Below is the updated documentation that includes comprehensive details for each device category, the system controllers, full API documentation, and many sample command examples. In the “Command Examples” section you’ll find a list of actual text prompts (explicit or implicit) along with the associated methods and the exact API call that would be made. This document is intended for internal use to help developers, QA engineers, and API integrators understand how to interact with the system.

Smart Home Simulation System Documentation

Version: 2.0 **Last Updated:** 2025-02-13

Table of Contents

---

1. Overview

The Smart Home Simulation System is built in Unity to simulate a full smart home environment. Every IoT device (lights, fans, ACs, geysers, fridges, washing machines, TVs, exhausts, and induction devices) is managed by its own controller script. Two central managers control the system:

- **IotManager:**
  Holds references (arrays and Inspector-editable settings) for all device types, maintains fast lookup dictionaries by unique device ID, and applies state changes locally in the Unity simulation.

- **NetworkManager:**
  Runs an asynchronous HTTP server inside Unity (using HttpListener with async/await) to expose a RESTful API so external clients (Python, Java, JavaScript, etc.) can query and control the devices remotely.

---

2. Device Categories & Methods

Each device controller (e.g., LightController, FanController, etc.) exposes methods to change its state and a `getStatus()` method that returns a JSON string of its current state.

1. Lights (Light.cs)

Methods/Features:

- `toggle(bool state)`
  — Turns the light on (`true`) or off (`false`).
- `setIntensity(float value)`
  — Sets brightness (range 0 to 2).
- `setHue(string hexValue)`
  — Changes the light color using a hexadecimal color code.
- `getStatus()`
  — Returns a JSON object with the current state.

Status Example:

```
{
  "category": "light",
  "id": "light001",
  "toggle": true,
  "intensity": 0.8,
  "color": "#FFA500"
}
```

2. Fan (Fan.cs)

Methods/Features:

- `toggle(bool state)`
  — Powers the fan on/off.
- `setRPM(int rpm)`
  — Sets the fan's speed in RPM.
- `getStatus()`
  — Returns a JSON status.

Status Example:

```
{
  "category": "fan",
  "id": "fan001",
  "toggle": true,
  "RPM": 400
}
```

3. Air Conditioner (AC.cs)

Methods/Features:

- `toggle(bool state)`
  — Switches the AC on/off.
- `setFanSpeed(int level)`
  — Sets internal fan speed (0: off, 1: low, 2: medium, 3: high).
- `setTemperature(int temp)`
  — Sets target temperature.
- `toggleEcoMode()`
  — Toggles eco-friendly mode.
- `getStatus()`
  — Returns current settings as JSON.

Status Example:

```
{
  "category": "ac",
  "id": "acHall",
  "toggle": true,
  "fanSpeed": 2,
  "temperature": 22,
  "eco": false
}
```

4. Geyser (Geyser.cs)

Methods/Features:

- `toggle(bool state)`
  — Turns the geyser on/off.
- `setTemp(int temp)`
  — Sets water temperature.
- `getStatus()`
  — Returns JSON status.

Status Example:

```
{
  "category": "geyser",
  "id": "geyserMasterBath",
  "toggle": true,
  "temp": 55
}
```

5. Fridge (Fridge.cs)

Methods/Features:

- `toggle(bool state)`
  — Powers the fridge on/off.
- `setTemp(int temperature)`
  — Sets the main compartment temperature.
- `setFreezeTemp(int temperature)`
  — Sets the freezer temperature.
- `getStatus()`
  — Returns JSON status including door statuses.

Status Example:

```
{
  "category": "fridge",
  "id": "fridgeKitchen",
  "toggle": true,
  "mainTemp": 4,
  "freezeTemp": -18,
  "mainDoorStatus": false,
  "freezeDoorStatus": true
}
```

6. Washing Machine (WashingMachine.cs)

Methods/Features:

- `setToggle(bool state)`
  — Turns the washing machine on/off.
- `startJob(string mode)`
  — Starts a wash cycle (e.g., "LIGHT", "MEDIUM", "HEAVY").
- `getStatus()`
  — Returns the current job details in JSON.

