# Medical Connectivity Library

## Users Guide

Document Number:MEDCONLIBUG
Rev. 5
05/2012

## *How to Reach Us:*


**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

**For Literature Requests Only:**
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com








Document Number: MEDCONLIBUG
Rev. 5

# Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document.

| Revision Number | Revision Date | Description of Changes |
|---|---|---|
| Rev. 1 | 10/2009 | Initial release. |
| Rev. 2 | 04/2010 | Updated Figure 2-1, Figure 4-1, and Section A.1.1.1, “Software Setup.” |
| Rev. 3 | 06/2010 | • Added support for CFV2 devices |
| Rev. 4 | 07/2011 | • Updated images in Appendix A |
| Rev. 5 | 05/2012 | Added chapters<br>• IEEE 11073 Manager<br>• PHDC Manager Demo Applications |

## Chapter 1 Before You Begin



## Chapter 2 Getting Familiar



## Chapter 3 Medical Connectivity Library Architecture



## Chapter 4 Developing New Device Specialization



## Chapter 5 IEEE 11073 Manager

## Appendix A Working with the Software



## Appendix B PAN USB Agent Demo



## Appendix C PAN Serial Bridge Demo



## Appendix D PHDC Manager Demo Application

# Chapter 1 Before You Begin

## 1.1 About Medical Connectivity Library

The increased use of electronic devices in the medical domain has created a need for the development of common protocol for data interchange between various devices. This problem is addressed by IEEE-11073 Part 20601 (Optimized Exchange) specification. Freescale Medical Connectivity Library is based on this specification. This library follows Continua design guidelines for implementation.

Freescale Medical Connectivity Library allows customers to implement various device specializations defined under this standard without worrying about details of implementation of this protocol. The library is transport independent, allowing users to use different transport technologies like Ethernet, USB, and so on.

## 1.2 About This Book

This book describes the Freescale Medical Connectivity Library architecture and also explains you how to create customized applications using this library. Table 1-1 shows the summary of chapters included in this book.

**Table 1-1. MEDCONLIBUG Summary**

| Chapter Title | Description |
|---|---|
| Before you begin | This chapter provides the prerequisites of reading this book. |
| Getting Familiar | This chapter provides the information about the Freescale Medical Connectivity Library software suite. |
| Medical Connectivity Library Architecture | This chapter discusses the architecture design of Freescale Medical Connectivity Library software suite. |
| Developing New Device Specialization | This chapter discusses the steps a developer must take to develop new device specialization applications using Freescale Medical Connectivity Library. |
| Working with Software | This chapter provides information on how to build, run, and debug drivers and applications. |
| PAN USB Agent Demo | This chapter provides the setup and running PAN USB Agent demo for MC9S08JM60 and MCF51JM128 devices. |

## 1.3 Reference Material

Use this book in conjunction with:

- *Medical Connectivity Library API Reference Manual* (document MEDCONLIBAPIRM)
- *Freescale USB Stack with PHDC API Reference Manual* (document MEDUSBAPIRM)
- *Freescale USB Stack with PHDC Users Guide* (document MEDUSBUG)

For better understanding, also refer to the following documents.

- Continua Design Guidelines (document ContinuaV1_DG_HL7_R1.pdf)
- IEEE Std 11073-20601TM-2008, Health informatics — Personal health device communication — Part 20601: Application profile — Optimized Exchange Protocol.
- IEEE P11073-10441TM, Health informatics — Personal health device communication — Part 10441: Device specialization — Cardiovascular fitness and activity monitor.
- IEEE P11073-10442TM, Health informatics — Personal health device communication — Part 10442: Device specialization — Strength fitness equipment.
- IEEE Std 11073-10408TM, Health informatics — Personal health device communication — Part 10408: Device specialization — Thermometer.
- IEEE Std 11073-10415TM, Health informatics — Personal health device communication — Part 10415: Device specialization — Weighing scale.
- IEEE Std 11073-10471TM, Health informatics — Personal health device communication — Part 10471: Device specialization — Independent living activity hub.
- ISO/IEEE P11073-10404, Health informatics — Personal health device communication — Part 10404: Device specialization — Pulse oximeter.
- ISO/IEEE P11073-10407, Health informatics — Personal health device communication — Part 10407: Device specialization — Blood pressure monitor.
- ISO/IEEE P11073-10417, Health informatics — Personal health device communication — Part 10417: Device specialization — Glucose meter.
- ISO/IEEE 11073-10101, Health informatics — Point-of-care medical device communication — Part 10101: Nomenclature.
- ISO/IEEE 11073-10201:2004, Health informatics — Point-of-care medical device communication — Part 10201: Domain information model.

## 1.4 Acronyms and Abbreviations

| | |
|---|---|
| CFV1 | ColdFire V1 (MCF51JM128 CFV1 device is used in this document) |
| CFV2 | ColdFire V2 (MCF52221and MCF52259 CFV2 devices are used in this document) |
| DIM | Domain Information Model |
| IDE | Integrated Development Environment |
| IEEE | The Institute of Electrical and Electronics Engineers |
| ISO | International Organization for Standardization |
| JM60 | MC9S08JM60 Device |
| MDS | Medical Device System |
| PHD | Personal Healthcare Device |
| PHDC | Personal Healthcare Device Class |
| RTSA | Real Time Sample Array |
| TIL | Transport Independent Layer |
| USB | Universal Serial Bus |

## 1.5 Important Terms

Table 1-2 shows the terms used throughout the book.

**Table 1-2. Important Terms**

| Term | Description |
|---|---|
| Continua Alliance | This is a consortium of companies to establish standards for the medical segment devices. |
| DemoJM | This is the physical hardware where the expansion card with the silicon is mounted. |
| Expansion Card | This is the card where the silicon is embedded and can be loaded on to the hardware board. |
| Shim | Transport used for Data Send/Receive. |
| PM Store | Persistent Metric Store |
| PM Segment | Persistent Metric Segment |

# Chapter 2 Getting Familiar

## 2.1 Introduction

The Freescale Medical Connectivity Library software suite contains Medical Connectivity Library (designed based on IEEE 11073-20601 specification and Continua Design guideline), USB Transport Layer (based on USB-PHDC Specification), and a sample application. This section intends to help you develop an understanding of the Medical Connectivity Library and to assist you in developing more device specializations as defined by the standard. The document is targeted for firmware application developers who would like to develop the applications using this library.

## 2.2 Software Suite

This suite contains Freescale Medical Connectivity Library, USB Transport Layer, and a sample application for JM60, CFV1, and CFV2 devices.

## 2.3 Directory Structure

The software suite has a standard directory structure. You can extend it easily to accommodate more applications and Transport Layers.

The following figure shows the directory structure.

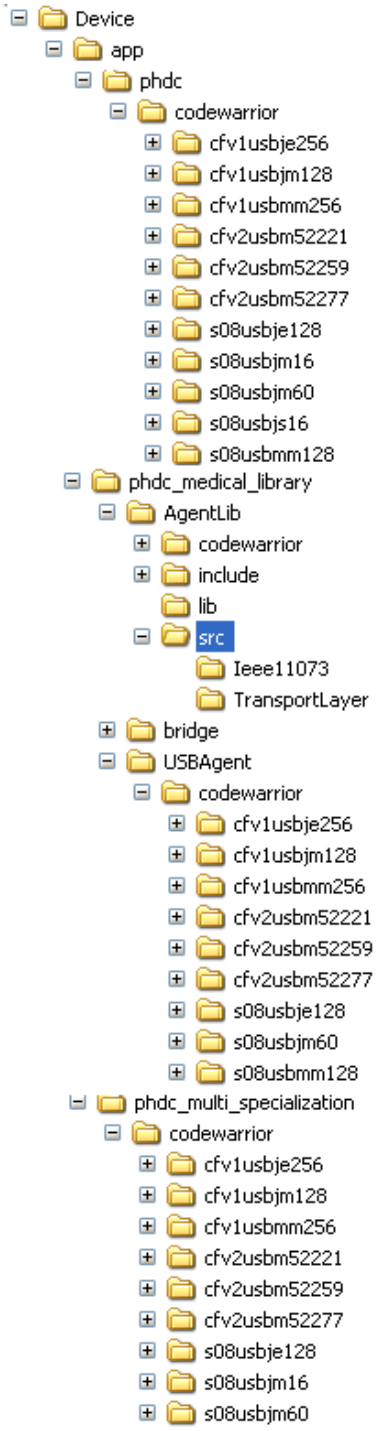


**Figure 2-1. Freescale Medical Connectivity Library Directory Structure**

# Chapter 3 Medical Connectivity Library Architecture

## 3.1 Architecture Overview

Figure 3-1 shows the Freescale Medical Connectivity Library architecture.

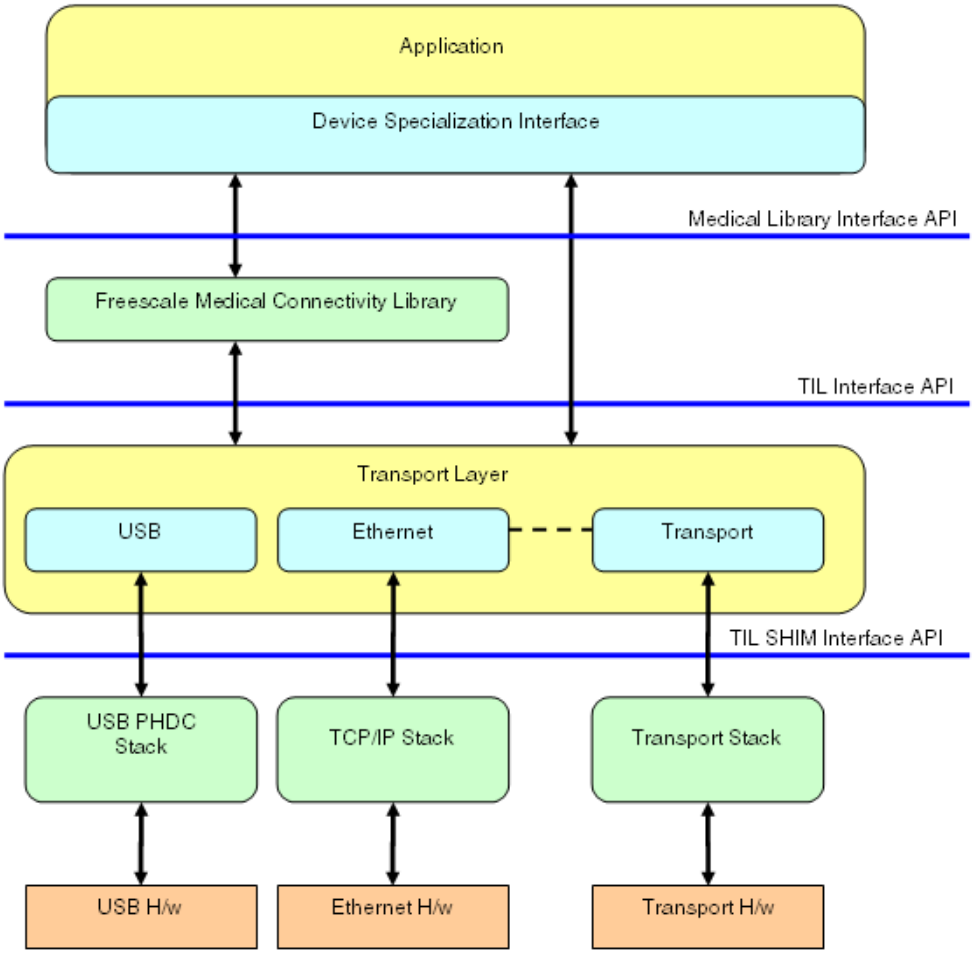


**Figure 3-1. Freescale USB Stack with PHDC Architecture**

Medical Connectivity Library is divided into two layers with application developed on top of them. The layered architecture helps the application developers concentrate on developing the application without being concerned about the other layers.

Transport Layer interacts with underlying Shim to provide reliable data communication to upper layer. This layer abstracts application and Medical Connectivity Library from details of how the data is sent or received through Shim. This layer interacts with underlying Shims with its defined interfaces, therefore allowing users to add different transport technologies as required.

Freescale Medical Connectivity Library Layer implements IEEE11073-20601 Standard (based on Continua guidelines). Freescale Medical Connectivity Library Layer defines interfaces for Device Specialization (part of Application Software) layer to communicate with Continua Manager. Device Specialization layer uses interfaces provided by this layer to send measurement and configuration data to Continua Manager.

As stated earlier, the layered architecture helps the application developers to develop applications. However, it does not limit the developer to interface lower layer APIs if they prefer to.

**CAUTION**

Simultaneous use of driver APIs and class APIs may have undefined behavior. In this case, the driver functionality will not work as defined in this document. Functioning of class drivers depends upon lower layer USB device controller state and USB bus state machine. If application invokes lower layer functions directly, then class driver state machine might get affected, leading to undefined behavior. However, this does not necessitates application to use class driver only. You can develop its application directly using lower layer driver APIs. In such a case, application needs to take care of class layer functionality.

## 3.2 Software Flows

This section describes the execution flow of the stack across various layers and modules.

### 3.2.1 Initialization Flow

Figure 3-2 describes application initialization flow.

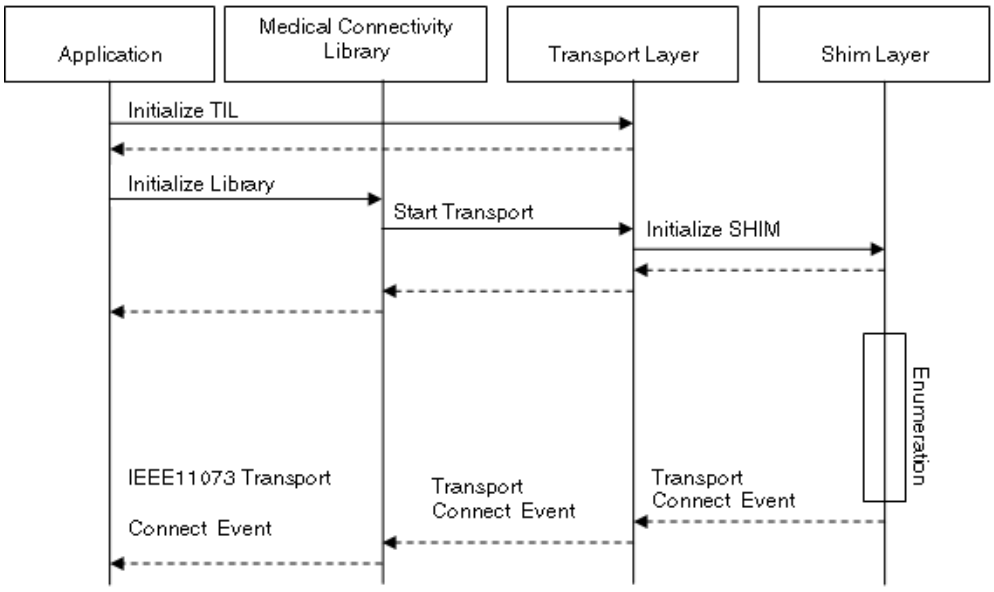


**Figure 3-2. Sequence for Library Initialization**

The application initialization sequence starts by initializing Transport Independent Layer (TIL). Using this interface, application registers various Shims that can be used as transport. After successful TIL initialization, application initializes Medical Connectivity Library with a callback and Shim Identification Code. This internally initializes Shim to send and receive data.

After Shim is initialized, enumeration process begins. After successful initialization, "Transport Connect" event is sent to TIL, which is passed to the Medical Connectivity Library and then to the application.

### 3.2.2 Association Flow

Figure 3-3 describes association process flow.

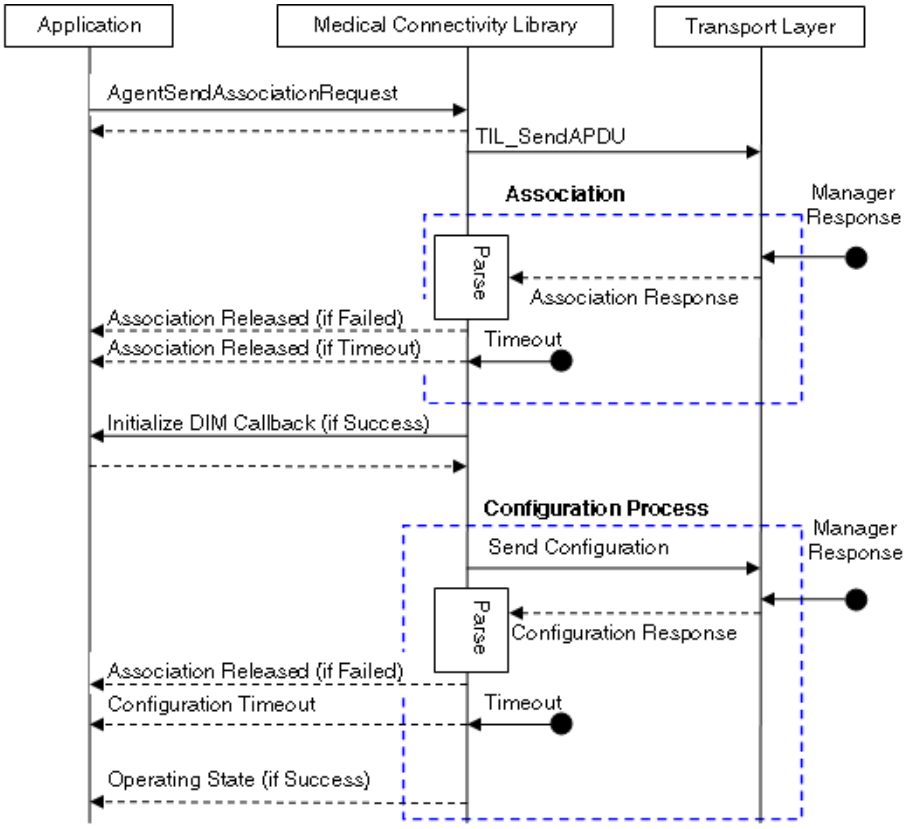


**Figure 3-3. Association Process Flow**

After receiving Transport Connect Event, application can start Association Process. Application uses AgentSendAssociationRequest interface to send Association Request to Continua Manager. If the request is successful, application receives a callback to initialize DIM. Once DIM is initialized, configuration process begins. This process is initiated by Medical Connectivity Library. After configuration process is successfully completed, application is given a callback Operating State. After this callback event is received, application can now send measurement data to Continua Manager.

### 3.2.3 Send Measurement Data Flow

Figure 3-4 and Figure 3-5 below describes how application can send confirmed/un-confirmed measurement data to Continua Manager.

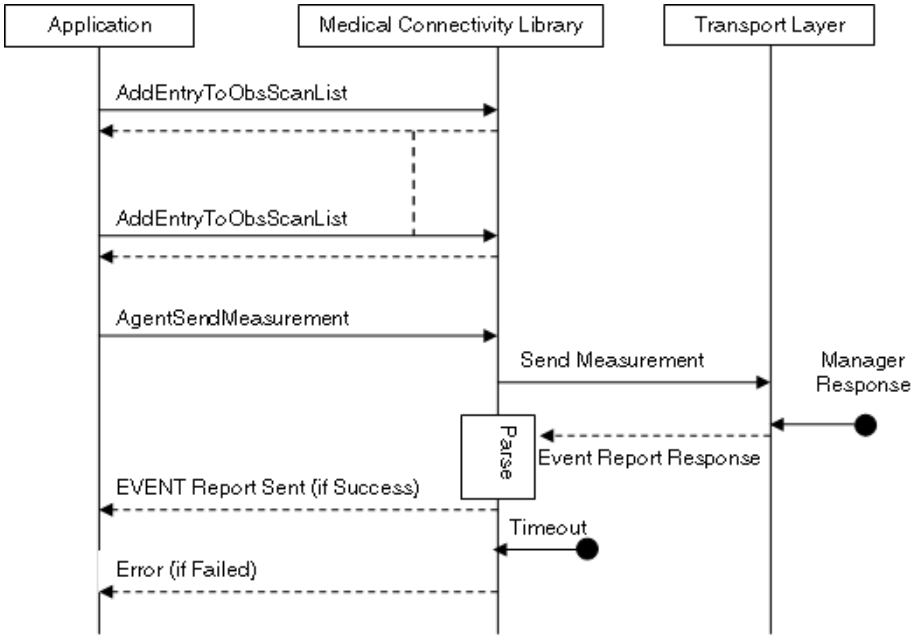


**Figure 3-4. Send Measurement Confirmed Flow**

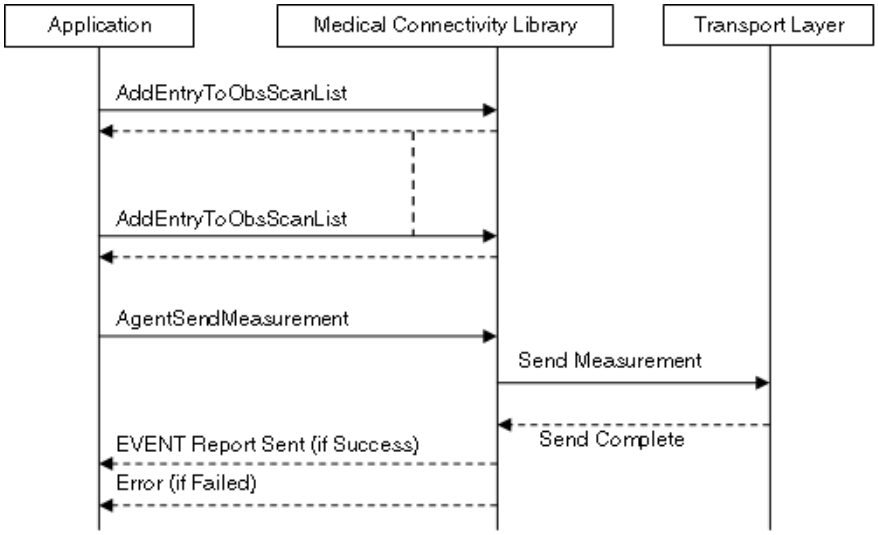


**Figure 3-5. Send Measurement Un-Confirmed Flow**

To send measurement data to Continua Manager, application creates an ObservationScanList. Application then calls AgentSendMeasurements interface to send data to Continua Manager. Application is informed by a callback event (EVENT Report Sent) if successful, otherwise an error is reported in the callback. Application should wait for these callback events before attempting to send another measurement.

To prepare ObservationScanList, AddEntryToObsScanList interface is provided to the application. This abstracts the internal details of this structure from the application.

