jDLMS User Guide

Table of Contents

1. Intro

jDLMS is a Java implementation of the DLMS/COSEM protocol. It can be used to develop individual DLMS/COSEM client/master and server/slave applications.

The jDLMS library supports communication over HDLC/serial, HDLC/TCP and Wrapper/TCP.

2. Using jDLMS

See the jDLMS Javadoc for documentation of the API. You may want to take a look at the source code of the ConsoleClient or the ConsoleServer which is part of the distribution.

2.1. Client

The following is a brief sample client that connects to a server and reads (using the GET service) the list of COSEM objects available in the logical device.

Listing 1. Sample connection from sample source folder.

```
InetAddress inetAddress = InetAddress.getByName("127.0.0.1");
SecuritySuite securitySuite = SecuritySuite.builder()
        .setPassword("Password".getBytes(StandardCharsets.US_ASCII))
        .setAuthenticationMechanism(AuthenticationMechanism.LOW)
        .setEncryptionMechanism(EncryptionMechanism.NONE)
        .build();

TcpConnectionBuilder connectionBuilder = new TcpConnectionBuilder(
inetAddress).setPort(6789)
        .setSecuritySuite(securitySuite)
        .setRawMessageListener(new RawMessageListener() {

            @Override
            public void messageCaptured(RawMessageData rawMessageData) {
                // TODO: log data
                // logger.debug(.. rawMessageData.getMessageSource() ..
            }
        });

try (DlmsConnection dlmsConnection = connectionBuilder.build()) {

    // IC_AssociationLn#OBJECT_LIST
    GetResult result = dlmsConnection.get(new AttributeAddress(15,
"0.0.40.0.0.255", 2));

    if (result.getResultCode() == AccessResultCode.SUCCESS) {
        DataObject resultData = result.getResultData();
        System.out.println(resultData.toString());
    }
} // closes the connection automatically at the end of this block
```

2.2. Server

The jDLMS server libray implements the DLMS/COSEM standard as folowing:

- interface classes (IC) can be defined by annotating the Java class with the CosemClass annotation, in addition the Java class must inherit the CosemInterfaceObject/CosemSnInterfaceObject class.
- COSEM methods and attributes are defined by annotating Java methods and attibutes with CosemMethod and CosemAttribute.
- instances of the a COSEM IC can then be added to a logical.
- logical devices can be registered on a server.

Listing 2. Sample IC implementation from sample source folder.

```
@CosemClass(id = 1, version = 0)
public class Data extends CosemInterfaceObject {

    @CosemAttribute(id = 2, type = Type.LONG64)
    private final DataObject value;

    public Data(DlmsInterceptor interceptor) {
        super("0.0.0.2.1.255", interceptor);

        this.value = DataObject.newInteger64Data(8649726893311919808L);
    }

    public DataObject getValue() {
        return value;
    }

    @CosemMethod(id = 1)
    public void operate() throws IllegalMethodAccessException {
        // implement this
    }

}
```

2.2.1. Annotation Processer

The jDLMS server comes with an annotaion processor. This annotation processor makes it easier/safer to develop a jDLMS server in an IDE like Eclipse or Intellij IDEA.

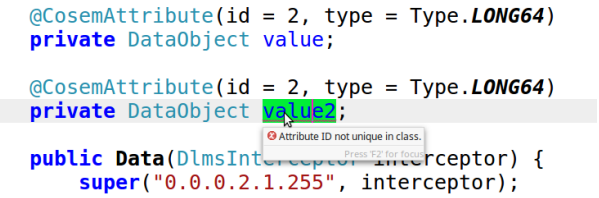

Figure 1. Annotations Processor marks an error in the Eclipse IDE.

The annotation processor can be applied using the following gradle build script. For futher information, see the gradle plugin project page https://github.com/tbroyer/gradle-apt-plugin#gradle-apt-plugin.

Listing 3. Using the jDLMS Annotaion Processor in a Gradle build.

```
plugins {
  id 'net.ltgt.apt-eclipse' version '0.15'
  id 'net.ltgt.apt-idea' version '0.15'
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'

repositories {
    mavenCentral()
}


dependencies {
    compile group 'org.openmuc', name: 'jdlms', version: '1.7.1'
    annotationProcessor group: 'org.openmuc', name: 'jdlms-annotation-
processor', version: '1.7.1'
}
```

2.3. Command Line Application

A command line application is part of the library that can be used to connect, read, write and scan DLMS devices. You can execute it using the *jdlms-console-client* script found in the folder *run-scripts*. Executing the script without any parameters will print help information to the screen.

Instead of running the application from the terminal you can create Eclipse project files as explained in our FAQs and run it from within Eclipse.

3. DLMS/COSEM Concepts

DLMS/COSEM is an application layer protocol specifically designed for communication with smart meters. The DLMS/COSEM standard is adopted by the IEC TC13 WG 14 into the IEC 62056 standard series.

