JBoss Communications
ISUP Stack User Guide

by Amit Bhayani, Bartosz Baranowski, and Oleg Kulikov

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/] 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` 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.

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/], against the product **JBoss Communications ISUP Stack** , or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: ISUPStack_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 ISUP Stack

The ISDN User Part (ISUP) is a protocol used to set-up, manage, and release trunk circuits voice carry voice and data between terminating line exchanges (e.g., between a calling party and a called party). ISUP is used for both ISDN and non- ISDN calls. However, calls which originate and terminate at the same switch do not use ISUP signaling.

ISUP is defined in ITU-T `Q.763` specification.

ISUP supercedes TUP (Telephony User Part) as the standard for call control between telephone exchanges.

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 Stack doesn't change the JBoss Communications Hardware Requirements, however it requires SS7 card.

Note

Desired is high performance machine to process voice and data channels.

2.1.2. Software Prerequisites

The Stack depends on:

- JBoss Communications MTP library
- JBoss Communications Stream library
- JBoss Communications M3UA library

2.2. JBoss Communications ISUP Stack 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.BETA3.

```
[usr]$ svn co ?/1.0.0.BETA3 isup-1.0.0.BETA3
```

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 isup-1.0.0.BETA3
[usr]$ mvn install
```

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

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 ISUP Stack is subject to changes as it is under active development!

Note

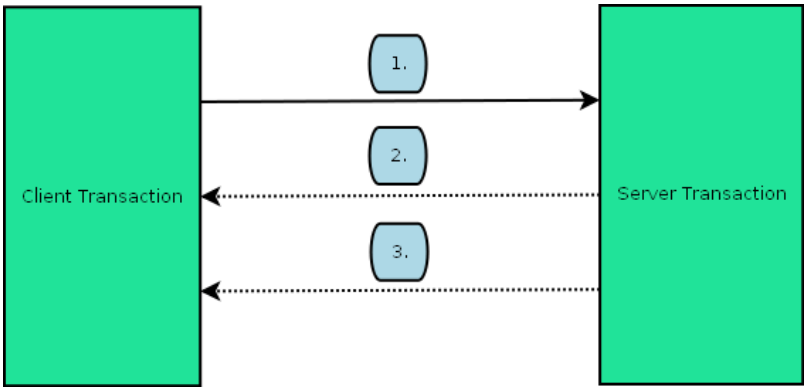
SS7 design allows ISUP to be used on top of SCCP, however currently JBoss Communications ISUP does not support this use case.

Note

For messages definition please refer to source or Q.763.

ISUP specification declares only messages and parameters to pass information about trunks and circuits setup. JBoss Communications ISUP introduces notion of transactions. Transaction groups messages which logically belong to single interaction (like associating circuit to call or releasing circuit). Server transaction is used as reference for incoming request. It provides means of sending answer back to originating peer. Client transaction is used as reference for outgoing requests. It provides means of sending request to remote peer. Also incoming answers will be delivered on client transaction which sent it.

Depending on type of action and messages involved, life time of transaction varies. Diagram below depicts transaction scope:



ISUP transactions overview

Note

Depending on transaction state machine messages #2 and #3 may not be present.

Following transactions types are defined by JBoss Communications ISUP :

Table 3.1. Transaction types

Name	Initial message	Intermediate messages	Final message
IAM	IAM	ACM	ANM
REL	REL		RLC
CGB	CGB		CGBA
GRS	GRS		GRA
CGU	CGU		CGUA
RSC	RSC		
UBL	UBL		UBA

Transactions are either terminated on receiving or sending final response or when certain time elapses without answer. In both cases stack user is notified that transaction is no longer valid.

Protocol

4.1. API

JBoss Communications ISUP Stack API is transaction oriented. Transaction are used to exchanged messages which contain parameter with some trunk specific information.

Below are listed interface with short explanation.

4.1.1. Transaction

Common part of transactions is defined as follows:

```
package org.mobicents.protocols.ss7.isup;

import org.mobicents.protocols.ss7.isup.message.ISUPMessage;

public interface ISUPTransaction {
   /**
    * Get unique transaction key associated with this transaction
    * @return
    */
   public TransactionKey getTransactionKey();
   /**
    * Determine if this transaction is server.
    * @return
    */
   public boolean isServerTransaction();
   /**
    * Get original message which started this transaction
    * @return
    */
   public ISUPMessage getOriginalMessage();
   /**
    * Determine if transaction has terminated properly.
    * @return
    */
   public boolean isTerminated();
   /**
    * Determine if transaction has timeout.
    * @return
    */
```

```
    public boolean isTimedout();

}
```

Above interface declares `getTransactionKey` method. It returns transaction key which is unique for certain transaction.

Client transaction is defined as follows:

```
package org.mobicents.protocols.ss7.isup;

import java.io.IOException;

public interface ISUPClientTransaction extends ISUPTransaction {
    /**
     * Send request for which this transaction has been created.
     * @throws ParameterRangeInvalidException
     * @throws IOException
     */
    public void sendRequest() throws ParameterRangeInvalidException, IOException;
    /**
     * State of this transaction
     * @return
     */
    public ISUPClientTransactionState getState();
}
```

Server transaction is defined accordingly to its role:

```
package org.mobicents.protocols.ss7.isup;

import java.io.IOException;
import org.mobicents.protocols.ss7.isup.message.ISUPMessage;

public interface ISUPServerTransaction extends ISUPTransaction{

    /**
     * Send answer via this transaction.
```

```
     * @param msg
     * @throws ParameterRangeInvalidException
     * @throws IllegalArgumentException
     * @throws IOException
     */
    public void sendAnswer(ISUPMessage msg) throws ParameterRangeInvalidException
        ,IllegalArgumentException, IOException;
    /**
     * Get state of this transaction.
     * @return
     */
    public ISUPServerTransactionState getState();

}
```

4.1.2. Factories

ISUP stack declares separate factories for messages and their parameters. However both interfaces are too big to have any use if mentioned in documentation. Please refer to source for details.

4.1.3. Stack

ISUP stack relies on MTP to provide transport. Stack interface is defined as follows:

```
package org.mobicents.protocols.ss7.isup;

public interface ISUPStack {

    public ISUPProvider getIsupProvider();

    public void stop();

    public void start();
}
```

Stack declares provider which allows user to access stack facilities, send messages statelesly and register listener. Provider is defined as follows:

```
package org.mobicents.protocols.ss7.isup;

import java.io.IOException;

import org.mobicents.protocols.ss7.isup.message.ISUPMessage;

public interface ISUPProvider {
    /**
     * Stateles message send over MTP. No state is maintained.
     *
     * @param msg
     * @throws ParameterRangeInvalidException
     * @throws IOException
     */
    public void sendMessage(ISUPMessage msg) throws ParameterRangeInvalidException, IOException;

    public void addListener(ISUPListener listener);

    public void removeListener(ISUPListener listener);

    public ISUPParameterFactory getParameterFactory();

    public ISUPMessageFactory getMessageFactory();
    /**
     * Create new client transaction if it does not exist.
     * @param msg
     * @return
     * @throws TransactionAlredyExistsException
     * @throws IllegalArgumentException
     */
    public ISUPClientTransaction createClientTransaction(ISUPMessage msg)
            throws TransactionAlredyExistsException, IllegalArgumentException;
    /**
     * Create new server transaction if it does not exist.
     * @param msg
     * @return
     * @throws TransactionAlredyExistsException
     * @throws IllegalArgumentException
     */
    public ISUPServerTransaction createServerTransaction(ISUPMessage msg)
            throws TransactionAlredyExistsException, IllegalArgumentException;
```

```
}
```

Provider allows to register listener. Listener contains set of call backs useful for ISUP user. It is defined as follows:

```
package org.mobicents.protocols.ss7.isup;

import org.mobicents.protocols.ss7.isup.message.ISUPMessage;

/**
 *
 * @author kulikov
 * @author baranowb
 */
public interface ISUPListener {

