Introduction

Behaviour and Twilight both have **FROM** and **TO** time between which they are valid.

They can be active on one or multiple days of the week. We only look at the FROM time to see if the behaviour is executed and will then always last until its TO time.

Behaviours do not overlap with other behaviours. Twilights do not overlap with other Twilights.

After leaving a room or sphere, there is a (currently hardcoded) timeout of 5 minutes before Crownstone turns off.

Behaviour Features:

- Presence aware
    - When there is **someone** in ROOM A, ROOM B or ROOM C
    - When there is **nobody** in ROOM A, ROOM B or ROOM C
    - Ignore presence (for timers etc.)
- Active times:
    - FROM
    - TO
    - Always (00:00 - 23:59:59)
- Time features:
    - Absolute time of day “15:32”
    - Sunrise +- 2 hours in 15 min increments
    - Sunset +- 2 hours in 15 min increments
- Action:
    - Turn ON
    - Turn ON but dimmed to X percent (10% increments)

Twilight features:

- Active times:
    - FROM
    - TO
    - Always (00:00:00 - 23:59:59)
- Time features:
    - Absolute time of day “15:32”
    - Sunrise +- 2 hours in 15 min increments
    - Sunset +- 2 hours in 15 min increments
- Action:
    - When turned on, dim to X percent instead (10% increments)

A Twilight will only ever REDUCE the light intensity.

Overrides and manual commands

If the behaviour says the light has to be ON, but the user switches it off with BLE, voice assistants or Switchcraft, the behaviour is in an OVERRIDE state.

You leave the room, and the 5 minute timeout has expired, the behaviour says the light has to be OFF. This matches the override state and the behaviour continues normally from then on, the behaviour is no longer in override state.

We call this override VALID_UNTIL_STATE_MATCH.

If the behaviour says the light has to be on @ 80% and you manually dim it to 40%, this is considered an override state, but still an ON state, light remains at 40%. This means that if you leave the room, the lights still turn off, and if you enter again, the lights will go on at 80%.

If it is 20:00, behaviour says 80%, you dim to 40% and at 21:00 twilight says dim to 50%, the light will NOT dim to 50% as twilight can only reduce light intensity.

If it is 20:00, behaviour says 80%, you dim to 60% and at 21:00 twilight says dim to 50%, the light will dim to 50% slowly (fade time should be tested).

When there is a conflict between behaviour dim percentage and twilight dim percentage, we choose the minimum value.

Definition *(Switch State)***:**
A description that represents an actual hardware state of a Crownstone. A pair of values consisting of:

- Boolean: relay_on
- Integer: pwm_duty_cycle (range from 0 to 100, both inclusive)

Definition *(Switch Intensity)***:**
Users may not understand the difference between relay and igbt's, so a *Switch State* is communicated to a user as a *Switch Intensity*. It will be up to firmware to decide what the corresponding relay/igbt values are. A *Switch Intensity* consists of:

- Integer: intensity (range from 0 to 100, both inclusive)

Definition *(Override Intensity)***:**
A *Switch Intensity* that is explicitly communicated by a user through switchcraft, tap-to-toggle, voice asistants or application UI elements. An *Override Intensity* may be undefined, and it may be the special value '*Translucent On*' signifying that the user wishes that the *Switch State* is *On* but wants to let the system decide which intensity is best suitable.

Definition *(Behaviour Intensity)***:**
A *Switch Intensity* that is generated by the firmware on a specific crownstone based on the *Active Behaviours* on that stone, at a specific moment in time, given a specific *Presence Description*. A *Behaviour Intensity* may be undefined.

Definition *(Aggregated Intensity)***:**
A *Switch Intensity* that is the result of combining *User Intensity* and *Behaviour Intensity* and the previous *Aggregated Intensity.* The *Aggregated Intensity* may be undefined, and will be undefined at startup.