DLMS/COSEM supports multiple lower layer protocols. Communication over TCP, UDP, RS-232, RS-485 and multiple power line protocols such as G3 is supported. Many meters offer an optical interface at the front of the meter. One can communicate over this interface using optical probes that convert the signal to RS-232 and back again.

In DLMS/COSEM the meter is called the *server* (when used over IP) or *slave* (when accessed over HDLC/serial). The entity accessing the meter is called the client or master respectively. In this manual we will always use the terms client and server.

In most cases DLMS/COSEM communication uses either *HDLC* or a special *Wrapper Layer* to add addressing information to the DLMS application layer PDUs.

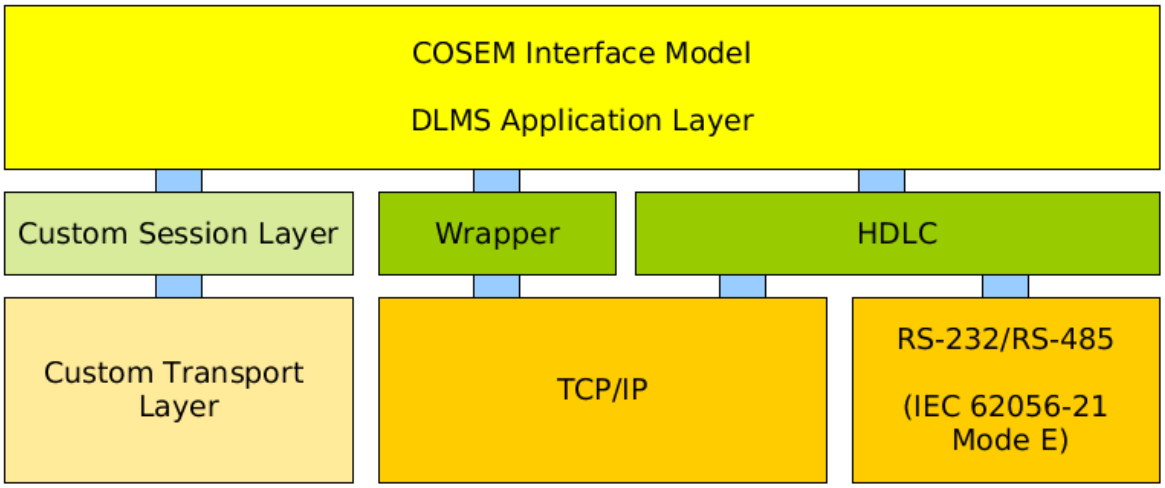

Figure 2. DLMS/COSEM communication layers

3.1. Physical and Logical Devices

In DLMS/COSEM, a physical smart meter device can host several so called logical devices. Each logical device has an address in the range [0, 16383]. As an example, a meter could consist of one logical device for electricity metering at address 18 and another one for a connected gas meter at address 67. No matter what type of smart meter it is, a DLMS/COSEM device must contain a special logical device called the *management logical device* at address 1. The content of this device may vary, but it must at least have a list of all logical devices inside the physical smart meter. You are always connecting to a single logical device with DLMS/COSEM.

