Common Tasks and FAQs

For the Carrier Open API

Version 1.2

March 7, 2025

Table of Contents

Note

- The following examples use JSON and JSONPath syntax. Doing the same with XML will be very similar.
- Unless otherwise stated, the following tasks are using the PUT `/systems/{serialNumber}/config` endpoint.
- Due to a few bugs (as of 2022-06-01) in PUT `/systems/{serialNumber}/config`, it is recommended that you perform a GET first, modify the JSON, and then PUT the entire document. The following examples are written assuming that you're modifying an existing JSON document instead of creating a new, minimal document. If you decide to create a minimal document instead, here are the API bugs you will face:
  - If `$.deadband` is not explicitly specified, it will be set to `null`. The workaround for this is GETting the deadband at/around registration (or just infrequently) since this value rarely changes and can only be changed by the installer/servicer.
  - If `$.zones[?(@.id=='1')]` is specified (for any specific zone ID), each `$.zones[?(@.id=='1')].program[..].periods` will be cleared. The workaround for this is GETting the program data just before PUTting changes. Unlike deadband, homeowners can change this at will.

Changing the Setpoints and/or Fan Speed

When running from the schedule or running a hold, the thermostat is always running an **activity**. Each activity is a collection of the setpoints and fan speed. There are four activities associated with the schedule (Home, Away, Sleep, and Wake) and one activity for provisional use (Manual).

Note that changing the Home, Away, Sleep, and Wake activities will impact the setpoints and/or fan speed used in the schedule. In other words, the schedule is just an instruction to “run activity X” at a particular time. When it’s time to start a new schedule period, the thermostat will “look up” the settings for the activity it’s about to start. So if you want to use a custom setpoint or fan speed, you will want to use the Manual activity.

See Setpoint and Temperature Limits for more information on setpoint limits and incrementation. See Fan Speed for more information on the fan speed definitions.

Setting an Activity Hold

The easiest way to change the setpoints and/or fan speed is just to set an activity hold. You only need to modify the hold configuration for one or more zones.

JSON Path	Description
`$.zones[?(@.id=='1')].hold.isEnabled`	`true` to immediately start a hold; otherwise, `false` to end a hold.
`$.zones[?(@.id=='1')].hold.activity`	The `id` of the activity to run: `Home`, `Away`, `Sleep`, or `Wake`.
`$.zones[?(@.id=='1')].hold.endTime`	The time, formatted as `HH:MM`, when the hold will end. See Notes on Hold End Time for some restrictions. If you omit `endTime` the hold will last indefinitely.
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

A common use case for this is when a homeowner wants to run the Home settings regardless of the schedule.

Setting a Manual Hold

If you want to change the current setpoints and/or fan speed to values other than those of the Home, Away, Sleep, or Wake activities, you need to modify the Manual activity. For each zone that you want to modify, update the Manual activity and hold configuration.

JSON Path	**Description**
`$.zones[?(@.id=='1')].activities[?(@.id=='Manual')].htsp`	The custom heat setpoint.
`$.zones[?(@.id=='1')].activities[?(@.id=='Manual')].clsp`	The custom cool setpoint.
`$.zones[?(@.id=='1')].activities[?(@.id=='Manual')].fan`	The custom fan speed.
`$.zones[?(@.id=='1')].hold.isEnabled`	`true` to immediately start a hold; otherwise, `false` to end a hold.
`$.zones[?(@.id=='1')].hold.activity`	The `id` of the activity to run: `Manual`.
`$.zones[?(@.id=='1')].hold.endTime`	The time, formatted as `HH:MM`, when the hold will end. See Notes on Hold End Time for some restrictions. If you omit `endTime` the hold will last indefinitely.
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

A common use case for this is when a homeowner just wants to “increase the heat/cool setpoint”. For example, if a homeowner wanted to increase the heat setpoint by three degrees, you could copy the settings from the current activity (see Viewing the Current Status) to the Manual activity, add three to the heat setpoint, and then set a Manual hold.

Ending an Ongoing Hold

If you want to end a finite hold before the end time, or you want to end an indefinite hold, you only need to disable the hold.

JSON Path	Description
`$.zones[?(@.id=='1')].hold.isEnabled`	`false` to end a hold.
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

Notes on Hold End Time

- `HH` can range from `00` to `23`.
- `MM` must be in increments of 15: `00`, `15`, `30`, or `45`.
- Since end time lacks a date component or time zone offset, this means that a finite hold cannot last longer than about 24 hours.
- The hold will end when the time displayed on the thermostat reaches the specified hold end time. See How do I determine the thermostat's time? for more information about determining the local time of the thermostat.
- By default, the thermostat sets the end time to the start of the next scheduled period. If you want to reproduce that behavior, see How do I determine when the current schedule period ends or the next one starts? for more information.

