

# **JBoss Communications Stream Library User Guide**

by Amit Bhayani, Bartosz Baranowski, and Oleg Kulikov

---

---

---

Preface .....	v
1. Document Conventions .....	v
1.1. Typographic Conventions .....	v
1.2. Pull-quote Conventions .....	vii
1.3. Notes and Warnings .....	vii
2. Provide feedback to the authors! .....	viii
<b>1. Introduction to JBoss Communications Stream Library .....</b>	<b>1</b>
<b>2. Setup .....</b>	<b>3</b>
2.1. Pre-Install Requirements and Prerequisites .....	3
2.1.1. Hardware Requirements .....	3
2.1.2. Software Prerequisites .....	3
2.2. JBoss Communications Stream Library Source Code .....	3
2.2.1. Release Source Code Building .....	3
2.2.2. Development Trunk Source Building .....	4
<b>3. Design Overview .....</b>	<b>5</b>
3.1. Stream protocol - Datalink .....	5
<b>4. Source overview &amp; Example .....</b>	<b>9</b>
4.1. Stream .....	9
4.2. Datalink .....	12
A. Revision History .....	17
Index .....	19

---

---

## Preface

# 1. 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](https://fedorahosted.org/liberation-fonts/) [https://fedorahosted.org/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.

## 1.1. 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 **Ctrl+Alt+F1** to switch to the first virtual terminal. Press **Ctrl+Alt+F7** 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 **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

*Mono-spaced Bold Italic Of 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.

## 1.2. 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"));
    }
}
```

## 1.3. 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.

## 2. 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 [Issue Tracker](http://bugzilla.redhat.com/bugzilla/) [http://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications Stream Library** , or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: StreamLibrary\_User\_Guide

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.



# Introduction to JBoss Communications Stream Library

In many places there is need for asynchronous I/O operations, be it file, socket or any other. There are asynchronous I/O libraries. However implementations are thread safe and restricted to some specific assumptions.

JBoss Communications Stream Library fills this gap. It aims to provide simple API which enables user to abstract I/O operation in any way he desires.



# Setup

## 2.1. Pre-Install Requirements and Prerequisites

Ensure that the following requirements have been met before continuing with the install.

### 2.1.1. Hardware Requirements

The library doesn't change the JBoss Communications Hardware Requirements.

### 2.1.2. Software Prerequisites

The library does not have dependencies.

## 2.2. JBoss Communications Stream Library Source Code

### 2.2.1. Release Source Code Building

#### 1. Downloading the source code



#### Important

Subversion is used to manage its source code. Instructions for using Subversion, including install, can be found at <http://svnbook.red-bean.com>

Use SVN to checkout a specific release source, the base URL is ?, then add the specific release version, lets consider 1.0.0.BETA1.

```
[usr]$ svn co ?/1.0.0.BETA1 stream-1.0.0.BETA1
```

#### 2. Building the source code



#### Important

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 stream-1.0.0.BETA1  
[usr]$ mvn install
```

Once the process finishes you should have the `binary jar` file in the `target` directory.

### 2.2.2. Development Trunk Source Building

Similar process as for [Section 2.2.1, “Release Source Code Building”](#), the only change is the SVN source code URL, which is NOT AVAILABLE.

