|  |
| --- |
| OMR demo architecture PLANOM  Technical Specification |
| FORGEROCK 201 Mission St. Ste 2900  San Francisco, CA 94105 |



## HIMSS OMR Demo – Medical Devices

### Introduction

The OMR Demo was created by the Ecosystems Engineering Team to be presented at HIMSS 2018 in Las Vegas, March 5th to March 9th 2018.The purpose of the Demo was to showcase Forgerock’s Identity Platform capabilities to integrate with Health Care Related Cloud Services in order to collect data from Medical Devices and share the data with authorized parties only via UMA policies.

### Use Case: Nonin 3230 Pulse-Oximeter Data Collection

**Participants:**

Medical Provider (Doctor)

Consumer Patient (Patient)

QualcommLife (2Net SP)

Forgerock’s Identity Platform (FRIAMP)

Nonin 3230 Pulse-Oximeter (Device)

ACME Medical Provider Portal

Data Collection Mobile Application

**Scenarios:**

The Patient is prescribed a medical device as part of treatment for the patient’s condition. The Doctor assigns the Device to the patient’s account. The Doctor provides instructions to the Patient to sign in to the Provider’s Portal Web Application and ‘activate’ the prescribed device, and by doing so, the Device is enabled to collect data from the Patient and transmit it to a cloud service. The device is enabled only when the Patient agrees to the way in which data will be shared with the Medical Provider; this is accomplished by collecting the Patient’s Consent at Device Activation Time in the Provider’s Portal.

The Patient takes a reading with the Device. The process is as follows:

* The Patient pairs the Device via Bluetooth with his/her mobile phone.
* Upon successful pairing, the Patient opens the Data Collection Mobile Application on the phone and logs in using his/her user name and password.
* Once logged in to the Mobile Application, the Patient then clips the Device on a finger and waits for the green light indicator on the Mobile Application.
* This provides the indication to the Patient that a reading has been taken by the Device.
* The data read by the Device is sent to QualcommLife’s 2Net Services Platform (2Net SP), which in turn broadcasts the reading to a consumer service stood up by the Medical Provider’s Organization.
* Once the information is collected in the Medical Provider’s data stores (Implemented using FRIAMP), the Doctor can access the data on the ACME Medical Provider Portal.

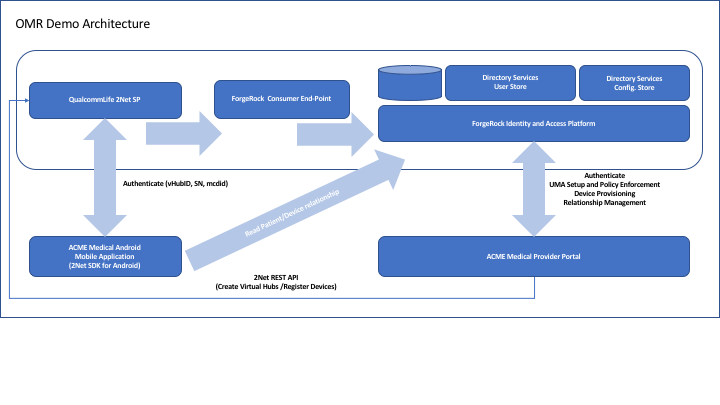
## OMR Demo Architecture

### Overview

The architecture of the Demo has the following layers:

* Applications Layer
  + Medical Provider Portal Web Application
  + Mobile Device Medical Application – Data Collector
* Cloud Services Layer
  + 2Net Cloud Services Platform
  + ACME Medical Data Consumer End-Point
  + ForgeRock Identity and Access Management APIs
    - Provisioning
    - Relationship Management
    - UMA
    - Authentication/Authorization
* Storage Layer
  + OpenIDM Data Store
  + OpenDS Directory Services – User Store
* Physical Layer
  + Medical Devices – 2Net Compatible
    - Nonin 3230 – Pulse Oximeter

The associations and dependencies amongst the components described above is shown in the diagram below:



As described in the picture above the ACME Medical Android Mobile Application communicates with the QualcommLife Cloud Services Platform using the Android SDK (Qualcomm Provides an IOS version of the SDK for iPad and iPhone applications).

The communication between the Mobile Application and the 2Net Cloud Services Platform has the purpose of authenticating the mobile device using the virtual hub associated to the Medical Device, an authentication code associated to the Device’s owner (Patient) and a digital signature for the Android Device being used to collect the data (MCDID) from the Medical Device (Patient’s Phone). Once the Device is authenticated, if someone else tries to login to the user’s account with a different Mobile Device, even if the credentials are compromised, this person won’t be able to connect and take any measurements with the associated Medical Device.

The ACME Medical Provider Portal establishes the relationship between a Patient and a Device by generating an Authentication Code and use it to create a Virtual Hub on 2Net Cloud Service Platform. The Portal uses REST APIs to communicate with 2Net CSP. This process needs to be completed before the data is collected and before the Doctor can get access to the Patient’s Device collected data.

When ACME Medical Provider Portal activates the Device, it also generates an UMA resource and an UMA policy giving access to the Doctor associated to the Patient. All these relationships are created and maintained in OpenIDM. Then this information is retrieved by the Mobile Application and the Portal when the Patient takes a reading and when the Doctor attempts to access a Patient’s data.

ForgeRock’s Identity and Access Management Platform also provides User Authentication and Authorization for Doctors and Patients.

## Implementation

### Demo Applications

#### ACME Medical Provider Portal

This is a Web Application developed using Grails Framework. It is written in Groovy. The source code is located under the OMRAdminPortal subdirectory of the GitHub repository.