Changing the Temperature Units

The thermostat supports displaying temperatures in both Fahrenheit and Celsius. Unfortunately, changing between these units is more involved than just a simple boolean: you'll have to change all the temperatures in the configuration as well.

Changing the Units

First the easy part.

JSON Path	Description
`$.tempUnits`	`f` for Fahrenheit; `c` for Celsius.

Changing the Temperatures, Setpoints, Etc.

See Setpoint and Temperature Limits for more information on setpoint limits and incrementation. Here are example functions to use in converting temperatures (in TypeScript):

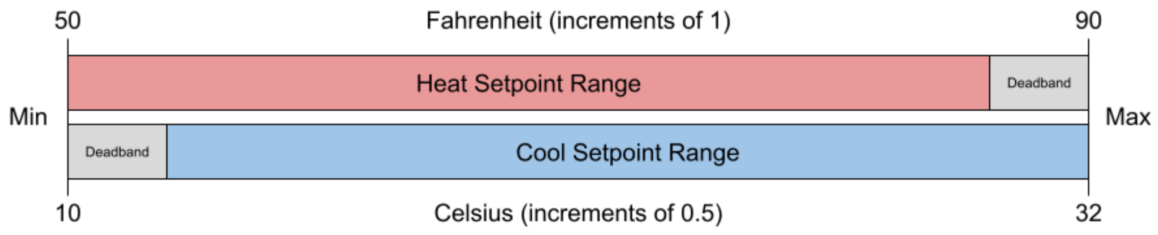
```
function convertFahrenheitToCelsiusSetpoint(f: number): number {
  // const float = (f - 32) * 5 / 9
  // const rounded = Math.round(float * 2) / 2
  // const bound = Math.max(10, Math.min(rounded, 32))
  // return bound

  return Math.max(10, Math.min(Math.round((f - 32) * 5 / 9 * 2) / 2, 32))
}

function convertCelsiusToFahrenheitSetpoint(c: number): number {
  // const float = (c * 9 / 5) + 32
  // const rounded = Math.round(float)
  // const bound = Math.max(50, Math.min(rounded, 90))
  // return bound

  return Math.max(50, Math.min(Math.round((c * 9 / 5) + 32), 90))
}

function convertFahrenheitToCelsiusOffset(fOffset: number): number {
  const cValues = [0, 0, 1, 1, 2, 3, 3, 4, 4, 5]

  return cValues[Math.max(0, Math.min(fOffset, 9))]
}

function convertCelsiusToFahrenheitOffset(cOffset: number): number {
  const fValues = [0, 2, 4, 5, 7, 9]

  return fValues[Math.max(0, Math.min(cOffset, 5))]
}
```

Here is a list of all the temperatures, setpoints, etc. that you'll need to modify.

JSON Path	Description
`$.vacation.htsp`	The heat setpoint to use during a vacation.
`$.vacation.clsp`	The cool setpoint to use during a vacation.

`$.utilityEvents.min`	The minimum setpoint allowed for all types of utility events.
`$.utilityEvents.max`	The maximum setpoint allowed for all types of utility events.
`$.utilityEvents.price.absHeatSetPoint`	The minimum setpoint allowed for price utility events.
`$.utilityEvents.price.absCoolSetPoint`	The maximum setpoint allowed for price utility events.
`$.utilityEvents.price.offset`	The maximum offset (subtracted from heat setpoint and added to cool setpoint) allowed for price utility events. Use the “offset” function, not the “setpoint” function.
`$.utilityEvents.demand.absHeatSetPoint`	The minimum setpoint allowed for demand utility events.
`$.utilityEvents.demand.absCoolSetPoint`	The maximum setpoint allowed for demand utility events.
`$.utilityEvents.demand.offset`	The maximum offset (subtracted from heat setpoint and added to cool setpoint) allowed for demand utility events. Use the “offset” function, not the “setpoint” function.
`$.zones.*.activities.*.htsp`	The heat setpoint for each zones' activities.
`$.zones.*.activities.*.clsp`	The cool setpoint for each zones' activities.