Definition *(Next Aggregated/Override Intensity)***:**
In principle the priority of intensities is: *Override > Behaviour > Current Aggregated Intensity*, ignoring any undefined values. I.e. The *Next Aggregated Intensity* is equal to *Override Intensity* unless it is undefined, in which case it is equal to *Behaviour Intensity* unless that is undefined, in which case it will equal the previous *Aggregated Intensity*, or undefined. In these cases, the *Next Override Intensity* is equal to the previous one.
However, there is one exception:

- When all of three values are defined, and
- when the previous *Aggregated Intensity* and the *Override Intensity* match in terms of 'on' or 'off', and
- When the *Behaviour Intensity* is opposite of the previous intensity,

then the *Next Override Intensity* is undefined, and the *Next Aggregated Intensity* will be equal to the *Behaviour Intensity*. This is the VALID_UNTIL_STATE_MATCH criterion.

Switchcraft

It is 20:00 and behaviour is 80% WHEN PRESENT, but your phone is off. You turn the light on for switchcraft, we use the 80% value from the behaviour as a best bet for user preference.

The Crownstone is now in an ON override state with value 80%.

Conflict avoidance

If there are multiple active behaviours or twilights at any given time we will resolve conflicts as follows:

- Behaviour and Twilight are two separate entities, overlapping there is expected.

Behaviour conflicts:

- If both behaviours are presence based:
    - Determine which behaviour presence is active (ie. you're in the room expected by the system)
    - If multiple are active and you fulfill the presence both conditions:
        - Room presence > Sphere presence
        - Single room presence > Multiple room presence
    - If multiple room based presence behaviour conflict:
        - the switchstate of the behaviour with the latest start time is used.
- If there is a behaviour that is only time based and another is presence based
    - the crownstone will apply the switchstate of the time based behaviour if the presence condition is not fulfilled, if it is, the the switchstate of the presence behaviour is used (presence > time)
- If there are two behaviours that are only time based
    - the last to start is used.

Twilight

- Last to start is used

To summarize:

- Presence > time
- Room based presence > sphere presence
- Single room presence > multi room presence
- Else: latest start time

If a user tries to upload a behaviour that conflicts with another (time of day TOGETHER with active day) it does not store it and returns -1 instead of the rule index.

We keep a deterministic hash of the behaviour together with the data model which we can use as a check if rules have been synced. Hash will be delivered by phone so that the firmware can check if the data model on the phone is up to date.

In the future, we could advertise a 'master hash' as part of the servicedata. This is just a hash of the individual behaviour hashes and can be used by a phone to quickly and without additional communication setup check if it is still in sync.

NEW

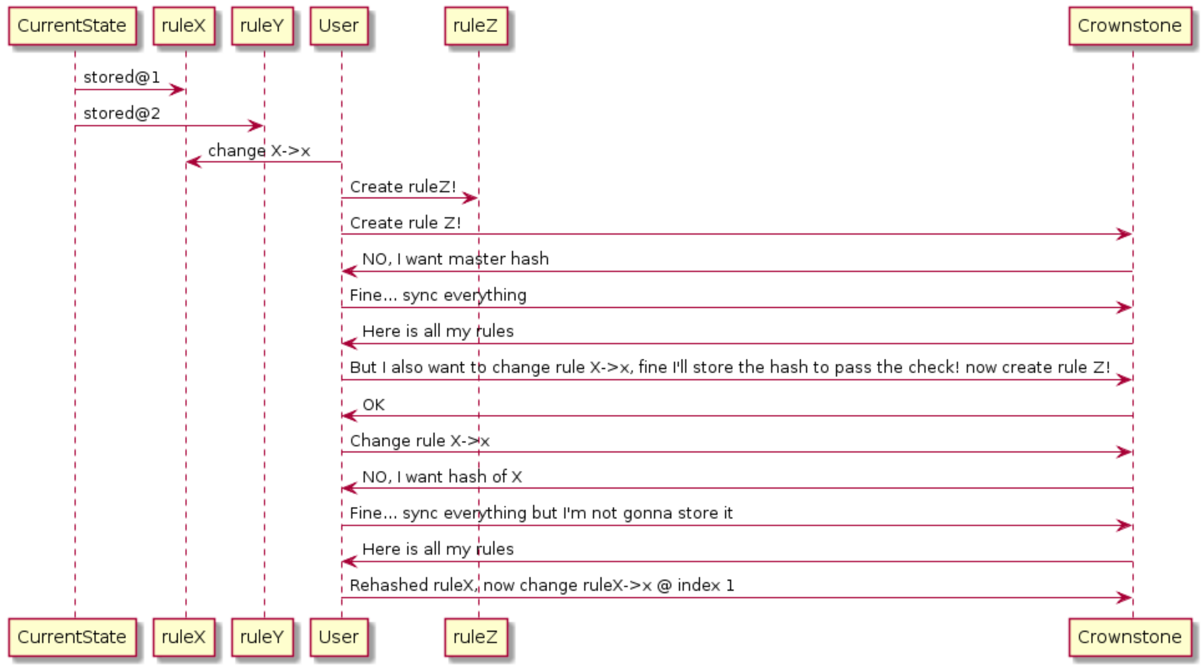
What we want to address:

- **Double writing of the same rule due to async of phones via cloud and async writing process**
- **Know when we are in sync with the Crownstone**

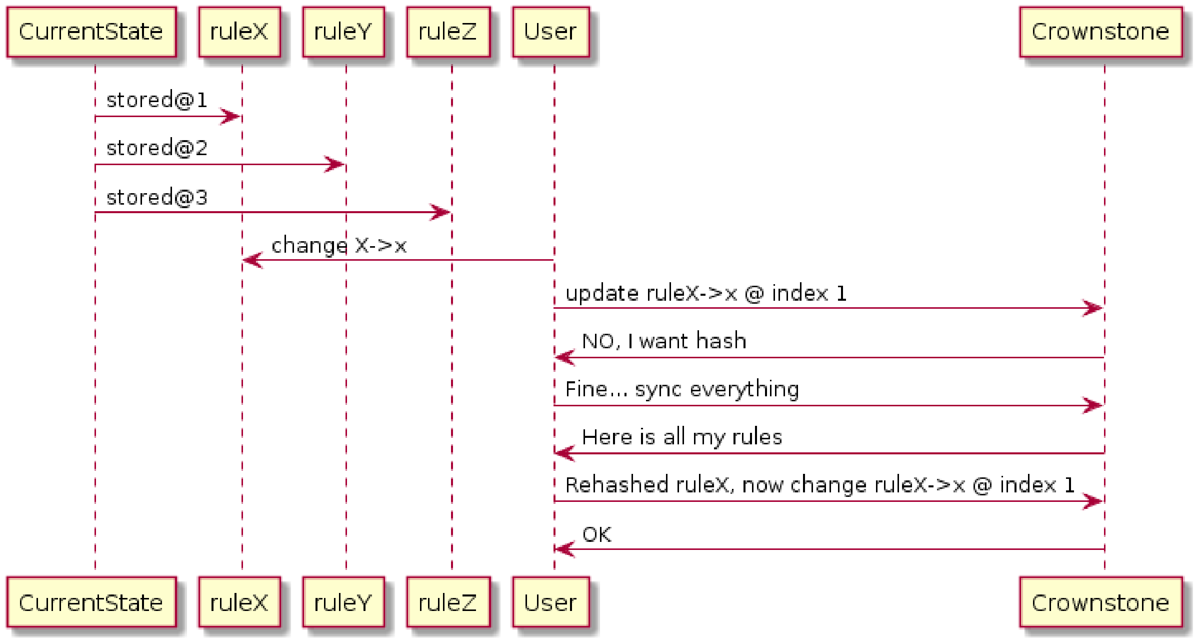
Current situation:

- **When saving: provide master hash of everything on the crownstone**
    - **This hash is built from all rules, except the new one.**
    - **This fails when I have changed 1 rule and created another since I cannot reproduce the master hash.**
        - **Workaround: store the last known hashes that I think are on the Crownstone**
            - **Problem: I want to change a behaviour and add one, disallowing changes until sync is wasted time.**
- **When updating: Provide hash of the previous version.**
    - **If I change the rule, pending sync, I dont have the previous version any more. This requires me to keep a history of previous versions, which has to also be synced over the cloud, which runs into the exact same syncing issues that we're trying to solve with the hashes**
        - **Workaround: store the hash that was on the crownstone without the behaviour as a sort of passphrase, which has to be also synced over the cloud.**
        - **The whole point of this is that we do not trust any cloud related syncing to work all the time. Why would we add another field here??**
- **When deleting: Provide hash of what you want to delete.**
    - **This is only possible if I do not alter the rule before deleting it. Otherwise the hash generated from the behaviour is different. All I want is to delete the behaviour that exists on index 4!**

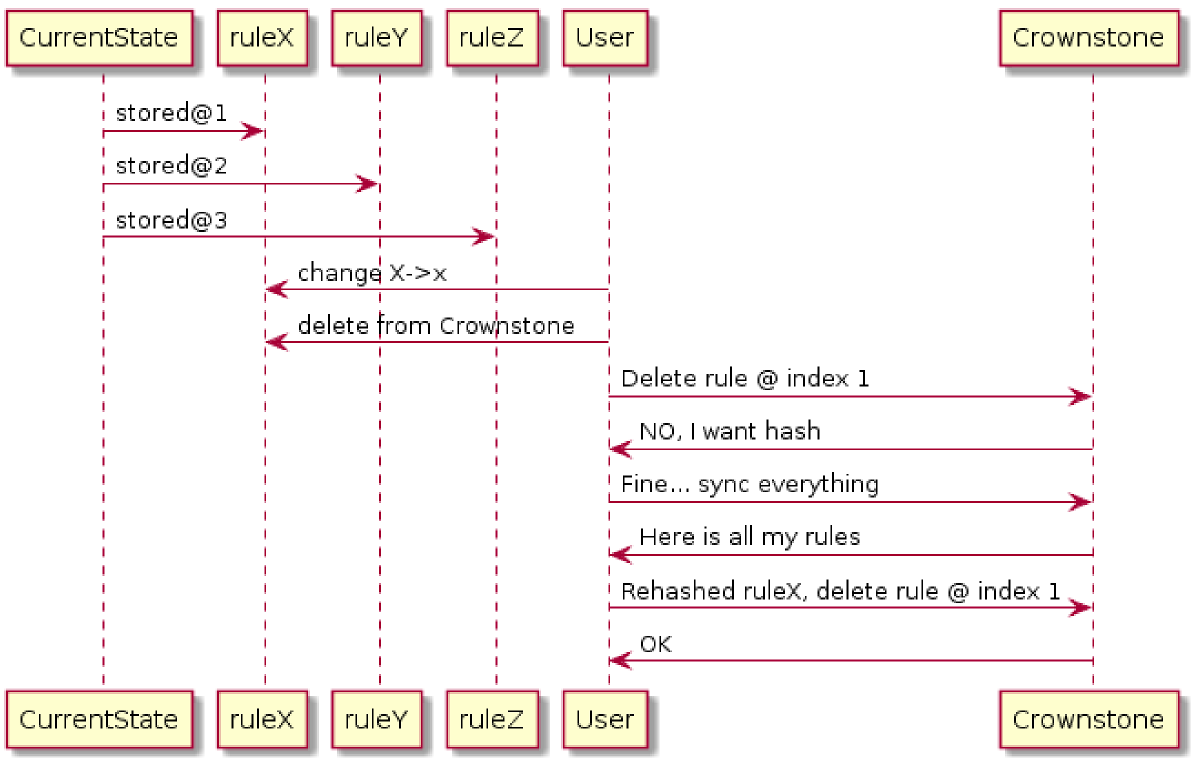
Summarizing: If I change a rule, my master hash is different assuming I dont keep a list of things that I think are on the Crownstone. If we want to keep up a list of things on the Crownstone, this has to also be synced through the cloud. My master hash is different, I have to sync, check if my rule exists, rehash and store the rule.



If I want to update a rule, I know that this rule lived in index 3. I do not have the hash of the previous one though, since it is changed pending storage on the Crownstone. So I have to I have to sync, check if my rule exists, rehash and update the rule.

