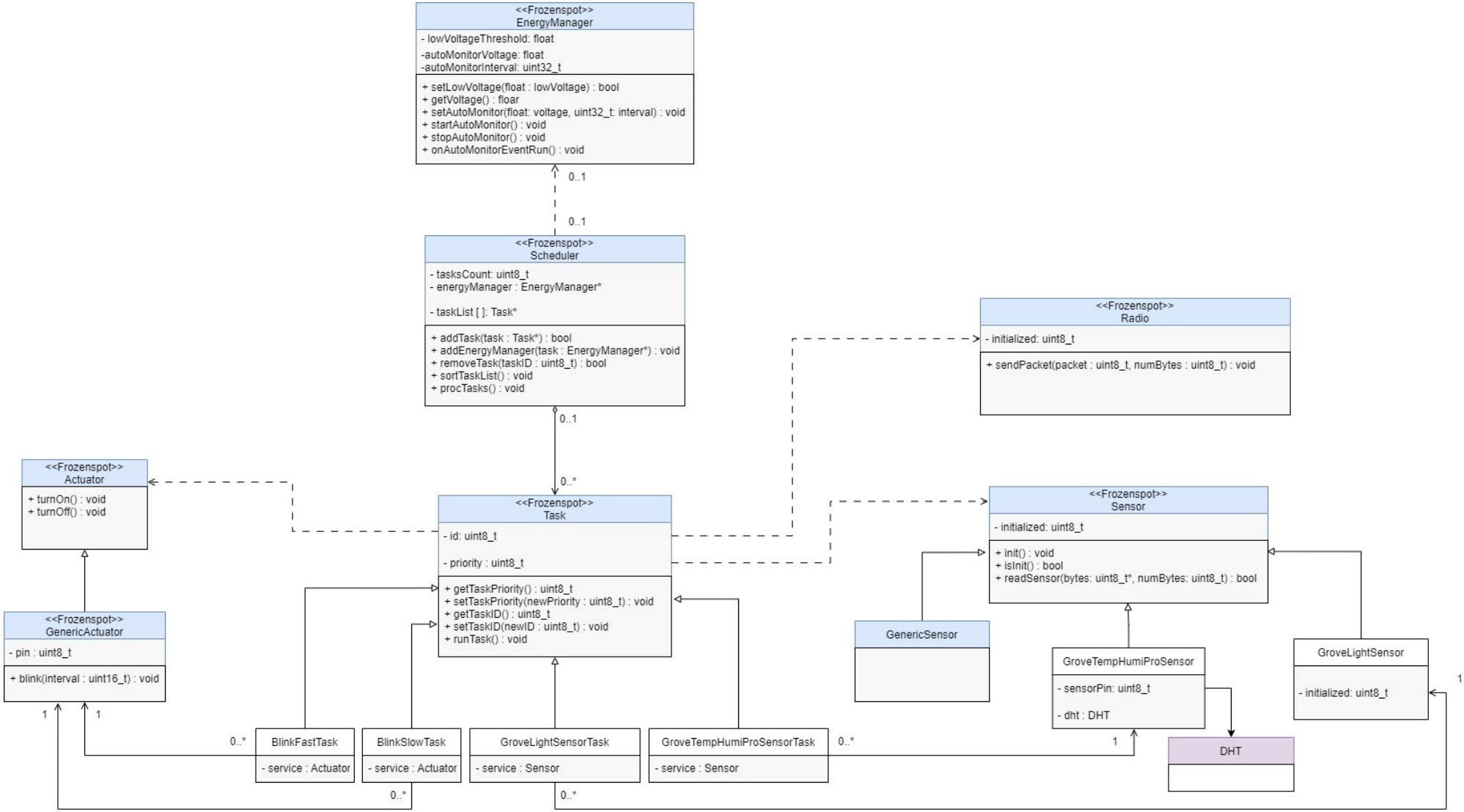
This module abstracts the underlying hardware components, such as sensors, radios, and actuators. It provides unified interfaces for communication with these hardware components, allowing for seamless integration with different configurations. The module enables developers to easily adapt SimpleEHF to various energy harvesting devices.