The IDE used for Development is IntelliJ IDEA by JetBrains, so the whole project is available in the repository under the OMRAdminPortal subdirectory as well.

The application is based on MVC architecture and leverages Spring framework for configuration management and internal bean wiring (Dependency Injection) – for more information about the Spring Framework please refer to the following link: <https://spring.io/>

The following diagram shows the relationships between the Domain objects in the Architecture:



###### OMRDevice

This class represents the Medical Devices that Doctors prescribe to Patients. The fields significant to the implementation with 2Net SP are:

* macAddress: This is the BTLE address the Patient’s mobile phone gets at pairing time.
* devSerial: This is the Device’s Serial Number which is tied to the MAC Address above
* vHubId: This is a unique identifier assigned by 2Net SP, and it represents the Patient’s Mobile Phone serving as a Virtual HUB. This is the link between the Medical Device and the cloud service exposed by 2Net SP.
* modelName: This is the Manufacturer Model of the Medical Device. This is also associated to the Device’s SN to make sure the type of device matches the provisioning record for the Device in 2Net SP’s Database.

The fields used for UMA grant flow implementation are the following:

* umaResourceID: This is the Id of the resource representing the Medical Device. This is used to create Resource Types and UMA Policies.
* umaROCredential: This is the Patient’s login credential which is used to obtain a PAT (Permission Access Token) used in the UMA Grant Flow to issue RPTs to Requesting Parties trying to access the Patient’s Medical Device data. For more information about UMA please review the following references: <https://docs.kantarainitiative.org/uma/rec-uma-core.html>.

###### OMRUser

This class is used to represent both Patients and Doctors. The difference is determined by the Roles that this user is a member of. Patients have no roles, Doctors do have roles. There are some attributes of this class that are used primarily during UMA grant flows, specifically the umaAccessToken. The umaAccessToken is the PAT for a Patient and the RPT for the Doctor.

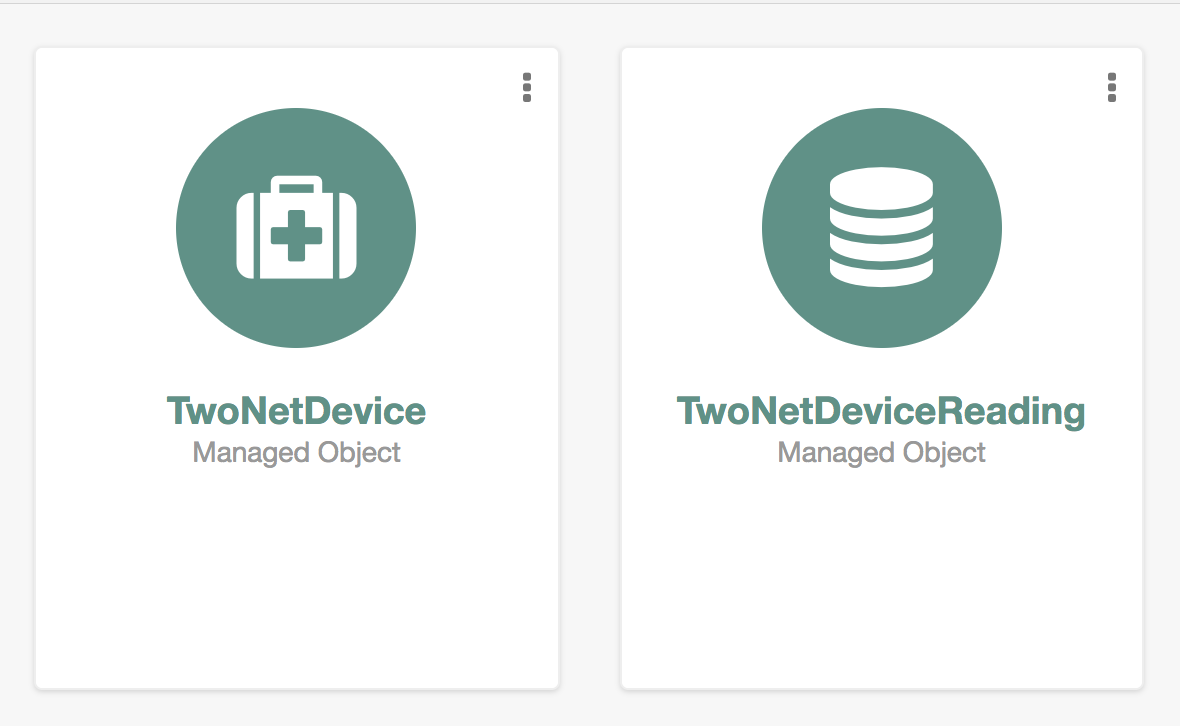
###### OMRReading

This class is used to represent a Medical Device Reading. Instances of this class are associated to a given device via the readings attribute in the OMRDevice class. The readings attribute is a list of instances of class OMRReading. The attribute that contains the information of the reading is a map of reading attributes. Each reading may contain multiple pieces of information, for instance: The pulse oximeter communicates two pieces of data: the Patient’s Heart Rate and the SP0 index (Oxygen Level). The best way to represent that was to store each individual piece of data as an entry in a Map object, that is why the devReading attribute is of type Map.

