A Brief Introduction to the **Splot Object Model** and **SMCP**

Robert Quattlebaum
rquattle@google.com
T2TRG/WISHI 2018-12-12

Google

Good morning everyone, my name is Robert Quattlebaum. Today I'll be giving a quick introduction to what I call the Splot Object Model and briefly touch on SMCP. In the interest of keeping this presentation short, I'm only going to go over the high-level concepts and only touch on a few interesting lower level details.

Splot Object Model Design Goals

- Emphasize monitoring and control of state
- Simple, easy to understand control surface
- Efficiently encapsulate existing IoT technologies, like Zigbee, Zwave, etc.
- Scalable to relatively medium-large systems

Google

The Splot Object Model is an experimental object model for facilitating the monitoring, control, and automation of individual things and groups of things. The model was designed with the following goals in mind:

 * Be straightforward for developers to understand and use,
 * expressive enough to satisfactorily wrap the behavior of other popular technologies in widespread use,
 * and flexible enough to be useful with large and complex systems

The model is currently being used to guide the development of experimental general-purpose IoT APIs as well as providing the basis for a new experimental application protocol: SMCP.

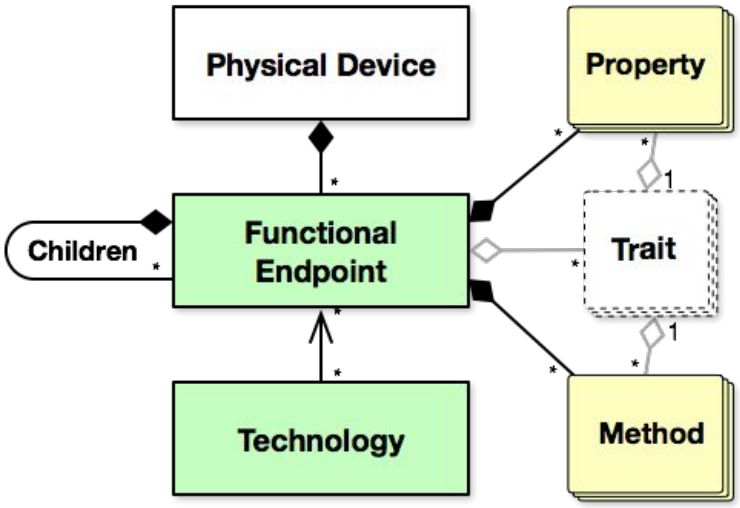
Functional Endpoints (FEs)

- ***Functional Endpoints*** are the fundamental control surface object
  - Physical devices ***host*** one or more *Functional Endpoints*
  - Monitored and controlled via ***Properties*** and ***Methods***
  - *Properties* and *Methods* are defined by ***Traits***
- May directly own/host child functional endpoints
  - Used to implement scenes, object managers, etc.

Google

The most important object type in the model is called a "functional endpoint". Functional endpoints are intended to represent an interface for controlling or monitoring a thing. Physical devices host one or more functional endpoints. For example, a smart dimmable light bulb would be exposed in the model as a single functional endpoint. On the other hand, a smart power strip would have one functional endpoint per power socket.

Splot Object Model UML



Google

Functional endpoints have "properties", "methods", and "children". For example, to toggle a smart light bulb on or off, you would toggle the on-off value property. To create a new "scene" based on the current state, you would call the "save" method, resulting in a new child functional endpoint representing the new scene. To manually tweak that scene, you would adjust the properties on that newly created child.

Functional endpoints aren't required to only represent permanent hardware features: they can also represent software features. For example, groups are special functional endpoints which allow you to change state (or invoke methods) on several independent functional endpoints with a single action.

Traits

- A named/versioned collection of property and method definitions.
- Not actually an object you directly interact with, more like a loose contract a functional endpoint adheres to
- The closest Splot gets to having a "schema"
- A functional endpoint uses one or more Traits to define its properties
- Traits provide additional structure to property naming
- Custom traits are allowed

Google

Properties and methods are defined by traits. Traits are about as close as the model gets to having schemas. Some traits---like the Level trait---expose a single behavior, whereas other traits---like the ambient environment trait---can expose more than one behavior.

