```
/**
 * Turf Stress Monitor SmartApp
 *
 * Monitors sensor data to determine turf stress tiers (Dormant, Normal, Moderate, High, Severe)
 * and updates a child device with the recommended mow height, blade height, and irrigation amount.
 */

definition(
    name: "Turf Stress Monitor",
    namespace: "yourNamespace",
    author: "Your Name",
    description: "Monitors turf stress tiers and adjusts settings based on sensor thresholds.",
    category: "Convenience",
    iconUrl: "https://s3.amazonaws.com/smartapp-icons/Convenience/Cat-Convenience.png",
    iconX2Url: "https://s3.amazonaws.com/smartapp-icons/Convenience/Cat-Convenience@2x.png"
)

preferences() {
    page(name: "mainPage", install: true, uninstall: true) {
        section("Sensor Devices") {
            input "dailyHighTempDevice", "capability.temperatureMeasurement", title: "Daily High-Temp Device", required: true, multiple: false, submitOnChange: true
            input "dailyHighTempAttribute", "enum", title: "Select Daily High-Temp Attribute", options: getDeviceAttributes(dailyHighTempDevice), required: true
            input "avgHighTempDevice", "capability.temperatureMeasurement", title: "7-Day Avg High Temp Device", required: false, multiple: false, submitOnChange: true
            input "avgHighTempAttribute", "enum", title: "Select 7-Day Avg High Temp Attribute", options: getDeviceAttributes(avgHighTempDevice), required: false
            input "dailyLowTempDevice", "capability.temperatureMeasurement", title: "Daily Low-Temp Device", required: true, multiple: false, submitOnChange: true
            input "dailyLowTempAttribute", "enum", title: "Select Daily Low-Temp Attribute", options: getDeviceAttributes(dailyLowTempDevice), required: true
            input "avgLowTempDevice", "capability.temperatureMeasurement", title: "7-Day Avg Low Temp Device", required: false, multiple: false, submitOnChange: true
            input "avgLowTempAttribute", "enum", title: "Select 7-Day Avg Low Temp Attribute", options: getDeviceAttributes(avgLowTempDevice), required: false
            input "soilTempDevice", "capability.temperatureMeasurement", title: "Soil Temp Device", required: true, multiple: false, submitOnChange: true
            input "soilTempAttribute", "enum", title: "Select Soil Temp Attribute", options: getDeviceAttributes(soilTempDevice), required: true
```

```
            input "uvDevice", "capability.illuminanceMeasurement", title: "UV Index Device",
required: true, multiple: false, submitOnChange: true
            input "uvAttribute", "enum", title: "Select UV Index Attribute", options:
getDeviceAttributes(uvDevice), required: true
            input "humidityDevice", "capability.relativeHumidityMeasurement", title: "Humidity
Device", required: false, multiple: false, submitOnChange: true
            input "humidityAttribute", "enum", title: "Select Humidity Attribute", options:
getDeviceAttributes(humidityDevice), required: false
        }
        section() {
            href "tierSettings", title: "Configure Tier Thresholds", description: "Tap to set thresholds
for Dormant, Normal, Moderate, High, and Severe tiers."
        }
    }
}

page(name: "tierSettings", title: "Tier Settings", install: false, uninstall: false) {
    section("Dormant Tier Thresholds (3+ matches)") {
        input "dormantHighTempMax", "number", title: "Max Daily High Temp for Dormant (°F)",
defaultValue: 70
        input "dormantAvgHighTempMax", "number", title: "Max 7-Day Avg High Temp for Dormant
(°F)", defaultValue: 65
        input "dormantLowTempMax", "number", title: "Max Daily Low Temp for Dormant (°F)",
defaultValue: 60
        input "dormantAvgLowTempMax", "number", title: "Max 7-Day Avg Low Temp for Dormant
(°F)", defaultValue: 50
        input "dormantSoilTempMax", "number", title: "Max Soil Temp for Dormant (°F)",
defaultValue: 60
        input "dormantUVIndexMax", "number", title: "Max UV Index for Dormant", defaultValue: 3
    }
    section("Normal Tier Thresholds (all must pass)") {
        input "healthyHighTempMax", "number", title: "Max Daily High Temp for Healthy (°F)",
defaultValue: 86
        input "healthyAvgHighTempMax", "number", title: "Max 7-Day Avg High Temp for Healthy
(°F)", defaultValue: 86
        input "healthyLowTempMax", "number", title: "Max Daily Low Temp for Healthy (°F)",
defaultValue: 70
        input "healthySoilTempMax", "number", title: "Max Soil Temp for Healthy (°F)",
defaultValue: 80
        input "healthyUVIndexMax", "number", title: "Max UV Index for Healthy", defaultValue: 6
