InnoVision Medical

19 South Street,
Baltimore, MD 21202

(443) 992-7450

Getting Started with the InnoVision Medical Device Connector Library (1.0.8)

The InnoVision Medical Device Connector Library simplifies connectivity to medical devices for software developers. The Connector Library supports measurements, settings, alarms, and waveforms. It does this by encapsulating the details of connectivity using a connector abstraction model. The architectural foundation of the Connector Library includes a base DeviceConnector class that is extensible. InnoVision offers various vendor-specific implementations that convert medical device data from the vendor-specific protocol to the current medical device connectivity standards defined by IHE, IEEE, and HL7 consortiums. The Connector Library employs a listener pattern, so developers will need to extend the DeviceListener class to handle the output created by the DeviceConnector.

As a developer, you can set up your connector in four simple steps.

1) Provide a configuration for runtime settings and properties
2) Specify a socket implementation
3) Create a DeviceListener implementation to handle data
4) Initialize the connector with the necessary properties

Step One: Provide a configuration for runtime settings and properties

A default configuration is provided for simplicity. If more functionality is desired, the IMLConfig class can be extended.

```
std::shared_ptr<DefaultConfig> config = std::make_shared<DefaultConfig>();
```

Step Two: Specify a socket implementation

InnoVision provides a default socket implementation; however, if more control over network details is desired then the SocketFactory class can be extended.

```
  std::shared_ptr<SocketFactory> sockFactory(new DefaultSocketFactoryImpl());
  config->setSocketFactory(sockFactory);
```

Step Three: Create a DeviceListener implementation to handle data

```
std::shared_ptr<ExampleDeviceListener> deviceListener =
std::make_shared<ExampleDeviceListener>();
  config->setDeviceListener(deviceListener);
```

Step Four: Initialize the connector with the necessary properties

Each connector requires a common software configuration and a properties abstraction. The configuration is represented by our API's configuration model and has attributes that all connectors will use such as the socket implementation. The properties are an abstraction based on variations in the vendor-specific protocol. Properties may vary by protocol and the available properties are included in each connector's documentation.

Example

Here is an example of how a Dräger ventilator that uses the Medibus protocol would be implemented. The properties below show a serial configuration for 19200 baud rate, 8 data bits, even parity, and 1 stop bit on a Linux serial port device ttyConnector1.

```
std::map<std::string, std::string> props;
props["tty"] = "/dev/ttyConnector1";
props["baud"] = "19200";
props["dataBits"] = "8";
props["parity"] = "even";
props["stopBits"] = "1";
medibus = new MedibusConnector(*config.get(), props);
```

After the initial setup, the only thing left to do is to run the connector! The connector takes care of the details to provide your DeviceListener implementation with the device's data.

```
medibus->run();
```

This article gave an overview of how the InnoVision Medical Device Connector Library can make medical device connectivity simple for software developers. Please reach out with any questions or for more information. info@innovisionmedical.com