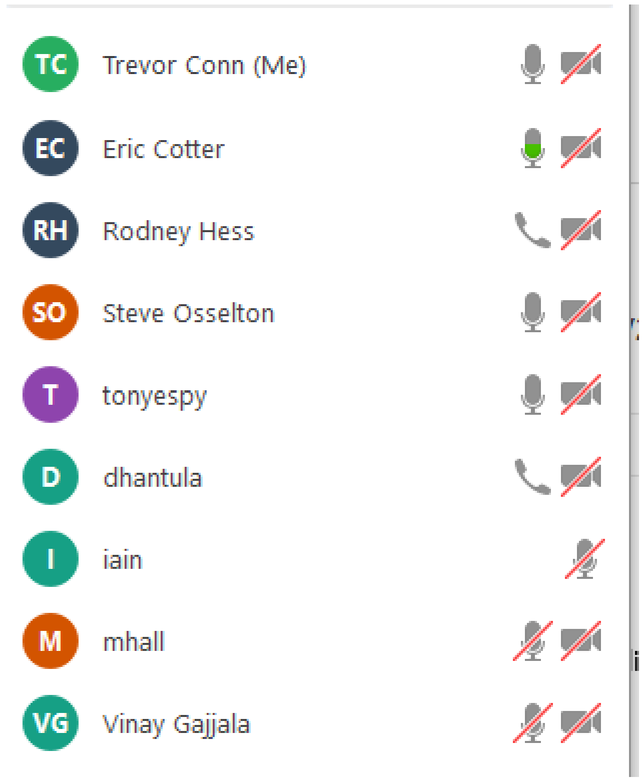
Device Services Meeting (2nd-July-2018)

Attendees


TC Trevor Conn (Me)
EC Eric Cotter
RH Rodney Hess
SO Steve Osselton
T tonyespy
D dhantula
I iain
M mhall
VG Vinay Gajjala

Scheduler Requirements

- Overview of scheduler requirements by Eric
- Discussion of where state is managed for scheduling
  - Endpoints provided from scheduler, metadata is the store
  - Device Service calls metadata directly on init
  - Java Device Services have the option of using internal schedule
- Desired – Device Services operate on schedule internally
  - Question: Does support-scheduler's responsibility change to become more of a mgmt API for scheduling?
  - Device Services could then fetch schedules, be updated by scheduler on changes
  - Device Services operates on schedule stored internally
  - Decisions on the changing responsibility of support-scheduler involve broader discussion with architecture group, SysMgmt
  - Ramifications for how device-virtual works
    - Which we need to convert to Go anyway
  - Need more discussion on required data structure – which approach?
    - Modify ScheduledEvent
      - This might make more sense, see DeviceVirtual as example
    - Modify DeviceProfile
- See Tony's emailed comments at the end of this document

- Based on the discussion, Eric to produce a V3 of his requirements document

Device SDK Requirements

- Tony revising his requirements draft (currently V7) based on feedback from Cloud Tsai
    - Device Profile documentation included in above, Appendix C
- Open issues
    - /all command endpoints
        - Data transforms on readings – one item out of many may fail so do we fail the whole operation?
            - If one reading in a collection of many has an overflow, how to handle?
            - If assertions cause overflow, how to handle?
            - Possible changes to response data structure to group success versus failure
        - Actuation on this endpoint? Same parameters applied to every device. Does this make sense? Probably not.
    - DeviceObject == DeviceResource – confusing mix of terms
        - Attributes attribute (Java: Object / Go: interface{})
    - DeviceProfile
        - Do we need additional command section? What is this being used for?
- **GO SDK Update**
    - Tony to complete assessment of remaining work
    - Meeting next week to discuss resource needs
    - Need to figure out repo names, how does it affect import path?
    - Request Jeremy to create repo once name is decided
- **C SDK Update**
    - C client API ready in perhaps another week
    - Need to figure out repo names, how does it affect import path?
    - Request Jeremy to create repo once name is decided
- **Binary Requirements Update**
    - No feedback received due to release
    - Will leave this out there for another week
- **Please add agenda item for Data Mgmt (Assaf's proposal) for next week**

On 6/29/18 11:45 AM, E.Cotter@Dell.com wrote:
Trevor pointed out a few thing which were unclear. I tided those up a bit.
Here is V2 of that email/discussion.