### 3.2.4 Send Person Measurement Flow

Figure 3-6 and Figure 3-7 below describes how application can send confirmed/un-confirmed person measurement data to Continua Manager.

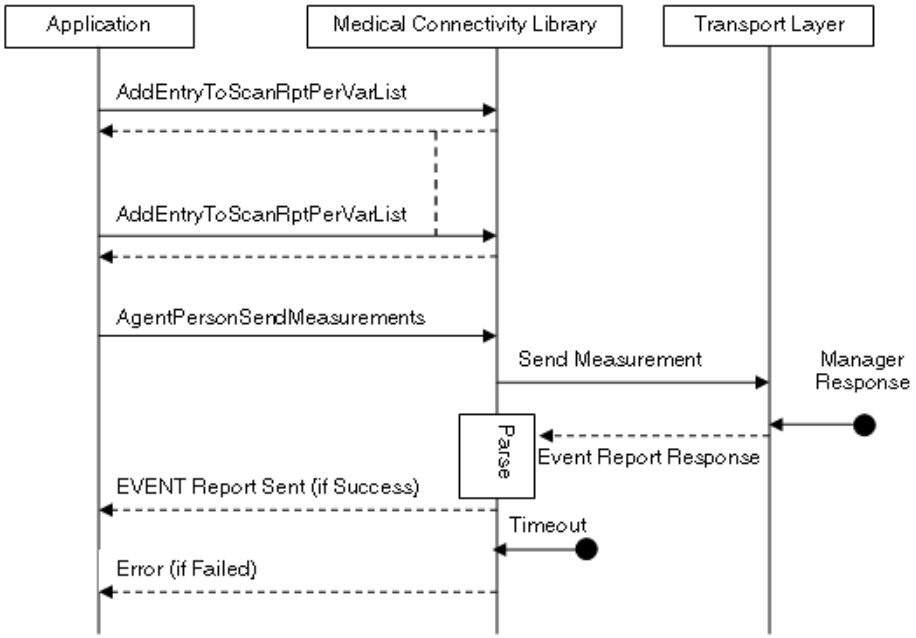


**Figure 3-6. Send Person Measurement Confirmed Flow**

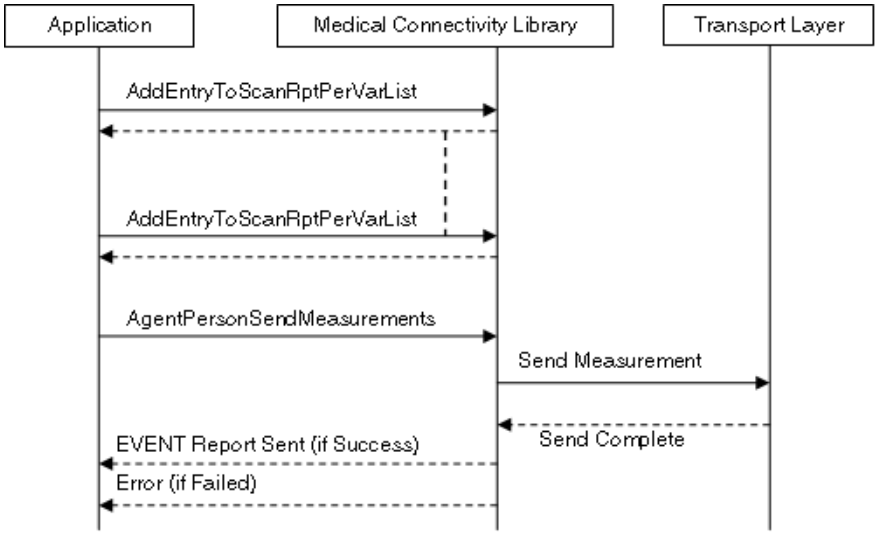


**Figure 3-7. Send Person Measurement Un-Confirmed Flow**

To send measurement data to Continua Manager, application creates a ScanReportPerVarList. Application then calls AgentSendPersonMeasurements interface to send data to Continua Manager. Application is

informed by a callback event (EVENT Report Sent) if successful, otherwise an error is reported in the callback. Application should wait for these callback events before attempting to send another measurement.

To prepare ScanReportPerVarList, AddEntryToScanRptPerVarList interface is provided to the application. This abstracts the internal details of this structure from the application.

# Chapter 4 Developing New Device Specialization

## 4.1 Introduction

This chapter discusses how you can create new device specialization applications based on Medical Connectivity Library.

## 4.2 Directory Structure

The following figure shows the directory structure of the PAN USB Agent application.

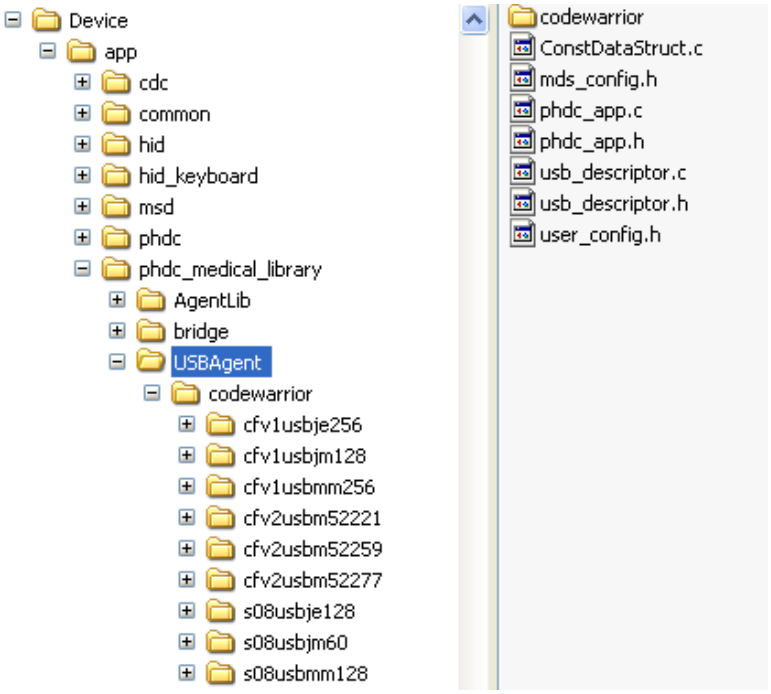


**Figure 4-1. PAN USB Agent Application Directory Structure**

For developing a new device specialization, change the following pre-existing files:

- ConstDataStruct.c
- phdc_app.c
- phdc_app.h

## 4.3 Medical Connectivity Library Interfaces

This section discusses the structures of the device specialization defined in the file ConstDataStruct.c.

### 4.3.1 Data Proto List

**Sample Structure:**

```
/* Data Proto List */
const intu8 g_DataProtoList[] =
{
      0x00, 0x01,
      0x00, 0x30,
      /* Data Proto ID = 20601 */
      0x50, 0x79,
      /* Data Proto Length  */
      0x00, 0x2c,
      /* PROTOCOL_VERSION1 */
      0x80, 0x00, 0x00, 0x00,
      /* Encoding rules */
      0x80, 0x00,
      /* NOM_VERSION1 */
      0x80, 0x00, 0x00, 0x00,
      /* Functional units */
      0x00, 0x00, 0x00, 0x00,  /* Device Cannot Enter Test Association */
      /* SYS_TYPE_AGENT */
      0x00, 0x80, 0x00, 0x00,
      /* system_id */
      0x00, 0x0e,
      'F','S','L','M','E','D','I','C','A','L','0','0','1', 0x0,
      /* dev_config_id */
      0x40, 0x00, /* --- user modifiable */
      /* data_req_mode_capab */
      0x08, 0x81, /* --- user modifiable */
      /* maximum number of parallel agent initiated data requests*/
      MAX_AGENT_DATA_COUNT, /* --- user modifiable */
      /* maximum number of parallel manager initiated data requests*/
      MAX_MANAGER_DATA_COUNT,
      /* optionList */
      0x00, 0x00, 0x00, 0x00
};
```

In the above structure, the user modifiable parameters are marked in red color. The device configuration Id for standard configuration is in the range 0x0001 — 0x3FFF (both inclusive). These configurations are fixed. If you want to create a configuration apart from the standard configuration, the configuration Id should be in the range 0x4000 — 0x7FFF (both inclusive). These configurations are called extended configurations. You can change data request mode capability flags depending on the requirement. The maximum number of parallel agent initiated data requests can be either 0 or 1.

**NOTE**

- Library does not support manager initiated data transmission (as per Continua Design guidelines).

- If agent initiated data transmission is not supported, at least one Scanner should be a part of the configuration.

### 4.3.2 DIM Variable

**Sample Structure:**

```
/* DIM variable */
DIM g_DIM =
{
      /* Pointer to MDS */
      &g_Mds, /* --- user modifiable */
      /* Device Configuration ID */
      EXTENDED_CONFIG_START /* --- user modifiable */
};
```

In the above structure, the user modifiable parameters are marked in red color. &g_Mds is a pointer to MDS which is the top-most object. The device configuration Id for standard configuration is in the range 0x0001 — 0x3FFF (both inclusive). These configurations are fixed. If you want to create a configuration apart from the standard configuration, the configuration Id should be in the range 0x4000 — 0x7FFF (both inclusive). These configurations are called extended configurations.

### 4.3.3 MDS Object

Medical Device System (MDS) represents the topmost object. Each device has one MDS class. The MDS represents the identification and status of the device through its attributes. Each class has a nomenclature code for its identification and some set of attributes. Out of all the attributes, some attributes are mandatory to implement (if that class is instantiated), and some are conditional that is, they should be present if the required condition is met. The rest of the attributes are optional. Each object has a "Handle" attribute that is used to identify the object for operations. The value of this attribute is unique. For MDS object, the handle is fixed to zero.

**Sample Structure:**

```
/* MDS Object */
MDS g_Mds =
{
      /* MDS Handle */
      0x0000,  /* --- not modifiable */
      /* Sys type */
      #ifdef MDS_ATTR_SYS_TYPE
      {
          4224,
          8136
      },
      #else
          /* Type Per Var List */
          TypeVerList*)&g_MdcAttrSysTypeSpecList,
      #endif /* MDS_ATTR_SYS_TYPE */

          /* System Model */
          (SystemModel*)&g_MdcAttrSysModel,
          /* System Id */
```

```
        (octet_string*)&g_MdcAttrSysId,
        /* device Configuration ID */
        0x4000,
        /* pointer to attribute val map */
    #ifdef MDC_FIXED_DATA_FORMAT
        (AttrValMap*)&g_MdcAttrValMap,
    #endif /* MDC_FIXED_DATA_FORMAT */

    /* pointer to production spec */
    #ifdef MDS_PROD_SPEC
        (ProductionSpec*)&g_MdcAttrIdProdSpecn,
    #endif /* MDS_ABSOLUTE_TIME */

    /* Mds Time Info */
    #ifdef MDS_SETTABLE_TIME
    {
        (
        #ifdef MDS_MGR_SET_TIME
            MDS_TIME_MGR_SET_TIME |
        #endif
        #ifdef  MDS_HIRES_TIME
            MDS_TIME_CAPAB_HIGH_RES_RELATIVE_TIME |
        #endif
        #ifdef MDS_RELATIVE_TIME
            MDS_TIME_CAPAB_RELATIVE_TIME |
        #endif
        #ifdef MDS_SET_CLOCK
            MDS_TIME_CAPAB_SET_CLOCK |
        #endif
        #ifdef MDS_RTC
            MDS_TIME_CAPAB_REAL_TIME_CLOCK |
        #endif
        0),
        MDC_TIME_SYNC_NONE,
        TIME_SYNC_ACCURACY_UNKNOWN,
        TIME_RESOLUTION_UNKNOWN,      /* Absolute Time Resolution */
        TIME_RESOLUTION_UNKNOWN,      /* Relative Time Resolution */
        TIME_RESOLUTION_UNKNOWN       /* High Res Time Resolution */
    },
    #endif /* MDS_SETTABLE_TIME */

    /* Absolute Time Stamp */
    #ifdef MDS_ABSOLUTE_TIME
    {
        0x20,     /* Century */
        0x09,     /* Year */
        0x07,     /* Month */
        0x22,     /* Day */
        0x19,     /* Hour */
        0x30,     /* Minute */
        0x10,     /* Second */
        0x00      /* Second Fraction */
    },
    #endif /* MDS_ABSOLUTE_TIME */

    /* Relative Time */
    #ifdef MDS_RELATIVE_TIME
```

```
        0x00000000,
    #endif /* MDS_ABSOLUTE_TIME */

    /* Hi Resolution Relative Time */
    #ifdef MDS_HIRES_TIME
    {
        0x00, 0x00, 0x00, 0x00, 0x00,0x00, 0x00, 0x00
    },
    #endif /* MDS_ABSOLUTE_TIME */

    /* Absolute Date and Time Adjust */
    #ifdef MDS_ADJUST_DATE_TIME
    {
        0x00, 0x00, 0x00, 0x00, 0x00,0x00
    },
    #endif /* MDS_ABSOLUTE_TIME */

    /* Power Status */
    #ifdef MDS_POWER_STATUS
        ON_MAINS,
    #endif /* MDS_POWER_STATUS */

    /* Battery Charge */
    #ifdef MDS_BATT_CHARGE
        /* 100% Charge */
        100,
    #endif /* MDS_BATT_CHARGE */

    /* Battery Measure */
    #ifdef MDS_BATT_REMAIN
    {
        /* 1 Day of Battery Remaining */
        0x1,
        MDC_DIM_DAY
    },
    #endif /* MDS_BATT_REMAIN */

        /* pointer to Reg Certified Data List */
        (RegCertDataList *)&g_MdcAttrRegCertDataList,

    /* Confirm Timeout  - 3 secs */
    #ifdef MDS_CONFIRM_TIMEOUT
        0x00005dc0,
    #endif
        /* Configuration Selected */
        0,       /* --- not modifiable */
        /* Configuration Count */
        2,
        /* pointer to an array of configurations */
        (CONFIGURATION (*)[])&g_Configuration
};
```

The user modifiable attribute values are marked in red color. You can modify all the attributes, except for Handle attribute. The configuration selected parameter, in the above structure, is modified in the library. The device can support multiple configurations whose number is given by the configuration count

followed by an array of configurations. The index of the configuration accepted by the manager is updated in the configuration selected parameter in the library.

### 4.3.4 Device Configuration

There are zero or more Numeric, Real-Time Sample Array, Enumeration, Scanner, or PM Store objects associated with an MDS object. There are zero or more PM Segments that contain persistent metrics associated with a PM Store. Numeric, Real-Time Sample Array, and Enumeration are derived from a common Metric base class that contains common and shared attributes. In general, Numeric objects represent episodic measurements, Real-Time Sample Array objects represent continuous samples or wave forms, Enumeration objects represent event annotations, and PM Stores along with PM Segments provide a persistent storage mechanism for metrics that are accessed by the Manager at a later time. In addition, a Scanner object facilitates the reporting of agent initiated data transfers. All these objects form a part of the device configuration.

**Sample Configuration Structure:**

```
/* Device Configuration */
const CONFIGURATION g_Configuration[] =
{
      /* Device Configuration ID */
      0x4000,
      #ifdef MDS_NUMERIC
          /* Numeric object count */
          2,
          /* Pointer to an array of Numeric classes */
          (NUMERIC (*)[])&g_Numeric,
      #endif
      #ifdef MDS_RTSA
          /* RTSA object count */
          1,
          /* Pointer to an array of RTSA classes */
          (RTSA (*)[])&g_Rtsa,
      #endif
      #ifdef MDS_ENUMERATION
          /* Enumeration object count */
          1,
          /* Pointer to an array of Enumeration classes */
          (ENUMERATION (*)[])&g_Enum,
      #endif

      #ifdef MDS_EPISOIDIC_SCANNER
          /* Episoidic Scanner object count */
          1,
          /* Pointer to an array of Episoidic Scanner classes */
          (EPICFGSCANNER (*)[])&g_EpiScanner,
      #endif
      #ifdef MDS_PERIODIC_SCANNER
          /* Periodic Scanner object count */
          1,
          /* Pointer to an array of Periodic Scanner classes */
          (PERICFGSCANNER (*)[])&g_PeriScanner,
      #endif
      #ifdef MDS_PMSTORE
```

```
            /* PM Store object count */
            1,
            /* Pointer to an array of PM Store classes */
            (PMSTORE (*)[])&g_PMStore
        #endif
};
```

The whole configuration structure is user modifiable. The configuration structure sets the number of Numeric, RT-SA, Enumeration, Episodic Scanner, Periodic Scanner, PM Store classes and the pointers to these classes. It also provides the device configuration Id.

**NOTE**

- The configuration Id of the first configuration should be the one specified in the Data Proto List.
- If agent initiated data transmission is not supported (specified in data request mode flags in Data Proto List), at least one Scanner should be a part of the configuration.

## 4.3.5 Metric Object

The Metric class is the base class for all objects representing measurements, status, and context data. The Metric class is not instantiated. As a base class, it defines all common attributes, methods, events, and services that are common for all objects representing measurements.

**Sample Structure:**

```
#ifdef MDS_METRIC
METRIC g_Metric[] =
{
        /* Numeric(Temperature) Class Handle */
        (intu16)0x0002,
        /* TYPE */
        {
            MDC_PART_SCADA,
            MDC_TEMP_BODY
        },
        /* Metric Spec Small */
        (intu16)(
        MSS_AVAIL_INTERMITTENT | MSS_AVAIL_STORED_DATA |
        MSS_UPD_APERIODIC | MSS_MSMT_APERIODIC | MSS_ACC_AGENT_INITIATED
        ),
        /* Optional Attributes */
        (intu16)(OPT_MET_UNIT_CODE | OPT_MET_ATTRVALMAP),
        /* SupplementalTypeList */
        (SupplementalTypeList*)&g_Supplemental_type_list,
        /* Metric Struct Small */
        {
            MS_STRUCT_SIMPLE, 0
        },
        /* Measurement Status */
        MS_VALIDATED_DATA,
        /* MetricIdPhysioList */
        {
```

```
            0,
            (OID_Type *)&gMetricId
        },
        /* Metric ID Part */
        NOM_PART_OBJ,
        /* Unit Code */
        MDC_DIM_DEGC,
        /* Attribute Val Map */
        (AttrValMap*)&g_AttrValMap,
        /* source handle reference*/
        0x0001,
        /* Label String */
        (octet_string*)&g_label_string,
        /* Unit Label String */
        (octet_string*)&g_unit_label_string
};
#endif
```

The user modifiable attribute values are marked in red color. You can modify all the attributes. The operational attribute flag tells which conditional and optional attributes of this class are implemented by the user.

### 4.3.6 Numeric Object

An instance of a Numeric class represents a numerical measurement. This class is derived from the Metric base class.

**Sample Structure:**

```
/* Numeric classes */
#ifdef MDS_NUMERIC
const NUMERIC g_Numeric[] =
{
      /* Metric class Pointer */
      &g_Metric[0],
      /* Optional Attribute */
      1,
      /* Accuracy */
      0x00000001
};
#endif
```

The user modifiable attribute values are marked in red color. &g_Metric[0] sets the pointer to the metric base class. The operational attribute flag tells which conditional and optional attributes of this class are implemented by the user.

### 4.3.7 Real-Time Sample Array (RT-SA) Object

An instance of the Real-Time Sample Array (RT-SA) class represents a wave form measurement. This class is derived from the Metric base class.

**Sample Structure:**

```
const RTSA g_Rtsa[] =
{
      /* Metric class Pointer */
      &g_Metric[3],
      /* sample period */
      0x00000002,
      /* Simple Sa Observed Value */
      (octet_string*)&g_RtsaSimpSaObsVal,
      /* Scale Range Spec */
      0x00,
      (ScaleRangeSpec8 *)&g_scale8,
      /* Sa Spec */
      0x00a,          /*array size */
      0x08, 31,        /* sample type */
      SMOOTH_CURVE    /* sa flags */
};
```

The user modifiable attribute values are marked in red color. &g_Metric[3] sets the pointer to the Metric base class. All attributes of the RT-SA object class are mandatory.

### 4.3.8 Enumeration Object

An instance of the Enumeration class represents status information and/or annotation information. The values of the Enumeration object are coded in the form of normative codes (as defined in ISO/IEEE Std 11073-10101) or in the form of free text. This class is derived from the Metric base class.

**Sample Structure:**

```
/* Enumeration Object Class */
#ifdef MDS_ENUMERATION
ENUMERATION g_Enum[] =
{
      /* Pointer to metric class */
      &g_Metric[2],
      /* operational attribute flag */
      1,
      /* Nom Partition */
      NOM_PART_SITES
};
#endif
```

The user modifiable attribute values are marked in red color. &g_Metric[2] sets the pointer to the Metric base class. The operational attribute flag tells which conditional and optional attributes of this class are implemented by the user.

### 4.3.9 Scanner

A Scanner object is an observer and 'summarizer' of object attribute values. It observes attributes of Metric objects (for example, Numeric objects) and generates summaries. The Scanner class is an abstract class defining attributes, methods, events, and services that are common for its subclasses. As such, it cannot be instantiated. More specialized Scanner classes are derived from the Scanner base class.

**Sample Structure:**

```
#ifdef MDS_SCANNER
SCANNER g_Scanner[] =
{
      /* Episoidic class handle */
      5,
      /* Operational Stat */
      0, /* --- not modifiable */
      /* Handle Attribute Val Map */
      (HandleAttrValMap*)&g_HandleAttributeValMap,
};
#endif
```

The user modifiable attribute values are marked in red color.

### 4.3.10 Configurable Scanner

The Configurable Scanner class is an abstract class defining attributes, methods, events, and services that are common for its subclasses (Episodic and Periodic Configurable Scanner objects). In particular, it defines the communication behavior of a configurable Scanner object. As such, it cannot be instantiated.

**Sample Structure:**

```
/* Configuration scanner class */
#ifdef MDS_CFGSCANNER
CFGSCANNER g_CfgScanner[] =
{
      /* Optional Attributes */
      3,
      /* pointer to scanner class */
      (SCANNER*)&g_Scanner[0],
      /* confirm mode */
      0x0001,
      /* confirm timeout - 2 secs */
      16000,
      /* transmit window */
      0x0001
};
#endif
```

The user modifiable attribute values are marked in red color. &g_Scanner[0] sets the pointer to the Scanner base class. The operational attribute flag tells which conditional and optional attributes of this class are implemented by the user.

### 4.3.11 Episodic Configurable Scanner

The Episodic Configurable Scanner objects are used to send reports containing episodic data, that is, data that does not have a fixed period between each data value. A report is sent whenever one of the observed attributes changes value. However, two consecutive event reports shall not have a time interval less than Min-Reporting-Interval.