Setpoint and Temperature Limits



Deadband is the minimum difference between the heat and cool setpoints. For example, assume that deadband is `2`, heat setpoint is `70`, and cool setpoint is `74`. If the homeowner touches a down button for the cool setpoint four times, the resulting heat and cool setpoints should be at least `2` apart. One way to handle this would be changing the heat setpoint to `68` and the cool setpoint to `70`. Deadband is typically only changed by the installer/servicer, not the homeowner. It is **strongly recommended** that you leave deadband as is.

When the temperature units are in Fahrenheit:

- The minimum heat setpoint is `50`. The maximum heat setpoint is `90 - deadband`.
- The minimum cool setpoint is `50 + deadband`. The maximum cool setpoint is `90`.
- Setpoints and temperatures must be rounded to the nearest integer.

When the temperature units are in Celsius:

- The minimum heat setpoint is `10`. The maximum heat setpoint is `32 - deadband`.
- The minimum cool setpoint is `10 + deadband`. The maximum cool setpoint is `32`.
- Setpoints and temperatures must be rounded to the nearest half-integer.

Changing the Configuration Mode

The configuration mode represents the kinds of heating and/or cooling that the homeowner wants the thermostat to perform.

Changing the Mode

If you want to change the mode, you need to modify the following:

JSON Path	**Description**
`$.mode`	See Configuration Mode for all possible values.

Don’t Change the Type

Don’t confuse the thermostat’s mode with the type (`$.type`). For historical reasons, type is included in the configuration, but you shouldn’t change it. The thermostat will actually reject any change that includes a change to this value. However, the value for type may be useful to your application. See Configuration Type for all possible values.

Planning and Canceling a Vacation

When a vacation is ongoing, it overrides the schedule.

Planning a Vacation

Set the following values:

JSON Path	**Description**
`$.vacation.isPlanned`	`true` to schedule the vacation; otherwise, `false` to cancel the vacation.
`$.vacation.startDate`	The ISO-8601 date/time when the vacation should start (shifted to UTC). The time should be rounded to the nearest 15 minutes. Although the API will accept any start date, the thermostat will reject any date that isn’t in the future.
`$.vacation.endDate`	The ISO-8601 date/time when

	the vacation should end (shifted to UTC). The time should be rounded to the nearest 15 minutes. Although the API will accept any end date, the thermostat will reject any date that isn't at least 24 hours after the start date.
`$.vacation.htsp`	The minimum desired temperature during the vacation.
`$.vacation.clsp`	The maximum desired temperature during the vacation.
`$.vacation.fan`	The desired fan speed during the vacation.

Canceling a Vacation

Set the following value:

JSON Path	**Description**
`$.mode$.vacation.isPlanned`	`false` to cancel the vacation.

Viewing and Changing the Schedule

Each zone has its own list of activities (the group of setpoints and fan speed) and schedule. The schedule program contains a list of seven days, and each day contains a list of up to five periods. The periods specify which activity to run and when to start running it (rounded to the nearest 15 minutes).

See How do I determine when the current schedule period ends or the next one starts? for more information about how the schedule wraps to the next day or week.

Viewing the Schedule

The relevant parts of the config data to look at are:

JSON Path	Description
`$.zones[?(@.id=='1')].program`	The schedule, ordered by day, then period.
`$.zones[?(@.id=='1')].activities`	The list of activities that contain the setpoints and fan speed settings.
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

Changing the Schedule

When changing the schedule, you may add or remove period items from any of the `periods` arrays. However, each `periods` array must have at least one and no more than five items. You should NOT add or remove any of the days from the `program` array.

JSON Path	Description
`$.zones[?(@.id=='1')].program$.zones[?(@.id=='1')].program[?(@.id=='sunday')].periods[?(@.id=='2')].activity`	The name of the activity to run at a particular day/time. Think of this as a pointer to an item in the list of activities for the zone.
`$.zones[?(@.id=='1')].program[?(@.id=='sunday')].periods[?(@.id=='2')].time`	The time, formatted as `HH:MM`, (on a particular day) when the activity should start. This should be rounded to the nearest 15 minutes.
• `1` just represents the `id` for any zone. Substitute a different value as needed. • `sunday` just represents the `id` for any program day. Substitute a different value as needed. • `2` just represents the `id` for any day period. Substitute a different value as needed.	

As an example, if you wanted to add Wake and Sleep activities to a schedule that currently only has Home and Away, you could make the following change.