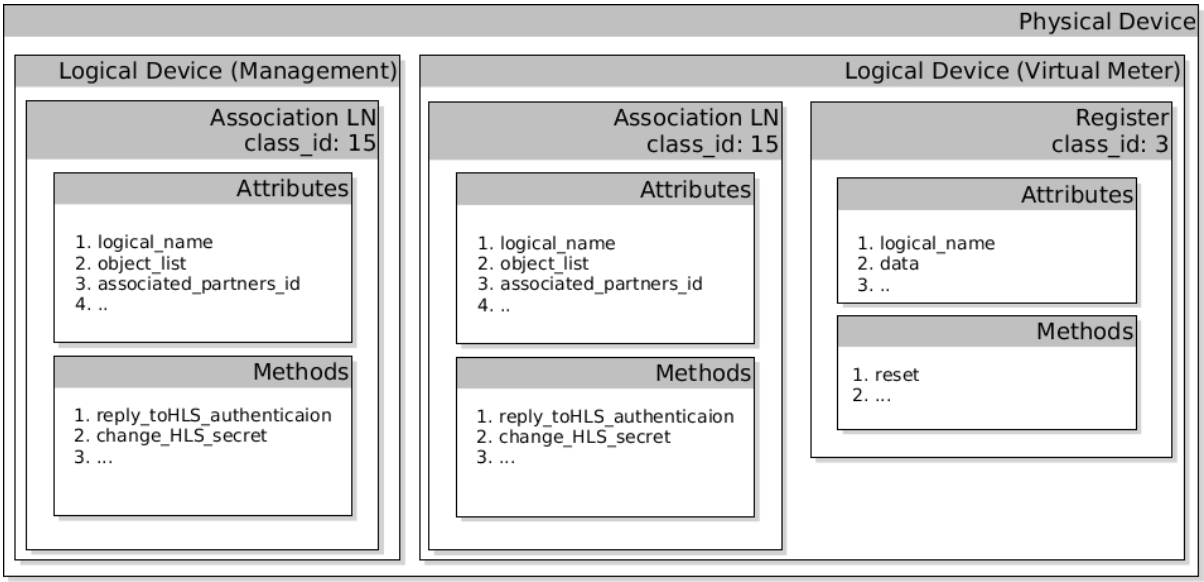

Figure 3. COSEM physical device

3.2. Server Address

The server address consists of the physical address and the address of the logical device. Over TCP/IP the physical address is equal to the IP address and port. Over serial the physical address (also referred to as the lower HDLC address) ranges from 0 to 16383. It can either be left empty (over RS-232) or not (RS-485).

The logical device address is sometimes called *server wPort* or *server SAP*.

3.3. Client ID

The client ID (also called *client SAP* or *client wPort*) defines the access level with which the client connects to the logical device. The client ID 16 is a special client ID, which refers to the *public client* for which no authentication or encryption is required.

3.4. Addressing Attributes and Methods

DLMS/COSEM groups the data of a logical device into objects. These objects are always instances of *interface classes* (IC) that are defined in IEC 62056-6-2.

Each COSEM object can contain several attributes and methods. These are identified using attribute ID and method ID respectively.

Attributes and methods can be addressed in two ways: using *logical name referencing* and *short name referencing*. The referencing method needs is negotiated during connection build up. Logical name referencing is recommend and is more commonly used.

3.4.1. Logical Name Referencing

A logical name is actually an OBIS code. It sometimes also called *instance ID*. It is a 6 byte number that uniquely identifies an object in a logical device. For example, the clock of a smart meter is always reachable under the address *[0, 0, 1, 0, 0, 255]*.

The meaning of an OBIS code is defined in IEC 62056-6-1. A list of all possible logical names is published by the DLMS UA in an MS Excel sheet. OBIS codes are in the format of *A-B:C.D.E*F* where each letter is one byte. Only bytes C and D are mandatory. The six bytes have the following meaning:

A	Medium: Defines the medium (e.g. electricity, gas) of the object. An object related to no medium (e.g. clock) is handled as abstract data.
B	Channel: The channel number allows to differentiate between different inputs (e.g. when a data concentrator is connected to several meters).
C	Defines the abstract of physical data items, related to the information source (e.g. reactive import energy).
D	Defines types, or the result of the processing of physical quantities identified with the value groups **A** and **C**, according to various specific algorithms (e.g. time integral).
E	The value group E defines further processing or classification of quantities identified by value groups **A** to **D** (e.g. tariff number).
F	Defines the storage of the data, identified by **A** to **E** (e.g. different billing periods). May be set to `0xFF` (`255`) where this is irrelevant.

Both attribute and method addresses consist of:

- Class ID
- Logical Name (i.e. Obis code or instance ID)
- Attribute ID / Method ID

The class ID of an object can be found in the Excel sheet mentioned earlier, while the exact attribute ID depends on the class ID and can be extracted the best way by consulting the document IEC 62056-6-2 or the Blue Book from the DLMS UA. Usually the first attribute (attribute ID 1) of an COSEM interface class (IC) is the logical name of the object. Further attributes refer to actual data (see section 4.5 of IEC 62056-6-2).

Example Addresses

Class ID	Logical Name	Attribute ID	Description
`SAP_ASSIGNMENT(17)`	`0.0.41.0.0.255`	`SAP_assignment_list(2)`	Contains the list of all logical devices and their SAP addresses within the physical device.
`ASSOCIATION_LN(15)`	`0.0.40.0.0.255`	`object_list(2)`	Contains the list of visible COSEM objects.

3.4.2. Short Name Referencing

The second way to address an attribute or method is by means of the so called *short address*. Short addresses are used for small devices and should only be used if the connected smart meter cannot communicate using logical names.

When short name referencing is used the meter still holds a unique logical name (i.e. Obis code) for each of its objects. In addition each object has 2 byte short name that maps to the logical name. Thus a client can address each attribute or method using 2 bytes only.

The address space of short addresses is not standardized like the logical names, meaning that the same address can lead to different objects on different devices.

jDLMS maps the Short Names to Logical Names.

3.5. DLMS Services

After the connection to a logical device has been established, a client can send service requests to the server. In order to access an attribute, you may invoke the GET or SET service on one or more attributes in a single request. Similarly the ACTION service may be invoked on methods.

3.5.1. Selective Access

The GET and SET services allow the client to access just a part of a specific attribute. The part is defined by some specific selective access parameters. These selective access parameters are defined as part of the attribute specification.

