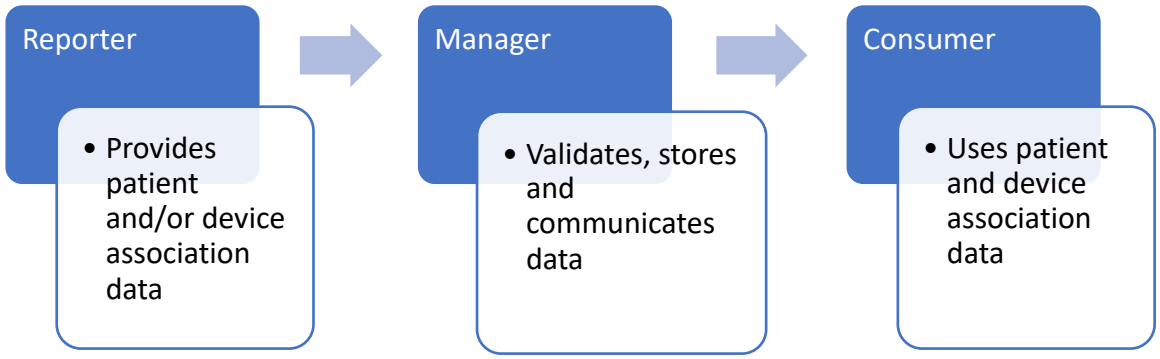
InnoVision Medical

19 South Street,
Baltimore, MD 21202

(443) 992-7450

Implementing Point of Care Identity Management (PCIM) as a Data Consumer

PCIM requires each message sent between devices and the EMR to include patient device association data. There are three parties to the PCIM protocol:



Medical devices can be both consumers and reporters. EMR systems can be reporters, managers, and/or consumers. Other consumers include alarm management systems and medical device or procedure information management systems. Other reporters include RTLS systems.

The following summarizes what the InnoVision Medical Patient Association Library has to offer the PCIM consumer medical device software developer.

Getting Started with the Patient Association Library

A PCIM manager is an actor that maintains and distributes an up-to-date store of device-patient associations. The API supports both a snapshot and a real-time association mode. The snapshot mode is as simple as sending a query for associations to a PCIM manager as needed by your implementation. The real-time mode creates a subscription to the PCIM manager and receives the latest associations on an interval.

Setting up to be a consumer of PCIM is straight-forward and described here. The following code demonstrates how to set up device-patient associations for your medical device in real-time and snapshot modes.

Initialization Phase

1. Create a *DeviceProfile* to represent your medical device in the device-patient association

```
DeviceProfile profile {};
profile.setEui64(EUI64Utils::createEUI64("11:22:33:44:55:66"));
profile.setSendingApp("InnoVision Consumer");
profile.setSendingFacility("InnoVision Facility");
profile.setModel("Test Model");
profile.setModelManufacturer("Test Manufacturer");
profile.setSerialNumber("123456789");
```

2. Create the *DevicePatientAssociationConsumer* that will be used to interact with the PCIM manager. The consumer takes the profile we created in step one as well as a host and port for the manager to communicate with.

```
DevicePatientAssociationConsumer dpac { profile, host, port };
```

Snapshot Mode:

1. The *DevicePatientAssociationConsumer* we created in the initialization phase can be used to query the manager for associations. If a specific query is not specified, then it will create a query based on the *DeviceProfile* we gave in the initialization phase.

```
std::vector<DevicePatientAssociation> initialAssociations = dpac.queryAssociations();
```

2. A specific query can be created and passed to the *queryAssociations* method on the *DevicePatientAssociationConsumer*.

```
std::string valueToFilterBy = "PID-ID-123456";
DevicePatientAssociationFilter filter0 { FILTER_BY_PID_ID, valueToFilterBy,
FILTER_OPERATOR_EQ };

std::vector<DevicePatientAssociationFilter> filters = { filter0 };

DevicePatientAssociationQuery query { "QUERY_TAG_001", filters };

std::vector<DevicePatientAssociation> associations = dpac.queryAssociations(query);
```

The vector of associations contains all the association events that match the query. This includes both active associations (associations without an end time). The *DevicePatientAssociation* class has a *DeviceProfile* to represent the device part of the association and a *Patient* to represent the patient part of the association. It also has the associate time and the disassociate time (if the latter is available).

Real-Time Mode:

1. Real-Time mode requires a handler function *(RealTimeAssociationHandler*) to be defined as all associations will be received asynchronously. This is an example handler function for incoming associations. Its purpose is to log out the details of the association events it receives. The *associationsHandler* takes a list of association events that include both associations and disassociations. The *isAssociated* method reveals which type of event it is.

```
void
associationsHandler(const std::vector<DevicePatientAssociation>& associationsIn)
{
  for(const DevicePatientAssociation& a : associationsIn) {
    if(a.isAssociated()) {
      Logger::getLogger()->log("Device-Patient Association Received: " +
        a.getPatient().getPatientID().getMrn() + " -> " +
        a.getDeviceProfile().getEui64() + " at " +
        a.getAssociationTime().getDateTime());
    }
    else {
      Logger::getLogger()->log("Device-Patient Disassociation Received: " +
        a.getPatient().getPatientID().getMrn() + " -> " +
        a.getDeviceProfile().getEui64() + " at " +
        a.getDisassociateTime().getDateTime());
    }
  }
}
```

2. Real-Time mode is enabled by starting it on the *DevicePatientAssociationConsumer*. Starting real-time mode requires an asynchronous association handler (*RealTimeAssociationHandler*). The *startRealTime* method sends a Device-Patient Subscription query based on the *DeviceProfile*. A custom *DevicePatientAssociationQuery* can also be used if desired.

```
  RealTimeAssociationHandler handler = &associationsHandler;

  isRunning = dpac.startRealTime(handler);
```

3. Real-Time mode can be disabled by calling the stop method on the *DevicePatientAssociationConsumer*.

```
  dpac.stopRealTime();
```