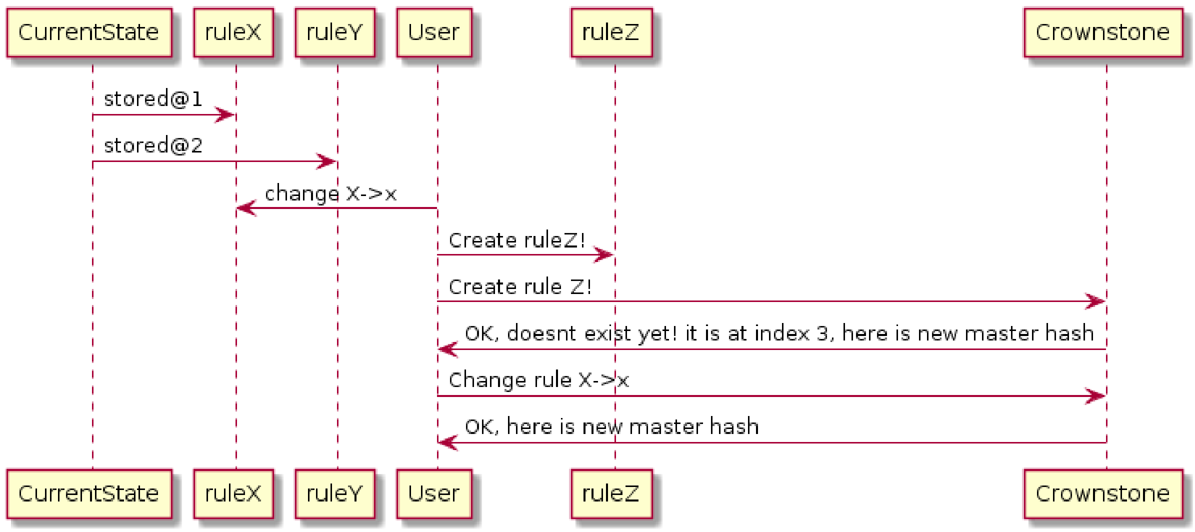


If I want to delete a rule, I have to present the hash of what to delete. But in the case of me changing the behaviour before I decide to delete it my hash is gone, so I have to I have to sync, check if my rule exists, rehash and delete the rule.

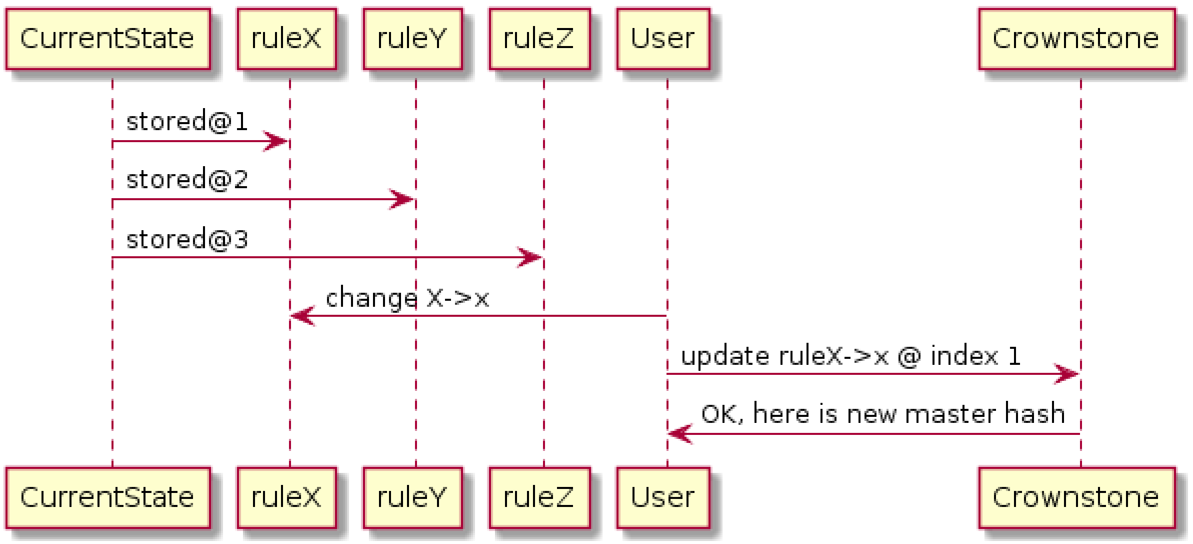


Solution:

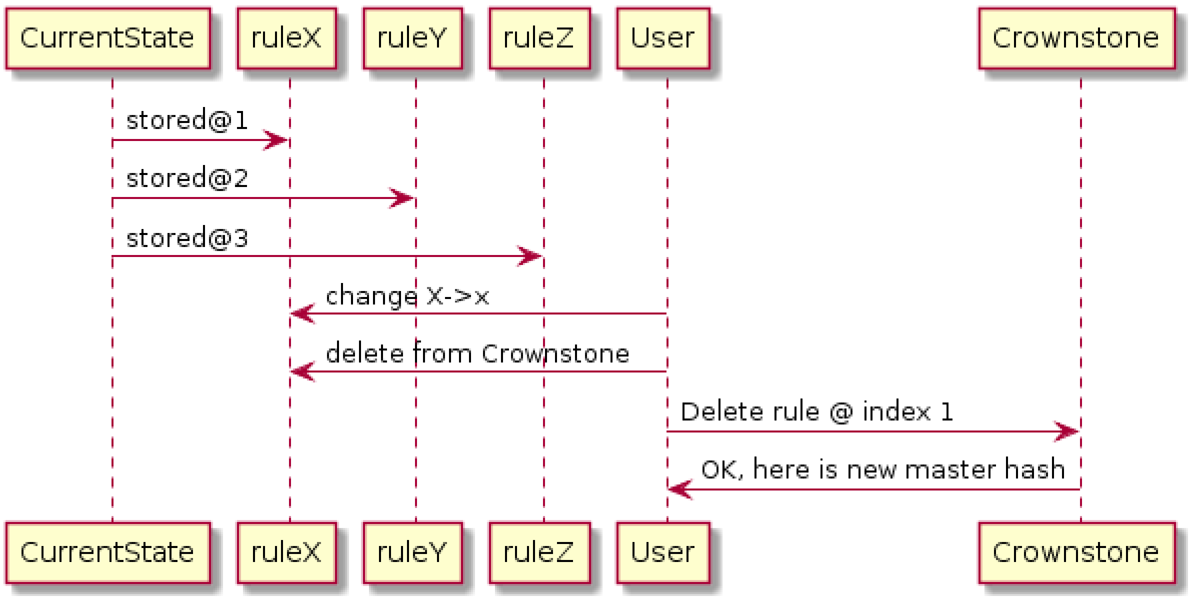
- **Propose hash of new rule to check duplicates. If not exist then store , otherwise return with the index.**



-
- **On change, just write to the index,** optionally propose the new hash and follow the addition rule



-
- **On delete, remove from the index,** optionally propose the new master hash that should result once this rule has been deleted to catch that it is already gone?



-
- **Use Master hash added to the return code after changes which can optionally be used for sync checking**
- **Use master hash in advertisement for state upkeep**

In the case of changing where the rule that I wanted to change is already changed by other people, we still want to change that rule! All the people with permission can overwrite each others data. Regardless if I was first, changed my rule, then the other came in and changed it into theirs or the other way around, the result will be that the last one to change it keeps it.

In the case of deleting the rule, suppose I want to delete rule 4. Someone else is changing rule 4. Still the last one wins, so I should just be able to delete rule 4.

Smart Schedules

These are behaviours that are initially time based (I will be on from sunset - 22:00) but with the condition:

- Afterwards, I'll stay on if someone is in the room
- Afterwards, I'll stay on if someone is in the sphere

This will be an option that will be provided to the rule. The Crownstone will set a presence based rule for at least an hour, after which the rule will be destroyed when all users have left the room / house.