Functional endpoints can implement several traits, which enables polymorphism. For example, a dimmable smart light bulb could implement the "On/Off" and "Level" traits, providing the ability to turn the bulb on or off as well as adjust its brightness. The level trait could also be implemented by non-lights, such as a smart window blind.

Property Sections

Each property is associated with a *section*: **State**, **Config**, or **Metadata**:

State

Controls and/or monitors fundamental *operational aspects* of the functional endpoint.

Ex: On/Off control, audio volume, current power draw, temperature etc.

Config

Controls how the functional endpoint *behaves*.

Ex: Maximum current before auto shutdown, dimmability, power-up scene, etc.

Metadata

Describes the functional endpoint and its capabilities.

Ex: Admin name, product name, UID, max current rating, max lumen output, etc.

Google

Properties, methods, and children are scoped by trait. Properties are additionally organized into one of three "sections": "State", "Config", or "Metadata".

State properties are used to directly monitor or control the primary behavior of the functional endpoint. For example, a boolean on/off value property would be in this section.

Config properties are used to indirectly change the behavior of the functional endpoint. For example, a current limit property that turns off the load when the specified current is exceeded would be in this section.

Metadata properties are used to identify the functional endpoint and describe its behavior in a static fashion. For example, a human-readable label property would be in this section, as would a maximum power property describing the maximum power draw of the device. Metadata properties are also used for discovery.

Properties can be fetched and changed individually or by section. This allows you to fetch or change an entire section with a single action.

Properties

- Primary interface to manipulate functional endpoints
- Operators: **get**, **observe**, **set**, **increment**, **toggle**, **add**, **remove**
- Types: **integer**, **float**, **boolean**, **string**, **data**, **array**, **dictionary**
- Can be fetched/changed individually or by *section*
- Organized by
  - **Section** (State, Config, Metadata)
  - **Trait** (On/Off, Level, Lock, Temperature, etc)

Google

Properties are generally preferred over methods.

Some properties can be mutated, meaning they can be altered in such a way that the resulting value is dependent on the previous value. This includes operations like "toggle" and "increment". Properties can only be mutated individually: you can't currently mutate several properties at once.

Automation Primitives

Pairings

Pairings allow the value of a property or section to be applied to the value of another property or section. Values can be directly replicated or can be transformed algebraically.

Rules

Rules describe actions (which behave like webooks) that are triggered when specific criteria are satisfied. Think “If This Then That” (IFTT), except without the cloud service. Rules can be used to make timers and periodic events.