The modular architecture of SimpleEHF allows developers to select and include only the necessary components based on their specific requirements. It promotes modularity, as each module encapsulates a specific set of functionalities and can be developed, tested, and maintained independently. This architecture provides flexibility and adaptability, allowing

developers with limited electronics knowledge to focus on application logic. By leveraging the modular design of SimpleEHF, developers can customize the framework to meet their specific requirements, expanding its capabilities as needed.

The accompanying UML class diagram provides a valuable reference for understanding the structural composition of an application utilizing the SimpleEHF framework and the relationships between its constituent classes. It offers a clear overview of each component's responsibilities and interactions, aiding developers in utilizing and extending the framework effectively.




<<Frozenspot>>
EnergyManager
- lowVoltageThreshold: float
-autoMonitorVoltage: float
-autoMonitorInterval: uint32_t
+ setLowVoltage(float : lowVoltage) : bool
+ getVoltage() : floar
+ setAutoMonitor(float: voltage, uint32_t: interval) : void
+ startAutoMonitor() : void
+ stopAutoMonitor() : void
+ onAutoMonitorEventRun() : void
0..1
0..1
<<Frozenspot>>
Scheduler
- tasksCount: uint8_t
- energyManager : EnergyManager*
- taskList [ ]: Task*
+ addTask(task : Task*) : bool
+ addEnergyManager(task : EnergyManager*) : void
+ removeTask(taskID : uint8_t) : bool
+ sortTaskList() : void
+ procTasks() : void
0..1
0..*
<<Frozenspot>>
Radio
- initialized: uint8_t
+ sendPacket(packet : uint8_t, numBytes : uint8_t) : void
<<Frozenspot>>
Actuator
+ turnOn() : void
+ turnOff() : void
<<Frozenspot>>
Task
- id: uint8_t
- priority : uint8_t
+ getTaskPriority() : uint8_t
+ setTaskPriority(newPriority : uint8_t) : void
+ getTaskID() : uint8_t
+ setTaskID(newID : uint8_t) : void
+ runTask() : void
<<Frozenspot>>
Sensor
- initialized: uint8_t
+ init() : void
+ isInit() : bool
+ readSensor(bytes: uint8_t*, numBytes: uint8_t) : bool
<<Frozenspot>>
GenericActuator
- pin : uint8_t
+ blink(interval : uint16_t) : void
1
1
GenericSensor
GroveTempHumiProSensor
- sensorPin: uint8_t
- dht : DHT
GroveLightSensor
- initialized: uint8_t
1
DHT
0..*
BlinkFastTask
- service : Actuator
BlinkSlowTask
- service : Actuator
0..*
GroveLightSensorTask
- service : Sensor
0..*
GroveTempHumiProSensorTask
- service : Sensor
0..*
1
1

4. Software Functional Description

The Simple Energy Harvesting Framework (SimpleEHF) is a software framework designed to facilitate the development of intermittent computing projects with energy harvesting on the Arduino platform. It provides a set of essential functionalities for managing the collection, storage, and efficient use of energy from intermittent sources, enabling task execution based on energy availability.

The key features of SimpleEHF include:

4.1. Energy management:

- ***Energy Monitoring:*** The framework provides functionality to monitor the available energy level in the energy buffer of the IoT device. It allows users to track the energy status and make informed decisions based on the energy readings.

- ***Energy Optimization Strategies:*** SimpleEHF enable the implementation of various strategies to optimize energy usage by the user. These strategies can be used to maximize the device's operational time by managing energy resources and prioritizing energy-intensive tasks.

4.2. Task scheduling:

- ***Task Creation***: Users can create tasks within the framework. Tasks represent specific operations or functions that need to be executed by the IoT device.

- ***Task Prioritization***: SimpleEHF allows users to assign priorities to tasks. Tasks with higher priorities are given preference in execution when energy is available.

- ***Energy-Based Task Execution***: The framework coordinates the execution of tasks based on the availability of energy. Scheduled tasks only execute when sufficient energy is present, ensuring efficient energy utilization.

- ***Sequential Task Execution***: SimpleEHF ensures that only one task is executed at a time to avoid conflicts and resource contention. This approach guarantees the stability and reliability of task execution in an intermittent energy environment.

