

How to create an Agent in IEEE-11073-PHD

October 22, 2012

1 Introduction

This document gives you a brief idea about how we created an IEEE-11073 compatible agent following the specification. The example code for the agents which we have created comes packaged with the Open Health Connector(OHC) framework. This framework which is based on publish-subscribe design pattern provides a plug and play platform for the agent which can register its capabilities with the EventHandler to subscribe for the specific data types. When the OHC platform receives the data corresponding to the registered capabilities, it publishes the received data to the agent who has subscribed its capabilities.

Outline The remainder of this article is organized as follows. Section 2 gives account of the IEEE-11073-PHD protocol and its motivation. We have provided with how to read and interpret the specification for the Pulse Oximeter Agent based on IEEE-11073 protocol in Section 3. Finally, Section 4 we give directions about how we have created the agent based on Java.

2 IEEE 11073- PHD Specification

ISO and IEEE 11073 standards enable communication between medical devices and external computer systems. This standard and corresponding IEEE 11073-104zz standards address a need for a simplified and optimized communication approach for personal health devices, which may or may not

be regulated devices. These standards align with, and draw upon, the existing clinically focused standards to provide easy management of data from either a clinical or personal health device.

Agents (e.g., blood pressure monitors, weighing scales, and pedometers) collect information about a person (or persons) and transfer the information to a manager (e.g., cell phone, health appliance, or personal computer) for collection, display, and possible later transmission. The manager may also forward the data to remote support services for further analysis. The information is available from a range of domains including disease management, health and fitness, or aging independently applications.

The communication path between agent and manager is assumed to be a logical point-to-point connection. Generally, an agent communicates with a single manager at any point in time. A manager may communicate with multiple agents simultaneously using separate point-to-point connections. The primary concentration is the interface and data exchange between the agents and manager. However, this interface cannot be created in isolation by ignoring the remainder of the solution space. Remaining cognizant of the entire system helps to ensure that data can reasonably move from the agents all the way to the remote support services when necessary. This path may include converting the data format, exchange protocols, and transport protocols across different interfaces. Much of the standardization effort is outside of the scope of the Personal Health Devices Working Group; however, aligning all standardization efforts allows data to flow seamlessly through the overall set of systems.

The application layers are, for the most part, not specific to any particular transport. Where necessary, this standard identifies assumptions that require direct support by a transport or a shim layer above the transport. This approach allows support for various transports. The definition of the transports is outside the scope of this standard and the working group. Above the transport layer is the Optimized Exchange Protocol (described in the IEEE 11073 standard). This protocol consists of two aspects: the application layer services and the definition of the data exchange protocol between agents and managers. The application layer services provide the protocol for connection management and reliable transfer of actions and data between agent and manager. The data exchange protocol defines the commands, agent configuration information, data format, and overall protocol. The Op-

timized Exchange Protocol provides the basis to support any type of agent. For a specific device type, the reader is directed to the device specialization for that agent to understand the capabilities of the device and its implementation according to this standard. The device specialization indicates which aspects of this standard to comprehend and where further information to implement the device is found.

Above the exchange protocol are device specializations that describe specific details relative to the particular agent (e.g., blood pressure monitor, weighing scale, or pedometer). The specializations describe the details of how these agents work and act as a detailed description for creating a specific type of agent. Additionally, they provide reference to a related standard for further details. The standard numbers reserved for device specializations range from IEEE Std 11073-10401 through IEEE Std 11073-10499, inclusive. When the collection of standards is being referenced, the term IEEE 11073-104zz is used where zz could be any number in the range from 01 to 99, inclusive.

Some device specializations describe broad categories of device types (e.g., the IEEE Std 11073-10441 models device types that promote cardiovascular activity such as step counters or exercise cycles). Other device specializations have a narrow focus on a single device type (e.g., IEEE Std 11073-10408 models thermometers). Specializations that address one or define more than one device types may also define profiles. A profile further constrains the model defined in a specialization to increase interoperability (e.g., the step counter profile utilizes a limited portion of the IEEE Std 11073-10441). The ISO/IEEE P11073-00103 [B8] technical report describes the overall personal health space with further definition of the underlying use cases and usage models.

This standard imports information from ISO/IEEE 11073-10201:2004 [B13] and ISO/IEEE 11073-20101:2004 [B14] as normative annexes. If there is a discrepancy between these standards, this standard takes priority. Because of the reuse of constructs from these standards, some of the names appear to be more clinically focused [e.g., medical device system (MDS) instead of personal health device system]; however, to maintain consistency, the traditional names have been preserved.