###### UMAResourceSet

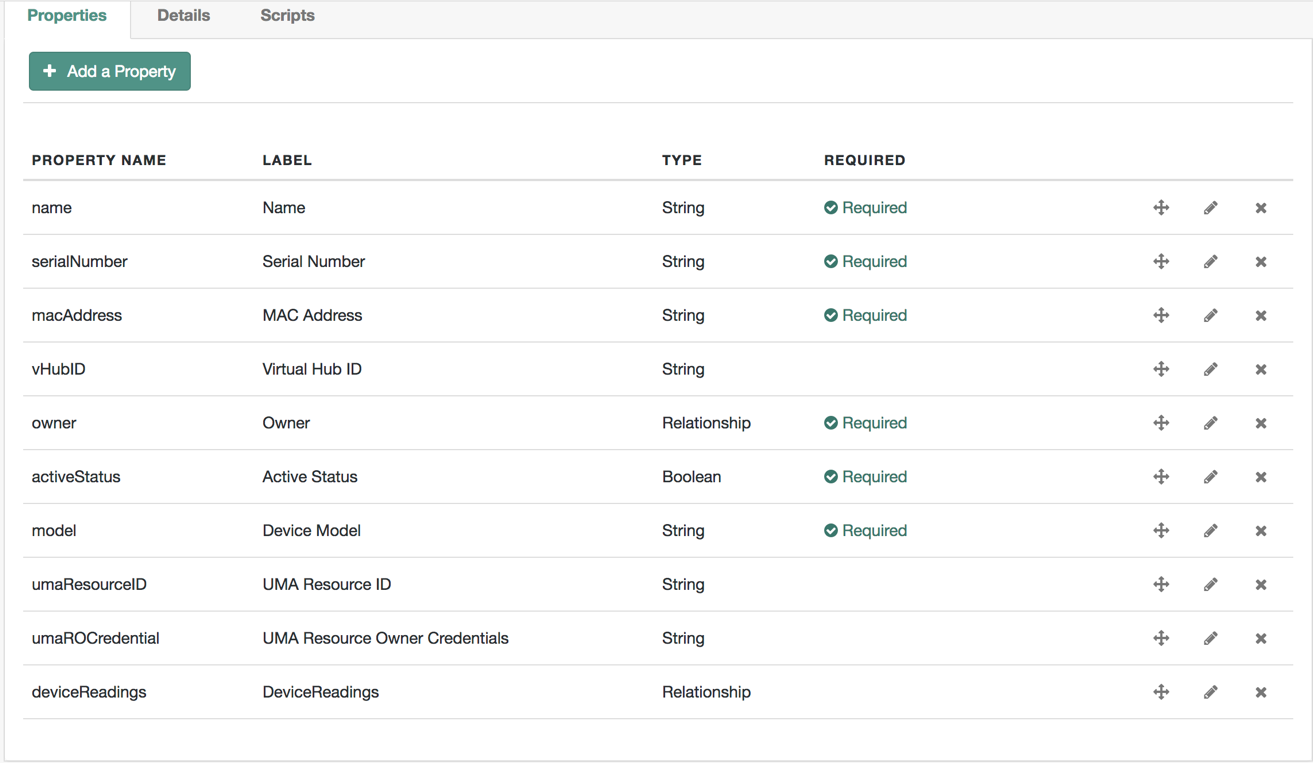
This class represents the Medical Device as an UMA Resource. This is used in the UMA Grant Flow to get access by the Doctor to the Patient’s Medical Device data. This is a very simple class which has just two attributes, and each instance of this class are associated to one and only one corresponding instance of a Medical Device (OMRDevice). The relationship is represented by attribute umaResourceID in class OMRDevice.

The Managed Objects in OpenIDM used in the OMR Demo are shown in the picture below:

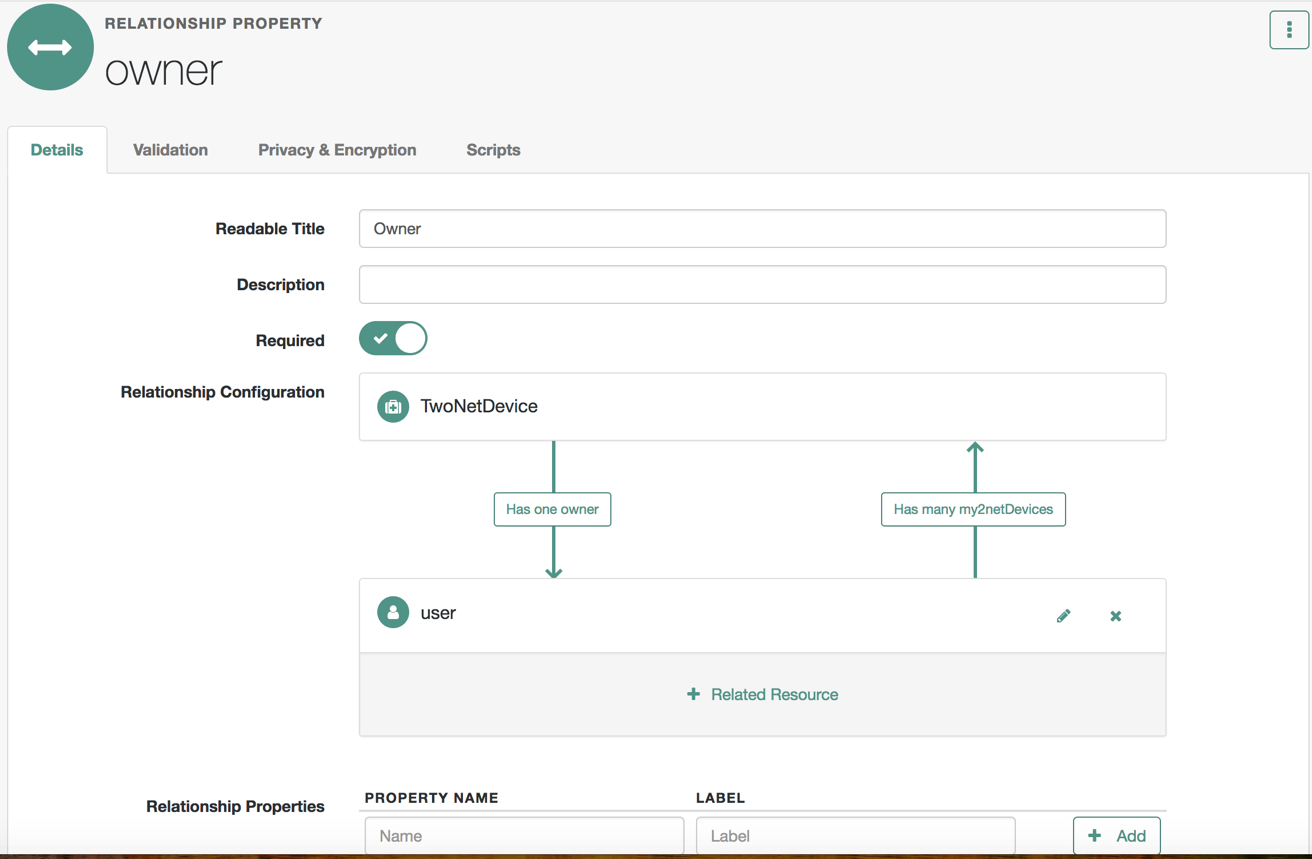


The following pictures show the attributes of each Managed Object:

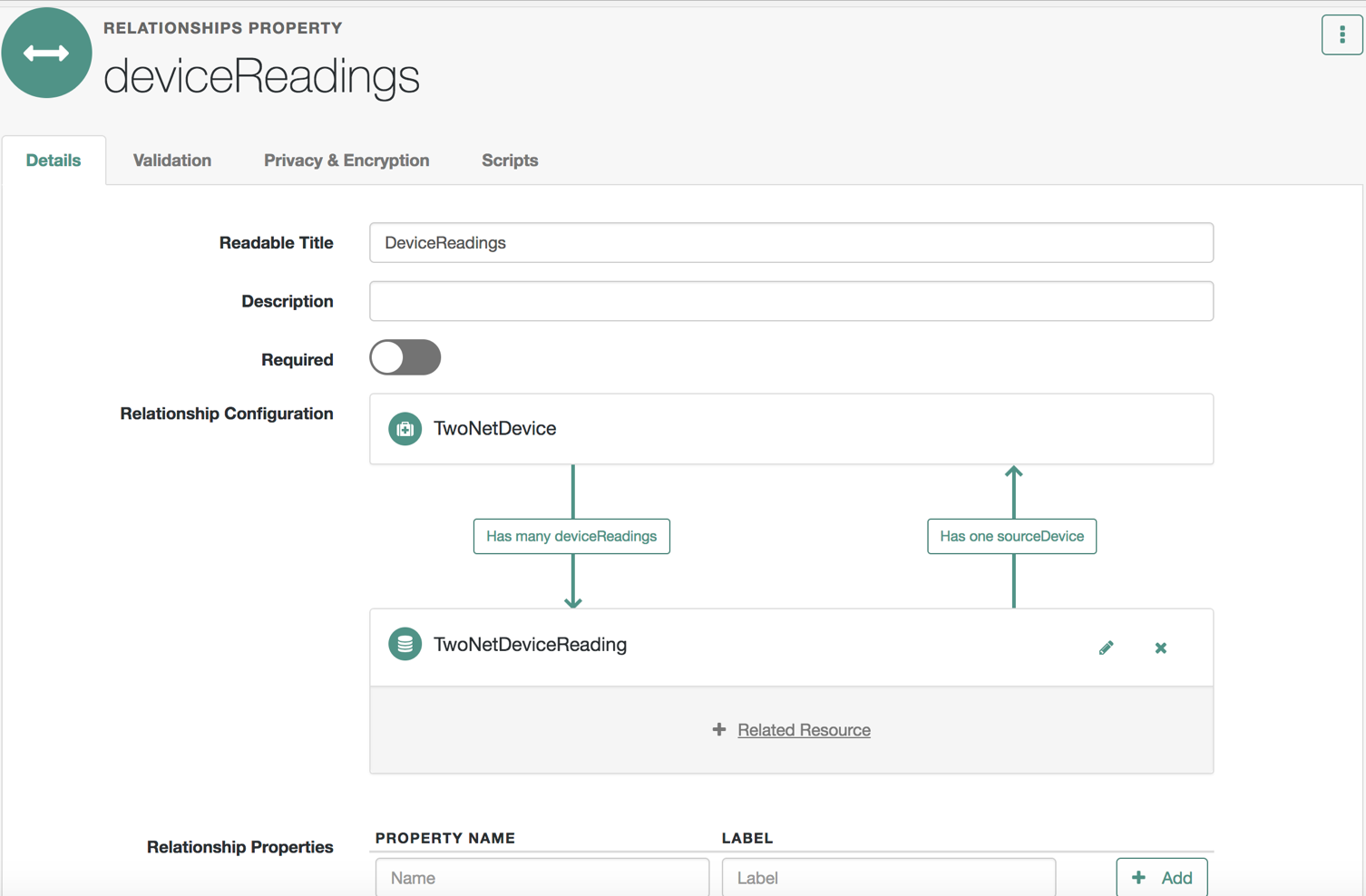
**TwoNetDevice**



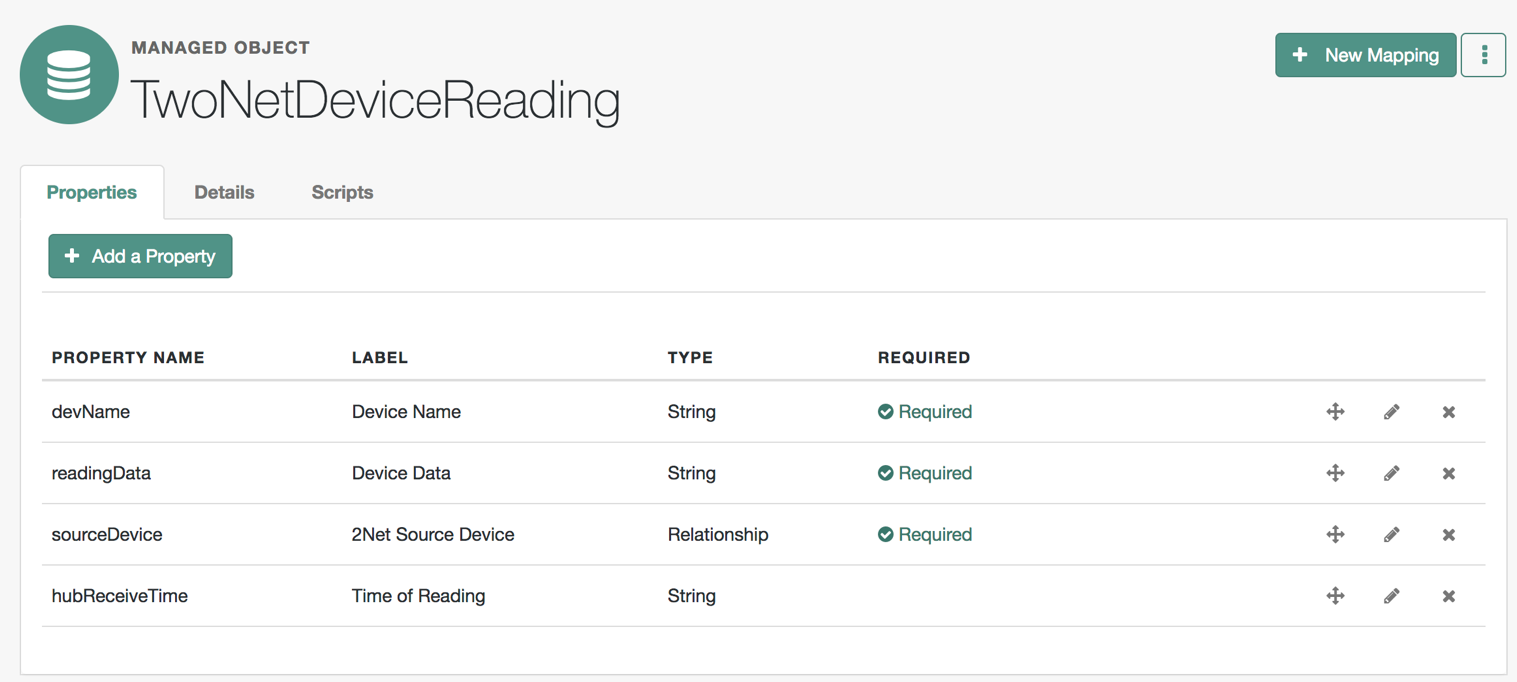
**Attribute owner Relationship Configuration**



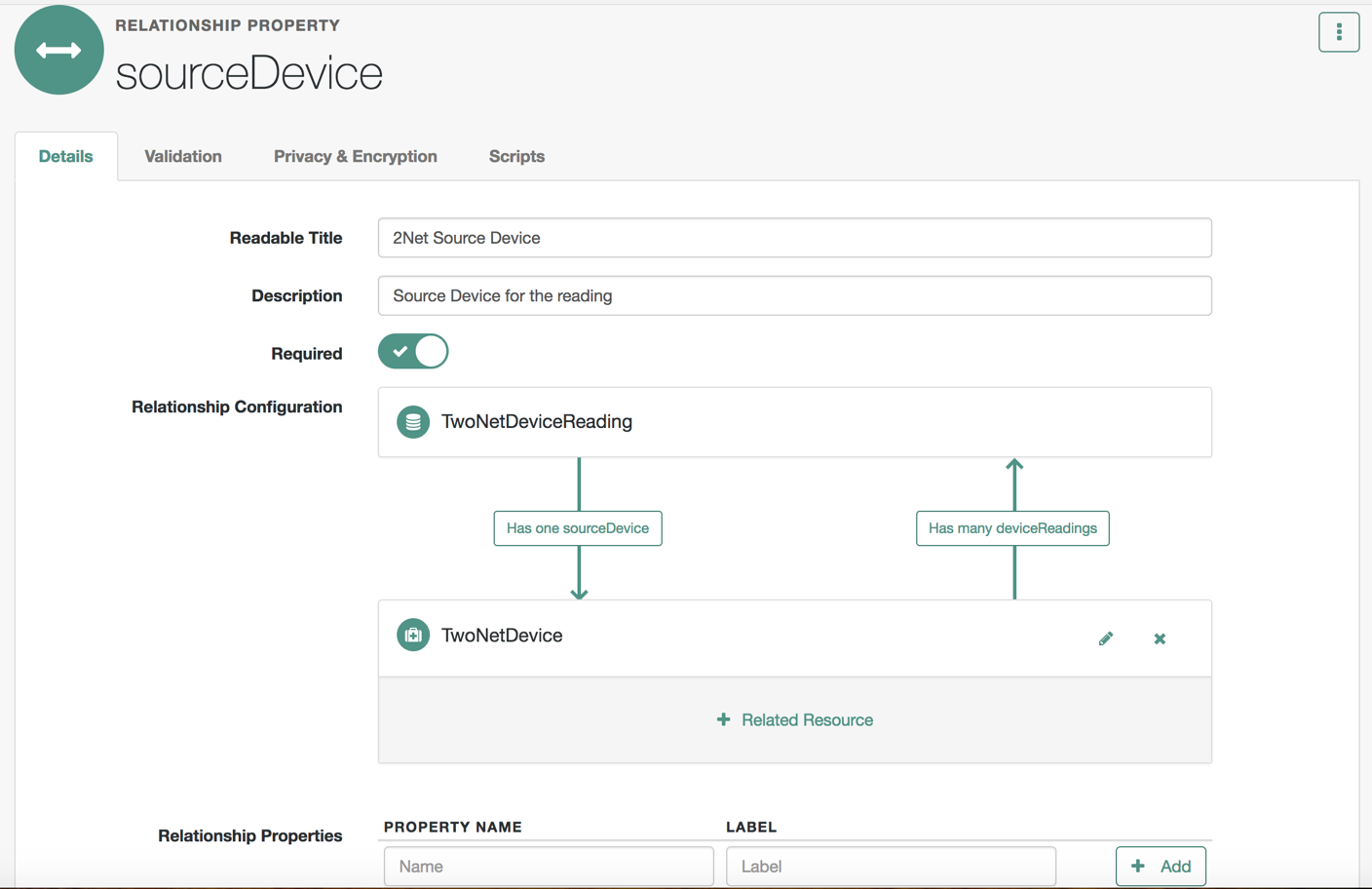
**Attribute devicesReadings Relationship Configuration**