4.3. Hardware access:

- ***Component Interfaces***: SimpleEHF provides an interface to communicate with hardware components such as sensors, radios, and actuators. This enables seamless integration

of these components into the energy harvesting system.

- ***Hardware Compatibility:*** The framework is designed to be compatible with the Arduino prototyping environment, allowing easy integration with Arduino-compatible microcontrollers and supporting libraries.

By providing these functionalities, SimpleEHF simplifies the development of intermittent computing applications with energy harvesting capabilities. It empowers users with limited electronics knowledge to harness the potential of energy harvesting and create efficient and reliable IoT solutions. The modular and extensible nature of SimpleEHF allows for customization and integration of specific hardware components, making it a versatile framework for a wide range of energy harvesting devices and applications.

5. User Manual

The Simple Energy Harvesting Framework (SimpleEHF) is a software tool designed to simplify the development of intermittent computing projects with energy harvesting capabilities. This user manual provides detailed instructions on how to effectively use the framework to create applications on the Arduino platform.

5.1. Getting Started

To use SimpleEHF is actually quite easy. After downloading the framework from here, unpack the compressed folder to your favorite location to work from. Depending on the IDE you are using, the following steps may differ. We will only describe the general settings you need to achieve with Arduino IDE.

The Simple Energy Harvesting Framework (SimpleEHF) caters to a diverse range of users and use cases. Some potential users and use cases include:

- ***Researchers and students***: Conducting experiments and studies related to intermittent computing and energy harvesting.
- ***IoT Developers***: Building energy-efficient IoT applications that rely on intermittent power sources.
- ***Hobbyists and Tinkerers:*** Exploring innovative projects with limited electronics knowledge but a desire to harness energy efficiently.

5.1.1. Installation

To setup SimpleEHF for Arduino IDE, you need to create a new sketch in Arduino IDE and move the unzipped SimpleEHF folder to the sketch folder.

1. Start Arduino IDE, and create a new sketch ('File' -> 'New')
2. Save your sketch with a name of your choice ('File' -> 'Save as...')
3. Unpack the repository folder to the desired location
4. Copy or move the src folder in the root of the unpacked repo folder to the Arduino sketch folder
5. Done! You are ready to develop incredible applications!

In the example folder you can find an application ready to run on Grove kits. You will need to install the required third party libraries through Arduino IDE though. The example is a good source of information on how to utilize the framework for a real application.

5.1.2. Framework Initialization

Once SimpleEHF is installed, you should import the disired classes of the framework to your code and instantiate then. A common application consists of one instance of Scheduler, multiple instances of components, multiple instaces of tasks and one instance of Energy Manager.

To initialize the framework in your project, follow these steps:

1. Import the necessary classes from the SimpleEHF framework.
2. Create an instance of the Scheduler class to manage task scheduling.
3. Initialize the Sensor, Actuator, Radio, and EnergyManager classes as per your project requirements.

5.2. Task operations

5.2.1. Creating a Task

To create a task in the SimpleEHF framework, follow these steps:

1. Define a new class that extends the Task class.
2. Implement the required methods and logic for your specific task. You must write all your logic in the method *runTask()*
3. Instantiate the created class
4. Set the priority and dependencies of the task as needed through the method setPriority().

5.2.2. Executing Tasks

To execute tasks using the SimpleEHF framework, follow these steps:

1. Add the tasks to the Scheduler using the method addTask().
2. Initialize the scheduler by calling the method procTasks();
3. The Scheduler will execute the tasks based on their priorities and available energy.

5.2.3. Removing Tasks

To remove tasks from the scheduler at runtime using the SimpleEHF framework, follow these steps:

1. Remove the task of Scheduler using the method removeTask().
2. The Scheduler will not execute the task anymore

5.3. Sensor, Actuator, and Radio Integration

5.3.1. Sensor Integration

To integrate a sensor into your project, follow these steps:

1. Define a new class that extends the Sensor class or use the provided GenericSensor class.
2. Implement the necessary methods to read data from the sensor.
3. Use the sensor data within your tasks or for energy management purposes.