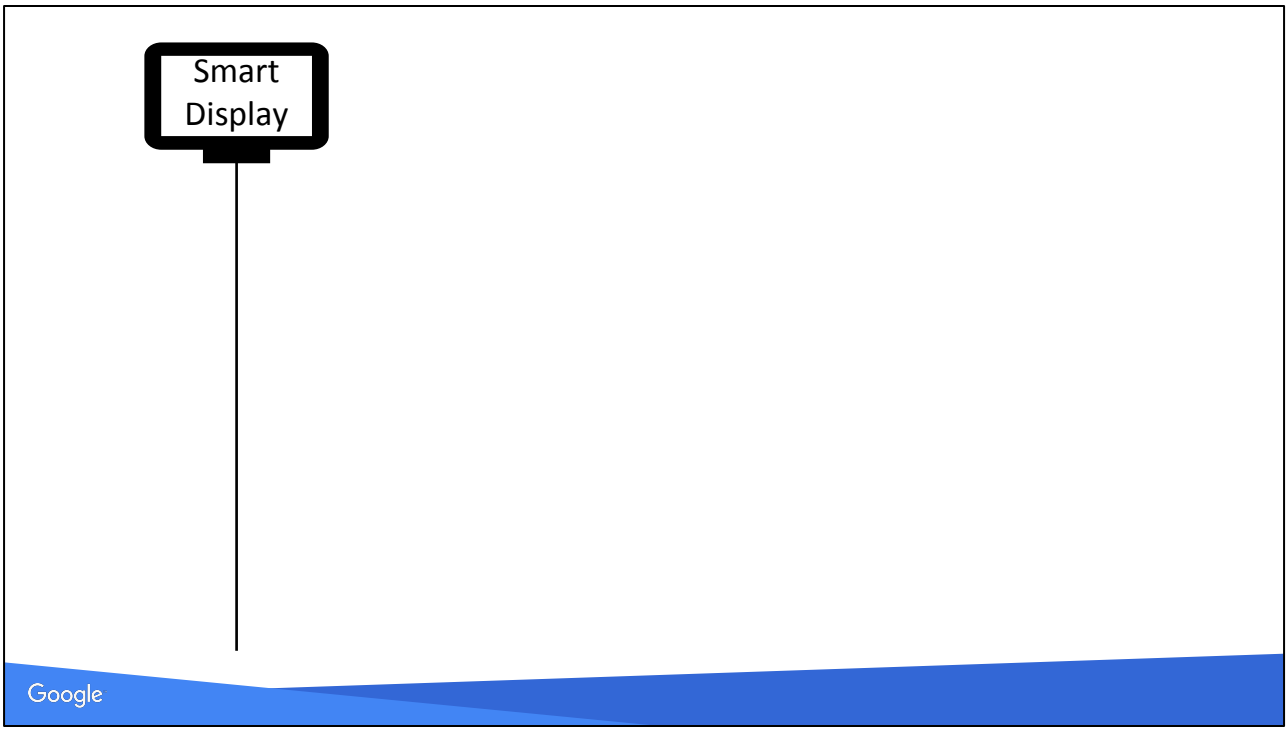
Both are intended to live on the devices they control, and both are *functional endpoints*, allowing them to be configured in-band.

Google

The relatively simple structure of the Splot Object Model allows us to define automation primitives that can themselves be configured in-band as functional endpoints. In SMCP, these would typically be hosted on one of the devices that they automate, but could also live on entirely separate devices.

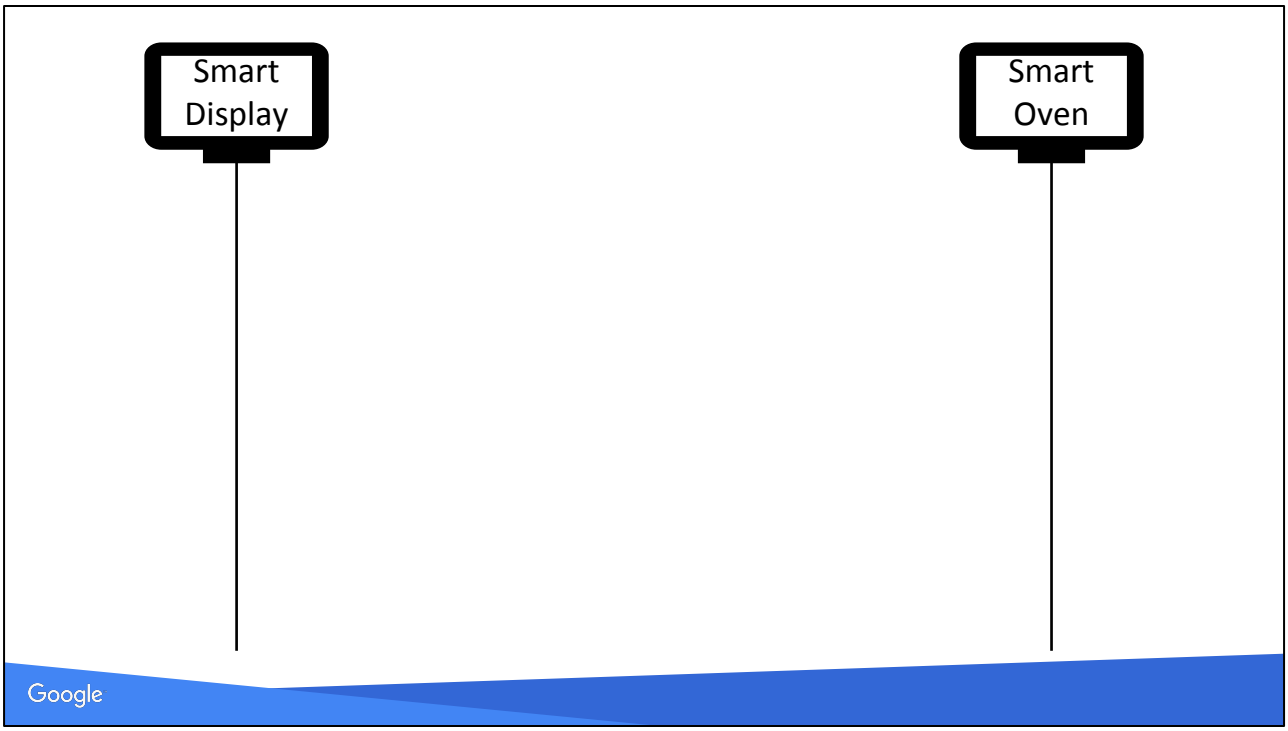
There are two automation primitives that are currently being defined for SMCP: Automation pairings and automation rules. Automation pairings are used to monitor changes in the value of a property and map that to the value of another property. Automation rules would be used to trigger actions (like a POST) when specific criteria are satisfied, somewhat like If-This-Then-That, except local. Using these two automation primitives, a rich set of intra-device behaviors can be configured.