Before

```
{
  "zones": [
    {
      "id": "1",
      "program": [
        {
          "id": "monday",
          "periods": [
            {
              "id": "1",
              "activity": "Away",
              "time": "08:00"
            },
            {
              "id": "2",
              "activity": "Home",
              "time": "18:00"
            }
          ]
        }
      ]
    }
  ]
}
```

After

```
{
  "zones": [
    {
      "id": "1",
      "program": [
        {
          "id": "monday",
          "periods": [
            {
              "id": "1",
              "activity": "Wake",
              "time": "06:00"
            },
            {
              "id": "2",
              "activity": "Away",
              "time": "08:00"
            },
            {
              "id": "3",
              "activity": "Home",
              "time": "18:00"
            },
            {
              "id": "4",
              "activity": "Sleep",
              "time": "22:00"
            }
          ]
        }
      ]
    }
  ]
}
```

NOTE:

- Irrelevant properties were omitted for brevity.
- You would need to make this change to every zone and day that the homeowner wants to modify. This example only shows the change to zone 1’s Monday schedule.
- If you also wanted to change the setpoints and/or fan speed for one or more activities, see the next section (Changing the Setpoints and/or Fan Speed on the Schedule).

Changing the Setpoints and/or Fan Speed on the Schedule

To change the setpoints and/or fan speed used in the schedule or an activity hold, you need to modify the activities settings for a zone.

JSON Path	Description
`$.zones[?(@.id=='1')].activities[?(@.id=='Home')].htsp`	The heat setpoint for the specified activity.
`$.zones[?(@.id=='1')].activities[?(@.id=='Home')].clsp`	The cool setpoint for the specified activity.
`$.zones[?(@.id=='1')].activities[?(@.id=='Home')].fan`	The fan speed for the specified activity.

- `1` just represents the `id` for any zone. Substitute a different value as needed.
- `Home` just represents the `id` for any activity. Substitute a different value as needed.

Changing the Humidity and Fresh Air Comfort Profiles

The humidifier and/or ventilator, if present, can be controlled automatically or with manual settings. Within `GET /systems/{serialNumber}/config` if `$.humidity.auto.isEnabled` is `true`, then the manual settings will be stored but ignored.

Configuring the Automatic Humidity Settings

Set the following values:

JSON Path	Description
`$.humidity.auto.isEnabled`	`true` to use the automatic humidifier and/or ventilator settings; `false` to use the manual settings.
`$.humidity.auto.humTarget`	The level, from `1` to `9`, of humidity desired. `1` will produce less humid air than `9`.
`$.humidity.auto.useVentilator`	`true` to allow the thermostat to use the ventilator, if present; otherwise, `false`.

Configuring the Manual Humidity Settings

Looking at `$.humidity.manualConfigs.*.activity`, there are three activities defined: `home`, `away`, and `vacation`. Here is when each humidity profile is used:

When running this activity in the first zone...
 	
`GET /systems/{serialNumber}/status`
 	
`$.zones[?(@.id=='1')].currentActivity`
	...this humidity profile is used.
	
	`GET /systems/{serialNumber}/config`

`home`, `wake`, `sleep`, or `manual`	`$.humidity.manualConfigs[?(@.activity=='home')]`
`away`	`$.humidity.manualConfigs[?(@.activity=='away')]`
`vacation`	`$.humidity.manualConfigs[?(@.activity=='vacation')]`

Once you have identified the humidity profile(s) that you want to modify, update the following settings:

JSON Path	**Description**
`$.humidity.manualConfigs[?(@.activity=='home')].humTarget`	The level, from `1` to `9`, that the humidifier will target while heating. `1` will produce less humid air than `9`.
`$.humidity.manualConfigs[?(@.activity=='home')].dehumTarget`	The level, from `1` to `9`, that the humidifier will target while cooling. `1` will allow less humid air than `9`.
`$.humidity.manualConfigs[?(@.activity=='home')].overcoolEnabled`	`true` if the thermostat can overcool by up to 3°F to aid with reaching the humidity desired; otherwise, `false`. (NOTE: even if `true`, overcool will not function if the indoor temperature is below 70°F/21°C.)
`$.humidity.manualConfigs[?(@.activity=='home')].useHumidifier`	`true` to allow the thermostat to use the humidifier; otherwise, `false`.
`$.humidity.manualConfigs[?(@.activity=='home')].ventConfigs.*.mode`	The mode in which the ventilator will be operated. See Ventilator Mode for all possible values.
`$.humidity.manualConfigs[?(@.activity=='home')].ventConfigs.*.maxSpeed`	The maximum speed at which the ventilator will run. See Ventilator Max Speed for all possible values.

- `home` just represents the `activity` for any humidity profile. Substitute a different value as needed.