****

**TwoNetDeviceReading**



**Attribute sourceDevice relationship configuration**



#### OMR Mobile Application

This is an Android Mobile Application which reads data from the Medical Device via Bluetooth. A pre-pairing of the Medical Device with the mobile phone of the Patient is required prior to using the application.

Technical Details

The Medical Devices have a BTLE MAC Address which is stored in the record created in OpenIDM. This MAC Address is one piece of data used in the Authentication process against 2Net SP.

The mobile application extracts the information from OpenIDM via a custom End-Point created for this purpose. The end-point URL is:

[https://openidm.aeet.fridam.aeet-forgerock.com/openidm/endpoint/twonetcommdata?\_queryFilter=userName%20eq%20'<userName>'](https://openidm.aeet.fridam.aeet-forgerock.com/openidm/endpoint/twonetcommdata?_queryFilter=userName%20eq%20'%3cuserName%3e')

The parameter userName is the login ID of the Patient signing in to the Mobile Application. The REST call is authenticated via the X-OpenIDM-Username and X-OpenIDM-Password headers using the Patient’s login credentials. These credentials are collected by the Mobile Application Login screen and passed to the function making the call to OpenIDM. The information returned is then used to Authenticate the Device against the 2Net SP.

The custom end-point returns the following information about the logged in user:

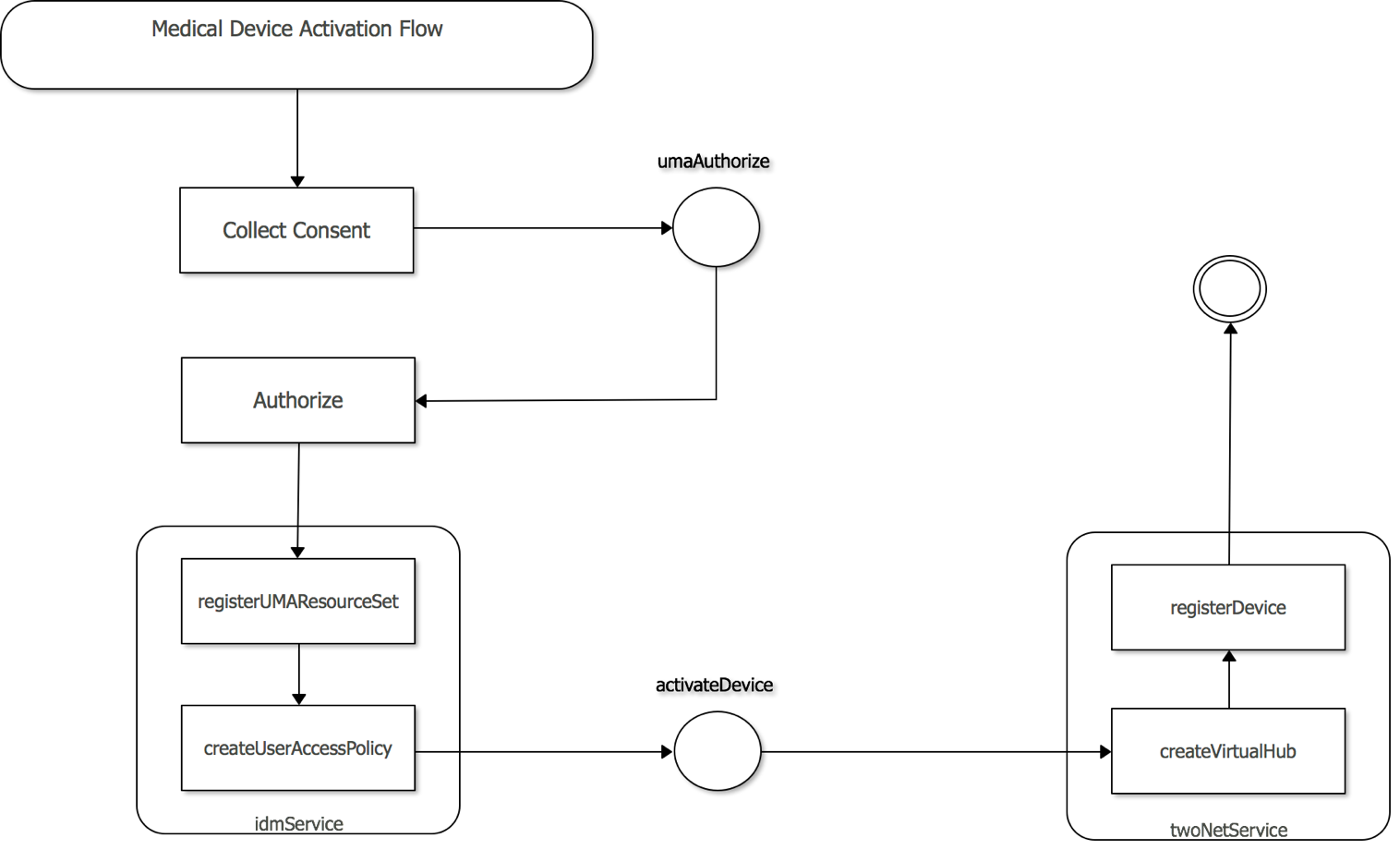
2Net Authentication Code: This is a random code generated when the Patient activates the assigned Medical Device. Then, it is stored in the Patient’s user record in OpenIDM.