3 Pulse Oximeter using IEEE 11073-PHD

3.1 Pulse Oximeter Device Specializations

In the context of personal health devices in the ISO/IEEE 11073 family of standards, a pulse oximeter, also called an oximeter, provides a noninvasive estimate of functional oxygen of arterial haemoglobin (SpO₂) from a light signal interacting with tissue, by using the time-dependent changes in tissue optical properties that occur with pulsatile blood flow. Applying the Beer-Lambert law of light absorption through such an arterial network, the fraction of oxygenation of arterial haemoglobin can be estimated. This estimate, normally expressed as a percentage by multiplying that fraction by 100, is known as SpO₂. Occasionally, this estimate may be referenced as information applicable to pulse oximetry.

3.2 Device Types

Pulse oximeter systems with applicability in the personal health space may take on a variety of configurations and sensor compositions, and their configurations have suitability in different personal health application spaces. Pulse oximeter equipment comprises a pulse oximeter monitor, a pulse oximeter probe, and a probe cable extender, if provided. Some oximeters are all-in-one assemblies, where the optical probe, processing, and display components are in a single package. Other oximeters may consist of separate sensor and processing/display components. Still others may place the sensor and signal processing in one component, and send that information into an external component for display and storage. In addition, other configurations may add storage capability into the system. This implies that different information models may be best suited for each particular device configuration.

3.3 Device Configurations

Although agents typically have a static configuration, it is permissible and desirable for an agent to support multiple configurations, one of which would be active at any given time. Pulse oximeters may have a rich set of features that can be combined into a collection of different configurations, one of which can be selected by the manager during configuration. Two general categories of configurations exist. The first category is known as the set of

standard configurations. These are intended to describe a relatively limited feature set of a single device specialization, which have predefined configuration ID codes. Managers may be pre-loaded with these configurations, in which case the configuration process is eliminated and immediate operation is allowed. The second category involves the set of extended configurations. These configurations are more flexible in that they may include concepts particular to one or more device specializations or include other features as defined in this standard

4 How to create a simple PulseOximeter Agent?

The process of creating an agent involves three steps of :

4.1 Association

1. The version of the association procedure used by the agent shall be set to `assoc-version1` (i.e., `assocversion = 0x80000000`).
2. The `DataProtoList` structure element of the data protocol identifier shall be set to `data-proto-id-20601` (i.e., `data-proto-id = 0x5079`).
3. The `data-proto-info` field shall contain a `PhdAssociationInformation` structure, which shall contain the following parameter values
4. The version of the data exchange protocol shall be set to `protocol-version1` (i.e., `protocol-version = 0x80000000`).
5. At least the MDER shall be supported (i.e., `encoding-rules = 0x8000`).
6. The version of the nomenclature used shall be set to `nom-version1` (i.e., `nomenclature-version = 0x80000000`).
7. The `functional-units` field may have the test association bits set, but shall not have any other bits set.
8. The `system-type` field shall be set to `sys-type-agent` (i.e., `system-type = 0x00800000`).

9. The system-id field shall be set to the value of the System-Id attribute of the MDS object of the agent. The manager may use this field to determine the identity of the pulse oximeter with which it is associating and, optionally, to implement a simple access restriction policy.
10. The dev-config-id field shall be set to the value of the Dev-Configuration-Id attribute of the MDS object of the agent.

4.2 Configuration

4.2.1 General

The agent enters the Configuring state if it receives an Association Response of accepted-unknown-config. In this case, the configuration procedure as specified in IEEE Std 11073-20601-2008 shall be followed. The following subclauses specify the configuration notification and response messages for a pulse oximetry agent with standard configuration ID 0x0190. Normally, a manager would already know the standard configuration. However, for the purposes of our example, it does not.

4.2.2 Standard Configuration

The agent performs the configuration procedure using a Remote Operation Invoke — Confirmed Event Report message with an mdc config event to send its configuration to the manager (see IEEE Std 11073-20601-2008). The ConfigReport structure is used for the event-info field.

4.3 Operation

4.3.1 General

Measurement data and status information are communicated from the pulse oximetry agent during the Operating state. If not stated otherwise, the operating procedure for a pulse oximetry agent of this standard shall be as specified in IEEE Std 11073-20601-2008.