Viewing the Current Status

To view information about what the thermostat is currently doing, look at the `GET /systems/{serialNumber}/status` endpoint. Here are some of the more notable values:

JSON Path	Description
`$.timestamp`	The UTC time when the server last received a status update from the thermostat.
`$.isDisconnected`	`false` if the thermostat has polled the server within the last 15 minutes; otherwise, `true`.
`$.mode`	Describes the nature of the HVAC system's current activity. See Status Mode for all possible values.
`$.oat`	The outdoor air temperature, if an OAT sensor is connected; otherwise `null`. NOTE: Unlike all other setpoints/temperatures in the status data, this value will always be an integer and in Fahrenheit regardless of what temperature units the thermostat is using. If needed, convert the value to Celsius using a standard (non-bound) formula.
`$.levels.*.value`	The percentage **used** (not remaining) of each type of consumable accessory expressed as an integer percentage. See Consumable Accessory IDs for all possible values.

`$.zones[?(@.id=='1')].rt`	The current room temperature in the preferred temperature units.
`$.zones[?(@.id=='1')].rh`	The current room humidity as an integer percentage.
`$.zones[?(@.id=='1')].currentActivity`	The current activity for the zone. This is similar to the schedule activity except that it also accounts for activity holds and other events, such as occupancy sensor overrides. See Configuration Activity for all possible values.
`$.zones[?(@.id=='1')].htsp`	The heat setpoint associated with the current activity.
`$.zones[?(@.id=='1')].clsp`	The cool setpoint associated with the current activity.
`$.zones[?(@.id=='1')].fan`	The fan speed associated with the current activity. See Fan Speed for all possible values.
`$.zones[?(@.id=='1')].hold.isRunning`	`true` if the thermostat is running a hold; otherwise, `false`.
`$.zones[?(@.id=='1')].hold.endTime`	The time, formatted as `HH:MM`, when the hold will end. This time is in the thermostat's local time, not UTC.
`$.zones[?(@.id=='1')].conditioningStatus`	Describes the current and planned HVAC tasks in terms of moving air through the ducts. See Conditioning Status for all possible values.
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

FAQs

For how long are access tokens valid?

Access tokens expire after an hour. However, refresh tokens last forever. The best practice is to use the refresh token at the start of a session (however, you define a session), and then use the resulting access token during the session. If you expect sessions that last longer than an hour, you will need to use the refresh token again. The one hour expiry is non-negotiable.

My app is getting an access token, but not a refresh token.

In addition to the OAuth scopes used for the various API endpoints, your app also needs to request the `offline_access` scope.

How do I know which serial number(s) to use?

You can get a list of all thermostats owned by a homeowner by requesting `GET /user/locations`. At `$.locations.*.systems.*._links.system.href`, you can get the base URL to use for each thermostat's data. If you wish, you can extract the serial number from each URL.

What are zones/zoning?

Typically, zones represent different rooms, floors, or sections of a home. The exact way that a home is partitioned into zones may differ between installers. So the thermostats themselves just use the term “zone”. Each zone can be named, so the homeowner should be able to determine which zone they're targeting when making changes.

However, most homes don't support zoning because this requires additional hardware and installation. In these cases, the thermostat UI hides references to zones/zoning and substitutes “room”, “home”, or similar phrases where appropriate. However, in terms of the API and data model, this room/home data is still stored in the zone data, as the first and only zone.

To determine if zoning is supported by the HVAC system, reference the `GET /systems/{serialNumber}/config` data at `$.features[?(@.id=='zoning')].isEnabled`. (This value is read-only.) If this value is `true`, then zoning is supported and each `$.zones.*` represents a zone. If the value is `false`, then zoning is not supported and `$.zones[0]` or `$.zones[?(@.id=='1')]` represents the only "zone".

Note that certain values (such as configuration mode, ventilator/humidifier settings, etc.) are scoped to the entire HVAC system, and not to individual zones.

How do I determine the thermostat's time?

Throughout this document, you may have noticed that some times, such as hold end times and schedule times, are formatted as `HH:MM`. The best way to interpret such a time is as "when the thermostat UI shows the time as..." However, if your app requires a more precise definition of when these events occur, you will need to know what time zone the thermostat is in or approximately what time the thermostat is displaying.

Although the thermostat allows homeowners to sync the time with the server and choose a time zone if they wish, these settings are currently not sent to the API. However, looking at the `GET /systems/{serialNumber}/status` endpoint at `$.localTime`, we can get an approximation. This represents the time displayed on the thermostat when it sent its status data. (Note: due to the time required to build and send network data, you should assume `$.timestamp` and `$.localTime` will be a few seconds out of sync and perform your own corrections/rounding.)