Status Example:

```
{
  "category": "washingmachine",
  "id": "washerHall",
  "toggle": true,
  "jobMode": "MEDIUM",
  "jobStartedTime": "2025-02-13T14:30:00Z",
  "remainingTime": "00:45:00"
}
```

7. Television (TV.cs)

Methods/Features:

- `setToggle(bool state)`
  — Turns the TV on/off.
- `setVolume(int volumeLevel)`
  — Adjusts the volume.
- `setChannel(int channelNumber)`
  — Changes the channel.
- `setSource(string source)`
  — Changes the input source.
- `getStatus()`
  — Returns the current TV settings as JSON.

Status Example:

```
{
  "category": "tv",
  "id": "tvHall",
  "toggle": true,
  "volume": 40,
  "channel": 97,
  "source": "HDMI1"
}
```

8. Exhaust (Exhaust.cs)

Methods/Features:

- `toggle(bool state)`
  — Activates or deactivates the exhaust.
- `setLevel(int level)`
  — Adjusts speed (1: low, 3: high).
- `getStatus()`
  — Returns JSON status.

Status Example:

```
{
  "category": "exhaust",
  "id": "exhaustKitchen",
  "toggle": true,
  "level": 2
}
```

9. Induction (Induction.cs)

Methods/Features:

- `setHeatLevel(int level)`
  — Sets the heat level (0–3).
- `getStatus()`
  — Returns the current induction status as JSON.

Status Example:

```
{
  "category": "induction",
  "id": "inductionKitchen",
  "heatLevel": 2
}
```

---

3. System Controllers

A. IotManager.cs

Description:

- Acts as a universal controller that holds references to all device types (lights, fans, ACs, etc.).
- Maintains Inspector-editable settings lists for each category.
- Builds internal dictionaries for fast device lookup based on unique IDs.
- Applies state changes from the Inspector to each device in real time.

Usage:

- Attach the IotManager component to a central GameObject in Unity.
- Populate the arrays and settings lists via the Inspector.
- Changes made in the Inspector (or via API commands) update the simulation immediately.

B. NetworkManager.cs

Description:

- Provides external API access by running an asynchronous HTTP server (using HttpListener with async/await).
- Processes API requests to query or control devices.
- Uses IotManager's fast dictionary lookups to locate devices by unique ID.
- Returns JSON responses indicating device status or command success.

Usage:

- Attach the NetworkManager component to a dedicated GameObject.
- Set the server port (default 8080) and assign the IotManager reference.
- On startup, the NetworkManager listens for HTTP requests so that external clients can interact with the simulation.

---

4. API Documentation & Usage

Base URL Structure:

http://<host>:<port>/{deviceType}/{deviceId}/{command}?param1=value1&param2=value2

Common API Endpoints:

- **Get Device Status:**

- **Example:**
  `GET http://localhost:8080/light/light001/getStatus`

Response:
```
{
  "category": "light",
  "id": "light001",
  "toggle": true,
  "intensity": 0.8,
  "color": "#FFA500"
}
```

- **Set Light Intensity:**
  - **Example:**
    `GET http://localhost:8080/light/light001/setIntensity?value=1.5`

Response:
{"status": "success"}

- **Toggle a Device (e.g., TV):**
  - **Example:**
    `GET http://localhost:8080/tv/tvHall/toggle?state=true`

Response:
{"status": "success"}

- **Set AC Temperature:**
  - **Example:**
    `GET http://localhost:8080/ac/acRoom1/setTemperature?value=20`

Response:
{"status": "success"}

- **Other Commands:**
  Similar endpoints exist for fans (`/fan/{id}/setRPM`), geysers (`/geyser/{id}/setTemp`), fridges (`/fridge/{id}/setTemperature` or `/fridge/{id}/setFreezeTemp`), washing machines (`/washingmachine/{id}/startJob?mode=HEAVY`), exhausts

(`/exhaust/{id}/setLevel`), and induction devices (`/induction/{id}/setHeatLevel`).

Note:
Currently, all commands use the GET method for simplicity. In production, consider using POST/PUT for commands that modify state.

---

5. Command Examples (Text Prompt → Associated Method(s) → API Call)

Below are many sample commands in the requested format. Each entry shows an example text prompt (explicit or implicit), the associated device method(s) that would be invoked, and the corresponding API call.