**Sample Structure:**

```
/* Episoidic scanner object class */
#ifdef MDS_EPISOIDIC_SCANNER
EPICFGSCANNER g_EpiScanner[] =
{
      /* Optional Attributes */
      OPT_EPISCAN_MIN_RPT_INT,
      /* pointer to configuration scanner class */
      (CFGSCANNER*)&g_CfgScanner[0],
      /* minumum reporting interval - 2 secs */
      16000
};
#endif
```

The user modifiable attribute values are marked in red color. &g_CfgScanner[0] sets the pointer to the Configurable Scanner base class. The operational attribute flag tells which conditional and optional attributes of this class are implemented by the user.

### 4.3.12 Periodic Configurable Scanner

Periodic Configurable Scanner objects are used to send reports containing periodic data, that is, data sampled during fixed periods. It buffers any data value changes to be sent as part of a periodic report. Event reports shall be sent with a time interval of Reporting-Interval. The number of observations for each Metric object is dependent on the Metric object's update interval and the Scanner's Reporting-Interval.

**Sample Structure:**

```
/* Periodic scanner object class */
#ifdef MDS_PERIODIC_SCANNER
PERICFGSCANNER g_PeriScanner[] =
{
      /* pointer to configuration scanner class */
      (CFGSCANNER*)&g_CfgScanner[1],
      /* Scan reporting period - 1 sec */
      8000
};
#endif
```

The user modifiable attribute values are marked in red color. &g_CfgScanner[1] sets the pointer to the Configurable Scanner base class. The scan reporting period sets the Scanner's reporting interval.

### 4.3.13 PM Store Object

An instance of the PM Store class provides long-term storage capabilities for Metric data. The data is stored in a variable number of PM Segment objects.

**Sample Structure:**

```
/* PM Store Object Class */
PMSTORE g_PMStore[] =
{
      /* optinal attribute flag */
      0x0f,
      /* PM Store Object handle */
      0x04,
      /* PM Store Capability */
      (
      #ifdef MULTI_PERSON_SUPPORT
          PMSC_MULTI_PERSON |
      #endif
      #ifdef SAMPLE_PERIOD
          PMSC_PERI_SEG_ENTRIES | /* if sample period attr present in PM Store or in all PM
          Segments, then this shud be set */
      #endif
      PMSC_CLEAR_SEGM_REMOVE | PMSC_CLEAR_SEGM_BY_TIME_SUP |
      PMSC_EPI_SEG_ENTRIES | PMSC_CLEAR_SEGM_BY_LIST_SUP),
      /* sampling algorithm */
      ST_ALG_NO_DOWNSAMPLING,
      /* max PM Segments entries */
      0x04,
      /* actual num of segment entries presently used */
      0x02,
      /* operational state, if data is actively added it should be 0x01 */
      0x00,
      /* PM Store Label String */
      (octet_string*)&g_pm_label_string,
      /* Sample Period */
      0x000000ff,
      /* num of PM Segments instantiated */
      0x02,
      /* Clear segment timeout */
      0xFFFFFFFF,
      /* PM Segment Count */
      0x02,
      /* Pointer to array of PM Segments */
      (PMSEGMENT (*)[])&g_PmSegment
};
```

The user modifiable attribute values are marked in red color. You can modify all the attributes. The PM Store object structure also sets the PM Segments instantiated followed by the pointer to an array of PM Segment structures.

### 4.3.14 PM Segment Object

An instance of the PM Segment class represents a persistently stored episode of measurement data.

**Sample Structure:**

```
PMSEGMENT g_PmSegment[] =
{
      /* optional attribute flag */
      0x01FF,
      /* instance number */
      0x0001,
      /* PM Segment entry map */
      (PmSegmentEntryMap*)&g_PmSegEntryMap,
      /* Person ID */
      #ifdef MULTI_PERSON_SUPPORT
          0x0001,
      #endif
      /* operational state */
      0x0000,
      /* Sample period */
      0x05,
      /* segment label string */
      (octet_string*)&g_PmSeg_label_string1,
      /* Segment Start Time */
      0x20, 0x09, 0x09, 0x13, 0x02, 0x00, 0x00, 0x00,
      /* Segment End Time */
      0x20, 0x09, 0x09, 0x13, 0x04, 0x00, 0x00, 0x00,
      /* Date and time adjustment */
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      /* usage count */
      0x00,
      /* Segment Statistics */
      (SegmentStatistics*)&g_SegStat,
      /* pointer to Segment data */
      NULL,
      /* Confirm timeout = 4 secs */
      0x00007D00,
      /* Transfer timeout = 4 secs */
      0x00007D00
};
```

The user modifiable attribute values are marked in red color. You can modify all the attributes. Whenever an entry is added to the PM Segment, the library updates the Segment Start Time (if implemented), Segment End Time (if implemented), and Usage Count (if implemented).

## 4.4 Application Design

This section discusses the application design. The application is made up of the main application function, application task, and the callback function.

### 4.4.1 Main Application Function

The main application function initializes Transport Independent Layer (TIL) and Medical Connectivity Library. It also registers the application callback function. The main application function uses the following C code:

```
void TestApp_Init(void)
{
      DisableInterrupts;
      <Application Buffers Initialization Code>
      <Application Specific Initialization Code goes here>

      /* Initialize TIL */
      TIL_Initialize((PTIL)&g_Til);
      /* Initialize IEEE11073 and start Transport */
      (void)Ieee11073Initialize((PTIL)&g_Til, SHIMID, MedAppCallback);

      EnableInterrupts;
      while(TRUE)
      {
          __RESET_WATCHDOG();
          <Application Specific Code goes here>
          new_app_task();
      }
}
```

### 4.4.2 Application Task

The application task performs application specific functionality. Sample application code below uses Keyboard inputs to send measurement data to Continua Manager. The application task uses the following C code:

```
static void new_app_task(void)
{
      ERR_CODE err = ERROR_SUCCESS;
      if(kbi_stat > 0)
      {
          switch(kbi_stat & KBI_STAT_MASK)
          {
              case SEND_BPM_MSR : /* PTG1 is pressed */
              {
                  <Application Specific Code goes here>
              }
              break;

              case SEND_TEMPERATURE_MSR : /* PTG2 is pressed */
              {
                  <Application Specific Code goes here>
              }
              break;
```

```
                case SEND_EPISOIDIC_SCANNER_MSR: /* PTG3 is pressed*/
                {
                    <Application Specific Code goes here>
                }
                break;

            /* otherwise */
            default:break;
            }
        /* reset status after servicing interrupt */
        kbi_stat = 0x00;
        }
        return;
}
```

### 4.4.3 Callback Function

This is the application callback function. This callback function is registered during initialization in the main application function. This function performs various tasks depending upon the event received from the Medical Connectivity Library. The callback function uses the following C code:

```
static void MedAppCallback(IEEE11073_EVENT event_id, void *pvoid)
{
        switch(event_id)
        {
            case IEEE11073_ASSOCIATION_RELEASING:
                <Application Specific Code goes here>
            break;
            case IEEE11073_ASSOCIATION_RELEASED:
                <Application Specific Code goes here>
            break;
            case IEEE11073_TRANSPORT_CONNECT:
                <Application Specific Code goes here>
            break;
            case IEEE11073_TRANSPORT_DISCONNECT:
                <Application Specific Code goes here>
            break;
            case IEEE11073_GET_DATAPROTO:
                <Application Specific Code goes here>
            break;
            case IEEE11073_OPERATING:
                <Application Specific Code goes here>
            break;
            case IEEE11073_EVNTRPT_SENT:
                <Application Specific Code goes here>
            break;
            case IEEE11073_CLEAR_PMSEGMENT:
                <Application Specific Code goes here>
            break;
            default:
                <Application Specific Code goes here>
            break;
          }
          return;
}
```

# Chapter 5 IEEE 11073 Manager

## 5.1 Introduction

This chapter describes the IEEE 11073-based PHDC Manager implementation using USB host stack functionality for transport purposes.

The IEEE 11073 Manager's purpose is to enable seamless interoperability between personal healthcare devices (such as glucose meters, pulse oximeters, thermometers, etc) and USB hosts. The USB Class definition for personal healthcare devices provides a generic mechanism by which standardized messages can be sent over USB

## 5.2 Folder structure

The PHDC manager source code includes:

- Include sub-folder:
  - ieee11073.h: contains agent structure definition ieee11073_comm.h: contains definition of application events, application Callback function pointer and function prototypes for communication layer
  - ieee11073_dec.h: describes function prototypes for event report decoding functionality
  - ieee11073_phd_types.h: defines all phdc structures for PHDC manager library ieee11073_non_codes.h: contains the nomenclature codes used by IEEE_11073ieee11073_sl.h: describes function prototypes for service layer
  - type.h: define basic data type used for PHDC manager libraryieee11073_timer.h: defines time object structure and function prototype serving time out functionality
  - TransportLayer/TIL.h: defines communication callback function pointer and function prototype for TIL layer
  - TransportLayer/UsbShimManager.h: defines constants, structures, function prototypes used for SHIM layer
- "Src sub-folder:
  - Ieee11073/ieee11073_comm.c: implements all functions of communication layer. It deals with analyzing APDU data from receive and send callback function called in lower level layer.
  - Ieee11073/ieee11073_dec.c: implements all decoding functions: It supports to decode:
    - Fix format
    - Variable format
    - Grouped format

For

- Numeric class

  - Real time sample array class
  - Enumeration class
  - MDS class
- ieee11073/ieee11073_sl.c: implement all service layer's functions which can be called by applications
- TransportLayer/TIL.c: implements TIL functions
- TransportLayer/USbShimManager.c: implements all functions of SHIM layer.

## 5.3 Module Usage/Aim

The PHDC Host class driver provides an interface to the USB Host controller, allowing the application layer to handle the data exchange with the Agent using standard PHDC commands in the scope of gathering the personal healthcare data.

The PHDC Host class provides the following functions:

- Manages a class interface with the connected device consisting in 3 communication pipes corresponding to the attached device endpoints (1 Bulk IN, 1 Bulk OUT endpoint and 1 Interrupt IN Endpoint)
- PHDC data sending with Metadata support.
- PHDC data receiving with Metadata support
- PHDC Send Class Request function with SET_FEATURE, CLEAR_FEATURE, GET_STATUS requests supportSend Complete Event indication to the application layer
- Receive Complete Event indication to the application layer
- Send Control Requests Complete Event indication to the application layer

## 5.4 Functional description

### 5.4.1 MQX and Bare-Metal System Interraction

The PHDC Host class driver uses the MQX or bare-metal stack provided Host common API services in order to access the USB bus and manage USB device communication. The Freescale MQX ™ USB Host Stack provides an abstraction of the USB hardware controller consisting in:

- Host API
- Chapter9 API
- Common-Class API

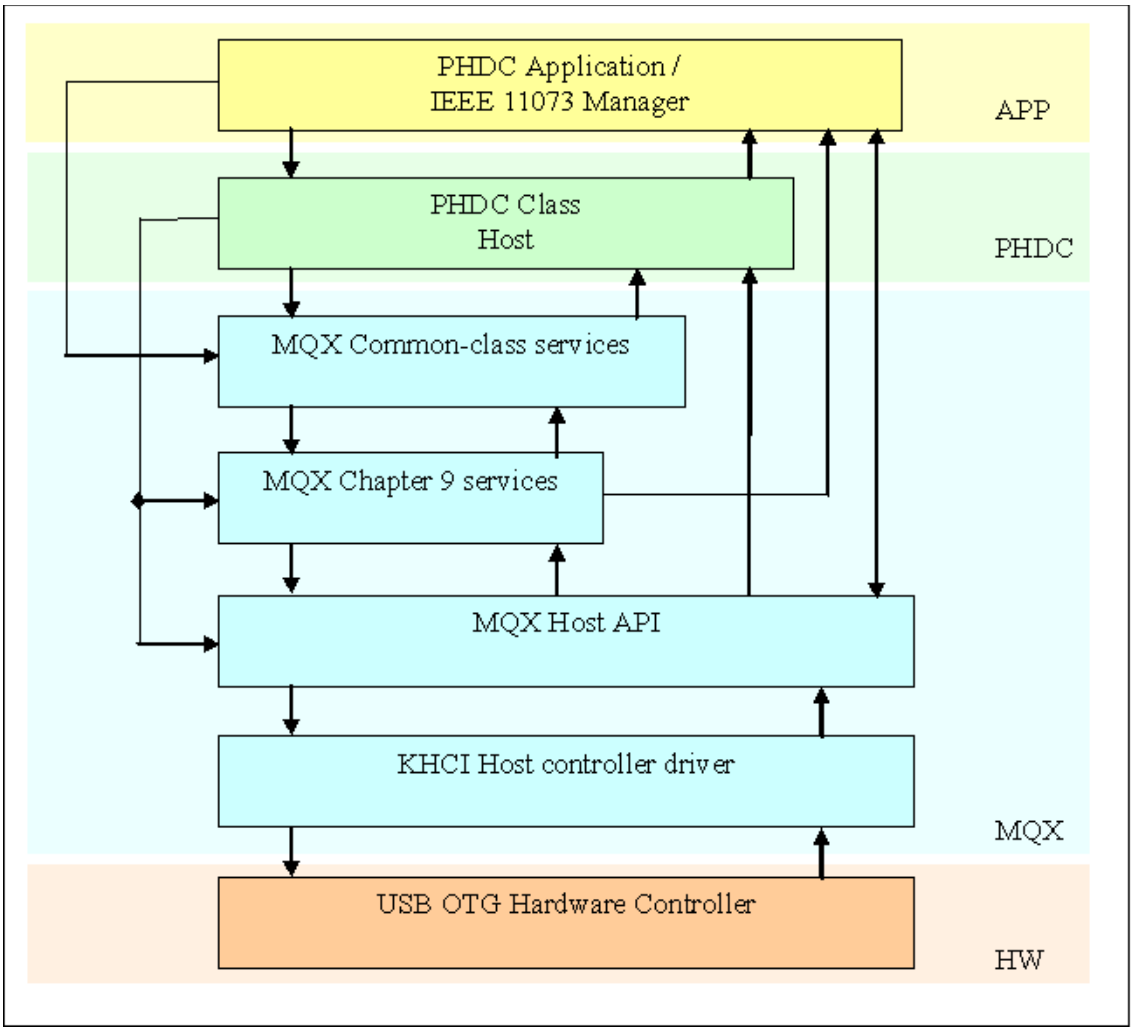


**Figure 5-1. PHDC Host class interaction with the MQX USB services**

The following sub-chapters present the interaction between the Application / PHDC Class driver and the used MQX USB layers.

## 5.4.2 MQX Host API

The MQX Host API is an abstraction of the host controller driver, providing interfaces independent of the underlying USB controllers.

The PHDC Application / IEEE 11073 Manager is responsible of initializing the Host controller and handle the USB attach and detach events issued by this layer.

The following MQX Host API services are used by the Application layer:

- **_usb_host_init** this function will initialize the USB Host stack. This function is called under application control as the application can manage multiple interfaces.

The PHDC Host class is responsible of handle the Bulk / Interrupt data transfers with the connected device using the MQX Host API pipe send / receive services:

- **_usb_host_send_data** this is a non blocking function that schedules a block of data for USB transmission and registers a callback for the send data complete event. The PHDC class calls this function following the Application request of sending a PHDC data (metadata or opaque data) to the connected agent.
- **_usb_host_recv_data** this is a non blocking function that schedules an USB data reception and registers a callback for the data received complete event. The PHDC class calls this function following the Application request of interrogating the connected Agent for measurement data.

### 5.4.3 MQX Chapter 9 Services

The MQX Chapter 9 layer implements the dedicated services specific to the USB Ch9 commands.

The Application / IEEE 11073 Manager is responsible of handling the relevant Chapter 9 indication events:

- **USB_ATTACH_EVENT** this event is sent to the application as the result of a PHDC device connection to the Host. The application uses this event to select the PHDC class interface on the Host (See the MQX Common Class)
- **USB_DETACH_EVENT** this event is sent to the application when the Device has disconnected from the host.

The PHDC Host class driver uses the MQX Chapter 9 services to transmit PHDC Specific requests (SET_FEATURE, CLEAR_FEATURE, GET_STATUS) using the control pipe:

- **_usb_hostdev_cntrl_request** this function is used by the MQX Ch9 Layer to process standard USB, class or vendor specific control pipe device requests. The PHDC class uses this service to send the PHDC class specific requests to the device and also USB standard requests as CLEAR_FEATURE (ENDPOINT_HALT)

### 5.4.4 MQX Common-class Services

The MQX Common-Class layer implements the common-class USB specifications. It interacts with the Host API as well as with the Chapter 9 layer in order to enumerate the attached device. After the device descriptors are identified, the common class layer searches for applications that are registered for the class or device plugged in.

The Application / IEEE 11073 Manager is responsible of handling the PHDC interface selection after the device has attached and getting the full descriptors of the PHDC device for a proper identification of the connected agent and QoS managing.

- _usb_hostdev_select_interface this function will select a new interface on the attached device, open the pipes associated to that interface and create the pipe bundle. This function also initializes the corresponding class driver by calling its registered initialization function. The application calls this interface after receiving the USB_ATTACH_EVENT from the Host API in order to set the PHDC interface and to initialize the PHDC class.

The PHDC Host class driver uses the Common-Class services to collect the PHDC class specific descriptors (QoS descriptor and the optional Metadata descriptor) for all the endpoints opened by the attached device:

- _usb_hostdev_get_descriptor this function returns the requested descriptor based on the provided descriptor type and descriptor index. The PHDC class calls this function in order to get the identification information from the connected device, after receiving the USB_ATTACH event from the Host API. The PHDC uses the PHDC specific descriptors in order to collect the data from the connected agent with regards to the supported QoS.

## 5.5 System Decomposition

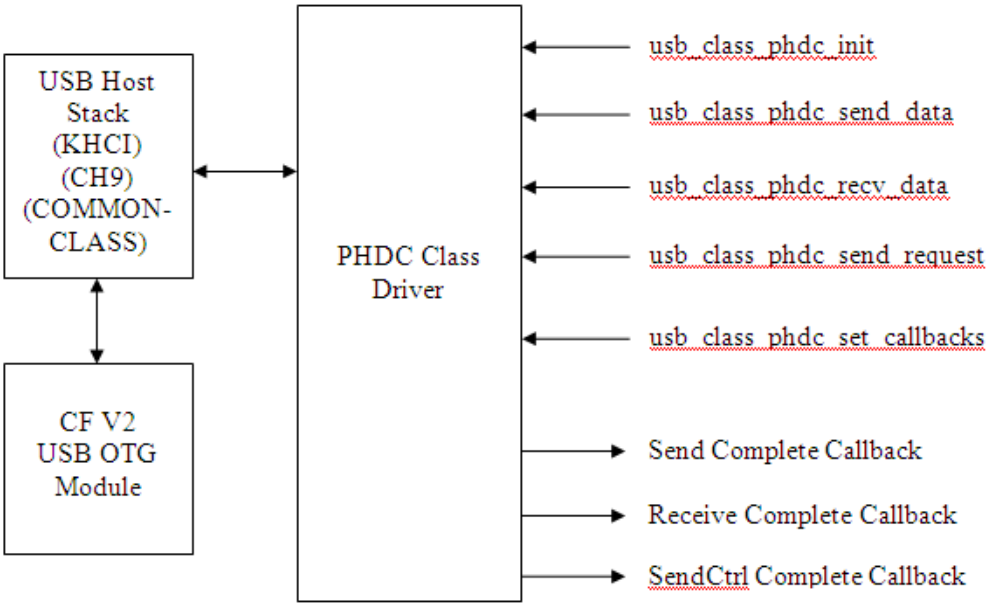


**Figure 5-2. PHDC class module decomposition**

# Appendix A Working with the Software

## A.1 Introduction

This chapter gives you insight on how to use the Medical Connectivity Library software. The following sections are described in this chapter:

- Preparing the setup
- Building the application
- Running the application

Knowledge of CodeWarrior IDE will be helpful to understand this section. While reading this chapter, practice the steps mentioned.

To take you through this chapter, USB Agent Demo for the MCS08JM60 is used as an example.

### A.1.1 Preparing the Setup

#### A.1.1.1 Software Setup

Refer to Readme file for compatibility with Freescale USB Stack with PHDC.

To install Medical Connectivity Library software setup:

1. Double-click MEDCONLIB_SW.exe file..
2. The Freescale Medical Connectivity Library Setup window appears. The following example shows the demonstration for Medical Connectivity Library 1.0 installation. You can follow the same instructions for new versions.

**Example:**

1. Click on the **Next** button to continue with Medical Connectivity Library 1.0 installation.


**Figure A-1. Freescale Medical Connectivity Library 1.0 Setup Wizard**

2. In Figure A-2, click on the **I Agree** button to accept the license agreement.

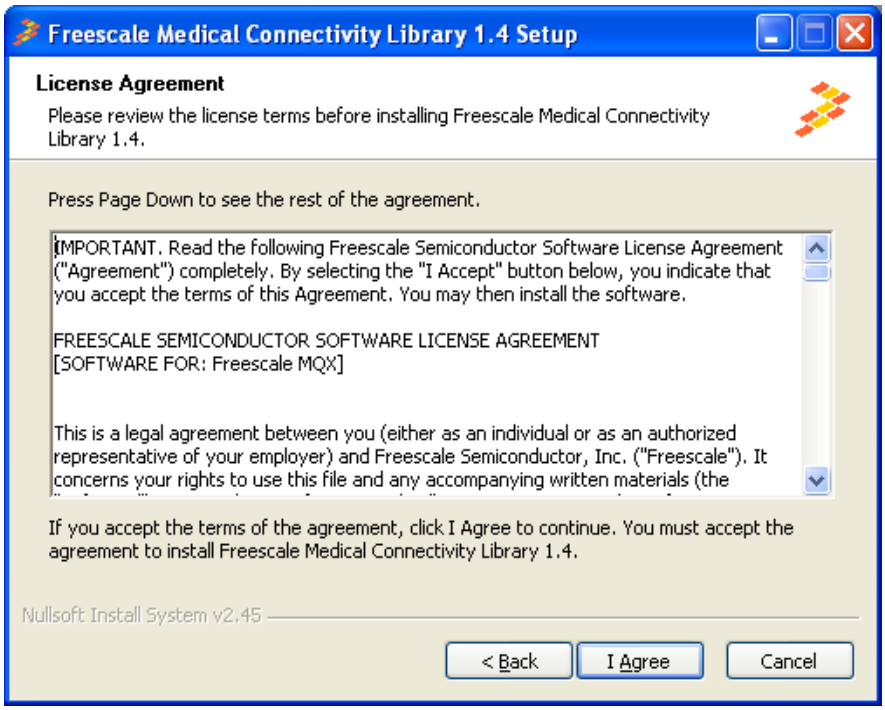


**Figure A-2. Freescale Medical Connectivity Library 1.0 Setup License Agreement**

3. In Figure A-3, select Medical Connectivity Library and PAN USB Agent demo application to install and click on the **Next** button.

