Rotate

Initialization

validateArguments(args)

Checks if required arguments are present and valid.

Epoch Parsing

parseEpoch(epochStr)

Parses the `epoch` argument and initializes the epoch time.

ChangeAt Rotation Setup

parseChangeAtTime(chageAtArr)

Parses and sorts `changeAt` times, divides them into Erinn and real-time entries.

registerTimeEvents()

Registers callbacks for Erinn and real-time changes.

calculateInitialRotationChangeAt()

Calculates the initial rotation based on the sorted `changeAt` times.

ChangeEvery Rotation Setup

calculateInitialRotationChangeEvery()

Calculates the initial rotation and schedules the first update based on `changeEvery`.

Value Retrieval

getCurrentValue(inx)

Returns the current value from the list based on the rotation.

Update Rotation

updateRotation(e)

Updates the rotation and display based on the current time and event triggers.

Query

Initialization

validateArguments(args)

Checks if required arguments (`id`, `entry`) are present and valid. It ensures there's exactly one `id` and handles the structure of `entry`.

Timer Selection and Configuration

selectTimer(id)

Selects the timer element based on the provided `id`.

configureDisplaySettings(args)

Sets up the display format for current and future values, with defaults for countdown display.

Query Setup

parseQueryArgs(args)

Extracts and prepares the query parameters (`lookFor`, `callIt`) from `args`.

Update Function

updateDisplay()

Updates the display with current information. This function is called to refresh the UI.

Value Retrieval

determineValue(idx)

Implements the logic to determine whether the index points to a 'current' or 'future' value based on the initial query condition and the index's parity

Time Retrieval

queryIndex(needed)

Searches through the timer values to find the index where the query condition is met the required number of times.

calculateTime(idx)

Determines the time associated with a given index, taking into account whether the index is seeking a 'current' or 'future' state and adjusting for any continuous sequences of the query condition being met.

Compress

Initialization

validateArguments(args)

ensure that `args` contains all necessary information and that this information is correctly formatted. It's particularly important to check the `subtype` field, as the compression function relies on another timer type to function.

Subtype Timer Initialization

initializeSubtypeTimer(args, list)

The compress function works by wrapping another timer type (specified in `args.subtype`). This step involves initializing this underlying timer and storing it for future queries.

QuerySetup

query(idx)

navigates through the underlying timer, skipping over consecutive identical values to find the index corresponding to the compressed timer's index. This function needs to be carefully designed to avoid infinite loops, particularly in cases where the timer might have long sequences of identical values.

Value Retrieval

getValueAtIndex(idx)

Given an index in the compressed timer, this function finds the corresponding value in the underlying timer. This involves using the `query` function to find the right index in the underlying timer and then retrieving the value at that index.

Time Retrieval

getValueAtIndex(idx)

Similar to `getValueAtIndex`, but instead of retrieving a value, this function retrieves the time associated with a given index in the compressed timer.

Update Function

updateDisplay()

Updates the rotation and display based on the current time and event triggers.

Select

Initialization

validateArguments(args)

ensure that `args` and `list` contain the necessary information for timer configuration, such as ensuring `list` items have the correct format and that `args` contains any required settings for the timer.

Timer Selection and Configuration

selectTimer(id)

initializes internal variables like `erinnEntries` and `realEntries` to manage Erinn and real-world times, acting as a form of timer configuration based on the provided `list`.

configureDisplaySettings(args)

implicitly configures display settings by parsing `list` entries and sorting them into Erinn and real-world categories, setting the groundwork for how times will be displayed and updated.

QuerySetup

parseQueryArgs

parses the `list` to separate and sort Erinn and real-world times, which is a form of query setup. It prepares the data for future queries on whether a given time is in Erinn or the real world, and what the current time label should be.

Time Retrieval

queryIndex(needed)

This is implicitly handled within the `query` function, which determines the correct index for Erinn or real-world times based on the current state and the passed index (`idx`).

calculateTime(idx)

This is implicitly handled within the `query` function, which determines the correct index for Erinn or real-world times based on the current state and the passed index (`idx`).

Update Function

updateDisplay()

Updates the rotation and display based on the current time and event triggers.