    /**
     * Called once stack receives proper ISUP message. Message holds reference to ongoing transaction if it exists.
     * @param message
     */
    public void onMessage(ISUPMessage message);
    // etc
    /**
     * Called once transaction times out. Transaction times out if there
     * is no answer received within configured time.
     */
    public void onTransactionTimeout(ISUPClientTransaction tx);
    /**
     * Called once transaction times out. Transaction times out if there
     * is no answer received within configured time.
     */
    public void onTransactionTimeout(ISUPServerTransaction tx);
    /**
     * Called once transaction ends its lifecycle.
     */
    public void onTransactionEnded(ISUPClientTransaction tx);
    /**
     * Called once transaction ends its lifecycle.
     */
    public void onTransactionEnded(ISUPServerTransaction tx);
```

```
    //transport methods
    /**
     * Method called when transport provider is not able to send/rcv messages,
     * any calls to send methods after this callback will throw exception
     */
    public void onTransportDown();
    /**
     * Method called when transport provider is able to send/rcv messages,
     * any calls to send methods after this callback are welcome.
     */
    public void onTransportUp();


}
```

4.1.4. Messages

Each messsage defines different setters and getters - since each message type has different set of parameters. However all messages defined by JBoss Communications ISUP Stack have common part:

```
package org.mobicents.protocols.ss7.isup.message;

public interface ISUPMessage extends ISUPComponent {

    /**
     * Get mandatory field, CIC.
     * @return
     */
    public CircuitIdentificationCode getCircuitIdentificationCode();
    /**
     * Set mandatory field, CIC.
     * @return
     */
    public void setCircuitIdentificationCode(CircuitIdentificationCode cic);

    /**
     * Returns message code. See Q.763 Table 4. It simply return value of static
     * constant, where value of parameter is value of MESSAGE_CODE
     *
```

```
     * @return
     */
    public MessageType getMessageType();

    /**
     * Adds parameter to this message.
     * @param param
     * @throws ParameterRangeInvalidException - thrown if parameter is not part of message.
     */
    public void addParameter(ISUPParameter param) throws ParameterRangeInvalidException;
    /**
     * Returns parameter with passed code.
     * @param parameterCode
     * @return
     * @throws ParameterRangeInvalidException - thrown if code does not match any parameter.
     */
    public ISUPParameter getParameter(int parameterCode) throws ParameterRangeInvalidException;
    /**
     * Removes parameter from this message.
     * @param parameterCode
     * @throws ParameterRangeInvalidException
     */
    public void removeParameter(int parameterCode) throws ParameterRangeInvalidException;

    /**
     * Return reference to transaction if it exists.
     * @return
     */
    public ISUPTransaction getTransaction();

    /**
     * @return
   *         true - if all requried parameters are set
   *         false - otherwise
   *
     */
    public boolean hasAllMandatoryParameters();

}
```

4.2. Configuration

ISUP stack depends on MTP to provide transport. Please refer to MTP documentation for configuration options supported. Below is list of ISUP specific configuration properties:

Table 4.1. ISUP Configuration Properties

Property Name	Description	Property Type	Default Value
isup.client.timeout	Value of timeout in milisecond. It controls timeout of client transaction. This value must be lower than `isup.general.timeout`	java.lang.Long	30.000
isup.general.timeout	Value of timeout in millisecond. It controls how long transaction object lingers in stack before its released - in case no action is performed.	java.lang.Long	120.000

4.3. Example

```
package org.mobicents.protocols.ss7.isup.impl;

import java.io.IOException;

import org.mobicents.protocols.ss7.isup.ISUPClientTransaction;
import org.mobicents.protocols.ss7.isup.ISUPListener;
import org.mobicents.protocols.ss7.isup.ISUPMessageFactory;
import org.mobicents.protocols.ss7.isup.ISUPParameterFactory;
import org.mobicents.protocols.ss7.isup.ISUPProvider;
import org.mobicents.protocols.ss7.isup.ISUPServerTransaction;
import org.mobicents.protocols.ss7.isup.ISUPStack;
import org.mobicents.protocols.ss7.isup.ParameterRangeInvalidException;
import org.mobicents.protocols.ss7.isup.TransactionAlredyExistsException;
import org.mobicents.protocols.ss7.isup.message.AddressCompleteMessage;
import org.mobicents.protocols.ss7.isup.message.AnswerMessage;
import org.mobicents.protocols.ss7.isup.message.ISUPMessage;
import org.mobicents.protocols.ss7.isup.message.InitialAddressMessage;
```