## Design Overview

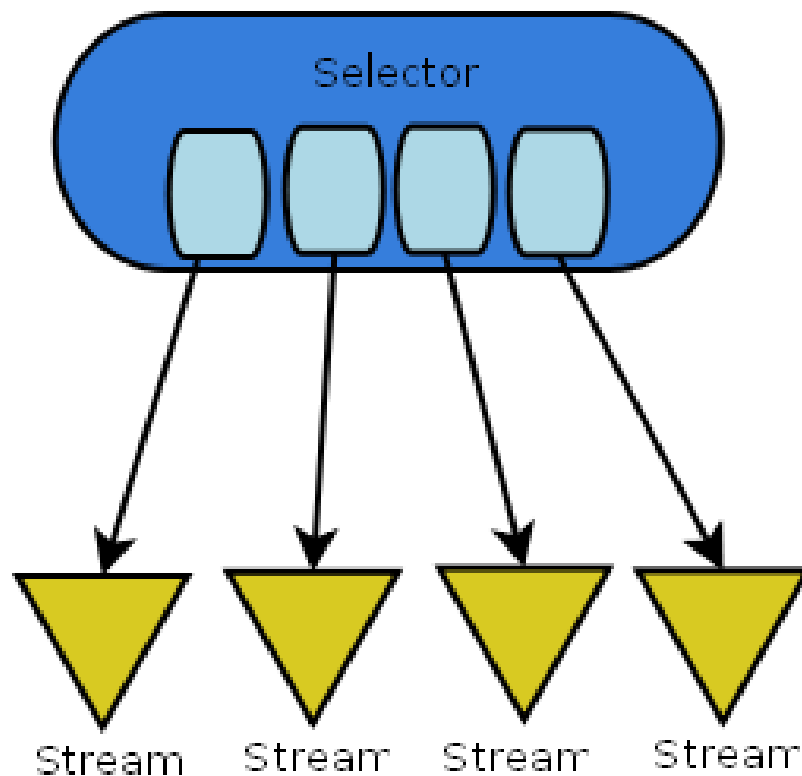


### Important

Be aware, JBoss Communications Stream Library is subject to changes as it is under active development!

JBoss Communications Stream Library builds layer of abstraction with API similar to NIO . Abstraction is built with three main components:

- Selector - performs stream queries and presents user with streams ready for I/O operations.
- SelectorKey - represents stream in selectors space
- Stream - straight as it is, a stream of data



IO overview

### 3.1. Stream protocol - Datalink

Beside simplified stream API , there is also need for SCTP(SCTP is not available in JDK6) like streaming. JBoss Communications Stream Library provides such feature over defined stream API .

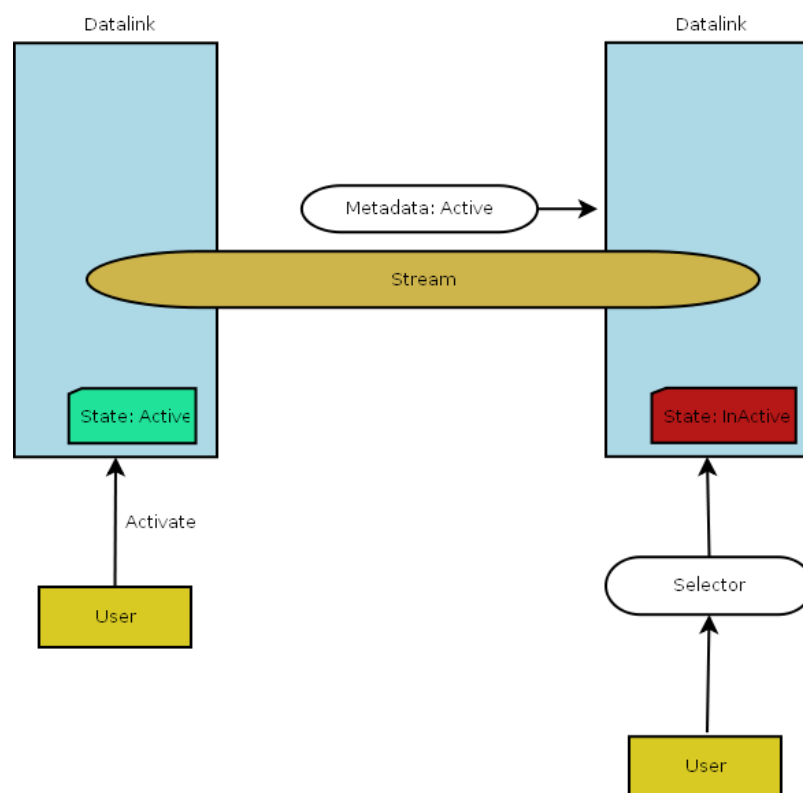
## Chapter 3. Design Overview

---

Datalink is a proprietary protocol designed to provide following:

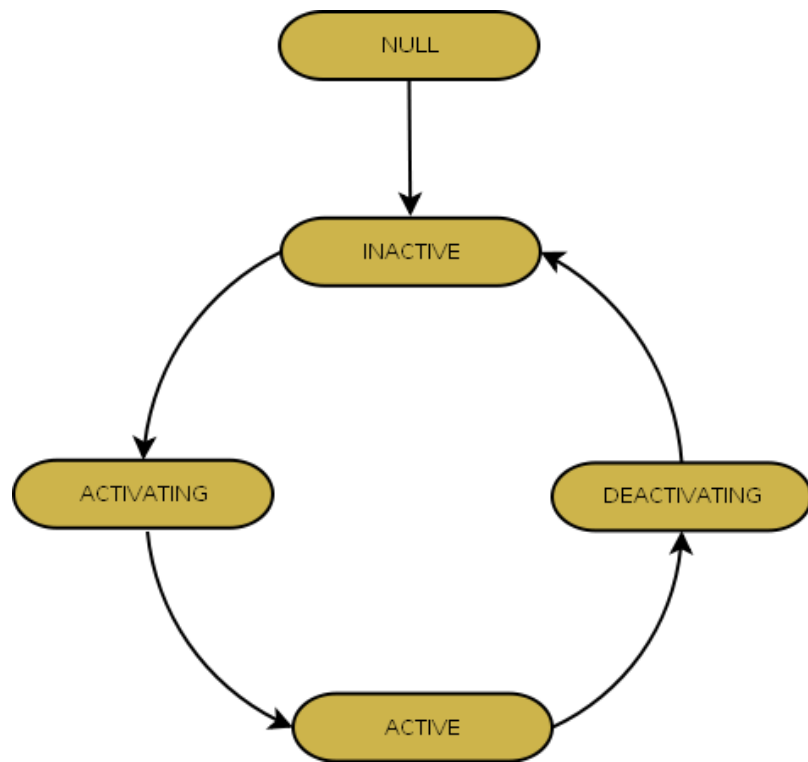
- peer to peer transport
- reliable delivery - retransmissions
- stream status metadata - provide user with stream status info(active,inactive, down...). Metadata is information which informs peers about readiness to receive data for instance.
- async I/O

Generally Datalink can be imagined as follows:



Datalink overview

Datalink follows state machine which reassembles one depicted on diagram below:



Datalink FSM





# Source overview & Example

## 4.1. Stream

As mentioned stream module is centered on three main interfaces:

- org.mobicens.protocols.stream.api.Stream - this class declares sets of methods to perform read and write operations, it looks as follows:

```
public interface Stream {

    /**
     * Registers this stream with the given selector, returning a selection key.
     * This method first verifies that this channel is open and that the given initial
     * interest set is valid.
     *
     * If this stream is already registered with the given selector then the selection key
     * representing that registration is returned after setting its interest set to the
     * given value.
     *
     * @param selector
     * @param op The selector with which this channel is to be registered
     * @return
     */
    public SelectorKey register(StreamSelector selector) throws IOException;

    public int read(byte[] b) throws IOException;

    public int write(byte[] d) throws IOException;

    /**
     * Closes this streamer implementation. After closing stream its selectors are invalidated!
     */
    public void close();

    /**
     * Returns the provider that created this stream.
     *
     * @return
     */
    public SelectorProvider provider();
}
```

```
}
```

- org.mobicens.protocols.stream.api.StreamSelector - this interface defines methods which are used to interrogate registered stream for IO readiness.

```
public interface StreamSelector {

    public static final int OP_READ = 0x1;
    public static final int OP_WRITE = 0x2;

    /**
     * Performs query of registeres stream. Returns set of keys pointing to streams ready to perform IO.
     * @param operation - operation which streams are queried. Value is equal to on of OP_X.
     * @param timeout
     * @return
     * @throws IOException
     */
    public Collection<SelectorKey> selectNow(int operation, int timeout) throws IOException;

    /**
     * Checks if selector has been closed.
     * @return
     */
    public boolean isClosed();

    /**
     * closeses selector, removes all stream from internal register.
     */
    public void close();

    /**
     * Returns registered streams.
     * @return
     */
    public Collection<Stream> getRegisteredStreams();
}
```

- org.mobicens.protocols.stream.api.SelectorKey - this interface declares contact for object representing stream in selector:

```
public interface SelectorKey {  
    /**  
     * Attach application specific object to this key. When underlying stream is  
     * ready for IO and key is returned, this attachment will be accessible.  
     *  
     * @param obj  
     */  
    public void attach(Object obj);  
  
    /**  
     * Gets attachemnt.  
     *  
     * @return  
     */  
    public Object attachment();  
  
    /**  
     * Returns validity indicator.  
     *  
     * @return  
     */  
    public boolean isValid();  
  
    /**  
     * Indicates if underlying stream is ready to read.  
     *  
     * @return  
     */  
    public boolean isReadable();  
  
    /**  
     * Indicates if underlying stream is ready to write.  
     *  
     * @return  
     */  
    public boolean isWritable();  
  
    /**  
     * Returns stream associated with this key  
     *  
     * @return  
     */  
    public Stream getStream();  
}
```

```
/**
 * Get selector for this key.
 *
 * @return
 */
public StreamSelector getStreamSelector();

/**
 * Cancels this key. Equals deregistration of stream
 */
public void cancel(); // Oleg verify this.
}
```