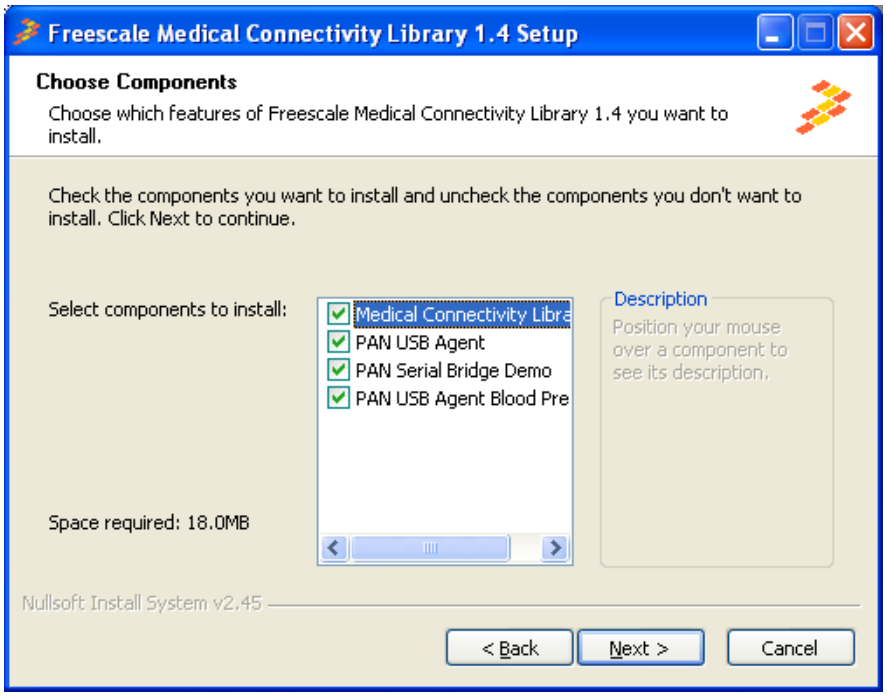


**Figure A-3. Freescale Medical Connectivity Library 1.0 Components**

4. In Figure A-4, select the location of the folder where you require to install the Freescale Medical Connectivity Library 1.0 software and click on the **Install** button.

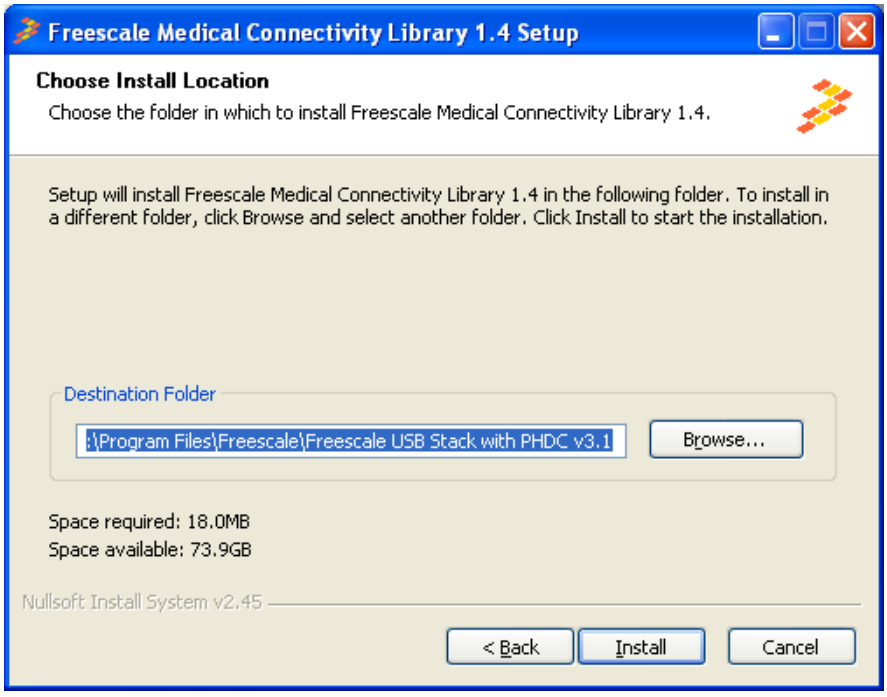


**Figure A-4. Freescale Medical Connectivity Library 1.0 Installation Folder Location**

**CAUTION**

Use the same destination folder where Freescale USB Stack with PHDC is installed.

5. Click on the **Finish** button to successfully complete the Freescale Medical Connectivity Library 1.0 Setup Wizard.

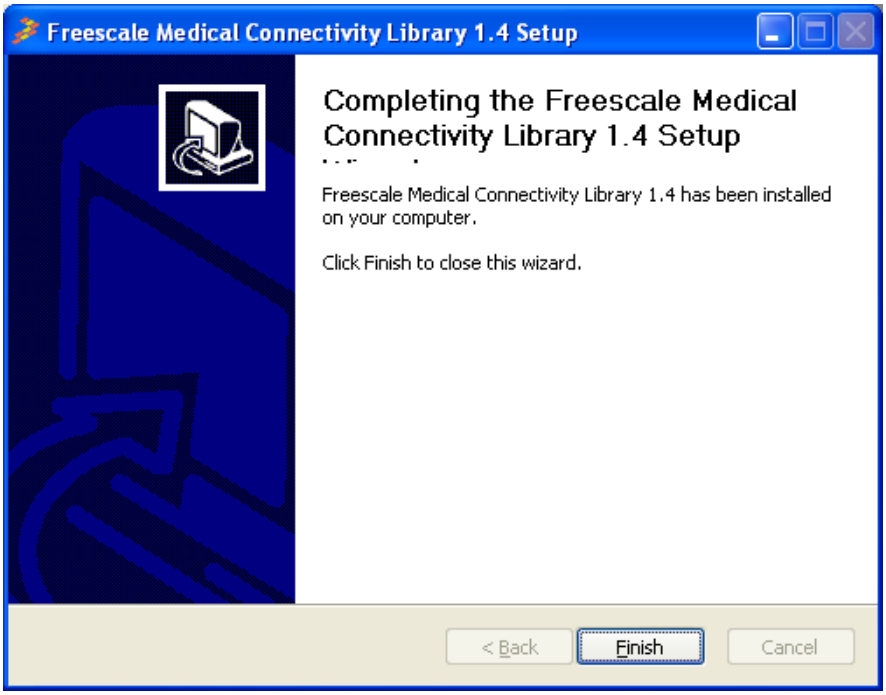


**Figure A-5. Freescale Medical Connectivity Library 1.0 Installation Finish**

To launch the Medical Connectivity Library project:

1. Click **Start** > **Programs** > **Freescale Medical Connectivity Library** > **Source**.

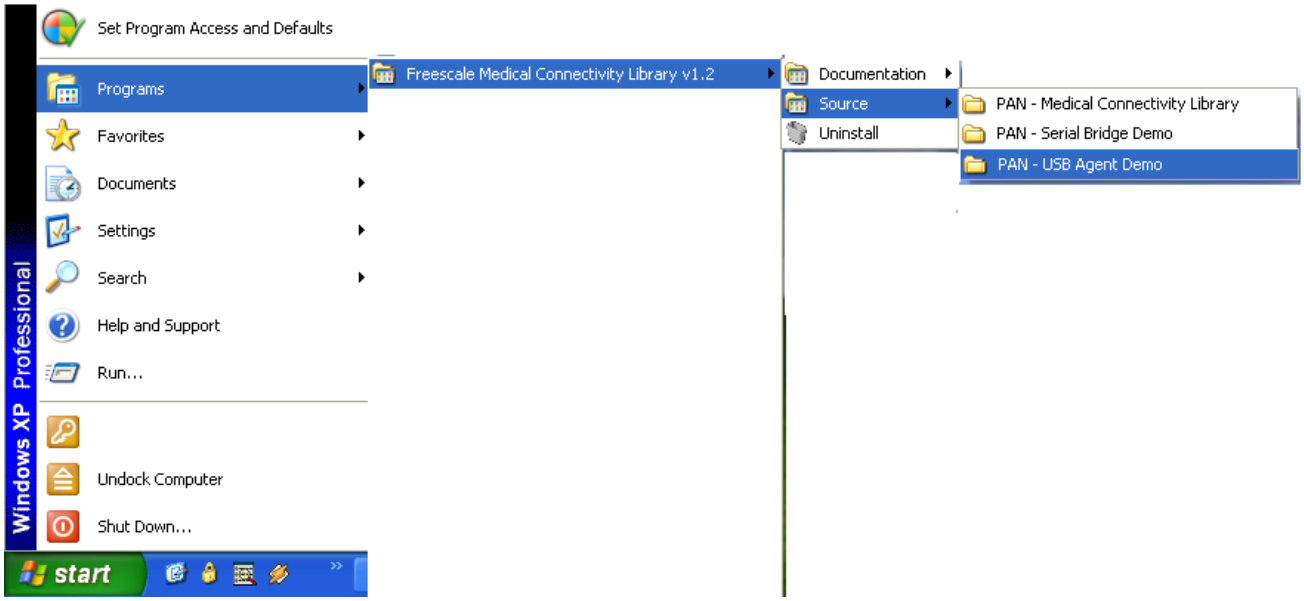


**Figure A-6. Freescale Medical Connectivity Library Source Program for Launch**

### A.1.1.2 Hardware Setup

- Make the connections as shown in Figure A-7.

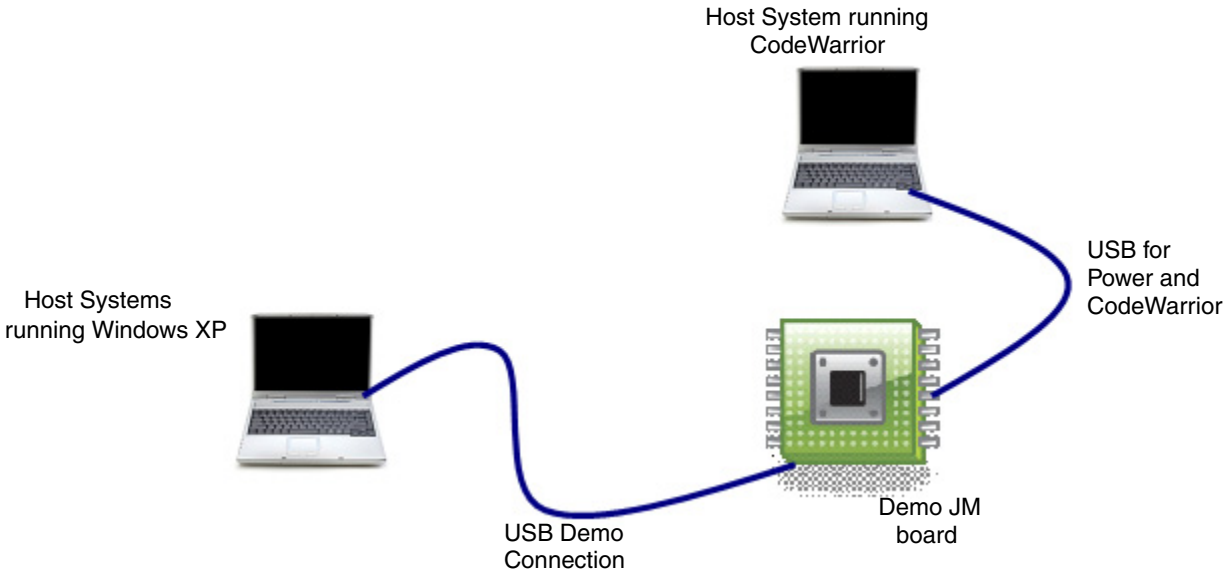


**Figure A-7. Medical Connectivity Library USB Setup**

— Make the first USB connection between the personal computer where the software is installed and the DemoJM board where the silicon is mounted. This connection is required to provide power to the board and downloading image to the flash.

— Make the second connection between the DemoJM board and the personal computer where the demo is run.

**NOTE**

Although, we have used two personal computers in Figure A-7, in reality you may achieve the same result by a single personal computer with two or more USB ports.

## A.1.2 Building the Application

The software for CFV1 is built using CodeWarrior 6.3. In addition, the software for CFV2 is built with CodeWarrior 7.2. Therefore, it contains application project files that can be used to build the project.

Before starting the process of building the project, make sure CodeWarrior 6.3 is installed on your computer.

To build the MC9S08JM60 project:

1. Navigate to the project file and open the s08usbjm60.mcp project file in CodeWarrior IDE.

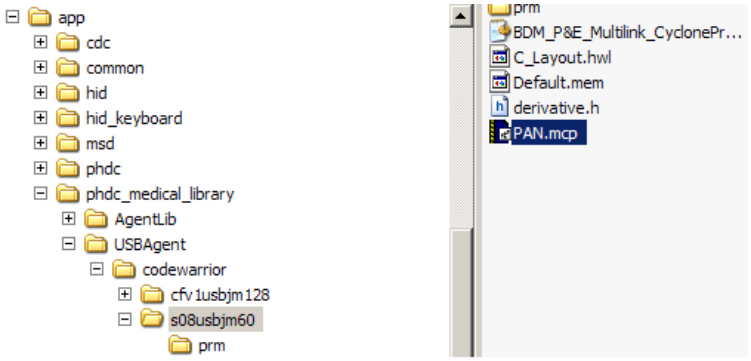


**Figure A-8. Open PAN.mcp Project File**

2. After you have opened the project, the following window appears. To build the project, click the button as shown in Figure A-9.

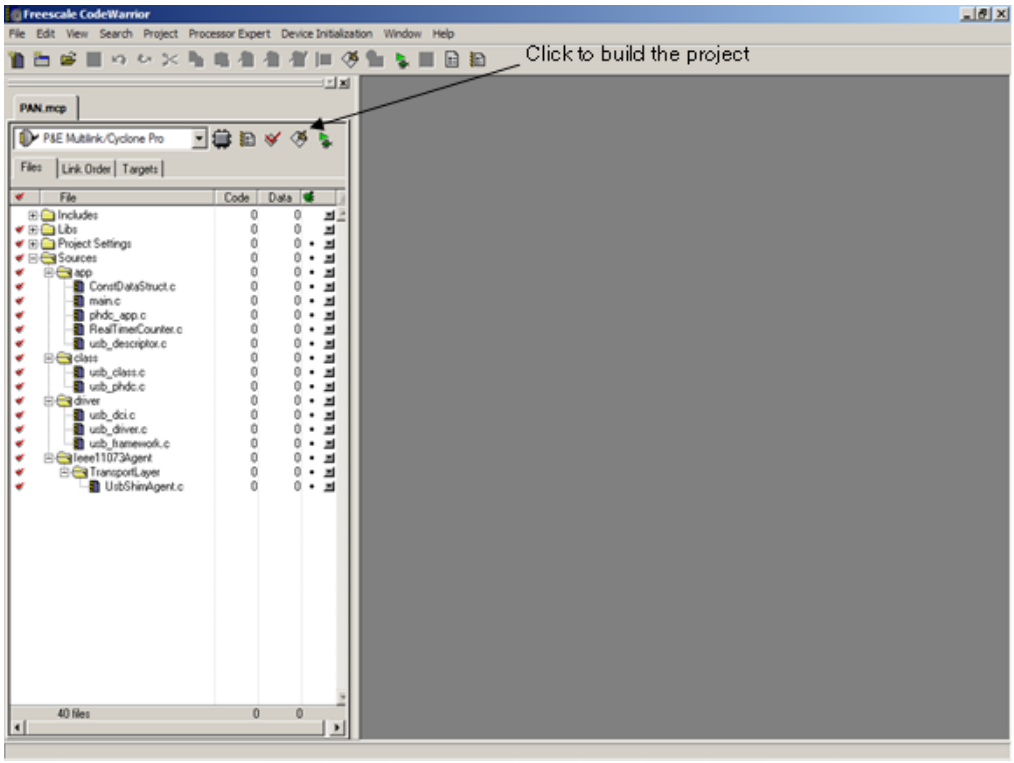


**Figure A-9. Build PAN.mcp Project**

3. After the project is built, the code and data columns must appear filled across the files.

**NOTE**

You must follow the above procedure to build CFV1and CFV2 projects also.

## A.1.3 Running the Application

Refer to the board documentation and CodeWarrior manual for details on how to program the flash memory on the evaluation board used. The following steps are presented as an example about how to run the application with DemoJM60 board using a P&E-micro debugger.

1. To run the application, click the button as shown in Figure A-10.

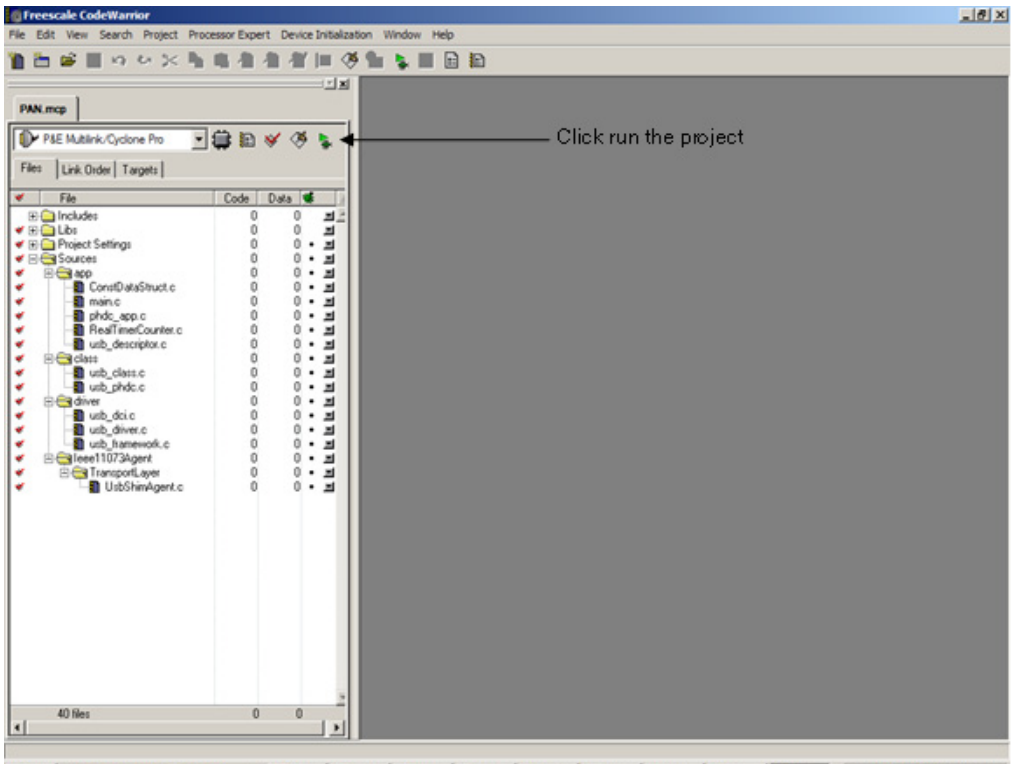


**Figure A-10. Running the Application**

2. The dialog box in Figure A-11 appears. Click on the **Connect (Reset)** button to connect to hardware as shown in Figure A-11.

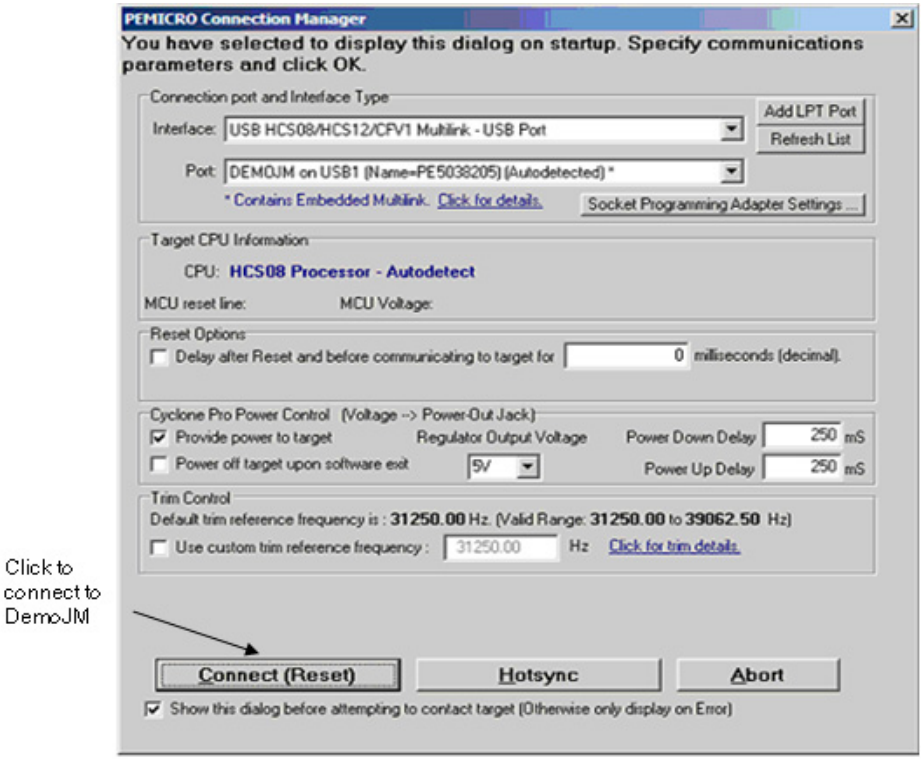


**Figure A-11. Connection Manager**

3. The pop-up in Figure A-12 appears. Click on the **Yes** button to load the built image to the JM60 flash.


**Figure A-12. Erase and Program Flash Pop-Up**

4. The pop-up in Figure A-13 appears to erase and program the built image to the JM60 flash.

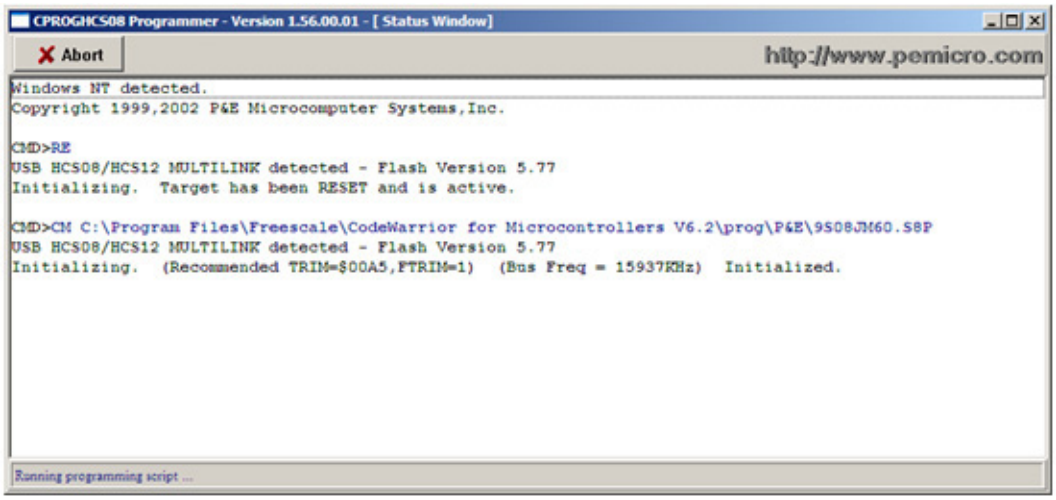


**Figure A-13. Image Programmed in Flash**

5. After the image is programmed in the flash, the debugger window as shown in Figure A-14 appears. Click on the Green Arrow as shown in Figure A-14 to run the programmed image.