A. Lighting and Device Management

- **Explicit:**
  Text Prompt: "Turn on the lights"
  Method: `LightController.Toggle(true)`
  API:
  `GET http://localhost:8080/light/light001/toggle?state=true`

- **Explicit:**
  Text Prompt: "Dim the living room lights to 40%"
  Method: `LightController.setIntensity(0.4)`
  API:
  `GET http://localhost:8080/light/light002/setIntensity?value=0.4`

- **Implicit:**
  Text Prompt: "It’s too bright in here."
  Method: `LightController.setIntensity(0.8)` (if already on)
  API:
  `GET http://localhost:8080/light/light003/setIntensity?value=0.8`

- **Implicit:**
  Text Prompt: "This room feels dull, change the light to a warm tone."
  Method: `LightController.setHue("FF4500")`
  API:
  `GET http://localhost:8080/light/light004/setHue?hex=FF4500`

B. Fan Controls

- **Explicit:**
  Text Prompt: "Turn on the fan."
  Method: `FanController.Toggle(true)`
  API:
  `GET http://localhost:8080/fan/fan001/toggle?state=true`

- **Explicit:**
  Text Prompt: "Set the fan speed to 400 RPM."
  Method: `FanController.setRPM(400)`
  API:
  `GET http://localhost:8080/fan/fan002/setRPM?value=400`

- **Implicit:**
  Text Prompt: "It's stuffy here, increase the fan speed."
  Method: `FanController.setRPM(500)`
  API:
  `GET http://localhost:8080/fan/fan003/setRPM?value=500`

C. Temperature Control (AC & Geyser)

- **Explicit (AC):**
  Text Prompt: "Turn on the AC and set the temperature to 21°C."
  Method:
  `ACController.Toggle(true)` and `ACController.setTemperature(21)`
  API:
  `GET http://localhost:8080/ac/acHall/toggle?state=true`
  `GET http://localhost:8080/ac/acHall/setTemperature?value=21`

- **Implicit (AC):**
  Text Prompt: "I'm feeling warm, can you cool it down?"
  Method: `ACController.setTemperature(18)`
  API:
  `GET http://localhost:8080/ac/acRoom1/setTemperature?value=18`

- **Explicit (Geyser):**
  Text Prompt: "Turn on the geyser and set the water temperature to 50°C."
  Method:
  `GeyserController.Toggle(true)` and `GeyserController.setTemp(50)`
  API:
  `GET`
  `http://localhost:8080/geyser/geyserCommonBath/toggle?state=true`

`GET`
`http://localhost:8080/geyser/geyserCommonBath/setTemp?value=50`

- **Implicit (Geyser):**
  Text Prompt: "I need hot water now."
  Method: `GeyserController.setTemp(55)`
  API:
  `GET`
  `http://localhost:8080/geyser/geyserMasterBath/setTemp?value=55`

D. Fridge and Washing Machine

- **Explicit (Fridge):**
  Text Prompt: "Turn on the fridge and set the temperature to 4°C."
  Method: `FridgeController.Toggle(true)` and
  `FridgeController.setTemperature(4)`
  API:
  `GET http://localhost:8080/fridge/fridgeKitchen/toggle?state=true`
  `GET`
  `http://localhost:8080/fridge/fridgeKitchen/setTemperature?value=4`

- **Implicit (Fridge):**
  Text Prompt: "The food is getting warm, lower the temperature."
  Method: `FridgeController.setTemperature(2)`
  API:
  `GET`
  `http://localhost:8080/fridge/fridgeKitchen/setTemperature?value=2`

- **Explicit (Washing Machine):**
  Text Prompt: "Turn on the washing machine and start a medium cycle."
  Method:
  `WashingMachineController.ToggleWashingMachine(true)` and
  `WashingMachineController.startJob("MEDIUM")`
  API:
  `GET`
  `http://localhost:8080/washingmachine/washerHall/toggle?state=true`
  `GET`
  `http://localhost:8080/washingmachine/washerHall/startJob?mode=MED`
  `IUM`

- **Implicit (Washing Machine):**
  Text Prompt: "I need a quick wash."