5.3.2. Actuator Integration

To integrate an actuator into your project, follow these steps:

1. Define a new class that extends the Actuator class or use the provided GenericActuator class.
2. Implement the required methods to control the actuator.
3. Use the actuator within your tasks or for energy management decisions.

5.3.3. Radio Integration

To integrate a radio module into your project, follow these steps:

1. Define a new class that extends the Radio class or use the provided GenericRadio class.
2. Implement the necessary methods to send and receive data using the radio module.
3. Utilize the radio module for communication purposes within your tasks.

5.4. Energy Manager

The SimpleEHF framework provides energy management capabilities to optimize energy usage in your project. Use the following steps to utilize this feature:

1. Create an instance of the EnergyManager class.
2. Set the low voltage threshold of the system by calling the method *SetLowVoltage()*.
3. Use the method *getLowVoltage()* to make informed decisions within your tasks.

You can also use the method setAutoMonitor() to configure a monitor that observes a user specified voltage periodically. When the voltage in the buffer equals the specified voltage, the auto monitor method calls the method onAutoMonitorEventRun(), which contains user logic to react to voltage events.

5.5. Best Practices

To maximize the effectiveness of the SimpleEHF framework, consider the following best practices:

1. Optimize your task scheduling to prioritize critical tasks and minimize energy consumption.
2. Regularly monitor and manage the energy resources to ensure efficient utilization.
3. Utilize the Sensor, Actuator, and Radio classes effectively to interact with external components.

By following this user manual and leveraging the various classes provided, you can create innovative applications that effectively utilize energy resources. Enjoy exploring the potential of the SimpleEHF framework and unlocking new possibilities in the realm of energy-efficient IoT applications.

Note: For detailed class references and method documentation, please refer to the SimpleEHF Framework API documentation.

6. Scenarios

6.1. Examples of Distinct Scenarios in which Users Successfully Utilize the Software.

These scenarios demonstrate the strengths of the Simple Energy Harvesting Framework (SimpleEHF) in facilitating research studies on intermittent computing and enabling IoT developers to create energy-efficient applications. The framework's modular architecture, task scheduling capabilities, and energy management features empower users with different levels of technical expertise to harness the benefits of energy harvesting technologies effectively.

Scenario 1: Researcher in Intermittent Computing Study

- **User Profile:** The user is a researcher with a strong technical background in computer science and expertise in the field of intermittent computing and energy harvesting.

- **Context:** The user is conducting a research study to explore the efficiency of energy harvesting techniques in intermittent computing. The study aims to analyze the impact of different scheduling algorithms on energy consumption and task execution in resource-constrained environments.

- **User Journey:**

Installation and Configuration*:*

The user downloads and installs the Simple Energy Harvesting Framework (SimpleEHF) on their research workstation.

They configure the framework by setting up the necessary hardware components, such as sensors and actuators, and defining the energy management strategies.

Experiment Setup:

The user designs a set of tasks with varying priorities and energy requirements using the Task class provided by SimpleEHF.

They integrate sensors to collect energy-related data and radio modules for communication purposes.

Execution and Data Collection:

The user utilizes the Scheduler class to schedule and execute tasks based on priority and available energy.

Throughout the experiment, the user monitors the energy consumption and task execution using the EnergyManager class.

Analysis and Evaluation:

The user evaluates the collected data to assess the effectiveness of different scheduling algorithms in optimizing energy usage.

They analyze the results, comparing task execution times, energy consumption, and overall system performance.

Utilization of Results:

The researcher utilizes the insights gained from the study to improve the design and implementation of intermittent computing systems.

The results contribute to academic publications, conferences, and further advancements in the field of intermittent computing.

Scenario 2: IoT Developer Creating Energy-Efficient Applications

- **User Profile**: The user is an IoT developer with basic knowledge of electronics and programming skills in object-oriented languages.

- **Context:** The user is developing an IoT application that relies on intermittent power sources, such as solar energy. The objective is to create an energy-efficient solution that minimizes power consumption and maximizes the utilization of available energy.

- **User Journey:**

Framework Installation:

The user downloads and installs the Simple Energy Harvesting Framework (SimpleEHF) from the repository.

They integrate the SimpleEHF framework into their preferred Integrated Development Environment (IDE) for ease of development.