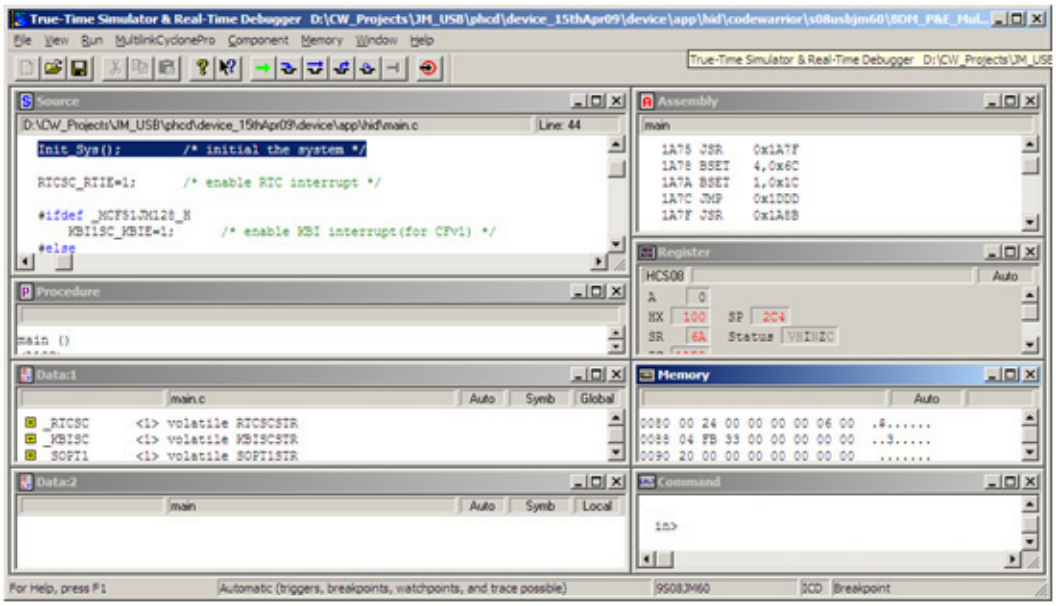


**Figure A-14. Simulator and Real-Time Debugger**

## A.2 Uninstall Freescale Medical Connectivity Library 1.0 Software

1. From your computer, click **Start** > **Settings** > **Control Panel** > **Add or Remove Programs**.

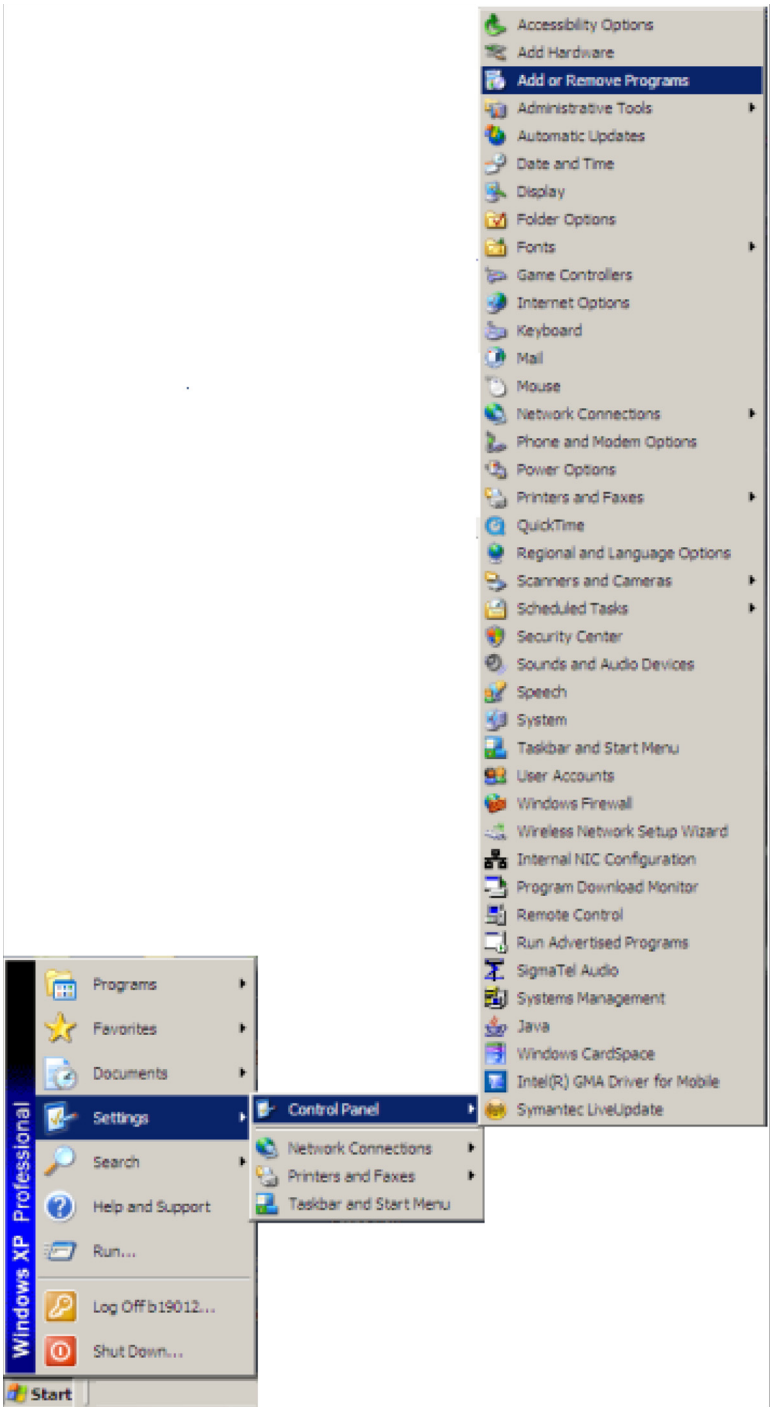


**Figure A-15. Add or Remove Programs Launch from Control Panel**

2. In Figure A-16, select **Freescale Medical Connectivity Library 1.0** and click on the **Change/Remove** button.

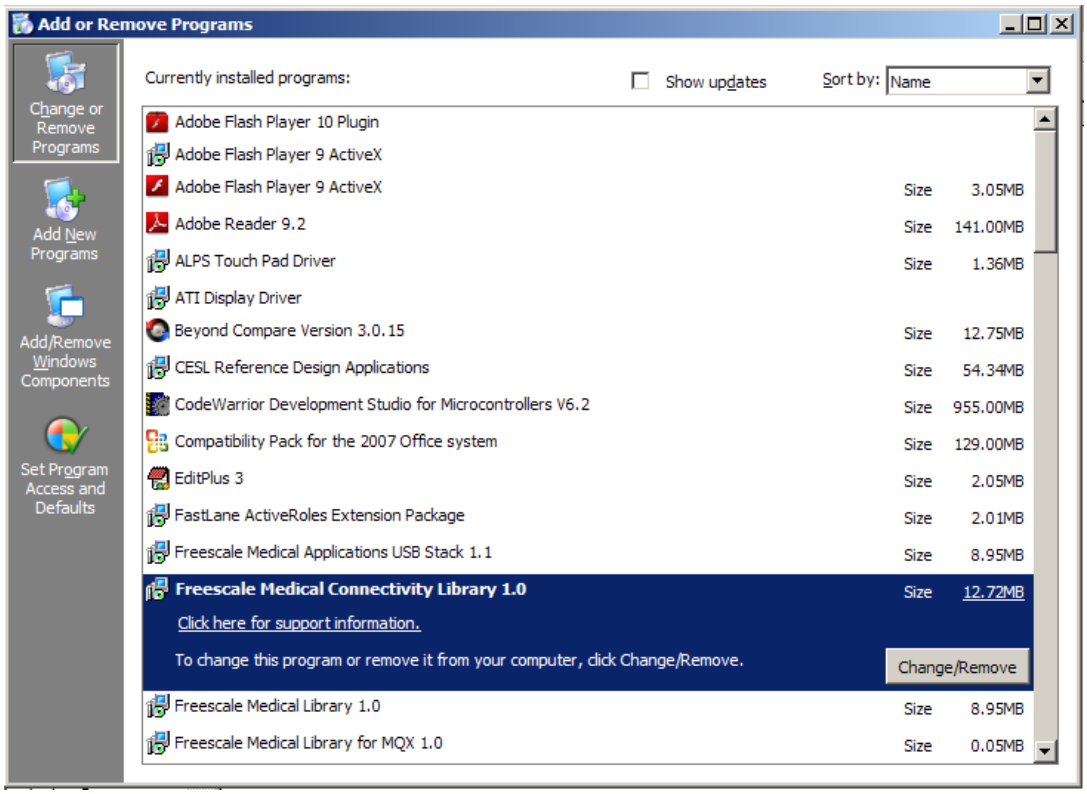


**Figure A-16. Add or Remove Programs**

3. The uninstall confirmation message appears. Click on the **Yes** button to uninstall.

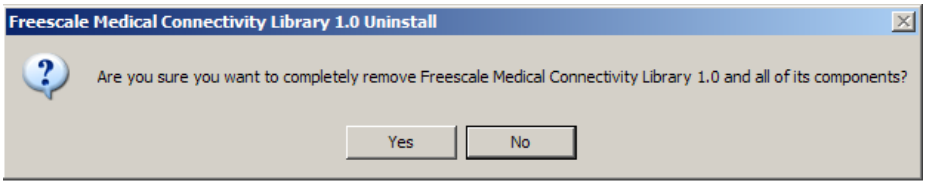


**Figure A-17. Freescale Medical Connectivity Library 1.0 Uninstall Confirmation Message**

4. A message box appears. Click on the **Ok** button to complete the uninstall operation.

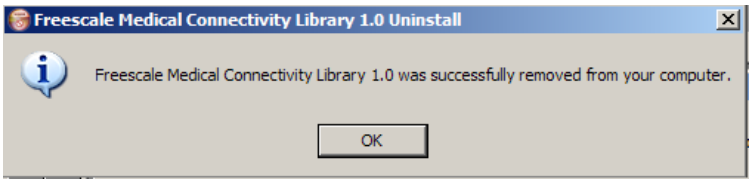


**Figure A-18. Freescale Medical Connectivity Library 1.0 Uninstall Completion Message**

## A.3 Important Files

Table A-1 shows the programming files that contain source code and header files.

**Table A-1. Important Files**

| Files | Description |
|---|---|
| device\app\phdc_medical_library\AgentLib\include\error.h | This header file contains error codes. |
| device\app\phdc_medical_library\AgentLib\include\ieee11073_dimstruct.h | This header file defines DIM structures to be used by device specialization. |
| device\app\phdc_medical_library\AgentLib\include\ieee11073_nom_codes.h | This header file contains IEEE11073 Nomenclature Codes. |
| device\app\phdc_medical_library\AgentLib\include\ieee11073_phd_types.h | This header file contains IEEE11073 Structure definitions. |
| device\app\phdc_medical_library\AgentLib\include\mds_config.h | This header file contains Medical Connectivity Library configuration (DO NOT CHANGE THIS FILE). |
| device\app\phdc_medical_library\AgentLib\include\MedAgentLibInterface.h | This header file contains Medical Connectivity Library Interface definitions. |
| device\app\phdc_medical_library\AgentLib\include\mempool.h | This header file contains Memory Management Interface. |
| device\app\phdc_medical_library\AgentLib\include\stack.h | This header file contains Buffer Stack Interface definitions. |
| device\app\phdc_medical_library\AgentLib\include\type.h | This header file contains basic data type definitions. |
| device\app\phdc_medical_library\AgentLib\include\TransportLayer\til.h | This header file contains TIL Interface definitions. |
| device\app\phdc_medical_library\AgentLib\include\TransportLayer\UsbShimAgent.h | This header file contains USB Shim Agent Interface definitions. |
| device\app\phdc_medical_library\AgentLib\src\TransportLayer\UsbShimAgent.c | This source file contains USB Shim Agent Source. |
| device\app\phdc_medical_library\AgentLib\lib\Ieee11073AgentLibCfv1.lib | This is Medical Connectivity Library for CFV1 devices. |
| device\app\phdc_medical_library\AgentLib\lib\ Ieee11073AgentLibJm60.lib | This is Medical Connectivity Library for JM60 device. |

# Appendix B PAN USB Agent Demo

Personal healthcare application interacts with the host system using IEEE-11073 – 20601 and (IEEE-11073 – 10415 (Weigh Scale), IEEE-11073 – 10407 (Blood Pressure Monitor), IEEE-11073 – 10417 (Glucose Meter), and IEEE-11073 – 10408 (Thermometer) protocol. To run this demo, a host system is required that runs the same IEEE-11073 protocols. One example of such implementation is done by Continua Alliance. In this demo, Continua Manager is used on the host system.

## B.1 Setting Up the Demo

1. Set the systems as described in Section A.1.1.2, "Hardware Setup."
2. Get the Continua Alliance (www.continuaalliance.org) CESL Reference Software V1.0 RC2.
3. Install the software on a host system
4. Program the JM60 flash with the PAN USB Agent Demo using CodeWarrior IDE. Refer to Section A.1.2, "Building the Application" for more details.

**NOTE**

CESL reference software is not provided as part of the suite. You will have to get this software independently from Continua Alliance.

## B.2 Running the Demo

After the system has been set, you must follow these steps to run the demo:

1. Turn on the DemoJM board. Found New Hardware window appears.


**Figure B-1. Found New Hardware Window**

2. Select **Install from a list or specific location (Advanced)** option as shown in Figure B-1, and click on the **Next** button. Search and installation options window appears as shown in Figure B-2.

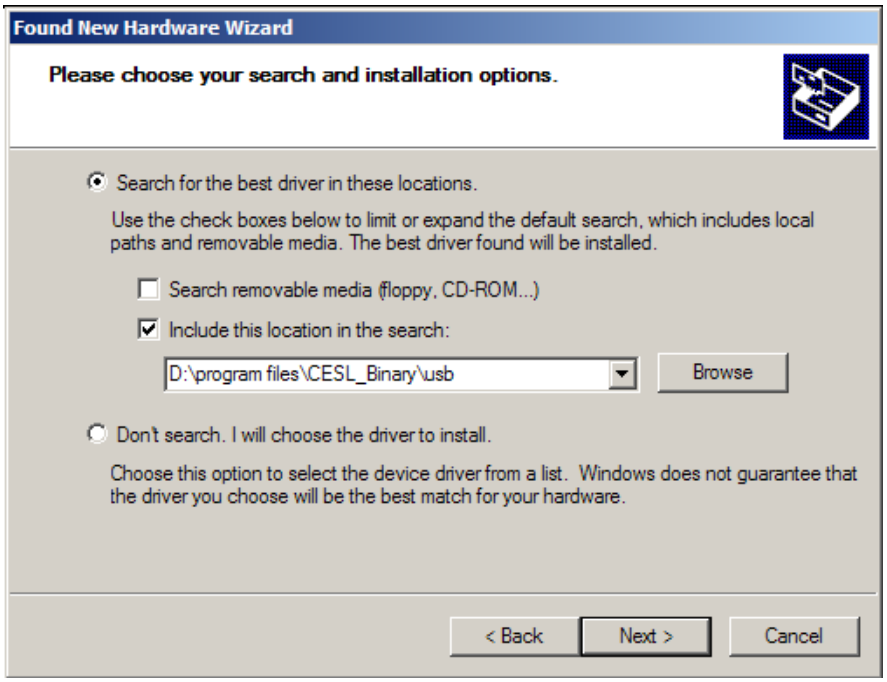


**Figure B-2. Search and Installation Options**

3. Point the search path to the bin directory where the Continua CESL software was installed and click on the **Next** button. The driver for the device will get installed.

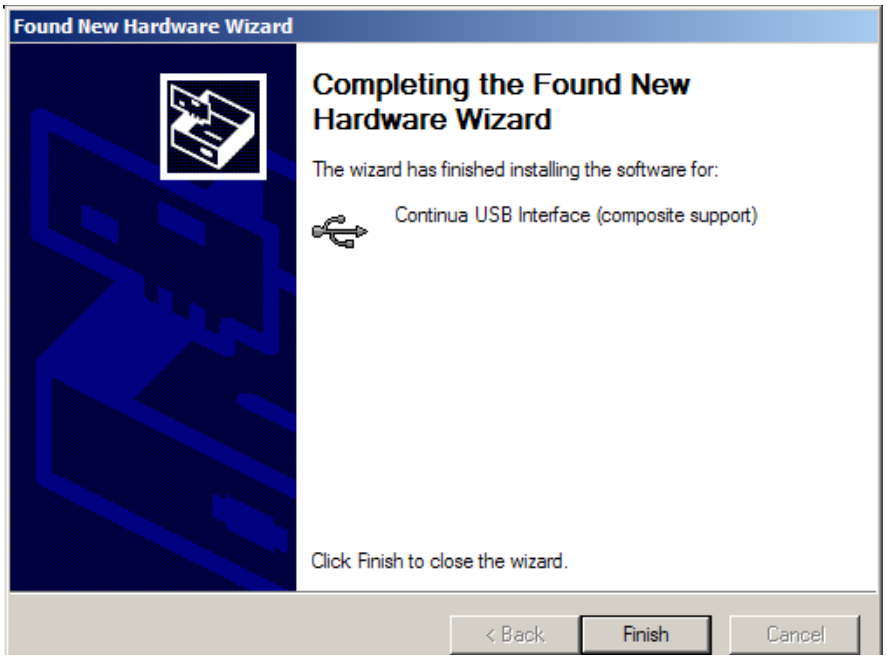


**Figure B-3. Installation Complete window**

4. To verify the installation, open the device manager. You must see the Continua USB Interface device entries.

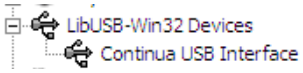


**Figure B-4. Continua USB Interface Device Entry in Device Manager**

5. Launch the Continua Manager from **Start** > **All Programs** menu as shown in Figure B-5.

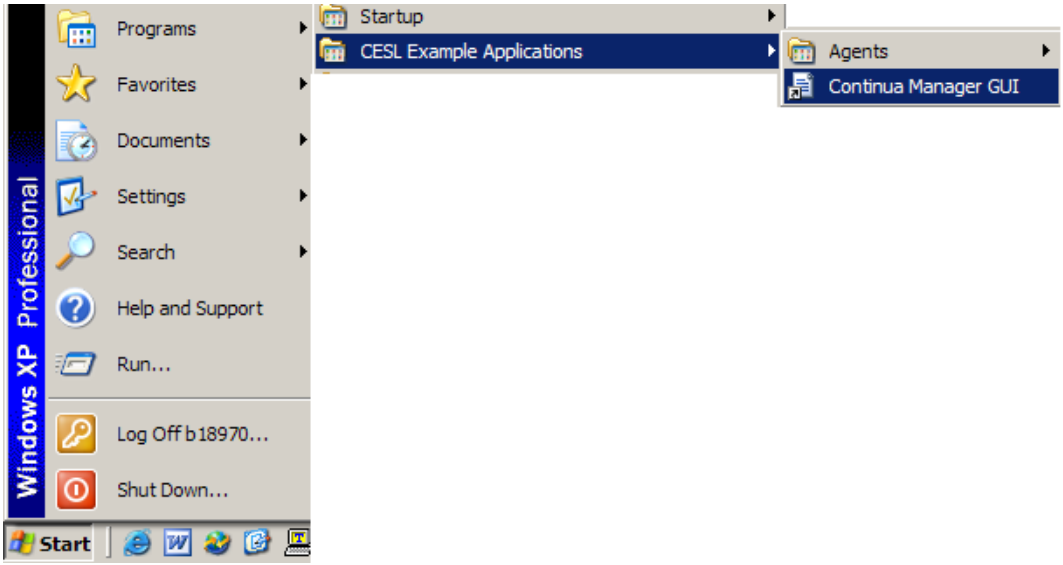


**Figure B-5. Launch Continua Manager**

6. The Continua Manager GUI opens as shown in Figure B-5. Enter the name of the skim directory and click on the **Start Transport** button.

