Geneva API Requests/ Responses

API Definition Principles for Specificity and Longevity

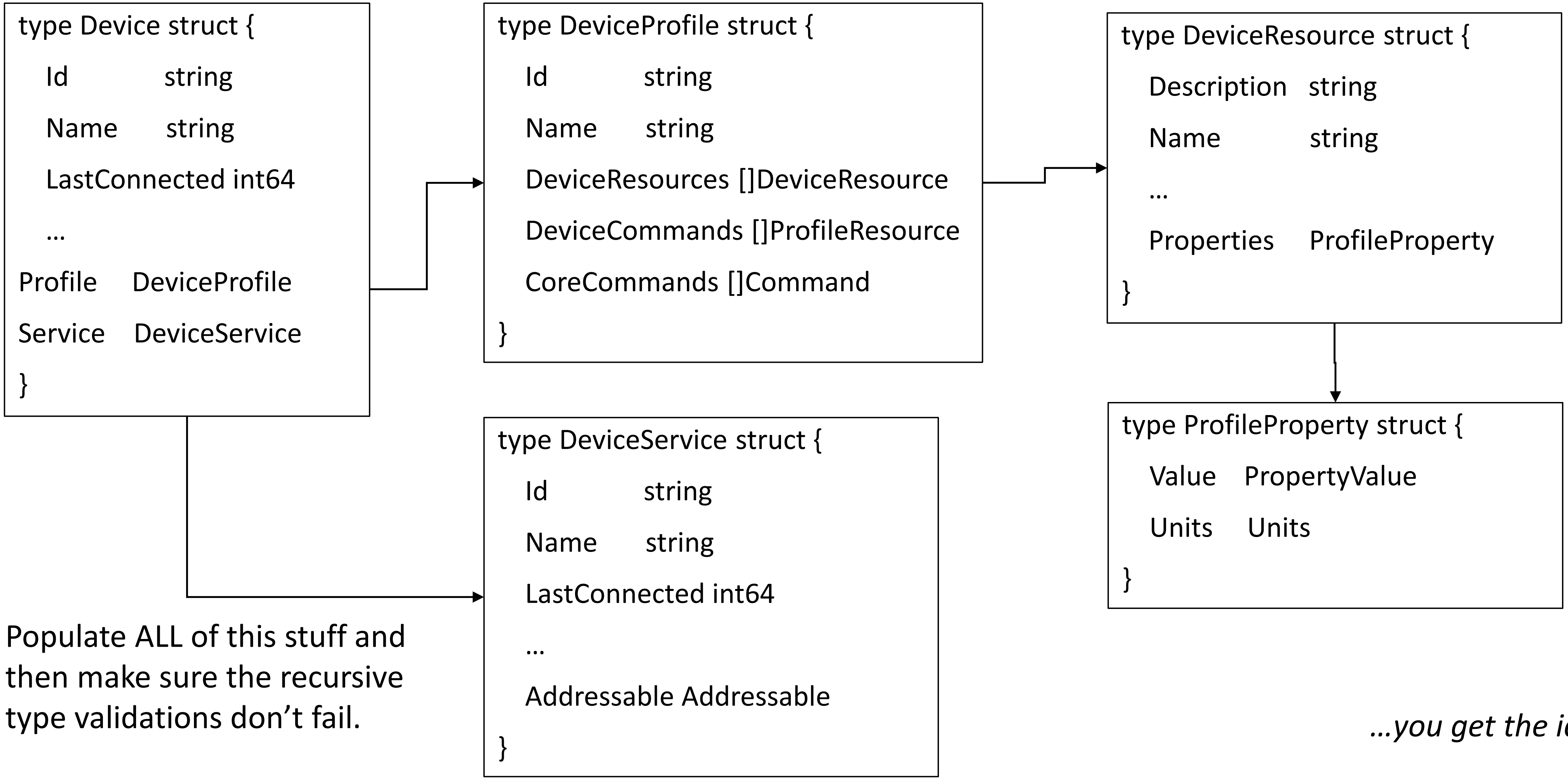
History of v1.x API

- Types marshaled as requests/responses were identical with internal representations of state in EdgeX Foundry services
- These types were defined in the edgex-go repo along with the service implementations
- For Edinburgh release, we split these types into go-mod-core-contracts. Benefits include