Method: `WashingMachineController.startJob("LIGHT")`
API:
`GET`
`http://localhost:8080/washingmachine/washerHall/startJob?mode=LIGHT`

E. Entertainment (TV) Controls

- **Explicit:**
  Text Prompt: "Turn on the TV."
  Method: `TVController.ToggleTV(true)`
  API:
  `GET http://localhost:8080/tv/tvHall/toggle?state=true`

- **Explicit:**
  Text Prompt: "Set the channel to 97."
  Method: `TVController.setChannel(97)`
  API:
  `GET http://localhost:8080/tv/tvHall/setChannel?value=97`

- **Implicit:**
  Text Prompt: "I'm in the mood for a movie, switch to HDMI2."
  Method: `TVController.setSource("HDMI2")` (and ensure TV is on)
  API:
  `GET http://localhost:8080/tv/tvHall/setSource?value=HDMI2`

- **Implicit:**
  Text Prompt: "Silence the living room."
  Method: `TVController.setVolume(0)`
  API:
  `GET http://localhost:8080/tv/tvRoom/setVolume?value=0`

F. Exhaust & Induction

- **Explicit (Exhaust):**
  Text Prompt: "Turn on the exhaust."
  Method: `ExhaustController.ToggleExhaust(true)`
  API:
  `GET`

`http://localhost:8080/exhaust/exhaustKitchen/toggle?state=true`

- **Implicit (Exhaust):**
  Text Prompt: "It's really stuffy in here, crank up the exhaust."
  Method: `ExhaustController.setLevel(3)`
  API:
  `GET http://localhost:8080/exhaust/exhaustKitchen/setLevel?value=3`

- **Explicit (Induction):**
  Text Prompt: "Set the induction heat level to 2."
  Method: `InductionController.SetHeatLevel(2)`
  API:
  `GET http://localhost:8080/induction/inductionKitchen/setHeatLevel?value=2`

- **Implicit (Induction):**
  Text Prompt: "I need the induction on high."
  Method: `InductionController.SetHeatLevel(3)`
  API:
  `GET http://localhost:8080/induction/inductionKitchen/setHeatLevel?value=3`

---

6. Example: 2BHK Flat Setup & Workflow

Imagine a 2BHK flat configured in Unity with the following devices:

- **Lights:** 10 devices (IDs: `light001` to `light010`)
- **Fans:** 5 devices (IDs: `fan001` to `fan005`)
- **ACs:** 3 devices (e.g., `acHall`, `acRoom1`, `acRoom2`)
- **Geysers:** 2 devices (e.g., `geyserMasterBath`, `geyserCommonBath`)
- **Fridge:** 1 device (`fridgeKitchen`)
- **Washing Machine:** 1 device (`washerHall`)
- **TVs:** 2 devices (e.g., `tvHall` and `tvRoom`)
- **Exhaust:** 1 device (`exhaustKitchen`)
- **Induction:** 1 device (`inductionKitchen`)

Workflow Example:

1. **Local Simulation:**

   - In the Unity Editor, the IotManager's Inspector is used to set initial states (e.g., turning on lights in the living room, setting AC temperature, etc.).
2. **Remote API Request – Status Query:**
   - An external system sends:
     `GET http://localhost:8080/light/light003/getStatus`
   - The NetworkManager parses the request, locates `light003` via IotManager, and returns its JSON status.
3. **Remote API Command – Control a Device:**
   - To turn off a fan (say, `fan002` in the living room), the client sends:
     `GET http://localhost:8080/fan/fan002/toggle?state=false`
   - The NetworkManager routes this to the appropriate FanController and returns a success message. The simulation updates immediately.
4. **Context-Aware Control:**
   - A voice command like "I'm feeling warm, can you do something?" might be processed (by an external NLP layer) to issue:
     - `GET http://localhost:8080/ac/acRoom1/toggle?state=true`
     - `GET http://localhost:8080/ac/acRoom1/setTemperature?value=18`
   - The AC in the room turns on and adjusts to a cooler temperature.

---

7. Summary

This documentation provides a robust overview of the smart home simulation system built in Unity. Key points include:

- **Device Controllers:**
  Each IoT device is managed by its own controller script, exposing methods to toggle, adjust settings, and return status in JSON.

- **IotManager:**
  Acts as a central hub, maintaining arrays and dictionaries for quick device lookup and updating device states from the Unity Inspector.

- **NetworkManager:**
  Runs an asynchronous HTTP server to handle RESTful API calls, allowing external applications to query and control devices using standardized endpoints.

- **API & Command Examples:**
  A wide range of explicit and implicit command examples are provided in the format: `Text Prompt → Method(s) → API Call`. These examples show exactly how natural

language commands or explicit instructions map to API calls, ensuring consistency in remote control and automation.

This integrated solution allows both local (Inspector-based) testing and remote control via API calls, making the smart home simulation scalable, flexible, and ready for integration with external systems.

For further details, troubleshooting, or enhancements, please refer to the internal developer wiki and project issue tracker.

Happy coding!