Assuming no network latency and that status update was sent at May 1, 2022 6:08:06 pm EDT, here are the types of values you should expect for `$.localTime`:

Value	Description
`2022-05-01T18:08:06-04:00`	An ISO-8601 timestamp with a time offset represents a homeowner that has explicitly chosen a time zone on the thermostat UI. The time offset is by far the most useful piece of information here because network latency can cause local time and the timestamp to be slightly out of sync. You can probably use your app's current time and just shift to the time offset sent by the thermostat.
`2022-05-01T18:08:06`	An unqualified ISO-8601 timestamp represents a homeowner that has not chosen a time zone on the thermostat UI. However, the time still represents

	the date/time displayed on the thermostat. So you can calculate the difference between UTC now and this value (accounting for Daylight Savings, etc. as well) and then round/infer which time zone the thermostat likely is in.
`null`	The thermostat is using older firmware that doesn't send local time. We recommend that you encourage the homeowner to update their firmware. There should be a popup on the thermostat to do so.

How do I determine when the current schedule period ends or the next one starts?

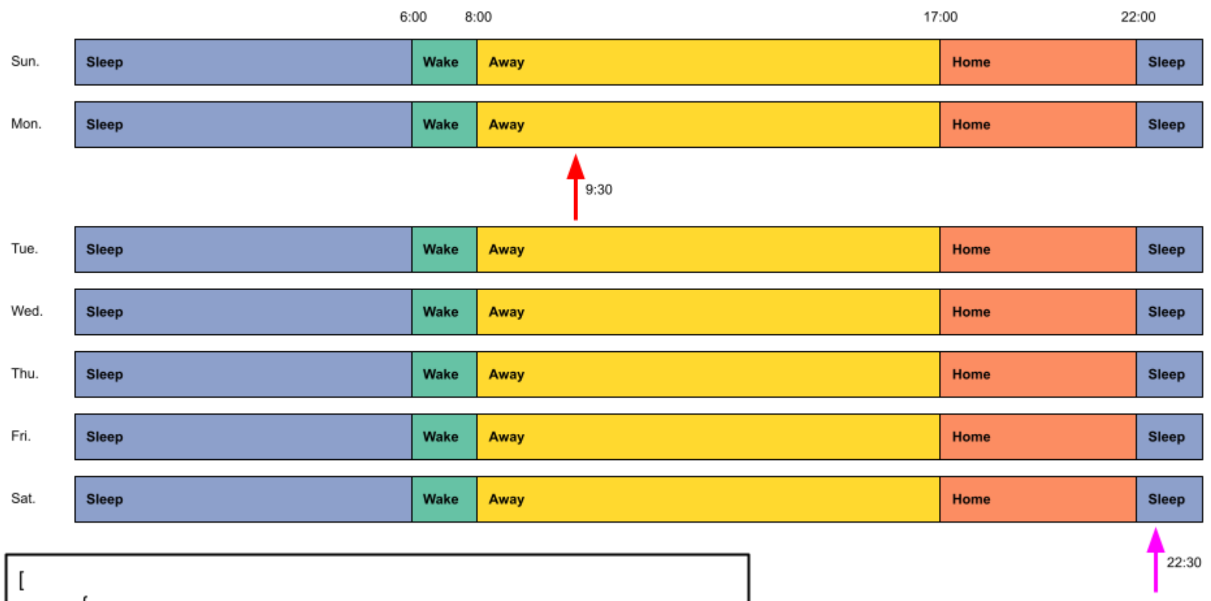
The relevant information to look at is:

Endpoint	**JSON Path**
`GET /systems/{serialNumber}/status`	`$.localTime`
`GET /systems/{serialNumber}/config`	`$.zones[?(@.id=='1')].program`
• `1` just represents the `id` for any zone. Substitute a different value as needed.	

Once you have determined where the thermostat's local time falls on the schedule/program, you can examine the following in this order:

1. the next period for the current day
2. the first period for the next day
3. the first period for Sunday

For example, assume the schedule for a zone looked like this:



```
[
        {
                "id": "monday",
                "periods": [
                        { "id": "1", "activity": "wake", "time": "06:00:00" },
                        { "id": "2", "activity": "away", "time": "08:00:00" },
                        { "id": "3", "activity": "home", "time": "17:00:00" },
                        { "id": "4", "activity": "sleep", "time": "22:00:00" }
                ]
        },
        { "id": "tuesday", "periods": [ … ] },
        …
]
```

If it is currently 09:30 on Monday (the red arrow), the next schedule period will start at 17:00 (for the Home activity). If it is currently 22:30 on Saturday (the magenta arrow), the next schedule period will start at 06:00 on Sunday (for the Wake activity).

