User Guide to Restcomm ASN Library 2.2.0-143

Table of Contents

Preface

Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the Liberation Fonts set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file *my_next_bestselling_novel* in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press `Enter` to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press `Enter` to execute the command.
>
> Press to switch to the first virtual terminal. Press to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

`Proportional Bold`

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For

example:

> Choose **System** > **Preferences** > **Mouse** from the main menu bar to launch `Mouse Preferences`. In the Buttons tab, click the Left-handed mouse check box and click **[Close]** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a `gedit` file, choose **Applications** > **Accessories** > **Character Map** from the main menu bar. Next, choose **Search** > **Find ›] from the `Character Map` menu bar › type the name of the character in the Search field and click [Next**. The character you sought will be highlighted in the Character Table. Double-click this highlighted character to place it in the Text to copy field and then click the **[Copy]** button. Now switch back to your document and choose **Edit** > **Paste** from the `gedit` menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the menu:>[] shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select from the **Preferences ›] sub-menu in the menu:System[** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or `Proportional Bold Italic`

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh username@domain.name` at a shell prompt. If the remote machine is *example.com* and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount file-system` command remounts the named file system. For example, to remount the */home* file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q package` command. It will return a result as follows: `package-version-release`.

Note the words in bold italics above —username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }

}
```

Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the {this-issue.tracker.ur}, against the product Restcomm ASN Library` `, or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: Restcomm ASN Library

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Chapter 1. Introduction to Restcomm ASN Library

Abstract Syntax Notation One (ASN.1) is the standard for describing data structures in telecommunication and computer networking world. ASN.1 provides a set of formal rules for describing the structure of objects. The specification describes abstract objects that are independent of machine-specific encoding techniques.

ASN defined data can be encoded using one of these encoding rules:

- Basic Encoding Rules (BER)
- Canonical Encoding Rules (CER)
- Distinguished Encoding Rules (DER)
- XML Encoding Rules (XER)
- Packed Encoding Rules (PER)
- Generic String Encoding Rules (GSER)

ASN.1, together with specific ASN.1 encoding rules, facilitates the exchange of structured data between application programs over networks by describing data structures in a way that is independent of machine architecture and implementation language.

ASN encoded data looks logically as follows:

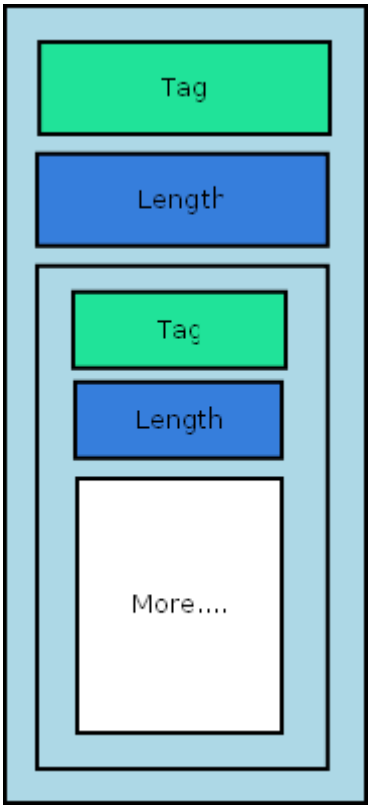

Figure 1. ASN encoding logical overview

Encoded data structure contains three elements:

Tag

Unique value, which identifies the type of data.

Length

Indicates the length of the current data structure.

Payload

Depending on the definition, this can be a simple value (like an integer), or it can carry another ASN encoded data structure.

Chapter 2. Setup

2.1. Restcomm ASN Library Source Code

2.1.1. Release Source Code Building

1. Downloading the source code

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at http://git-scm.com/

Use Git to checkout a specific release source, the Git repository URL is https://github.com/Restcomm/jasn, then switch to the specific release version, lets consider 2.2.0-143.

```
[usr]$ git clone git@github.com:RestComm/jasn.git
```

2. Building the source code

Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at http://maven.apache.org

Use Maven to build the binaries.

```
                    [usr]$ cd asn-2.0.4-SNAPSHOT
                    [usr]$ mvn install
```

Once the process finishes you should have the `binary` jar files in the *target* directory of `module`.

2.1.2. Development Trunk Source Building

Similar process as for Release Source Code Building, the only change is the GIT source code URL, which is https://github.com/Restcomm/jasn.

Chapter 3. Design Overview

Restcomm ASN Library is subject to changes as it is under active development.

Restcomm ASN Library has been designed as a simple library that enables the user to encode and decode streams according to ASN rules. It provides the user with the tools to process primitives and build more complex objects as defined in ASN.

Restcomm ASN Library does not provide an `ASN compiler`. Its sole purpose is to avoid costly processing and allow the user to implement the desired functionality in the optimal way.

Chapter 4. Protocol

4.1. Supported encoding rules

Restcomm ASN Library supports following the encoding rules:

- BER

4.2. 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.

4.3. 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);
    }
}
```

Appendix A: Revision History