List of Associated Medical Devices: A list of active devices associated to the Patient. This is maintained as a managed objects relationship in OpenIDM. Each Device returned will contain the attributes required for the authentication process against 2Net SP for that particular Device.

### UMA Configuration – OpenAM

The OMR Demo makes use of UMA features provided by OpenAM. These features are leveraged in the implementation of the Collection of Consent, which happens during the Medical Device activation flow.

The following diagram shows the Medical Device Activation Flow and the UMA configuration taking place:



*UMA Policy Setup Phase*

When the Patient activates the Medical Device the first step is to collect the Patient’s consent. Upon granting consent, the Authorize action is started. This makes use of the idmService by invoking methods registerUMAResourceSet and createUserAccessPolicy.

The registerUMAResourceSet method invokes OpenAM’s /openam/uma/resource\_set and sets the following payload as a POST:

{

“resource\_scopes” : [ “receive” ],

“name” : “OMRUMAResource<Patient Login ID><Medical Device SN>,

“type” : “omrDevice”,

“url” : “http://aeet-apps.fridam.aeet-forgerock.com/<patient login id>/<device SN>

}

This creates a Resource Type, a Resource Object and an empty subjects Access Policy. The User Access Policy for this UMA Resource can be updated later with actual subjects.

The subjects for the UMA Access Policy are setup by method createUserAccessPolicy which invokes the openam/json/users/<Patient’s Login ID>/uma/policies/<Policy ID> end-point with the following payload as a PUT request:

{

“policyId” : “Policy ID”,

“permissions” : [

{

“subject” : “<Doctor’s Login ID>”,

“scopes” : [“receive”]

}

]

}

**Note: Something to highlight here is the fact that the permissions are being added using a PUT. This is going to replace the Subjects with a new list every call made to the end point. If this is not the behavior you need (i.e. you need to incrementally add Subjects to the list), then you can use PATCH instead of PUT.**

In this particular implementation Patients only have one medical assignee. If you choose to implement multiple medical assignees or maybe physician referrals where the consent to share with additional parties is required to be collected prior to granting access, OpenAM APIs allow for a creation of a Consent Request Ticket that the Patient Approves/Rejects to Grant/Deny consent to the new requesting parties. This is described in the actual UMA grant flow when a Doctor tries to get access to the Patient’s Medical Device data.

Also, additional actions are taken during the Activation Process UMA Policy Setup phase, see the list below:

* The umaROCredential field of TwoNetDevice is also populated with the Patient’s password which is collected when the Patient successfully logs in to the OMR Provider Portal Web Application. This is needed for the UMA Grant Flow which requires a PAT, which is an Access Token retrieved via the OAuth Password Grant type, therefore, it requires the Patient’s (Resource Owner) credentials, and the Patient is not the one signed in at the time UMA Grant flow is executed, it is the Doctor.

**Note: For simplicity purposes, there is no encryption for the umaROCredential field. This may come in the future and you may decide to modify this as part of your implementation or extensions.**

*2Net SP – Registration phase*

This phase starts with the invocation of the registerDevice action of the PortalController. The source code for the registerDevice action is shown below:

**def** activateDevice() {  
 String umaResourceID = params.**umaResourceID** OMRDevice selectedDevice = findSelectedDevice()  
 String charset = ((**'A'**..**'Z'**) + (**'0'**..**'9'**)).join()  
 Integer length = 15  
 String authCode = RandomStringUtils.*random*(length, charset.toCharArray())  
 String activationDate = **new** Date().format(**"yyyy-MM-dd'T'HH:mm:ss.SSSZ"**, TimeZone.*getTimeZone*(**"UTC"**))  
 String finalDate = **new** StringBuilder(activationDate).insert(activationDate.length()-2, **':'**).toString()  
  
 String vHubId = **twoNetService**.createVirtualHub(authCode, finalDate)  
 println **"The Virtual Hub with ID** ${vHubId} **has been created."  
 twoNetService**.registerDevice(vHubId, selectedDevice.devSerial, selectedDevice.modelName, selectedDevice.macAddress, **"BTLE"**)  
 *// After success modify user with Authentication Code and set deviceActiveStatus to true* **securityContext**.loggedInUser = **idmService**.update2NetInfo(umaResourceID, **securityContext**.loggedInUser.loginId, **securityContext**.loggedInUser.idmId, vHubId, selectedDevice.idmId, authCode, **true**)  
 render(**view**: **"devices"**, **model**: [**securityContext**: **securityContext**, **uiConfig**: **uiConfig**])  
  