Enumerations

For backwards compatibility, parse all enumeration values in a case-insensitive manner.

Configuration Type

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.type`. This value is read-only.

Value	**Description**
`Heat`	The HVAC system is only capable of producing hot air.
`Cool`	The HVAC system is only capable of producing cold air.
`HeatCool`	The HVAC system is capable of producing both hot and cold air.

Configuration Mode

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.mode`.

Value	**Description**
`Off`	The thermostat will not run the fan, nor produce hot or cold air.
`FanOnly`	The thermostat will only run the fan according to the schedule and holds. If the fan speed is `off`, the fan will not run at all.
`Heat`	The thermostat will only command the HVAC system to produce hot air to raise the room temperature.
`Cool`	The thermostat will only command the HVAC system to produce cold air to lower the room temperature.
`Auto`	The thermostat will command the HVAC system to produce either hot or cold air to raise or lower (respectively) the room temperature.

Note that the configuration modes allowed depends on the configuration type:

Configuration Type	Configuration Mode
`HeatCool`	`Off`
	`FanOnly`
	`Heat`
	`Cool`
	`Auto`
`Heat`	`Off`
	`FanOnly`
	`Heat`
`Cool`	`Off`
	`FanOnly`
	`Cool`

Temperature Units

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.tempUnits`. NOTE: Changing this value will not adjust the setpoints automatically. You will need to manually adjust those values. See Changing the Temperature Units for more details.

Value	Description
`f`	Fahrenheit
`c`	Celsius

Time Format

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.timeFormat`.

Value	Description
`12`	Time will be displayed using a 12-hour clock with an AM/PM indicator.

Heat Source

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.heatSource`.

Value	Description
`system`	The thermostat will choose which heat source to use for heating.
`iduOnly`	Only the indoor heat source will be used for heating.
`oduOnly`	Only the outdoor heat source will be used for heating.

Fan Speed

Instances of this enumeration appear in:

- `GET /systems/{serialNumber}` and `GET /systems/{serialNumber}/config`
    - `$.vacation.fan`
    - `$.zones.*.activities.*.fan`
- `GET /systems/{serialNumber}/status`
    - `$.zones.*.fan`

Value	Description
`off`	This turns off the continuous fan. The thermostat will turn the fan on only when it needs to push hot or cold air. This is shown as "auto" on the thermostat.
`low`	The fan will continuously run on low, even if no hot or cold air is being pushed.
`med`	The fan will continuously run on medium, even if no hot or cold air is being pushed.
`high`	The fan will continuously run on high, even if no hot or cold air is being pushed.

Accessory IDs

Instances of this enumeration appear in:

- `GET /systems/{serialNumber}` and `GET /systems/{serialNumber}/config`
    - `$.reminders.*.id`
    - `$.accessoryStatusReset` (child property names)
    - `$.components.*.id` (except `filter`)
- `GET /systems/{serialNumber}/status`
    - `$.components.*.id`

Value	Description
`filter`	The air filter that is attached to the HVAC system.
`humidifier`	An optional humidifier attached to the HVAC system.
`ventilator`	An optional ventilator attached to the HVAC system.
`uvLamp`	An optional ultra-violet filter attached to the HVAC system.

Energy Fuel Type

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.energy.fuelType`.

Value	Description
`gas`	If present, the furnace will use natural gas.
`propane`	If present, the furnace will use propane.

Energy Fuel Unit

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.energy.gasUnit`. The value allowed depends on the fuel type.

Fuel Type	**Fuel Unit (Value)**	**Description**
`gas`	`therm`	Fuel usage should be displayed in therms.
`gas`	`gJoule`	Fuel usage should be displayed in gigajoules
`gas`	`cubicMeter`	Fuel usage should be displayed in cubic meters.
`propane`	`gallon`	Fuel usage should be displayed in gallons.
`propane`	`liter`	Fuel usage should be displayed in liters.

Configuration Activity

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at:

- `$.wholeHouse.hold.activity`
- `$.wholeHouse.activities.*.id`
- `$.zones.*.activities.*.id`
- `$.zones.*.program.*.periods.*.activity`

Value	**Description**
`Home`	The homeowner is in the home during normal, active hours.
`Away`	The homeowner is not in the home, but will return later in the day.
`Wake`	The homeowner is in the home, and is waking up.
`Sleep`	The homeowner is in the home, and is sleeping.
`Manual`	The homeowner is in the home, but is using provisional settings.
`none`	For `$.wholeHouse.hold.activity` only. No hold activity.

