Protocol

Table of Contents

Supported encoding rules

Restcomm ASN Library supports following the encoding rules:

- BER

API

Restcomm ASN Library is stream oriented. The user accesses ASN primitives by means of stream objects capable of proper decoding and encoding.

The following classes deserve explanation:

`org.mobicents.protocols.asn.Tag`
: This class defines static values that are part of header(Tag). Example values are tag values for Integer, BitString, etc.

`org.mobicents.protocols.asn.BERStatics`
: This class defines some static values that are specific for BER encoding, such as real encoding schemes(NR1,NR2...).

`org.mobicents.protocols.asn.External`
: This is a special class that is used to represent the "External" type. It is a special ASN type where "anything" can be used.

Input and Output stream
: Simple classes that are the core of this library. They allow for chunks of data to be read/written.

Examples

Simple decode integer primitive example:

```
// integer -128
byte[] data = new byte[] { 0x2, 0x1, (byte) 0x80 }; //encoded form
ByteArrayInputStream baIs = new ByteArrayInputStream(data);
AsnInputStream asnIs = new AsnInputStream(baIs);
int tag = asnIs.readTag();
if(Tag.INTEGER==tag)
{
    long value = asnIs.readInteger();
    //do somethin
}
```

Simple encode Real primitive example:

```
AsnOutputStream output = new AsnOutputStream();
output.writeReal(-3145.156d, BERStatics.REAL_NR1);
```

Complex example - how to decode some constructed data structure:

```
// mandatory
    private Long invokeId;

    // optional
    private Long linkedId;

    // mandatory
    private OperationCode operationCode;

    // optional
    private Parameter parameter;

public void doDecoding( AsnInputStream ais )
{

    int len = ais.readLength();
    if (len == 0x80) {
        throw new ParseException("Unspiecified length is not supported.");
    }

    byte[] data = new byte[len];
    if (len != ais.read(data)) {
        throw new ParseException("Not enough data read.");
    }

    AsnInputStream localAis = new AsnInputStream(new ByteArrayInputStream(data));

    int tag = localAis.readTag();
    if (tag != _TAG_IID) {
        throw new ParseException("Expected InvokeID tag, found: " + tag);
    }

    this.invokeId = localAis.readInteger();

    if (localAis.available() <= 0) {
        return;
    }

    tag = localAis.readTag();

    if (tag == Tag.SEQUENCE) {
        // sequence of OperationCode

        len = localAis.readLength();
```

```
        if (len == 0x80) {
            throw new ParseException("Unspiecified length is not supported.");
        }

        data = new byte[len];
        int tlen = localAis.read(data);
        if (len != tlen) {
            throw new ParseException("Not enough data read. Expected: " + len + ",
actaul: " + tlen);
        }
        AsnInputStream sequenceStream = new AsnInputStream(new ByteArrayInputStream
(data));

        tag = sequenceStream.readTag();
        if (tag == OperationCode._TAG_GLOBAL || tag == OperationCode._TAG_LOCAL) {
            this.operationCode = TcapFactory.createOperationCode(tag, sequenceStream);
        } else {
            throw new ParseException("Expected Global|Local operation code.");
        }

        if (sequenceStream.available() > 0) {
            tag = sequenceStream.readTag();
            this.parameter = TcapFactory.createParameter(tag, sequenceStream);

        } else {
            throw new ParseException("Not enought data to decode Parameter part of
result!");
        }
    } else {
        throw new ParseException("Expected SEQUENCE tag for OperationCode and
Parameter part, found: " + tag);
    }
}
```