  1.) Clients no longer have to import entirety of edgex-go

  2.) State internal to edgex-go can vary from request/response contracts (persistence model types, for example)
- However we still currently have some baggage

I Just Want to Add a Device ☹



```
type Device struct {
   Id           string
   Name       string
   LastConnected int64
   …
Profile    DeviceProfile
Service    DeviceService
}
```

```
type DeviceProfile struct {
   Id           string
   Name       string
   DeviceResources []DeviceResource
   DeviceCommands []ProfileResource
   CoreCommands []Command
}
```

```
type DeviceResource struct {
   Description   string
   Name             string
   …
   Properties     ProfileProperty
}
```

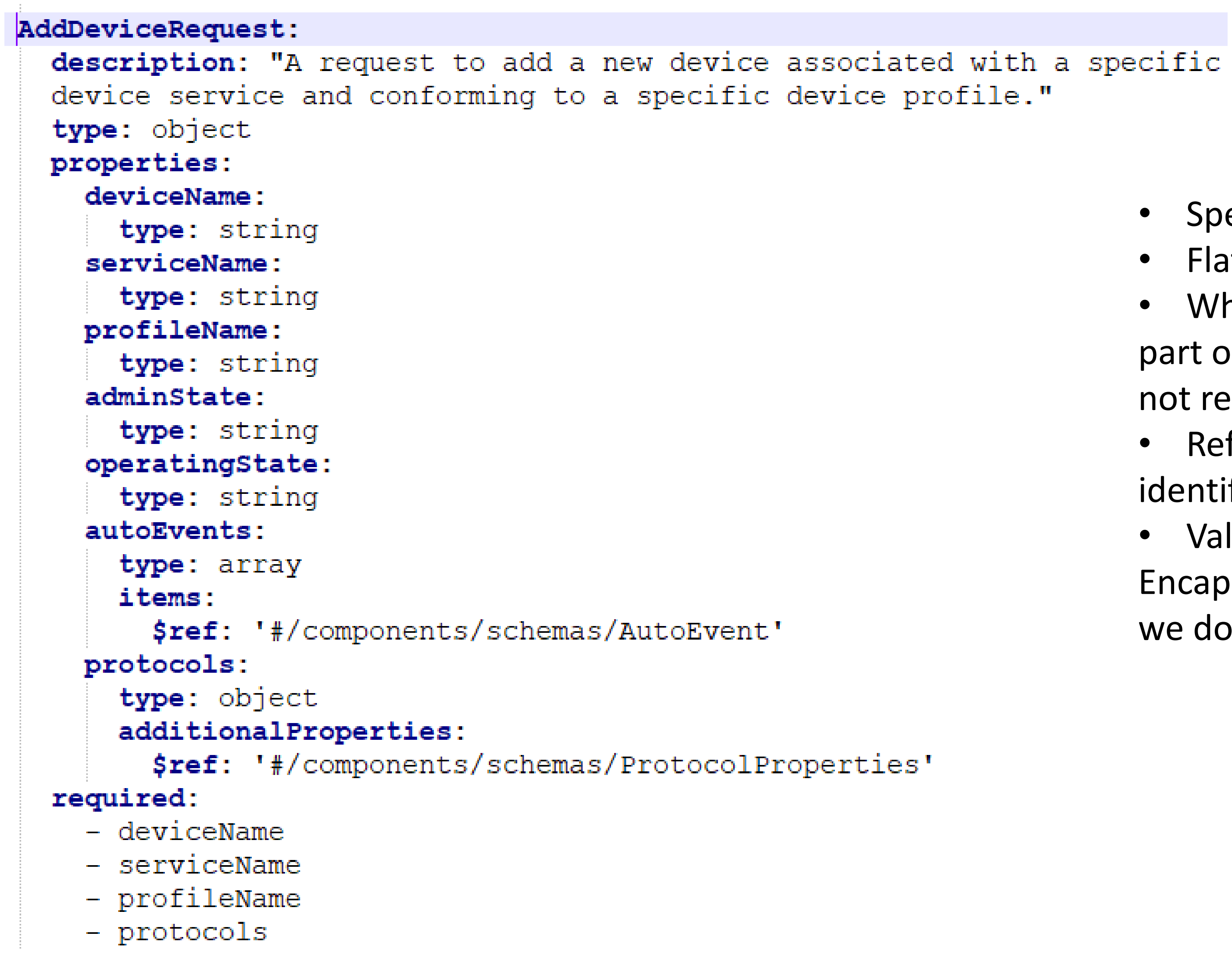
```
type ProfileProperty struct {
   Value    PropertyValue
   Units     Units
}
```

```
type DeviceService struct {
   Id           string
   Name       string
   LastConnected int64
   …
   Addressable Addressable
}
```