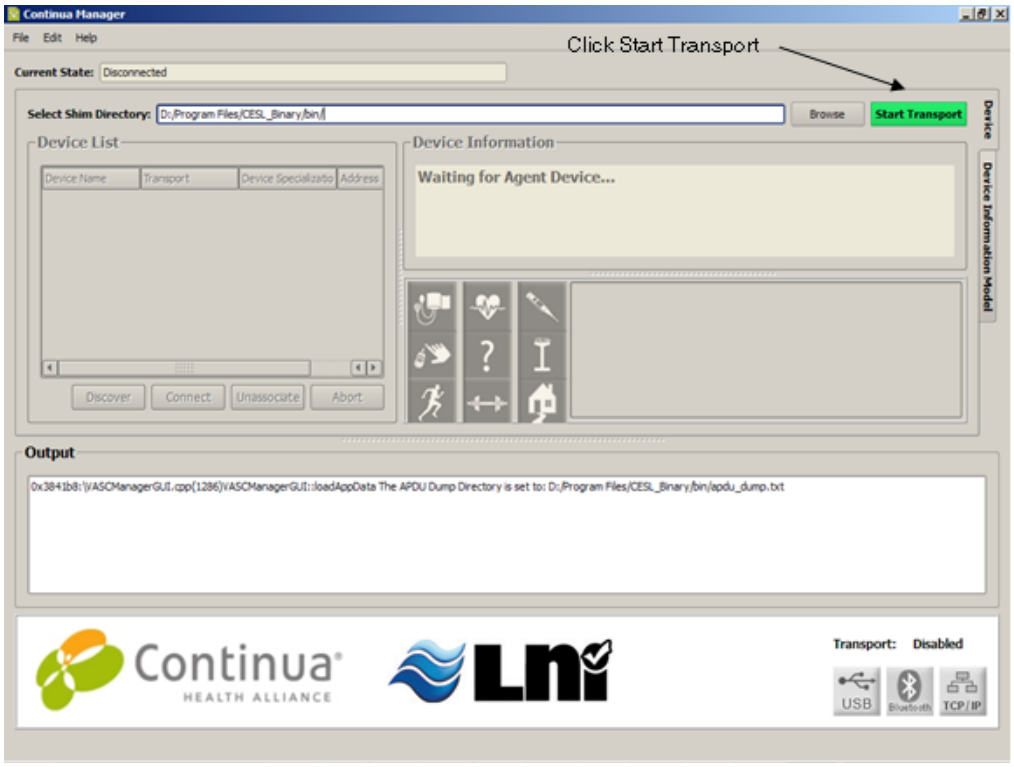


**Figure B-6. Host Entering Operating State**

7. The Continua Manager now enters the Operating State using Extended Configuration specialization. The Continua application window appears as shown in Figure B-7.

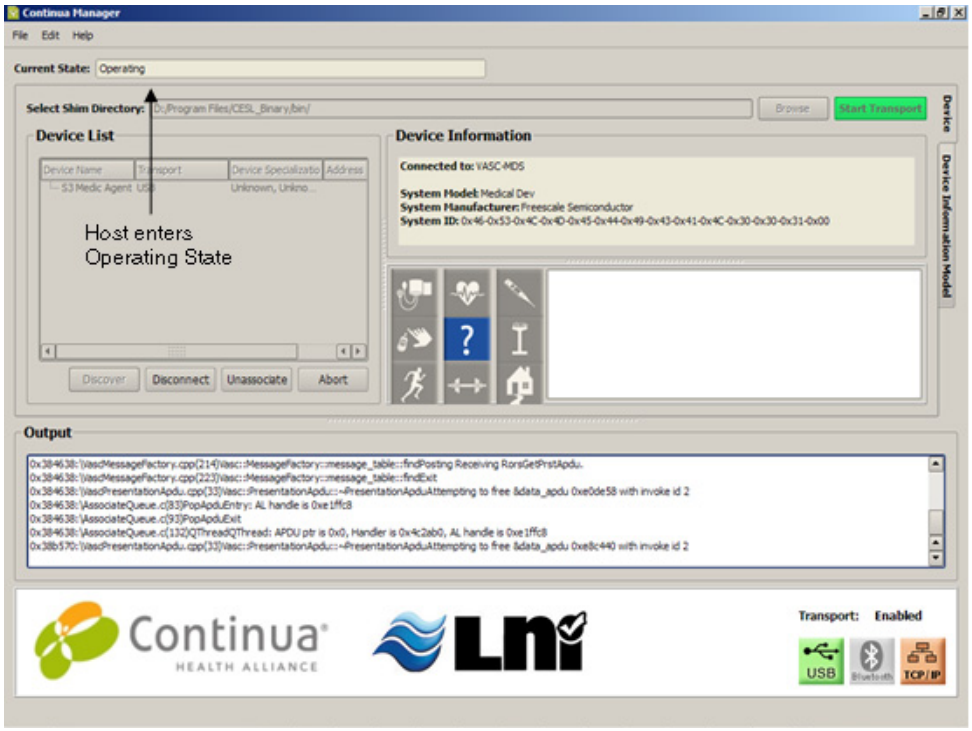


**Figure B-7. Host Entering Operating State**

8. After the host device is in operating state, **Push** Buttons on the device can be used to send weight measurements to the host. Figure B-8 shows the function of these buttons.

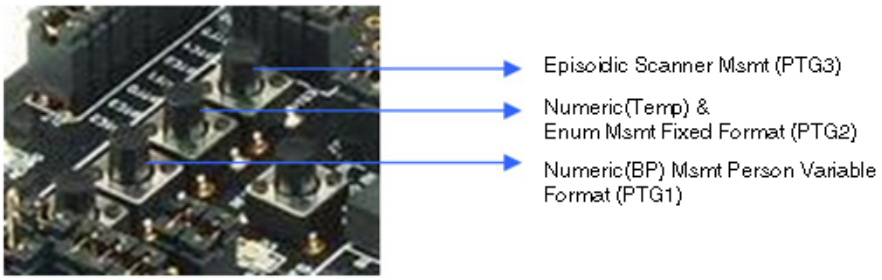


**Figure B-8. DemoJM Push Button Panel**

9. Figure B-9 shows Domain Information Model screen of Manager. This screen describes Agent Configuration.

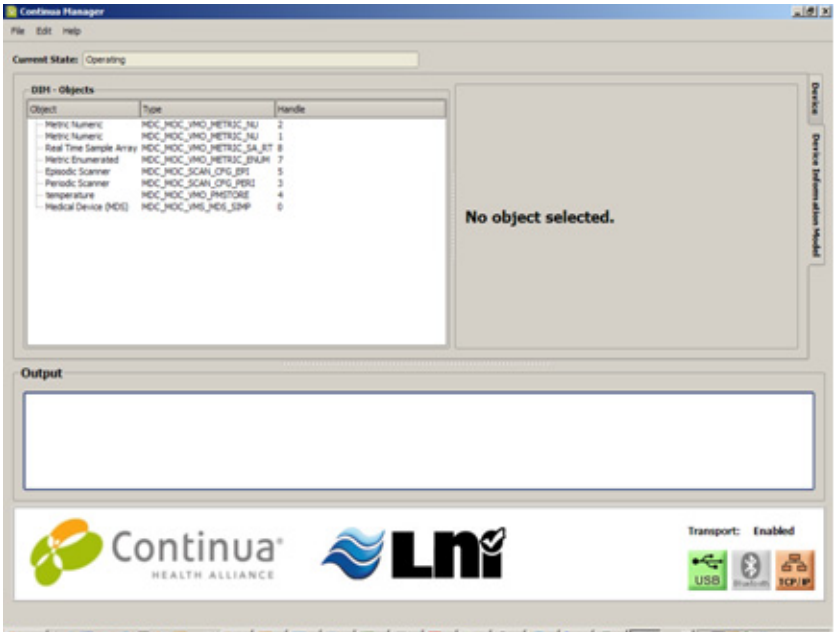


**Figure B-9. Domain Information Model screen**

10. When push button **PTG1** is pressed, Blood Pressure measurement in Person Variable Format is sent to host. Figure B-10 shows measurement data on host.

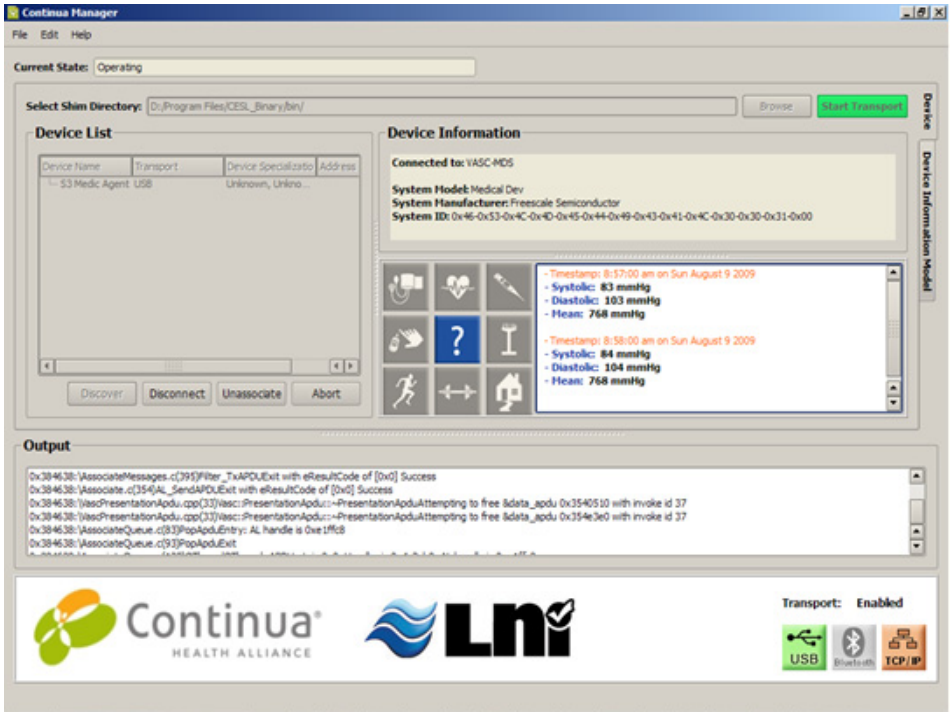


**Figure B-10. Blood Preasure measurement in Person Variable Format**

11. When push button **PTG2** is pressed, Temperature measurement and Enumeration class measurement in Fixed Format is sent to host. Figure B-11 shows measurement data on host.

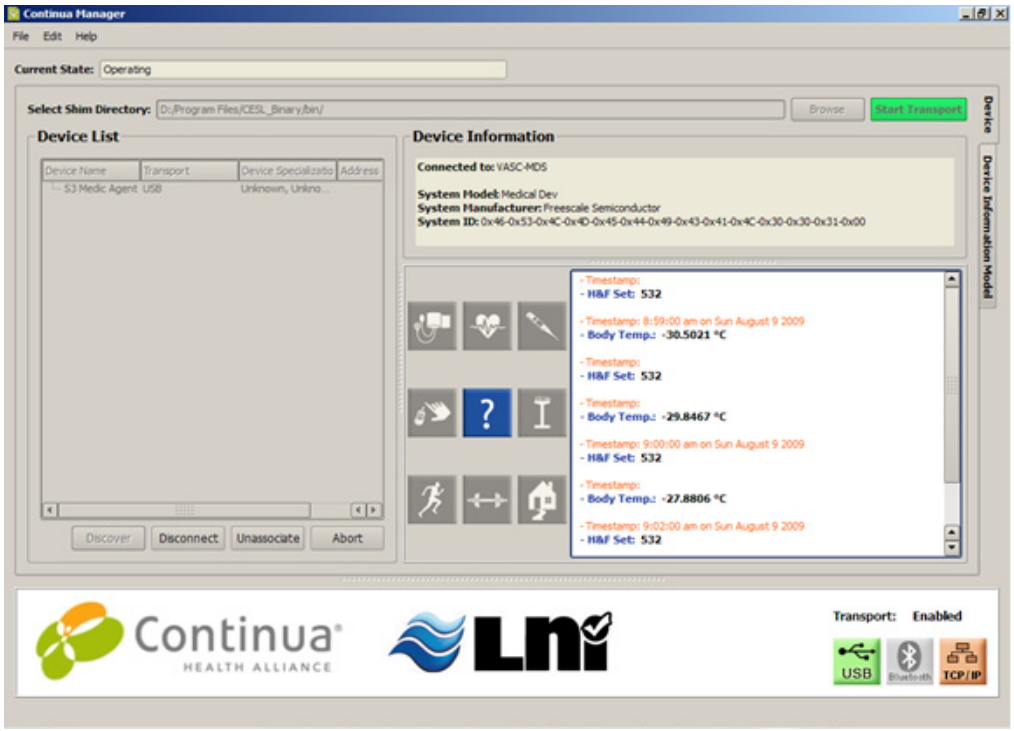


**Figure B-11. Temperature measurement and Enumeration class measurement in Fixed Format**

12. To enable Episoidic Scanner Measurement, go to **DIM** tab and select **Episoidic Scanner** on Continua Host. Click on **Enable Scanning** button.

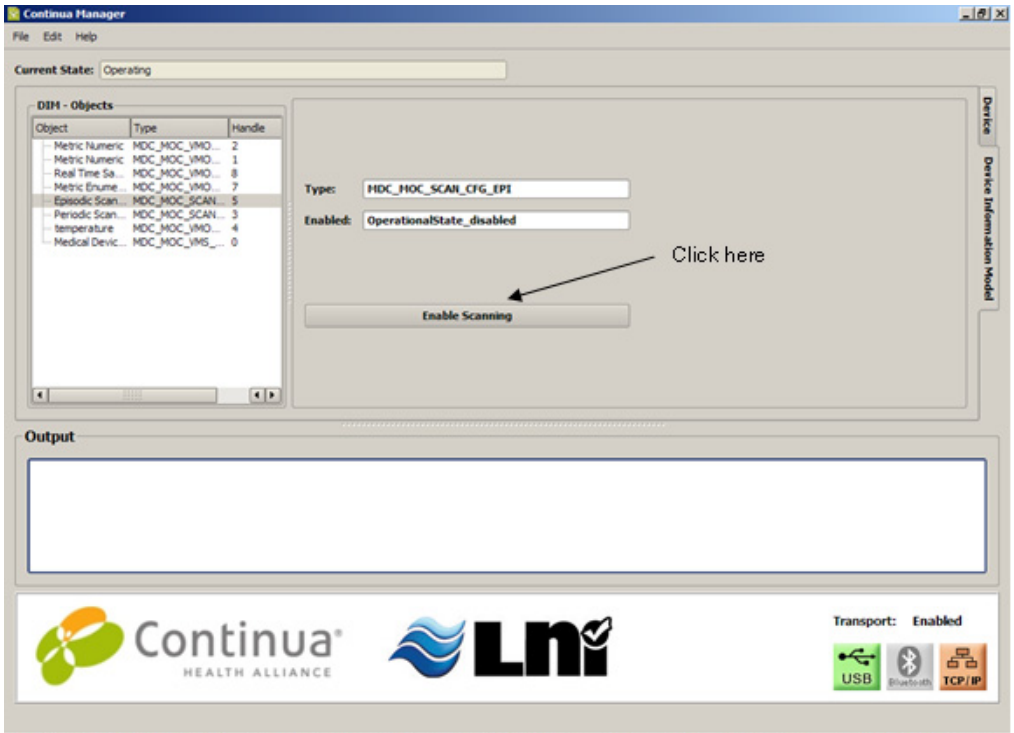


**Figure B-12. Enabling Episoidic Scanner**

13. Verify Episoidic Scanner is successfully enabled.

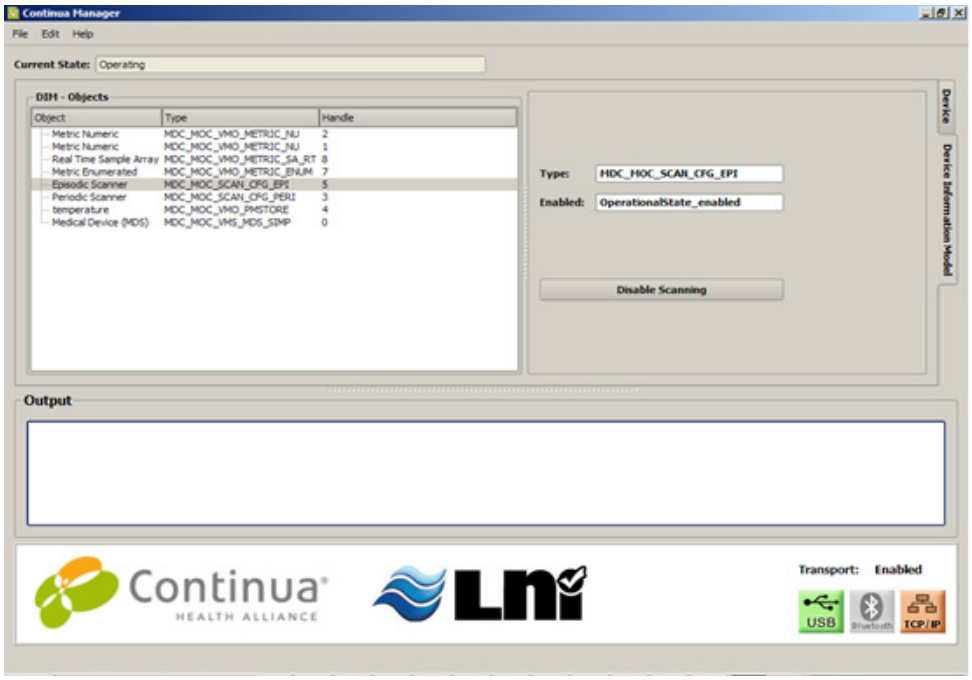


**Figure B-13. Episoidic Scanner enabled**

14. When push button **PTG3** is pressed, Episoidic Scanner measurement is sent in Grouped Format to host. Click on **Device** tab on Continua Host to see Episoidic Scanner data. Figure B-14 below shows measurement data on the host. If Episoidic Scanner is not enabled, then no data is sent to the host.

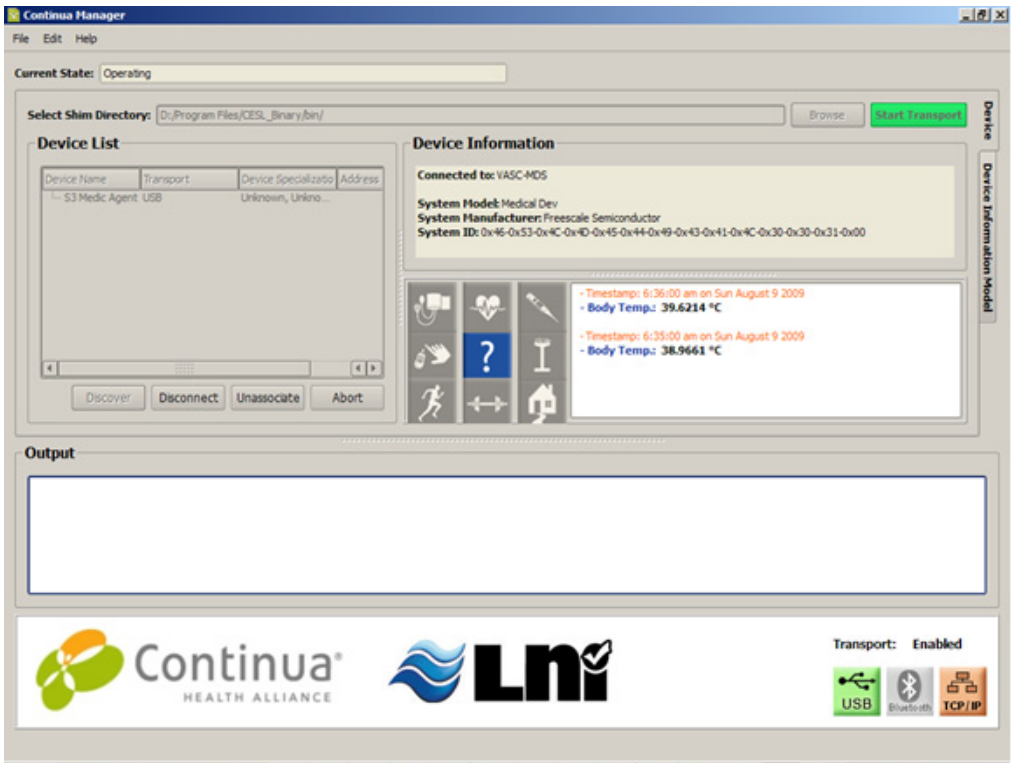


**Figure B-14. Episoidic Scanner measurement**

15. To disable Episoidic Scanner, go to **DIM** tab and select **Episoidic Scanner** on Continua Host. Click on **Disable Scanning** button.

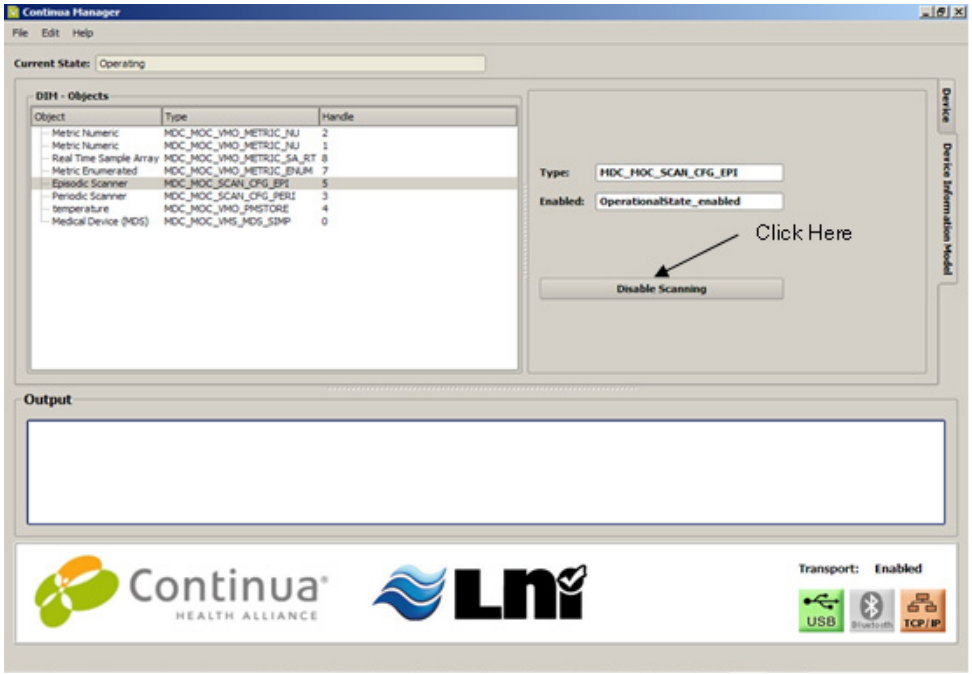


**Figure B-15. Disabling Episoidic Scanner**

16. To enable Periodic Scanner Measurement, go to **DIM** tab and select **Periodic Scanner** on Continua Host. Click on **Enable Scanning** button.