  
}

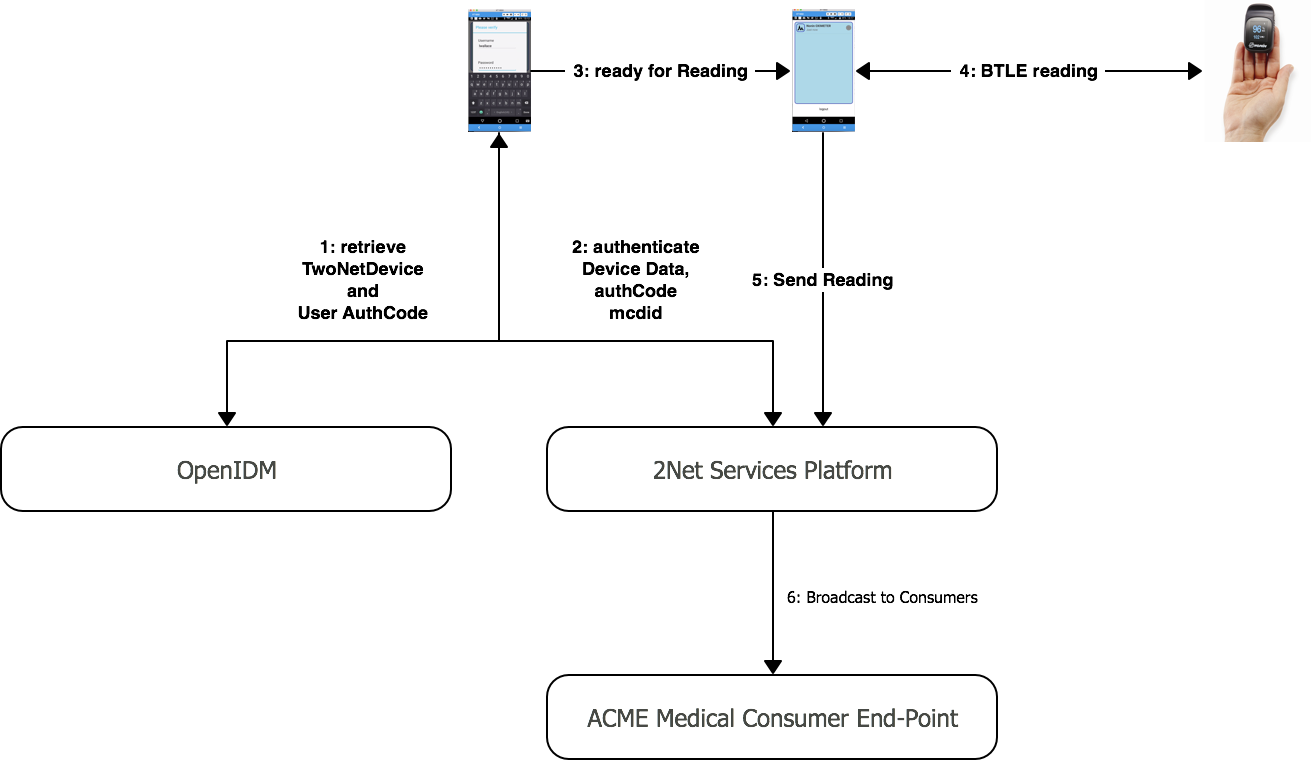
There are a few things happening before invoking the 2Net Services via the twoNetService API wrapper, see below:

* The Medical Device managed object entry is retrieved from OpenIDM using the PortalController’s protected method findSelectedDevice().
* An authentication code is generated using a Random String generator which generates a 15 character string using letters and numbers.
* The twoNetService createVirtualHub method is invoked using the authentication code generated above and the activation date. The resulting vHubId is stored for later.
* The twoNetService registerDevice method is called passing the selected device information (SN, MAC, Model and Interface Type) and the virtual Hub ID retrieved previously. What this means is that the Device is now trusted and the virtual hub used to send the actual readings to the Cloud is ready to go through the authentication process (described later).
* Some attributes of the Medical Device managed object entry in OpenIDM are updated: vHubID and umaResourceID.
* The Patient’s user profile attribute authCode2Net is also updated with the authentication code previously generated.

Once all the steps have been executed successfully, the Patient can use the ACME Medical OMR Mobile Application to take Device readings and send them to the 2Net SP Cloud Service, which in turn will broadcast the data to ACME Medical’s 2Net Consumer End-Point.

### Taking a Reading

The following diagram shows the process of taking a reading with the actual Medical Device:



The process of taking a reading is fairly simple. The Medical Device needs to be paired with the Patient’s mobile device prior to using the Mobile Application to take the reading (This process is not shown in the picture above).

1. Patient logs in to the Mobile Application – see login screen on the diagram above. The mobile application calls a custom end-point in OpenIDM to retrieve the Device Information and the Patient’s Authentication Code (authCode2Net) attribute.
2. Mobile Application tries to authenticate the Device against the 2Net Services Platform to match the vHubID and bind the MCDID of the Patient’s Phone to the Virtual Hub ID to which the Medical Device is registered. This protects from hacking.
3. The Mobile Application is ready to read data from the Oximeter (Medical Device).
4. The Pulse-Oximeter sends the data to the Mobile Application via BTLE.
5. The Mobile Application uses the 2Net Android SDK to submit the information to the QualcommLife Cloud Service.
6. The 2Net SP broadcasts the data to the subscribed consumers. The consumers are Web End-Points that are registered by 2Net Administrators to be linked to Customer Accounts. Part of the requirements to execute this demo with your own devices is to obtain a developer’s account on QualcommLife. The APIs used in the OMR Provider Portal require an API Key supplied as part of the provisioning of a Developer Account.