        input "healthyHumidityMin", "number", title: "Min Humidity for Healthy (%)", defaultValue:
45
        input "healthyHumidityMax", "number", title: "Max Humidity for Healthy (%)", defaultValue:
65
```

```
    }
    section("Moderate Tier Thresholds (any may apply)") {
        input "moderateHighTempMin", "number", title: "Min Daily High Temp for Moderate (°F)",
defaultValue: 86
        input "moderateAvgHighTempMin", "number", title: "Min 7-Day Avg High Temp for
Moderate (°F)", defaultValue: 86
        input "moderateLowTempMin", "number", title: "Min Daily Low Temp for Moderate (°F)",
defaultValue: 71
        input "moderateSoilTempMin", "number", title: "Min Soil Temp for Moderate (°F)",
defaultValue: 81
        input "moderateUVIndexMin", "number", title: "Min UV Index for Moderate", defaultValue: 6
        input "moderateHumidityMin", "number", title: "Min Humidity for Moderate (%)",
defaultValue: 45
        input "moderateHumidityMax", "number", title: "Max Humidity for Moderate (%)",
defaultValue: 65
    }
    section("High Tier Thresholds (2+ match)") {
        input "highHighTempMin", "number", title: "Min Daily High Temp for High (°F)",
defaultValue: 90
        input "highAvgHighTempMin", "number", title: "Min 7-Day Avg High Temp for High (°F)",
defaultValue: 90
        input "highLowTempMin", "number", title: "Min Daily Low Temp for High (°F)", defaultValue:
74
        input "highSoilTempMin", "number", title: "Min Soil Temp for High (°F)", defaultValue: 85
        input "highUVIndexMin", "number", title: "Min UV Index for High", defaultValue: 8
        input "highHumidityMin", "number", title: "Min Humidity for High (%)", defaultValue: 35
        input "highHumidityMax", "number", title: "Max Humidity for High (%)", defaultValue: 75
    }
    section("Severe Tier Thresholds (3+ match)") {
        input "severeHighTempMin", "number", title: "Min Daily High Temp for Severe (°F)",
defaultValue: 95
        input "severeAvgHighTempMin", "number", title: "Min 7-Day Avg High Temp for Severe
(°F)", defaultValue: 92
        input "severeLowTempMin", "number", title: "Min Daily Low Temp for Severe (°F)",
defaultValue: 77
        input "severeSoilTempMin", "number", title: "Min Soil Temp for Severe (°F)", defaultValue:
90
        input "severeUVIndexMin", "number", title: "Min UV Index for Severe", defaultValue: 9
        input "severeHumidityMin", "number", title: "Min Humidity for Severe (%)", defaultValue: 25
        input "severeHumidityMax", "number", title: "Max Humidity for Severe (%)", defaultValue:
85
    }
}
```

```
/**
 * Function to get attributes of the selected device.
 * This function will dynamically retrieve all attributes of the device.
 */
def getDeviceAttributes(device) {
    def attributes = []
    if (device) {
        try {
            attributes = device.supportedAttributes.collect { it.name }
            log.debug "Supported attributes for ${device.displayName}: ${attributes}"
        } catch (Exception e) {
            log.error "Error retrieving attributes for device ${device.displayName}: ${e.message}"
        }
    }
    return attributes
}

/**
 * Called when the SmartApp is installed
 */
def installed() {
    log.debug "[installed] Turf Stress Monitor installed."
    initialize()
}

/**
 * Called when the SmartApp settings are updated
 */
def updated() {
    log.debug "[updated] Turf Stress Monitor updated."
    unsubscribe()
    initialize()
}

/**
 * Subscribe to sensors and create virtual device if it doesn't exist
 */
def initialize() {
    // Check if the child device exists
    def child = getChildDevice("turfStressMonitorChild")

    if (!child) {
        log.debug "[initialize] Creating new child device."
        try {
```

```
        def deviceName = "Turf Stress Monitor Child"
        def deviceDriver = "Turf Stress Monitor Driver"
        def namespace = "yourNamespace"

        if (!deviceName || !deviceDriver || !namespace) {
            log.error "[initialize] Invalid parameters: deviceName: ${deviceName}, deviceDriver:
${deviceDriver}, namespace: ${namespace}"
            throw new IllegalArgumentException("Device name, driver, or namespace cannot be
null or empty.")
        }

        addChildDevice(namespace, deviceDriver, "turfStressMonitorChild", location.hub.id, [
            "name": deviceName,
            "label": "Turf Stress Monitor",
            "completedSetup": true
        ])
        log.debug "[initialize] Child device created successfully."