4.3.2 GET pulse oximeter MDS attributes

If the manager leaves the attribute-id-list field in the roiv-cmip-get service message empty, the pulse oximetry agent shall respond with a rors-cmip-get

service message in which the attribute-list contains a list of all implemented attributes of the MDS object. If the manager requests specific MDS object attributes, indicated by the elements in attribute-id-list, and the agent supports this capability, the pulse oximetry agent shall respond with a rors-cnip-get service message in which the attribute-list contains a list of the requested attributes of the MDS object that are implemented. It is not required for a pulse oximetry agent to support this capability. If this capability is not implemented, the pulse oximetry agent shall respond with a Remote Operation Error Result (roer) service message (see IEEE Std 11073-20601-2008) with the error-value field set to no-such-action(9).

4.3.3 Measurement Data Transmission

Measurement data transfer for a pulse oximetry agent of this standard may be initiated by either the agent or the manager (see agent- and manager-initiated measurement data transmission in IEEE Std 11073- 20601-2008). To limit the amount of data being transported within an APDU, the pulse oximetry agent shall not include more than 25 temporarily stored measurements in a single event report. If more than 25 pending measurements are available for transmission, they may be sent either using multiple event reports or by incorporating a persistent store facility. If multiple oximetry measurements are available, up to 25 measurements should be transmitted within a single event report. Alternatively, they may be transmitted using a single event report for each oximetry measurement. However, the former strategy is recommended to reduce overall message size and power consumption.

5 How to read an IEEE specification for an agent and use the JAC compiler to get Java attribute classes?

1. Suppose we want to create a corresponding Java code for the MDS attribute of SystemModel which comes under the Configuration phase.
2. We start by referring to the IEEE 11073-PHD spec which gives us the ASN.1 data structure format.

3. We can see that it is a sequence of two Octet String type of data types.
4. Start by creating a text file containing the data structure for the SystemModel.
5. Now for every enclosed data structure which is not a primary data type in ASN.1 we will write the corresponding structures for every such secondary data structure, until it has been decomposed to a data structure which consists of primary data types.
6. Now compile the text file using the open-source JAC compiler to get the corresponding Java classes
7. Now we can start by putting values in the data fields of the created Java classes.
8. For whichever datatypes which are not primary, we will first need to create the objects for those secondary data fields and then initialize the data fields with these objects.
9. Now we have to add the switch statements for the corresponding types in the AVA Type class for creating the objects of the classes dynamically. So in case of SystemModel attribute which we are trying to implement we will add a switch statement for the corresponding ID which is 2344 in this case. These IDs can be searched in the nomenclature code for IEEE 11073-PHD.
10. They can then be assigned to the parent classes of primary data types like sequence, sequenceof or integer and then we cast to the specific classes while filling up the data structures in the main program.
11. This has been implemented because the JAC compiler does not support the ANY type of data structure which can be assigned any data type based on the nomenclature code.
12. We have only supported the attributes corresponding to the nomenclature codes which are used in the six agents which we have implemented.
13. For extending the support for other attributes, we recommend to refer the IEEE 11073 document and add the switch cases for corresponding data fields.

14. This technique to dynamically cast to the child classes of primary data types like Integer, sequence and sequenceOf is adopted for whatever datatypes which have this ANY data type defined as an enclosed field.
15. Now for fixing the byte output streams which inherently seem to be encoded in the BER format, which as we know is a superset of DER encoding rules, but it seems that the Continua Manager which we are using in order to verify that our agent is compatible with the IEEE 11073 standard, supports only DER encoding format.
16. So we have to fix the output streams for definite encoding in which we do not write the length of the written data field on the stream and just write the values.
17. The version of JAC compiler which we are providing with the Open Health Connector project has the primary data types fixed for the DER encoding formats.
18. Now for every data type of primary type sequenceOf we have to add some code to the BERConstruct class. We could have added this code for the parent class of SequenceOf but it seems that the IEEE 11073 specification does not use this code for every child class of SequenceOf data type.
19. So for every child SequenceOf class we will have to check whether we need to write the total length of the enclosed data types on the byte output stream and then add the corresponding code to write the total length in the writeElement method of the BerConstruct class of the JAC Compiler for case of SequenceOf.

6 Future work

Though we were successful in implementing the essential basic features of the IEEE 11073-PHD standard and create a compatible agent, the protocol is too intensive to be implemented as an example. So we have made an effort to solely implement only those attribute classes which are required by the agents which come packaged along with the OHC framework. Also, the agent maintains a finite state machine which is useful for reliable communication. This involves reading the sequence numbers of received responses from the

manager and keeping a track for them. But for the purposes of giving a simpler example code, we have stopped at implementing only the essential features of this FSM required to communicate with the manager. The reader who is interested in further extending the project to support a FSM can refer to the device specialization of the agent and use the directions given above.

References

1. IEEE 11073-PHD Standard
2. IEEE 11703-PHD specification for Pulse Oximeter