Example use of this API looks as follows:

```
Stream s = ....
StreamSelector selector = ...
s.register(selector);

while(true)
{
    byte[] buff = new byte[...];
    Collection<SelectorKey> selected = selector.selectNow(selector.OP_READ,0); //0,
    immediate check
    for(SelectorKey key : selected)
    {
        int read = ket.getStream().read(buff);
        System.err.println("Read: "+read);
    }
    selected.clear();
}
```

## 4.2. Datalink

Datalink is basically small extension to async stream. Below example classes depict difference and use case:

```
import org.mobicens.protocols.link.DataLink;
import org.mobicens.protocols.link.LinkState;
import org.mobicens.protocols.link.LinkStateListener;
import org.mobicens.protocols.stream.api.SelectorKey;
import org.mobicens.protocols.stream.api.SelectorProvider;
import org.mobicens.protocols.stream.api.StreamSelector;

class XServer implements LinkStateListener
{
    private DataLink link;

    private volatile boolean started = false;
    private StreamSelector selector;

    private int rxCount, txCount;
    private InetAddress address, remote;

    public XServer(InetAddress address, InetAddress remote) throws Exception {
        link = DataLink.open(address, remote);
        link.setListener(this);
        selector = SelectorProvider.getSelector("org.mobicens.protocols.link.SelectorImpl");
        link.register(selector);
    }

    public void start() {
        started = true;
        new Thread(this).start();
        link.activate();
    }

    public void stop() {
        started = false;
        link.close();

        System.out.println("rx=" + rxCount);
        System.out.println("tx=" + txCount);
    }

    public void run() {
        byte[] rxBuffer = new byte[172];
        byte[] txBuffer = new byte[172];

        while (started) {
```

```
try {

    Collection<SelectorKey> keys = selector.selectNow(StreamSelector.OP_READ, 20);
    for (SelectorKey key : keys) {
        int len = key.getInputStream().read(rxBuffer);
        rxCount++;
        System.out.println("Read " + len + " bytes: " + Arrays.toString(rxBuffer));
    }

    keys.clear();
    keys = selector.selectNow(StreamSelector.OP_WRITE, 20);
    txBuffer[txCount%txBuffer]++;
    for (SelectorKey key : keys) {
        key.getOutputStream().write(txBuffer);
        txCount++;
    }

    Thread.currentThread().sleep(1000);
} catch (Exception e) {
    e.printStackTrace();
}

}

}

public void onStateChange(LinkState state) {
    System.err.println("DatalinkState: " + state);
}

}

class XClient implements LinkStateListener
{
    private DataLink link;

    private volatile boolean started = false;
    private StreamSelector selector;

    private InetAddress address, remote;

    public XClient(InetAddress address, InetAddress remote) throws Exception {
        link = DataLink.open(address, remote);
        link.setListener(this);
        selector = SelectorProvider.getSelector("org.mobicents.protocols.link.SelectorImpl");
        link.register(selector);
    }
}
```

```
}  
  
public void start() {  
    started = true;  
    new Thread(this).start();  
    link.activate();  
}  
  
public void stop() {  
    started = false;  
    link.close();  
  
}  
  
public void run() {  
    byte[] rxBuffer = new byte[172];  
    //byte[] txBuffer = new byte[172];  
  
    while (started) {  
        try {  
  
            Collection<SelectorKey> keys = selector.selectNow(StreamSelector.OP_READ, 20);  
            for (SelectorKey key : keys) {  
                int len = key.getInputStream().read(rxBuffer);  
  
                System.out.println("Read " + len + " bytes: " + Arrays.toString(rxBuffer));  
            }  
  
            keys.clear();  
            keys = selector.selectNow(StreamSelector.OP_WRITE, 20);  
  
            for (SelectorKey key : keys) {  
                key.getOutputStream().write(rxBuffer);  
            }  
  
            Thread.currentThread().sleep(1000);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
public void onStateChange(LinkState state) {  
    System.err.println("DatalinkState: "+state);  
}  
}
```



---

# Appendix A. Revision History

Revision History

Revision 1.0

Wed June 2 2010

BartoszBaranowski

Creation of the JBoss Communications Stream Library User Guide.



---

# Index

## **F**

feedback, viii