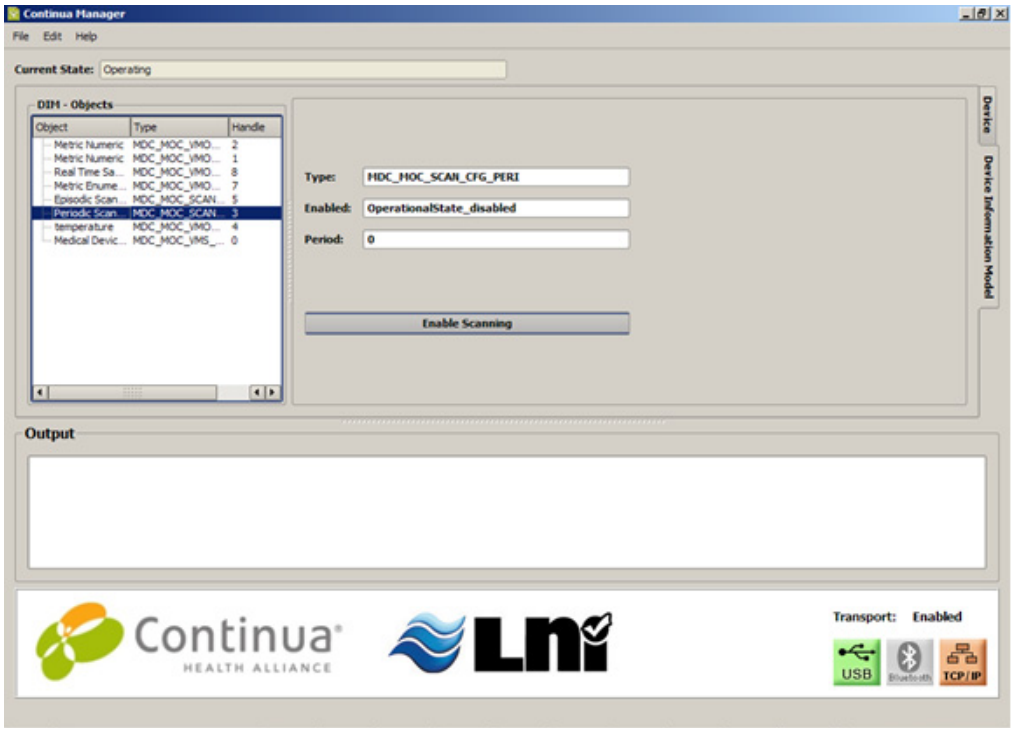


**Figure B-16. Enabling Periodic Scanner**

17. Verify Periodic Scanner is successfully enabled.

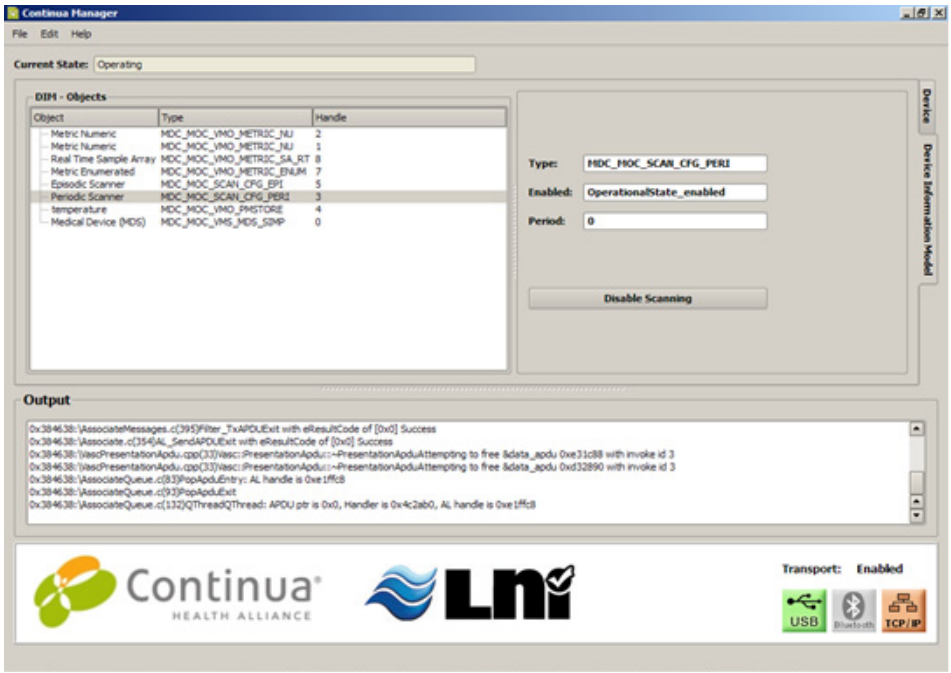


**Figure B-17. Periodic Scanner enabled**

18. Periodic Scanner data is automatically sent by agent. Click on **Device** tab on Continua Host. Measurement data gets updated on screen periodically. Figure B-18 below shows sample Periodic Scanner data.

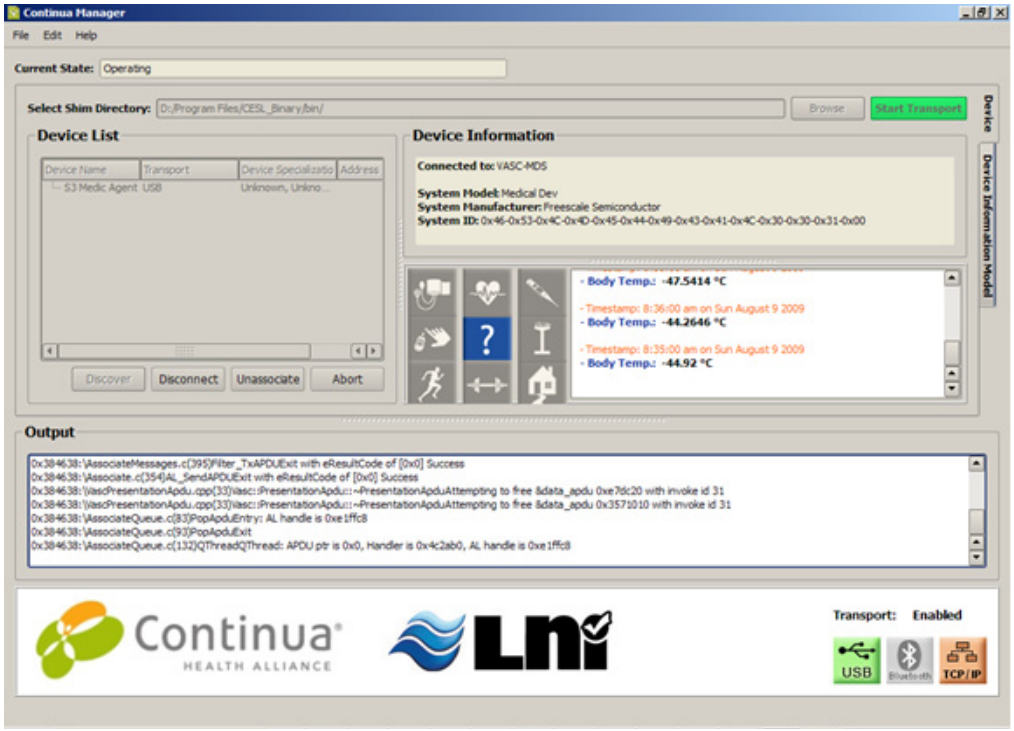


**Figure B-18. Periodic Scanner measurement**

19. To disable Periodic Scanner, go to **DIM** tab and select **Periodic Scanner** on Continua Host. Click on **Disable Scanning** button.

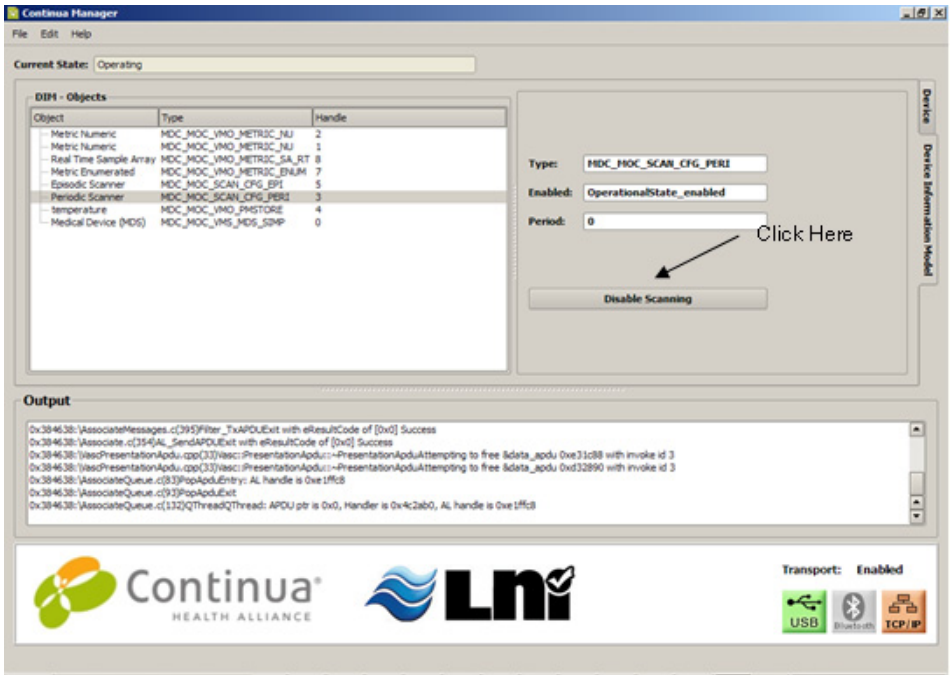


**Figure B-19. Disabling Periodic Scanner**

20. To fetch PM Store information, go to **DIM** tab and select **Temperature** (Name of PM Store) as shown in Figure B-20.

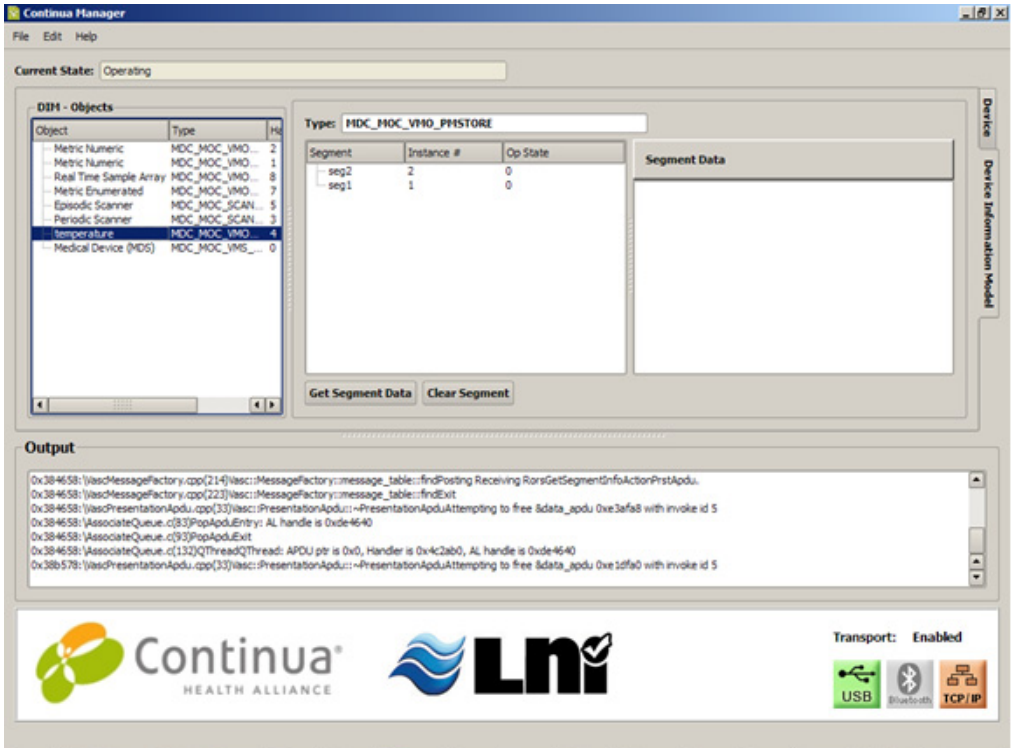


**Figure B-20. PM Store Segment information**

21. To fetch “seg2” data, click on **seg2** and press **Get Segment Data** button.

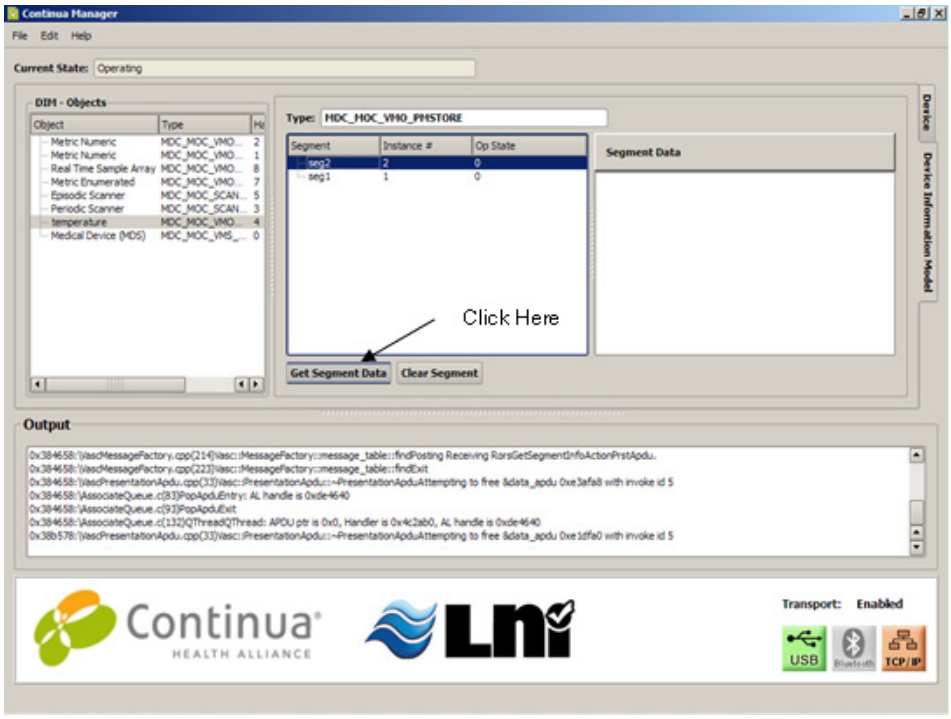


**Figure B-21. Fetch PM Segment data**

22. PM Segment "seg2" data is displayed on Continua Host as shown in Figure B-22.

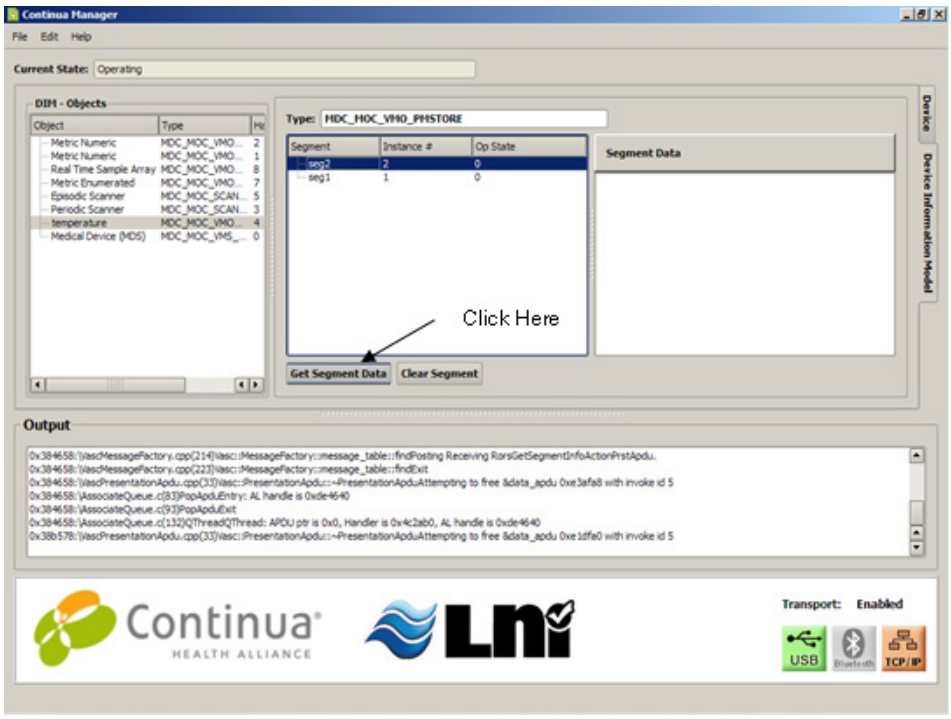


**Figure B-22. PM Segment data**

23. To delete PM Segment ("seg1"), click on **seg1** and then press **Clear Segment** button on Continua Host.

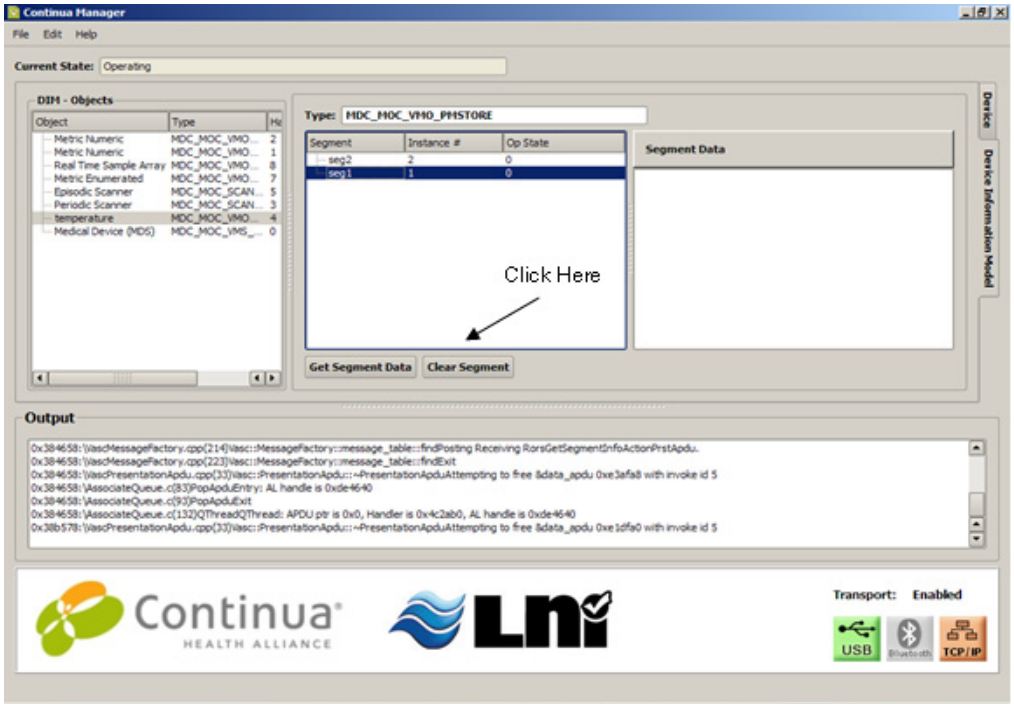


**Figure B-23. PM Segment delete**

24. After PM Segment is deleted, the Continua Manager is updated and "seg1" entry is removed.

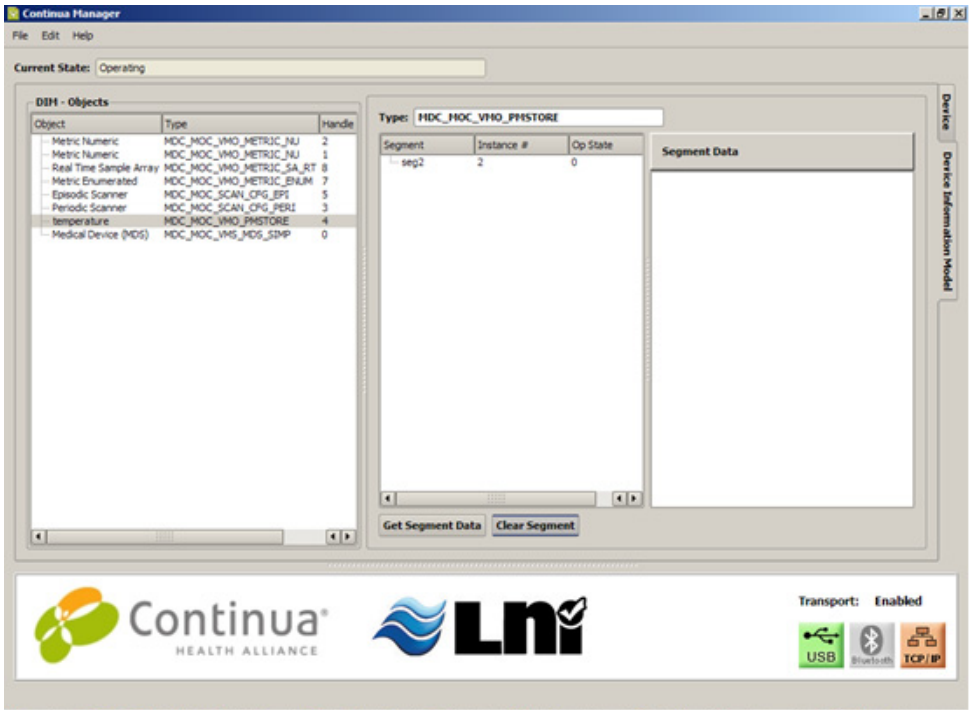


**Figure B-24. PM Segment delete verification**

# Appendix C PAN Serial Bridge Demo

The Serial Bridge demo demonstrates the PAN device demo working on a serial agent. The setup consists of two EVB51JM128 boards. The PHDC Medical Connectivity Library runs on one board with a serial agent. This device communicates with the Continua Host via Serial Bridge which is running on the other board. Continua Host Software detects Serial Bridge Device as PHDC Medical Connectivity Library USB Agent.

## C.1 Setting Up the Demo

1. Set the systems as shown below in Figure C- 1. The Serial connection between the two EVB51JM128 boards uses COM1 port.

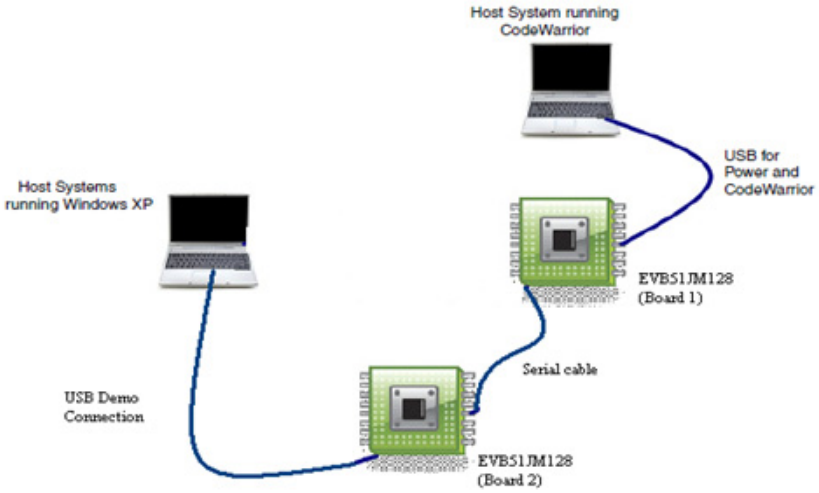


**Figure C-1. Hardware Setup**

2. Get the Continua Alliance (www.continuaalliance.org) CESL Reference Software V1.0 RC2.
3. Install the software on a host system
4. Program EVB51JM128 flash of Board 1 with the PHDC Serial Agent Demo application using CodeWarrior IDE.
5. Program the EVB51JM128 flash of Board 2 with the Serial Bridge demo application using CodeWarrior IDE.