      } catch (Exception e) {
        log.error "[initialize] Error creating child device: ${e.message}"
      }
   } else {
      log.debug "[initialize] Child device already exists."
   }

   // Subscribe to events for sensors
   subscribe(dailyHighTempDevice, dailyHighTempAttribute, sensorHandler)
   if (avgHighTempDevice) subscribe(avgHighTempDevice, avgHighTempAttribute,
sensorHandler)
   subscribe(dailyLowTempDevice, dailyLowTempAttribute, sensorHandler)
   if (avgLowTempDevice) subscribe(avgLowTempDevice, avgLowTempAttribute,
sensorHandler)
   subscribe(soilTempDevice, soilTempAttribute, sensorHandler)
   subscribe(uvDevice, uvAttribute, sensorHandler)
   if (humidityDevice) subscribe(humidityDevice, humidityAttribute, sensorHandler)

   runIn(5, calculateStress)

   // Schedule daily smoothing at midnight
   schedule('0 0 0 * * ?', finalizeDailyStress)
}

/**
 * Handler for all sensor events
```

```
 */
def sensorHandler(evt) {
    log.debug "[Sensor Event] ${evt.device.displayName} ${evt.name}: ${evt.value}"
    calculateStress()  // Call to calculate stress whenever a sensor event occurs
}

/**
 * Core stress calculation logic using user-defined thresholds
 */
def calculateStress() {
    log.debug "[CalculateStress] Running stress evaluation with detailed condition logging..."

    def dailyHighTemp = dailyHighTempDevice?.currentValue(dailyHighTempAttribute)?.toFloat()
    def dailyLowTemp = dailyLowTempDevice?.currentValue(dailyLowTempAttribute)?.toFloat()
    def avgHighTemp = avgHighTempDevice ?
avgHighTempDevice?.currentValue(avgHighTempAttribute)?.toFloat() : dailyHighTemp
    def avgLowTemp = avgLowTempDevice ?
avgLowTempDevice?.currentValue(avgLowTempAttribute)?.toFloat() : dailyLowTemp
    def soilTemp = soilTempDevice?.currentValue(soilTempAttribute)?.toFloat()
    def uvIndex = uvDevice?.currentValue(uvAttribute)?.toFloat()
    def humidity = humidityDevice?.currentValue(humidityAttribute)?.toFloat()

    def tierMap = ["Dormant": 1, "Normal": 2, "Moderate": 3, "High": 4, "Severe": 5]
    def reverseTierMap = tierMap.collectEntries { k, v -> [v, k] }

    def currentTier = "Normal"
    def mowHeight = 1.5
    def bladeHeight = 5
    def irrigationAmount = 0.25

    // === Dormant Tier ===
    int dormantMatches = 0
    if (dailyHighTemp < dormantHighTempMax) {
        dormantMatches++
        log.info "[Dormant Match] Daily High Temp ${dailyHighTemp} < ${dormantHighTempMax}"
    }
    // Add similar checks for other conditions...

    // === Tier Decision Logic ===
    if (severeMatches >= 3) {
        currentTier = "Severe"
        mowHeight = 2.5
        bladeHeight = 9
        irrigationAmount = 0.75
```

```
    } else if (highMatches >= 2) {
        currentTier = "High"
        mowHeight = 2.0
        bladeHeight = 7
        irrigationAmount = 0.65
    } else if (moderateMatches >= 1) {
        currentTier = "Moderate"
        mowHeight = 1.75
        bladeHeight = 6
        irrigationAmount = 0.50
    } else if (normalMatches >= 4) {
        currentTier = "Normal"
        mowHeight = 1.5
        bladeHeight = 5
        irrigationAmount = 0.25
    } else if (dormantMatches >= 3) {
        currentTier = "Dormant"
        mowHeight = 1.25
        bladeHeight = 4
        irrigationAmount = 0.0
    }

    log.debug "[CalculateStress] Final Tier Selected: ${currentTier}"

    // Use smoothed stress level for recommendations
    def smoothedTier = state.smoothedStressLevel ?: currentTier
    log.debug "[CalculateStress] Using smoothedTier for recommendations: ${smoothedTier}"

    // Update child device with recommendations based on smoothed tier
    def child = getChildDevice("turfStressMonitorChild")
    if (child) {
        child.updateStressData(smoothedTier, smoothedTier, mowHeight, bladeHeight,
irrigationAmount)
        log.debug "[CalculateStress] Child device updated successfully with tier data."
    } else {
        log.error "[CalculateStress] Child device not found. Unable to update."
    }

    // Save state + timestamped history
    state.currentStressTier = currentTier
    state.mowHeight = mowHeight
    state.bladeHeight = bladeHeight
    state.irrigationAmount = irrigationAmount
}
```