Last updated: 6/7/2019

Prior to the Delhi release, EdgeX Foundry and device services based on the Java SDK supported a feature called "dynamic device discovery". This feature was originally created in order to support wireless device protocols such as Bluetooth Low Energy, where a given device might not always be available. In such a scenario, the idea was to provide EdgeX with a set of filtering rules which could be used to dynamically add a "discovered" (i.e. available) device to the system if the rules produced a match.

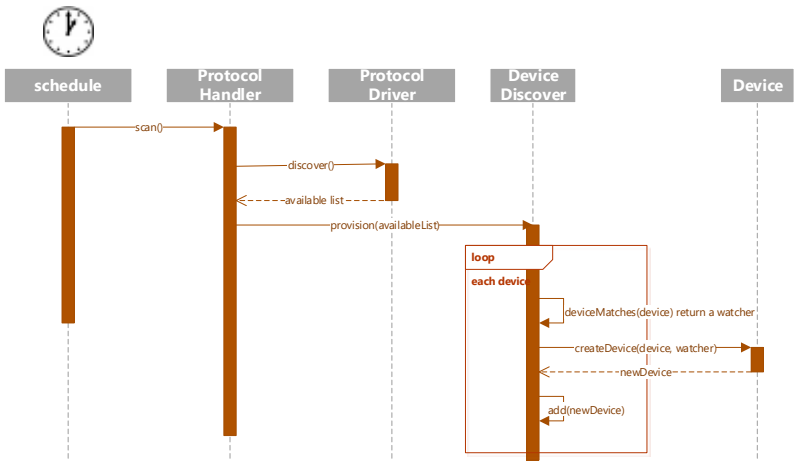
This mechanism relied on Core Metadata object called a ProvisionWatcher which was used to define filtering rules. A ProvisionWatcher has the following fields:

- name
- device-service - name of the owning service
- device-profile - profile assigned to new matching devices
- identifiers - map of attributes used to identify a device (eg. name, port, MAC, uuid, …)
- operating-state - initial operating state for new devices (optional)

  Example:
  default.watcher.name=Thermostats
  default.watcher.service=device-sdk
  default.watcher.profile=JC.RR5.NAE9.ConfRoom.Padre.Island
  default.watcher.name_identifiers=[246]

Note - identifiers can be fully-qualified or could use the wildcard ('*') for pattern matching. This allows a single ProvisionWatcher to be used to match multiple devices.

Device services (via code provided by the Java SDK) provided an endpoint called /discovery. Calling this endpoint (usually done on a scheduled basis) resulted in the device service calling an internal Scan method in the device service's ~~ProtocolDriver~~ ProtocolHandler. The result of the Scan method (working with the device service's was a list of available devices. For each device in this list, the ProtocolDriver provided a map of attributes that can be used to identify devices of a given protocol. For instance, for BLE devices this map might contain a MAC address and a service UUID.



The device SDK code would iterate through the scan results looking for new devices. If a new device was detected, then the SDK would attempt to match the device against any existing ProvisionWatcher objects associated with the

device service. When matches were found, the SDK would add the new device using the device profile and operating state specified in the ProvisionWatcher.

The final piece of the puzzle is Support Scheduler, which via Schedule and ScheduleEvent objects could be used to periodically call a device service's /discovery endpoint.

= Problems =

There are a few problems with this approach to dynamic device discovery.

1. Device deletion - if a device is deleted from the device service (and Core Metadata), on subsequent scans if the device was found to be available again, it would be re-added.

This problem is where the idea of device blacklists came from. In addition to deleting a device, you'd also add it to a blacklist object (in the ProvisionWatcher?). I'd argue that this is just adding to already complicated approach. Instead of black-listing the device, why not modify (or delete) the matching ProvisionWatcher before deleting the device?

> +1, in fact, this act could be a trigger that causes the launch of provision or unprovision/deregister or devices (outside of any schedule).

2. Scan are synchronous - the /discovery endpoint didn't support asynchronous scan results.

> Would we even need a /discovery endpoint? If the internal scheduler is used to kick off the request, why expose a synchronous (or asynchronous) endpoint?

3. ProvisionWatchers contain device profiles. This is the same problem we currently have with embedded device profiles in devices. This makes ProvisionWatchers fairly heavy-weight objects, and adds complexity if/when a device profile is updated.

> +1, Really just need the device profile name. The inclusion of the entire profile was a result of MongoDB storage arrangement.

4. Scanning in a fixed period is less than ideal, as scanning can impact performance and also may impacts battery life for certain devices. Ideally the logic for determining when to scan should be the responsibility of the device service, not pre-configured using Support Scheduler.

> I.e. – using the new internal DS scheduler?

Example Java configuration for BLE ProvisionWatchers:

```
default.watcher.name=BLE-Watcher,BLE-Watcher-2,BLE-Watcher-3
default.watcher.service=device-bluetooth,device-bluetooth,device-bluetooth
default.watcher.profile=SensorTag,XDK,BLELight
default.watcher.name_identifiers=.*CC2650.*,XDK.*,.*beLight.*
```

Food for thought…

Instead of the ProvisionWatcher approach, should devices be explicitly added individually and a new attribute added which indicates "presence"?

> Not sure I follow this. Do you mean “presence” as in enabled?