Populate ALL of this stuff and then make sure the recursive type validations don't fail.

…you get the idea…

Well Now You Can! ☺



```
AddDeviceRequest:
  description: "A request to add a new device associated with a specific
  device service and conforming to a specific device profile."
  type: object
  properties:
    deviceName:
      type: string
    serviceName:
      type: string
    profileName:
      type: string
    adminState:
      type: string
    operatingState:
      type: string
    autoEvents:
      type: array
      items:
        $ref: '#/components/schemas/AutoEvent'
    protocols:
      type: object
      additionalProperties:
        $ref: '#/components/schemas/ProtocolProperties'
  required:
    - deviceName
    - serviceName
    - profileName
    - protocols
```

- Specific request type to add device
- Flattened as much as possible
- Where nested types exist, they are part of the device definition itself and do not refer to other primary types
- Refer to other primary types by an identifier (in this case "Name")
- Validation of the request is still Encapsulated within the specific type, as we do today.

Looking toward a v2.x API

- We do not want to go through a v3.x exercise 12 months from now
- We need basic principles we can use to define a new API
  - Learn from the past
  - Allow for extensibility
- Preference for defining specification before implementation
  - Underway using OpenAPI 3.x specification (this is Swagger now)

Geneva API Guidelines Proposal (Requests)

- Request definition guidelines
  - GET/DELETE – The URL is the request. No additional type is needed
  - POST – This is an “ADD” operation. The request type should be named accordingly (e.g. AddDeviceRequest)
  - PUT – This is an “UPDATE” operation. The request type should be named accordingly (e.g. UpdateDeviceRequest)
    - This type provides the full state of the object being updated. Partial state updates each have their own specific routes (see later slide)
    - In provided example, this type tends to be identical to the respective Add request with the addition of the object’s ID property.
- All request types must implement self-validation

Geneva API Guidelines Proposal (Responses)

- In the case where an API returns a body, the content must be a marshaled type (JSON by default). No literal string return values.
- Response definition guidelines
  - GET (single item) – Return the requested type (e.g. Device)
    - If requested item is not found, return a 404
  - GET (list) – Return an array of the requested types. MUST support pagination via querystring parameters
    - If no items were found, return an empty array (200 HTTP status code)
  - DELETE – No content returned, 204 HTTP status code indicates success.
  - POST
    - If successful, return NewIdResponse type (e.g. provide the ID of newly inserted record)
    - If unsuccessful, return ErrorResponse type
  - PUT
    - If successful, return SuccessResponse type
    - If unsuccessful, return ErrorResponse type

Geneva API Guidelines Proposal (Routes)

- GET
  - Retrieving an item by ID or Name requires unique endpoints for each. No dual-purposing of routes.
  - Retrieving a list of items MUST support pagination via querystring parameters.
- POST
  - Only used for additions of new entities
  - Route should identify that entity with no additional cruft
    - E.g. "/api/v2/device"
- PUT
  - Only used for updates
  - If updating a specific property on an entity (like Device.LastReported) values specific to the operation should be on the Request type, not the route
    - /api/v2/device/lastreported
    - Example request:
      - {"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6", "time": 123456789, "notify": true}

Geneva API Guidelines Proposal (Routes – cont’d)

- DELETE
  - Deleting an item by ID or Name requires unique endpoints for each. No dual-purposing of routes.

For Example

- I’ve tried to apply these principles to core-metadata
- https://github.com/tsconn23/edgex-geneva-api