As noted earlier I can clear up and answer any questions you may have regarding this information.
Thanks for sharing Eric. Here are my comments:

1) "The schedulers goal looked primarily to solve issues with device event storage maintenance"

From my point of view working on device services for the past year, one of the main reasons the scheduler exists is to trigger readings to flow from a device service to Core Data. A secondary use in device services was to trigger a device service's /discovery endpoint used to trigger dynamic device scanning to determine if known devices have become available.

2) "Is a library currently..."

The heading is misleading. Also note that there are client libraries/packages that exist for Java and Go, but they aren't required (i.e. the scheduler implements a standard REST API).

3) Performs schedules and schedule events

You might want to note that there can be many schedule events which map to a single schedule.

3) Scheduler package which is executed by core

I'm not sure I understand the heading or text for this. What do you mean by "executed by core"? When I go back and check the legacy Java services, the only services that define scheduleEvent properties are device services.

4) Has a Scheduler client package...

See above comment re: client packages. Yes, they're nice to have, but aren't required. Also note that services can create as many Schedule objects as they need, and as mentioned above, can create multiple Events for each.

5) Schedules via an HTTP webservice API

"HTTP webservice API" --> "REST API"

Also note that the support-schedulers (both Go & Java) use the Core Metadata /schedule and /scheduleevent REST endpoints (via client packages) to retrieve the existing list of Schedules and ScheduleEvents. The only REST endpoint in the scheduler itself is a /callback endpoint used to update Schedules and ScheduleEvents when they're updated in Core Metadata.

6) No persistence

As noted above, Schedules and ScheduleEvents today are stored by Core Metadata, and this is where support-scheduler gets them from.

7) Username/Password support?

I'm not sure what this means. We currently don't support any inter-service authentication/authorization for any of the core, support, or device services.

8) Require clients "Services" to request their own...

This is correct. The legacy Java device services mentioned above use property files to defined Schedules and ScheduleEvents, and then uses a Core Metadata client to push them to Core Metadata.

9) A standalone Service

It already is...

10) Persistent Schedules

They already are...

11) Storage Othogiality

The service already has an in-memory cache. I think the question being asked is whether or not Core Metadata is the right place to store scheduling data? AFAIK, Core Metadata was designed so different storage layers could be implemented.

12) Ability to run Local as well as Remote

What's the use case for running shell scripts? Most OSes already have a system to do this?

Isn't a REST call to a service a remote job, or did you have something else in mind?

13) Implement Retries

I think you mean "the ability to detect failed ScheduleEvent REST calls and retry". If we were to support this, we'd probably want a bool and maybe a set of status codes in ScheduleEvent to know when to retry.

14) Timeouts

Isn't this a special case of retry, or do you mean something different?

15) ISO Standard

You might want to also consider IETF RFC2822 and RFC formats. See the Linux date command for examples.

16) Other Issues

The biggest issue I see with scheduling is the way scheduled readings work in the existing legacy device service designs.

The Java Device Service (aka DS) SDKimplements a scheme where a call to a DS command endpoint will trigger a reading from a specific device, or all devices, for a specific device resource. A GET command to this endpoint resultsa in the DS returning one or more events and readings but *additionally* triggers the DS to push the same events and readings to Core Data. In the case where the reading was triggered by a ScheduleEvent, the results to the REST call are basically dropped on the floor, so all that data is being sent for naught.

Furthermore, the Java SDK also designed an internal scheduler, which didn't rely on support-scheduler. It basically reads all the Schedules & ScheduleEvents from Core Metadata, and then implements it's own internal scheduler. The gross hack here, is that when ScheduleEvents fire, the DS makes a REST call to *itself*, and as above, it also drops the results in the floor, because all that really mattered was "priming the pump (so to speak)" to trigger the reading to get pushed to Core Data.

Cloud, who designed the Java device-virtual service realized this wasn't an ideal solution, and designed yet a third way of scheduling readings. For every device resource (an atomic value on a device that can be read/written) he defined a 'collectionFrequency', and then implemented yet another internal scheduler which triggers readings to be pushed to Core Data. Note, this only works for readings, not actuations.

I think it'd be possible to build a hybrid solution by extending the DeviceObject by adding a top-level frequency and delay (you wouldn't want all devices to send at the same exact time) attributes. This would only work for DeviceObjects (aka deviceResources in a device profile YAML file), not commands.

Finally, one last cheap option would be to add a new parameter to the DS command endpoint which says "don't send results back in reply".

That's it for now...

Regards,
/tony