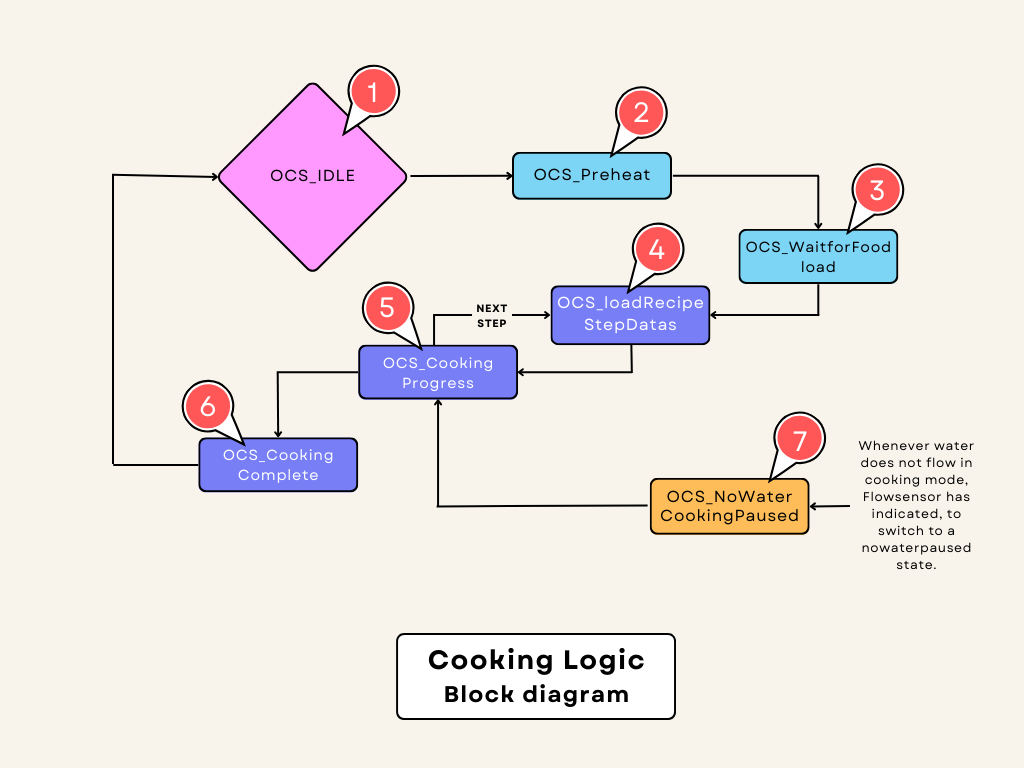
**Cooking Logic**

****

1. **Introduction**

This document provides documentation for the Oven Cooking Control software. The software is responsible for controlling the cooking process in an oven, including managing various cooking states, temperature control, and safety checks.

* This File Depends on The Recipe Config File and Cooking Config File.

1. **Header Inclusions**

* **Includes.h:** Header file that includes necessary system libraries and custom definitions.
* **COOKING\_H:** Custom header file for cooking-related functions.
* **DiagIDs.h**: Header file containing diagnostic error codes.
* **IOIF\_H:** Header file for input/output interface functions.
* **DEBUG\_H:** Header file for debugging functions.
* **DIAG\_H:** Header file for diagnostics functions.
* **FLOWSENSOR\_H:** Header file for flow sensor functions.

1. **Macro Definitions**

* **STEAM\_LEVEL\_RESOLUTION:** Defines the resolution for steam level control.
* **DRAIN\_VALVE\_ON\_TIME:** Defines the duration in seconds for the drain valve to be on (1 minute).
* **DRAIN\_VALVE\_OFF\_TIME:** Defines the duration in seconds for the drain valve to be off (5 minutes).

**RCR\_COUNTER\_TIMEOUT:** This Timeout value is used to report the error

1. **Global Variable Declarations**

**DvOpenFlag:** Boolean flag indicating the state of the drain valve (open/closed).And also handle the temp in multiple steps.

🡪This Flag is used to

The previous step Temperature is Higher than the Current step Temp, Continuously open\_state drain valve will go, after that, the Temp is equal to the Current step Temp, and the drain valve state Jumps to a Cyclic on-off state.

**CookingRuntimeCounter**: This variable is used to keep track of cooking runtime.

**RCR\_Counter**: This variable is represent by **RCR\_COUNTER\_TIMEOUT**.

**ovenCookingState**: Enumeration representing the current state of the oven cooking process.

1. **Cooking Main Function**

The cookingMain() function appears to be the core control logic for managing the cooking process in an oven. Let's break down the function step by step:

1. **Variable Declarations**:
   * Several pointers are declared and initialized to specific data structures, which are likely defined elsewhere in the code. These include **AndroidData**, **MachineStatus**, **recipeStepConfig**, and **DiagnosticsData**. These pointers are used to access and update various data related to the oven cooking process.
2. **Oven Light Control**:
   * The function checks the state of the **lightOn** flag in the **AndroidData** structure. If it's **TRUE**, it turns on the oven light using the **TURN\_ON\_LIGHT()** function; otherwise, it turns off the light with **TURN\_OFF\_LIGHT()**.
3. **Checking Oven State and AndroidData StartBit**:
   * It checks the current state of the oven (**ovenCookingState**) to determine if it's greater than the idle state (**OCS\_machineIdle**). If so, it checks whether the **startBit** in the **AndroidData** structure is **FALSE**. If the start bit is **FALSE**, it sets the **ovenCookingState** to **OCS\_CookingComplete**, indicating that the cooking process should be completed.
4. **Oven Pause Control**:
   * This section is entered if **ovenPauseControl** is **FALSE**, which means the oven is not paused or the door is not open.
   * The code enters a switch statement based on the **ovenCookingState** to handle different cooking process states.

**Cooking State Machine Explanation**

1. **OCS\_machineIdle**:
   * If the oven is in the idle state and the **startBit** is **TRUE**, it proceeds to check various conditions before starting the cooking process.
   * It checks if the "BOOT" bit (**MachineStatus->BOOT == FALSE**) which is used for ensure the all recipes base settings are received.
   * It sets the machine status to "preparingToCook" and sends this status.
   * It checks the recipe settings using **RecipeSettingsRxCbkCheck()**. If the check is successful (**TRUE**), it proceeds to check the cooking configuration using **CookingConfigCheck()**.
   * If the door is closed (**IsDoorClosed()**), it sets **MachineStatus->DCR** to **FALSE**, it will close the Door Open request pop-up notification in tablet.
   * If preheating is required (**rSconfig->preHeat == TRUE**), it sets the **ovenCookingState** to **OCS\_preHeat** and configures the cooking mode, as well as updates drain valve settings.
   * If preheating is not required, it sets the **ovenCookingState** to **OCS\_loadRecipeStepDatas**.
2. **OCS\_preHeat**:
   * In this state, it prepares the oven for cooking by turning on the heater (**TurnOnHeater()**).
   * It checks if the door open request (**MachineStatus->DOR**) is **TRUE**. If so, it opens the drain valve and waits for the door to be closed (**IsDoorOpen()**).
   * Once the door is closed, it transitions to **OCS\_waitForLoadTheFood**.
3. **OCS\_waitForLoadTheFood**:
   * In this state, it turns off the motor and heater.
   * It checks if the door is open (**IsDoorOpen()**). If so, it sets the machine status to "cookingPaused."
   * If the door is closed, it clears the open request flag and transitions to **OCS\_loadRecipeStepDatas**.
   * This state is represented by waiting for the load of the Food.
4. **OCS\_loadRecipeStepDatas**:
   * In this state, it prepares to load recipe data.

It checks if there are any recipe steps (**rSconfig->numberOfSteps <= 0**). If it is true number of steps is Zero. It is denoted the cooking step is not configured, and finally Report is sent to HMI.

* + If there are steps, it resets the cooking runtime counter and configures the cooking mode.
  + **if(currentStepIndex < rSconfig->numberOfSteps)**, This check is True cooking mode is Configure and DV Timeouts will Update.
  + It also opens the drain valve if the previous step's temperature is higher than the current step.
  + Finally, it transitions to **OCS\_CookingProgress**.

1. **OCS\_CookingProgress**:
   * In this state, it manages the cooking process.
   * It checks if the drain valve needs to be opened (**DvOpenFlag**) based on temperature conditions.
   * It monitors the cooking runtime and progresses to the next step when the runtime exceeds a certain limit.
   * If **enableMultiTrayMode** is **FALSE**, it continues the cooking process. Otherwise, it progresses to the next step.
   * The cooking runtime counter reaches once the runTimeIn100Millis, after that that cooking step is over and it is ready to move next step.
   * **recipeSteps[currentStepIndex].disable = TRUE,** This Flag indicates the current step cooking time is over and moves the next step of cooking Configure.
   * **if(currentStepIndex < rSconfig->numberOfSteps),** This check is Denoted to a number of steps that is lesser than the currentStepIndex, The cooking process is started again, if it is false the Cooking process moves on the cooking complete state.
2. **OCS\_CookingComplete**:
   * In this state, it completes the cooking process.
   * It resets various global variables and turns off all actuators.
   * It checks if the **startBit** is **FALSE** to transition back to **OCS\_machineIdle**.
3. **OCS\_NoWaterCookingPaused**:
   * In this state, it handles cooking pause due to lack of water.
   * It turns off the motor, closes the drain valve, and turns off the heater.
   * If water is available (**MachineStatus->NWF == FALSE**), it resumes the cooking process and transitions to **OCS\_CookingProgress**.
4. **Default Case**:
   * This is an empty default case that can be used for additional error handling.

Overall, the **cookingMain** function orchestrates the entire cooking process in an oven, managing different states, safety checks, and the execution of various actions such as heating, motor control, and door management. The specific behaviour of each state and function call depends on the configuration and requirements of the oven control system.