**NOTE**

CESL reference software is not provided as part of the suite. You will have to get this software independently from Continua Alliance.

## C.2 Running the Demo

After the system has been set, you must follow these steps to run the demo:

1. Power ON Serial Agent (Figure C-1 board 1).
2. Power ON Serial bridge (Figure C-1 board 2).
3. To run the demo, follow the steps given in Section B.2, “Running the Demo.” The push buttons used to send measurement data are that of the Serial Agent (Figure C-1 board 1). The Push button panel of EVB51JM128 is shown below in Figure C-2.

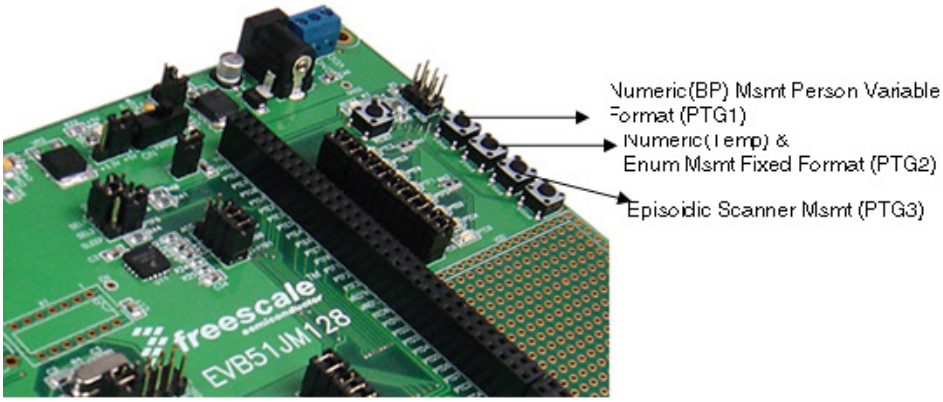


**Figure C-2. EVB51JM128 Push Button Panel**

# Appendix D PHDC Manager Demo Application

## D.1 Setting up the demo

### D.1.1 Hardware setup

This demo runs on M52259DEMO ColdFire board and follows the hardware setup below:

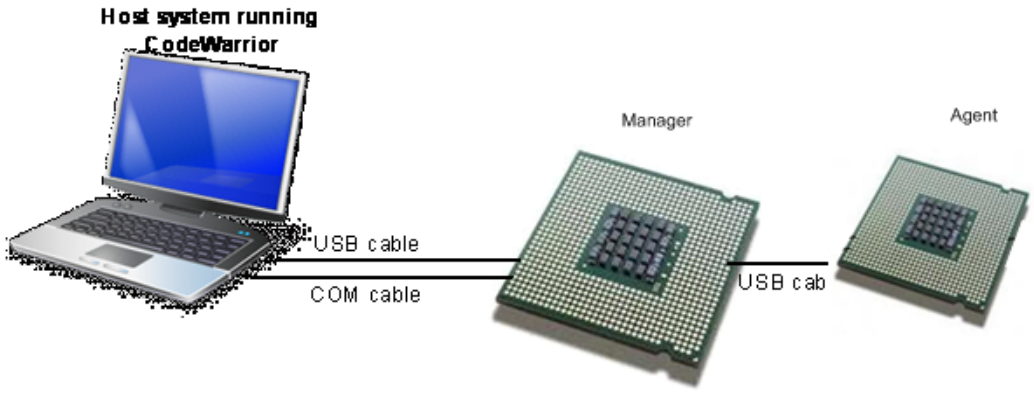


**Figure D-1. PHDC Manager demo setup**

### D.1.2 Set up HyperTerminal to get log

To ensure that application run correctly, the HyperTerminal is used on the PC to get events from the device. These steps are used to configure HyperTerminal:

1. Open HyperTerminal application as shown in Figure D-2

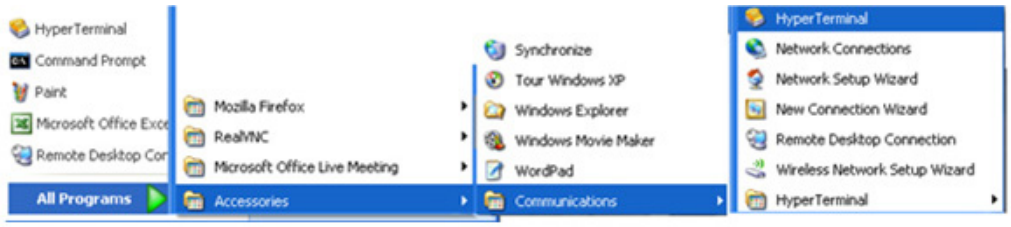


**Figure D-2. Launch HyperTerminal application**

2. The HyperTerminal opens as shown in Figure D-3. Enter the name of the connection and click on the OK button.

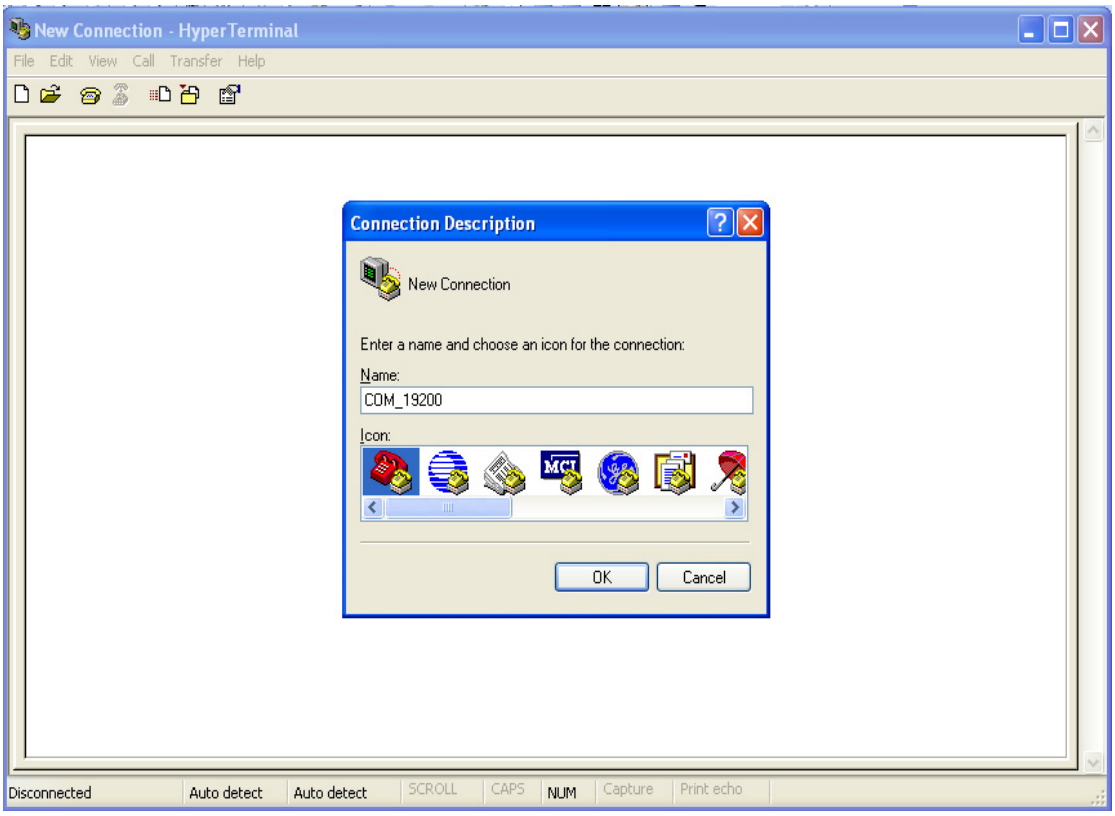


**Figure D-3. Hyper Terminal**

3. The window shown in the following figure appears. Select the COM port.

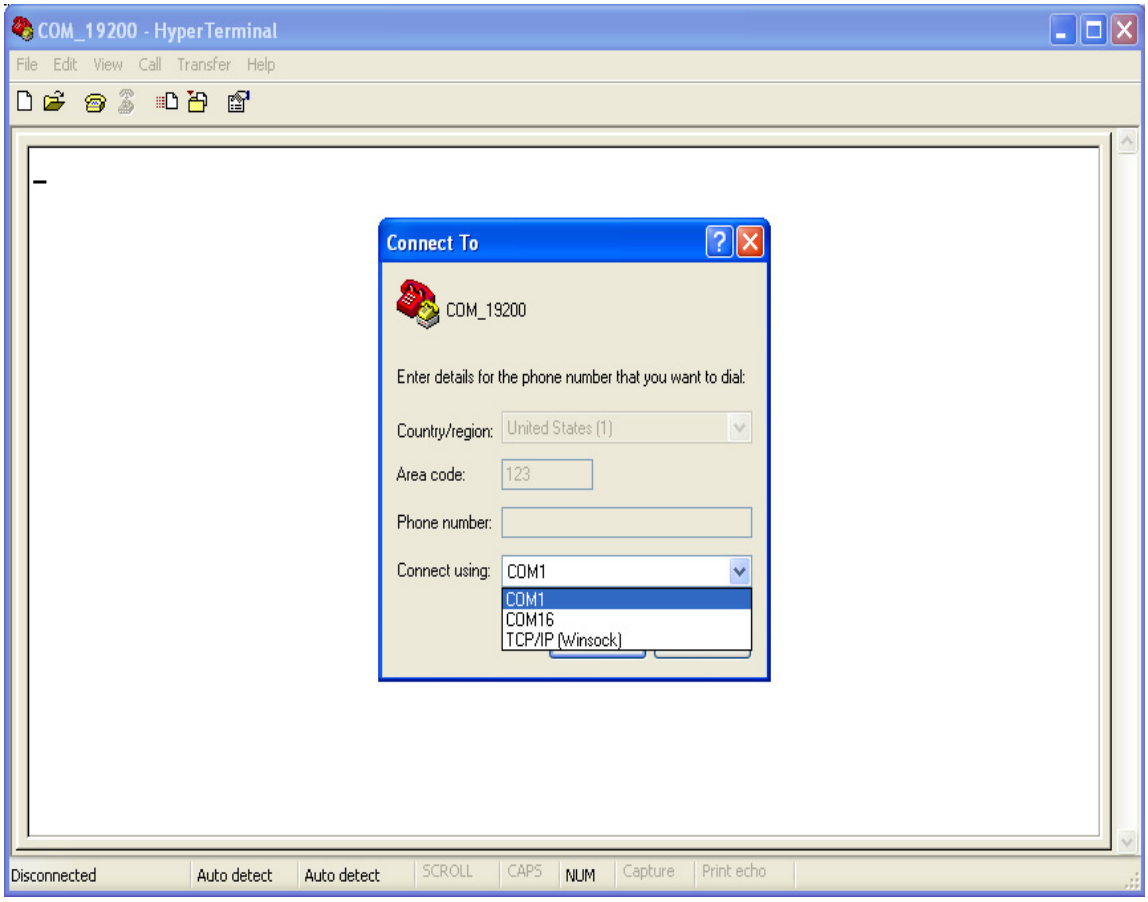


**Figure D-4. Connect using COM port**

4. Configure the COM port baud rate and other properties as shown in Figure D-5

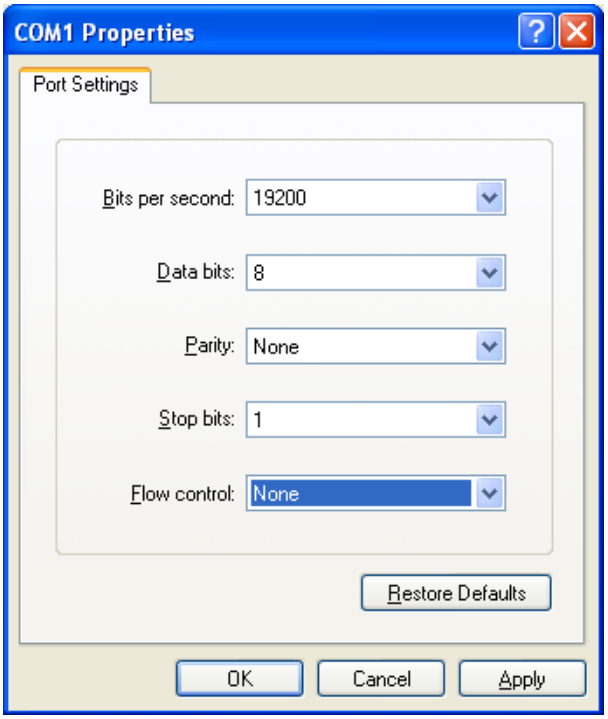


**Figure D-5. COM properties**

5. The HyperTerminal is now configured as shown in Figure D-6

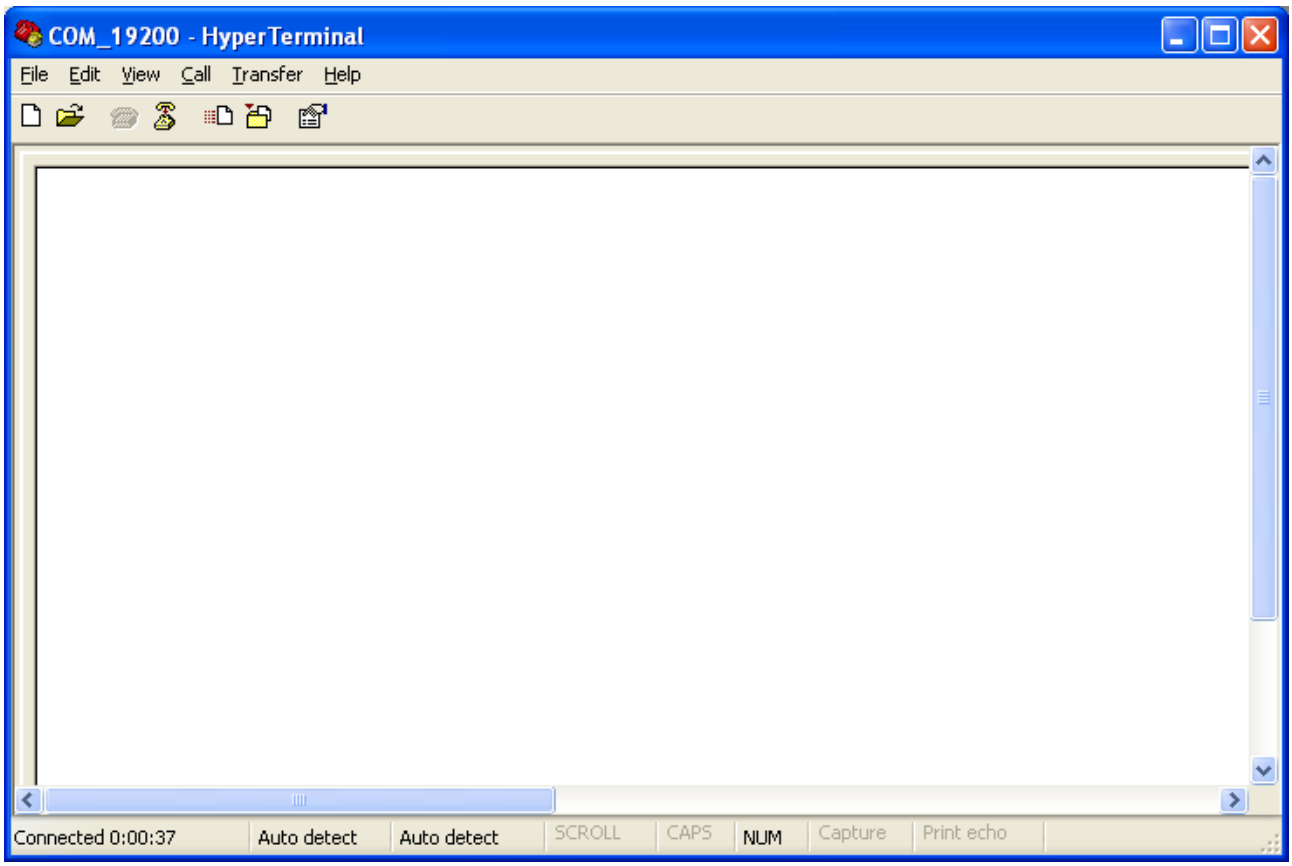


**Figure D-6. HyperTerminal**

### D.1.3 Running the demo

Perform the following steps to run demo:

1. Open and load the image of PHDC manager demo application to the board.
2. After the image has been loaded successfully, HyperTerminal appears as shown in Figure D-7

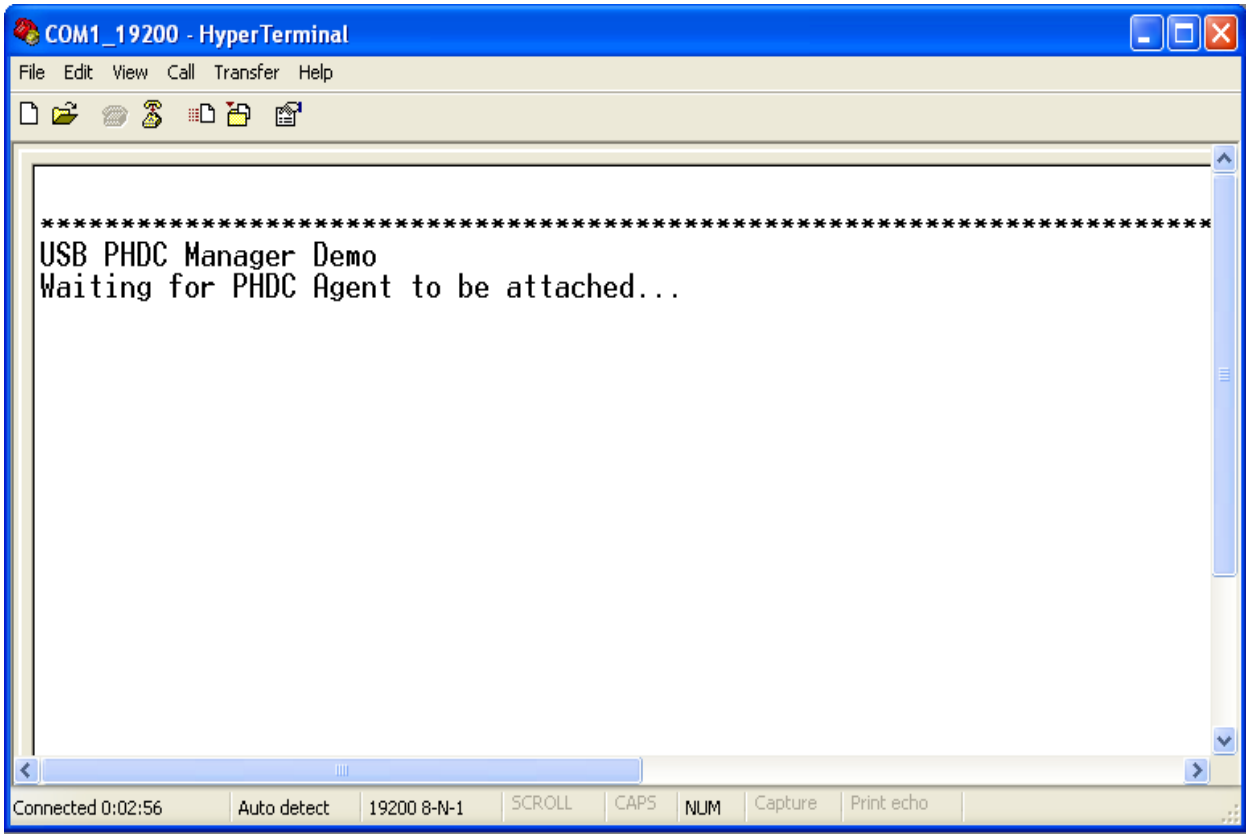


**Figure D-7. Using PHDC Manager Demo attached event**

3. Plug a PHDC agent device into the board. The phdc device examples which is included in this package can be used as PHDC agent device. The agent will be associated with manager and the manager enters operating state.

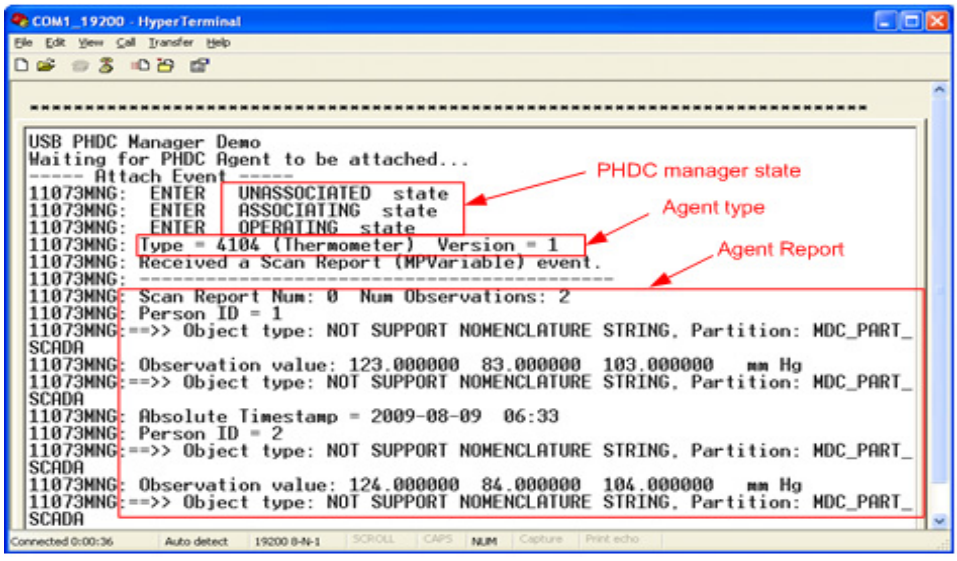


**Figure D-8. USB PHDC Manager Demo operating**