Software Configuration:

The user configures the framework by setting up the required hardware components, including sensors, actuators, and radio modules, extending the respective classes provided by SimpleEHF.

Task Definition and Prioritization:

The user creates tasks using the Task class, defining their functionalities and energy requirements.

They assign priorities to the tasks based on their criticality and energy constraints.

Task Execution and Energy Management:

The user utilizes the Scheduler class to schedule and execute tasks based on their priorities and available energy.

They use the EnergyManager class to monitor and optimize energy usage, ensuring efficient utilization of the available power.

Application Deployment:

Once the development is complete, the user deploys the energy-efficient IoT application in real-world scenarios, such as smart home automation or environmental monitoring systems.

Benefits and Utilization:

The user observes significant energy savings and increased reliability in their IoT application due to the efficient utilization of intermittent power sources.

The application's performance and longevity are improved, contributing to reduced maintenance and operating costs.

6.2. Examples of Distinct Scenarios in Which Users Encounter Difficulties Using the Software

These scenarios demonstrate potential challenges that users may face when utilizing the Simple Energy Harvesting Framework (SimpleEHF) in different contexts. While novice developers may encounter difficulties in the initial installation and configuration process, they can overcome these challenges by referring to the user manual and seeking guidance from available resources. On the other hand, advanced researchers may face limitations related to hardware compatibility, requiring them to adapt their projects or explore alternative solutions.

Scenario 1: Novice Developer with Limited Electronics Knowledge

- **User Profile:** The user is a novice developer with limited knowledge of electronics and programming skills.

- **Context:** The user is in an educational environment, aiming to learn and develop their skills in programming and IoT applications. They want to utilize the Simple Energy Harvesting Framework (SimpleEHF) to create energy-efficient projects as part of their coursework or personal projects.

- **User Journey:**

  Initial Installation:

The user downloads and installs the SimpleEHF framework but encounters difficulties during the installation process due to unfamiliarity with the required dependencies and configuration steps.

Configuration Challenges:

The user struggles to set up the hardware components, such as sensors and actuators, as they lack experience in integrating external devices.

The user finds it challenging to configure the framework correctly, resulting in errors and malfunctions when attempting to execute tasks.

Troubleshooting and Recovery:

The user realizes the need to consult the user manual for guidance on hardware integration and software configuration.

They follow step-by-step instructions, pay attention to error messages, and seek support from the developer community or forums to address the configuration challenges.

Learning from Mistakes:

The user learns the importance of thorough reading and understanding of the user manual to avoid common pitfalls during the installation and configuration process.

They realize the significance of paying attention to error messages and seek additional resources, such as tutorials or video guides, to enhance their understanding of the framework's functionalities.

Scenario 2: Advanced Researcher with Specific Hardware Constraints

- **User Profile:** The user is an advanced researcher with a strong technical background and expertise in intermittent computing. They possess extensive knowledge of electronics and programming.

- **Context:** The user is conducting research in a specialized field that requires the integration of specific hardware components not fully supported by the Simple Energy Harvesting Framework (SimpleEHF).

- **User Journey:**

Initial Exploration:

The user initially identifies the SimpleEHF framework as a potential solution for their research project due to its energy management and scheduling capabilities.

However, they encounter limitations when attempting to integrate specific hardware components, such as specialized sensors or actuators, not fully supported by the framework.

Diagnostic Process:

The user thoroughly examines the documentation and user manual provided by SimpleEHF to determine if there are workarounds or alternative approaches for integrating their specific hardware.

They investigate online forums, developer communities, and other resources to gather insights and seek advice from experts in the field.

Limitations and Adaptations:

The user realizes that the SimpleEHF framework may have limitations in terms of hardware compatibility and extensibility, especially for niche or custom-built hardware components.

They adapt their research project by either modifying the hardware setup to align with the capabilities of SimpleEHF or exploring alternative frameworks that better suit their specific hardware requirements.

Learning and Feedback:

The user provides feedback to the SimpleEHF developer community, highlighting the specific hardware constraints they encountered and suggesting potential improvements or extensions to support a broader range of hardware components.

They actively engage in discussions and collaborations with other researchers in the field to share their experiences and collectively contribute to the advancement of intermittent computing frameworks.