These two automation primitives can't cover all interesting use cases, but they are surprisingly flexible. One case in particular I think is worth discussing as a quick case study.

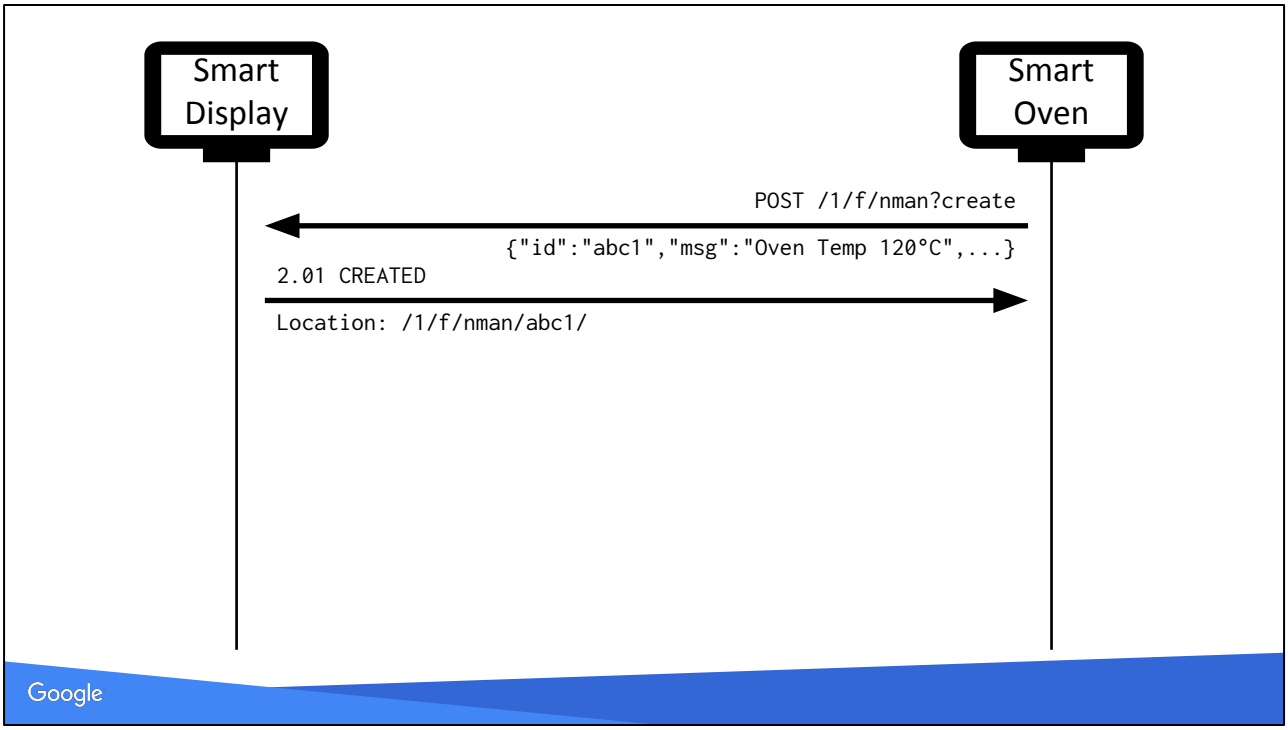


Let's say we have a smart display device that supports displaying notifications, which consist of a message and some optional actions that the user might want to take. These notifications would be received over the network using SMCP.

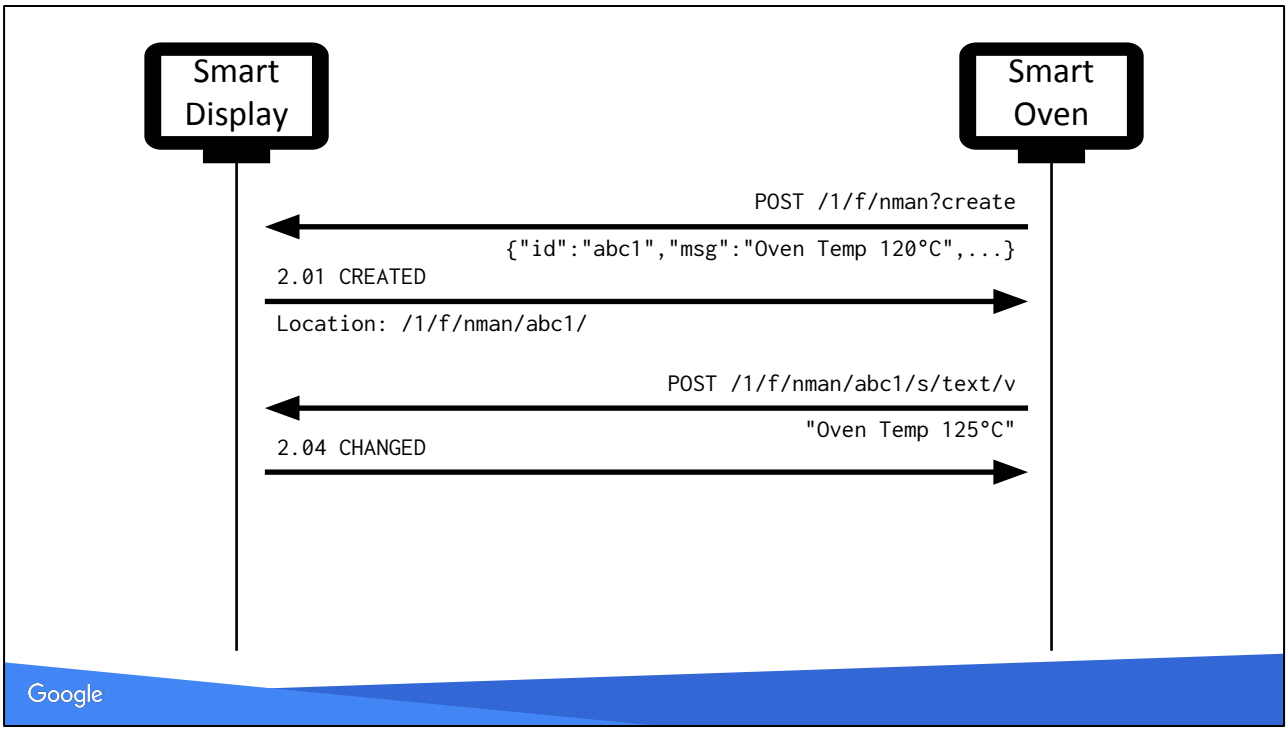
Notifications are managed using a notification manager functional endpoint. This functional endpoint has a method called "create", which when invoked creates an instance of a new notification—which is, itself, a child functional endpoint---allowing the notification to be updated after it has been created.