3.6. Security

The DLMS/COSEM standard differentiates between two security mechanisms: 1) the *authentication mechanism* (i.e. security level) and 2) the *encryption mechanism* (i.e. security suite). The two mechanisms often use the same keys but they can be chosen independently of each other and can be used in any combination.

3.6.1. Authentication Mechanism

The authentication mechanism is the mechanism by which client and server authenticate each other during connection build up. The following authentication mechanisms (also called security levels) exist:

Lowest Level Security (0)

No authentication is used.

Low Level Security (1, LLS)

Only the client is authenticated using a plain text password.

High Level Security (>1, HLS)

In this case both client and server are authenticated. First both client and server exchange *challenge strings* (e.g. a random strings). Then both use cryptographic algorithms on these *challenge strings* and send the result back. The cryptographic algorith used for authentication depends on the HLS level:

HLS 2

A manufacturer specific algorithm is used that is not standardized.

HLS 3

MD5 is used.

HLS 4

SHA-1 is used

HLS 5

Galois Message Authentication Code (GMAC) is used. The algorithm takes both the global encryption key (16 bytes) as well as the authentication key (16 bytes) as inputs.

HLS 6

using SHA-256

HLS 7

using ECDSA

The jDLMS library supports lowest level, low level as well as HLS 5 authentication at the moment.

3.6.2. Encryption Mechanism

The encryption mechanism is used to encrypt messages and/or add authentication tags to individual messages. Until recently only one encryption mechanism (also called security suite) existed. It has the the ID 0. By now new security mechanisms have been defined.

Encryption mechanism 0 is based on AES-GCM-128. It is the only mechanism currently supported by jDLMS. It uses the global unicast encryption key and, if available, the authentication key.

Optionally a client may send a so called dedicated key (i.e. a session key) to the server during connection build up. The dedicated key is then used instead of the global encryption key for the remaining communication of this connection. The dedicated key is a temporary key that is usually generated ad-hoc at connection time.

3.6.3. Keys

Several different keys exist which are used by the different security mechanisms.

Password/Secret

Used by LLS. The client sends the password to the server for authentication.

Authentication key

Used by authentication mechanisms such as HLS 5 as well as optionally by encryption mechanisms such as mechanism 0.

Global unicast encryption key

Used by authentication mechanisms such as HLS 5 as well as by encryption mechanisms such as mechanism 0.

Dedicated encryption key

A temporary session key that can optionally be used instead of the global encryption key by encryption mechanisms.

Master key

The master key is used for wrapping global keys. When changing the global encryption or authentication keys one has to wrap the new key before transmitting it.

4. Distribution

After extracting the distribution tar file the jDLMS library can be found in the directory *build/libs-all*. For license information check the *license* directory in the distribution.

4.1. Dependencies

Besides the jDLMS library the folder *build/libs-all/* contains the following external libraries:

jASN1

A library for ASN.1 BER encoding/decoding, MPLv2.0, https://www.openmuc.org/asn1/

jRxTx

This library for serial communication is only needed by jDLMS if communication over HDLC/serial is used. jRxTx is a fork of RxTx and is hosted at https://github.com/openmuc/jrxtx. The library is licensed under the LGPL(v2.1 or later) + linking exception.

jRxTx consists of a Java part located in the *dependencies* folder of the distribution and a native part that is equivalent to that of RxTx. Note that you have to install the native part of RxTx as explained in our FAQ.

Bouncy Castle (bcprov)

A library needed for security algorithms. It is only needed if jDLMS is used with encryption or high level authentication enabled.

The *dependencies* folder in the distribution contains more detailed license and copyright information about these dependencies.

5. Modifying and Compiling jDLMS

We use the Gradle build automation tool. The distribution contains a fully functional gradle build file (*build.gradle*). Thus if you changed code and want to rebuild a library you can do it easily with Gradle. Also if you want to import our software into Eclipse you can easily create Eclipse project files using Gradle. Just follow the instructions on our FAQ site.

5.1. A-XDR Compiler

The jDLMS distribution contains a subproject named *A-XDR Compiler*. It is an application developed to automatically compile the DLMS/COSEM ASN.1 data structures and generate classes that can be used to encode/decode A-XDR messages. The generated classes are part of jDLMS. You will *not* need this application if you simply want to use the jDLMS library.

6. Terminology

Client ID

Synonyms: client wPort, client SAP

COSEM

Companion Specification for Energy Metering

DLMS

Device Language Message Specification

HDLC

High-Level Data Link Control

HLS

High Level Security

Logical Device Address

Synonyms: server wPort, server SAP

OBIS

Object Identification System

SAP

Service Access Point

7. Authors

Developers:

- Dirk Zimmermann
- Albrecht Schall

Former developers:

- Stefan Feuerhahn
- Karsten Müller-Bier else::[]