Ventilator Mode

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.humidity.manualConfigs[?(@.activity=='home')].ventConfigs.*.mode`.

Value	Description
`auto`	Fresh air will be brought in automatically as needed.
`manual`	Fresh air will be brought in continuously.
`off`	The ventilator will not run.

Ventilator Max Speed

Instances of this enumeration appear in `GET /systems/{serialNumber}/config` at `$.humidity.manualConfigs[?(@.activity=='home')].ventConfigs.*.maxSpeed`.

Value	Description
`low`	Run the ventilator on low.
`low25`	Run the ventilator on low 25% of the time. This value is no longer supported, but remains for historical purposes. If you encounter it, interpret it as `low`.
`low50`	Run the ventilator on low 50% of the time. This value is no longer supported, but remains for historical purposes. If you encounter it, interpret it as `low`.
`low75`	Run the ventilator on low 75% of the time. This value is no longer supported, but remains for historical purposes. If you encounter it, interpret it as `low`.
`low100`	Run the ventilator on low 100% of the time. This value is no longer supported, but remains for historical purposes. If you encounter it, interpret it as `low`.
`med`	Run the ventilator on medium.
`high`	Run the ventilator on high.

Status Mode

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.mode`. This represents the current heating/cooling state, which is displayed in the lower-right corner on the thermostat's home screen.

Value	**Text Displayed in the Lower-Right Corner**
`Off`	*No text will be displayed.*
`Heat`	heating...
`Cool`	cooling...
`Dehumidify`	dehumidifying...
`HpHeat`	heat pump heating...
`HpElectHeat`	hp + electric heating...
`GasHeat`	gas heating...
`ElecHeat`	electric heating...
`Hydronic`	hydronic heating...
`HpGas`	hp + gas heating...
`HpHydronic`	hp + hydronic heating...

Current Activity

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.zones.*.currentActivity`.

Value	Description
`Home`	The homeowner is in the home during normal, active hours.
`Away`	The homeowner is not in the home, but will return later in the day.
`Wake`	The homeowner is in the home, and is waking up.
`Sleep`	The homeowner is in the home, and is sleeping.
`Manual`	The homeowner is in the home, but is using provisional settings.
`Vacation`	The homeowner is not in the home, and will not return soon. This happens when the homeowner plans a vacation.

Conditioning Status

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.zones.*.conditioningStatus`.

Value	Description
`idle`	The zone is not being conditioned and no conditioning is planned.
`active_heat`	The zone is currently being heated. Other zones waiting for cooling will have to wait for this session to terminate.
`active_cool`	The zone is currently being cooled. Other zones waiting for heating will have to wait for this session to terminate.
`prep_heat`	The zone needs heating, and the HVAC system is preparing the heat source to do so.
`prep_cool`	The zone needs cooling, and the HVAC system is preparing the cool source to do so.
`pending_heat`	The zone needs heating, and the HVAC system is waiting for an active cooling session to finish.
`pending_cool`	The zone needs cooling, and the HVAC system is waiting for an active heating session to finish.
`prep_pend_heat`	The HVAC system will begin heating the zone once the appropriate dampers have been opened/closed.
`prep_pend_cool`	The HVAC system will begin cooling the zone once the appropriate dampers have been opened/closed.
`fan`	The zone is currently being conditioned with air that is neither hot nor cold (fan-only mode). Other zones waiting for heating or cooling will have to wait for this session to terminate.

Consumable Accessory IDs

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.levels.*.id`.

Value	Description
`filter`	Identifies the air filter attached to the HVAC system, which is always present.
`humidifierPad`	Identifies the humidifier pad, if a humidifier is attached to the HVAC system.
`ventilatorPrefilter`	Identifies the ventilator prefilter, if a ventilator is attached to the HVAC system.
`uvLamp`	Identifies the ultra-violet lamp, if an ultra-violet filter is attached to the HVAC system.

Humidifier Status

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.components[?(@.id=='humidifier')].status`.

Value	Description
`off`	The humidifier is not currently running.
`on`	The humidifier is currently running.

Ventilator Status

Instances of this enumeration appear in `GET /systems/{serialNumber}/status` at `$.components[?(@.id=='humidifier')].status`.

Value	Description
`off`	The ventilator is not currently running.
`low`	The ventilator is running on low.
`high`	The ventilator is running on high.

`dehum`	The ventilator is dehumidifying.