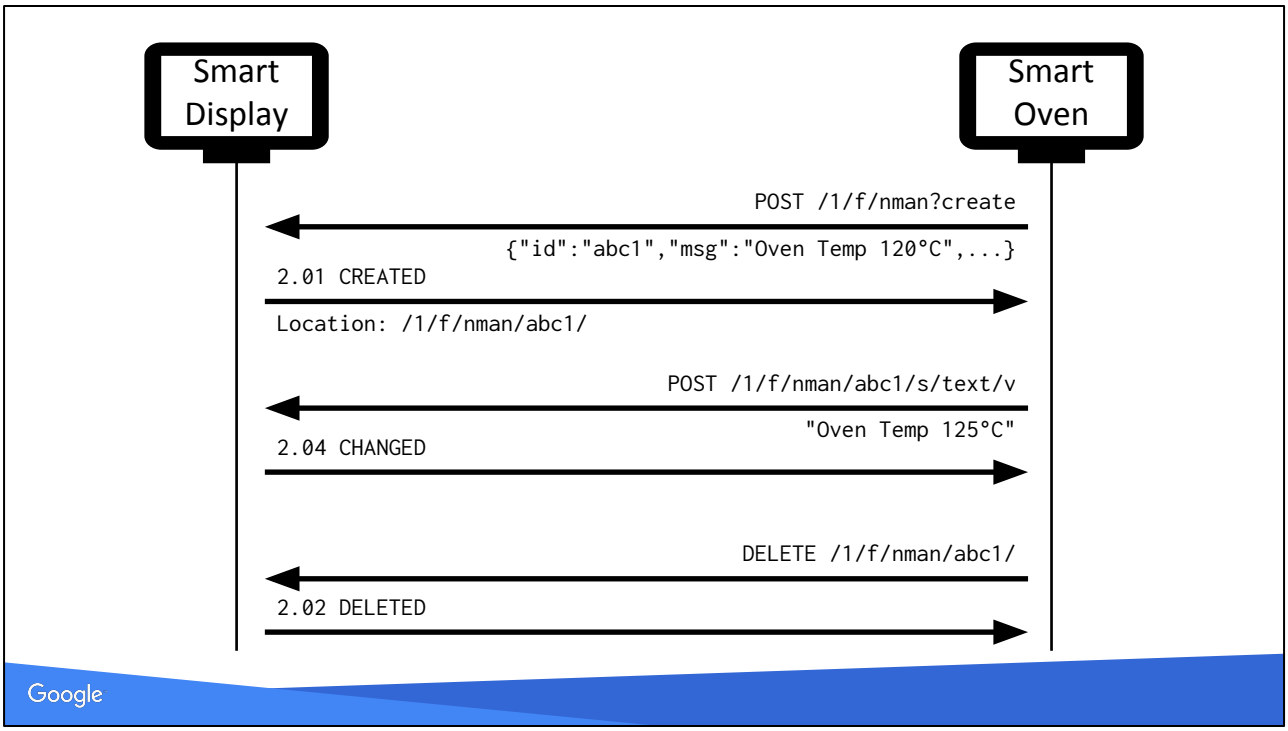
A smart oven might want to create a notification whenever the oven is turned on, giving the user the option to monitor the temperature or timer and perhaps give the user the ability to turn off the oven from the smart display.



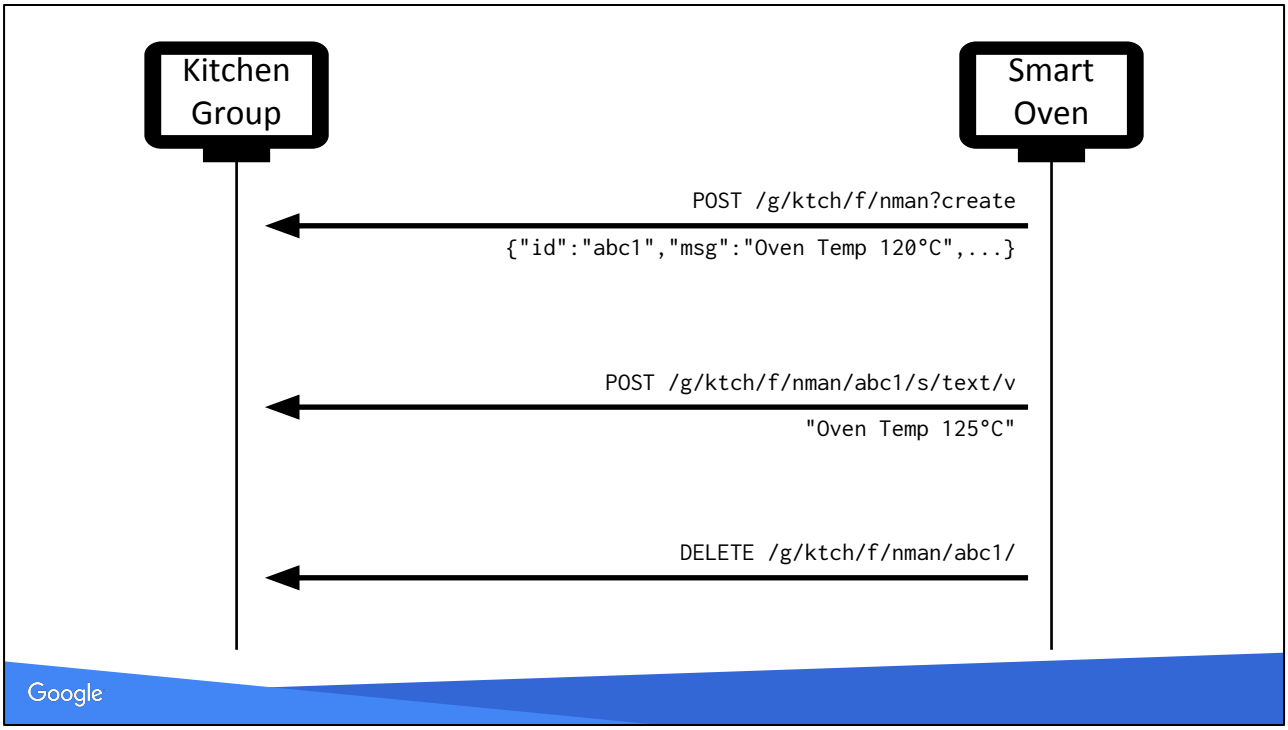
This smart oven invokes the "create" method on the notification manager on the smart display, passing the initial information about temperature and named actions the user could take (like turning off the oven). A new child functional endpoint that represents the notification is created as a result of the smart oven invoking the "create" method.



This functional endpoint would be updated as the oven temperature increases.

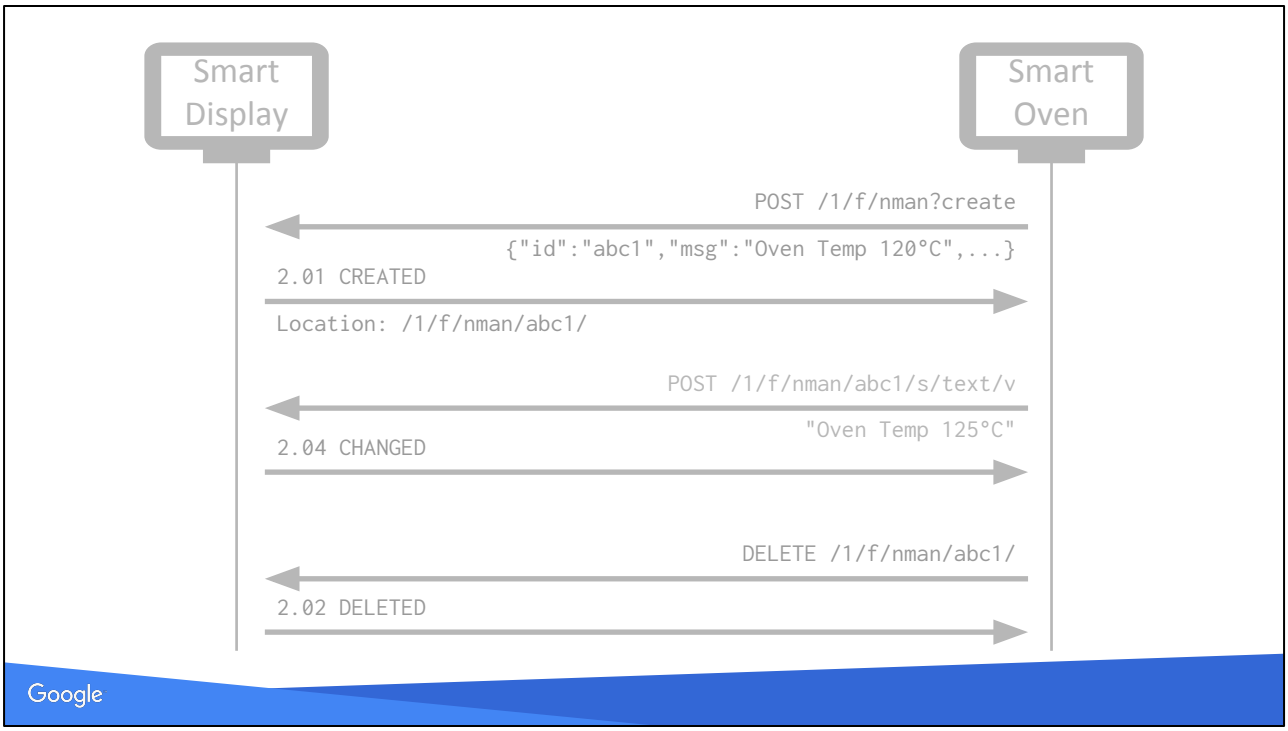


When the smart oven is eventually turned off, the functional endpoint for the notification is deleted by the oven and it disappears from the smart display.



Taking this idea a step further, the smart oven could perform these actions on a group instead of directly on a specific smart display. In this case you could have the notification appear on several smart displays, or perhaps even your smartphone (via a bridge device).

But this case still requires the smart oven to know what a notification manager is and how to modify, update, and eventually delete a notification.



However, notifications can be created and used via automation primitives, too. For example, if you wanted to display a warning on all the smart displays in the house whenever the garage door was left open for longer than an hour, you could set up an automation rule with an action that would create such a notification. You could even include a user action with the notification to allow the user to close the garage door from the notification. So even notification behaviors can be automated without hard coding them to specifically handle generating notifications.

Note that this even applies to the oven case (including temperature updates), but that would require more detail than I have time for here.

In residential settings all of the complexity of setting up such automation primitives and other behavioral relationships would be automated by apps: either based on smartphones or in the cloud. Advanced users and contractors would still be able to make manual tweaks for custom behaviors if they had the expertise.

This example highlights a strength of the SMCP approach: the burdens of supporting behavioral features like this are relocated from the device firmware to the configuration applications, while still allowing devices to directly communicate with each other (or through a third local device).

Conclusion

Google

Since this was a brief introduction to the Splot Object Model and only a superficial discussion of SMCP, there are a great many relevant details that have been omitted, but hopefully this presentation has given you an idea of how some of these parts fit together. Additionally, more formal documentation is forthcoming early next year.