```
import org.mobicents.protocols.ss7.isup.message.parameter.CalledPartyNumber;
import org.mobicents.protocols.ss7.isup.message.parameter.CallingPartyCategory;
import org.mobicents.protocols.ss7.isup.message.parameter.CircuitIdentificationCode;
import org.mobicents.protocols.ss7.isup.message.parameter.ForwardCallIndicators;
import org.mobicents.protocols.ss7.isup.message.parameter.NatureOfConnectionIndicators;
import org.mobicents.protocols.ss7.isup.message.parameter.TransmissionMediumRequirement;


public class ClientIAM implements ISUPListener {
    private MTPProvider mtpProvider;
    private ISUPStack isupStack;
    private ISUPProvider provider;
    private ISUPMessageFactory factory;
    private ISUPParameterFactory parameterFactory;


    private ISUPClientTransaction ctx;


    public ClientIAM(MTPProvider mtpProvider) {
        super();

        Properties props = new Properties();
        props.setProperty("mtp.drive","m3ua");
        props.setProperty("mtp.opc","123");
        props.setProperty("mtp.apc","321");
        props.setProperty("mtp.address.local","192.168.1.1:123");
        props.setProperty("mtp.address.remote","192.168.1.12:321");
        //no need to set timeout, default value is ok;

        this.isupStack = new ISUPStackImpl();
        this.isupStack.configure(props);
        this.isupStack.start();
        this.provider = this.isupStack.getIsupProvider();
        this.factory = this.provider.getMessageFactory();
        this.parameterFactory = this.provider.getParameterFactory();
    }

    public void start() throws IllegalArgumentException, TransactionAlredyExistsException
        , ParameterRangeInvalidException, IOException {
        InitialAddressMessage iam = this.factory.createIAM();

        // create obligatory params!
        NatureOfConnectionIndicators nais = this.parameterFactory.createNatureOfConnectionIndicators();
```

```
        nais.setContinuityCheckIndicator(
            NatureOfConnectionIndicators._CCI_PERFORMED_ON_PREVIOUS_CIRCUIT);
        nais.setEchoControlDeviceIndicator(NatureOfConnectionIndicators._ECDI_INCLUDED);
        nais.setSatelliteIndicator(NatureOfConnectionIndicators._SI_TWO_SATELLITE);

        ForwardCallIndicators fcis = this.parameterFactory.createForwardCallIndicators();
        fcis.setEndToEndInformationIndicator(true);
        fcis.setEndToEndMethodIndicator(fcis._ETEMI_PASSALONG);
        fcis.setInterworkingIndicator(false);
        fcis.setIsdnAccessIndicator(true);

        CallingPartyCategory cpg = this.parameterFactory.createCallingPartyCategory();
        // ?
        cpg.setCallingPartyCategory((byte) 1);

        TransmissionMediumRequirement tmr = this.
            parameterFactory.createTransmissionMediumRequirement();
        tmr.setTransimissionMediumRequirement(tmr._MEDIUM_14x64_KBIT_UNRESTRICTED);

        CalledPartyNumber cpn = this.parameterFactory.createCalledPartyNumber();
        cpn.setAddress("123455");
        cpn.setNumberingPlanIndicator(cpn._NPI_ISDN);
        cpn.setNatureOfAddresIndicator(cpn._NAI_NATIONAL_SN);
        cpn.setInternalNetworkNumberIndicator(cpn._INN_ROUTING_ALLOWED);

        iam.setNatureOfConnectionIndicators(nais);
        iam.setForwardCallIndicators(fcis);
        iam.setCallingPartCategory(cpg);
        iam.setTransmissionMediumRequirement(tmr);
        iam.setCalledPartyNumber(cpn);

        CircuitIdentificationCode cic = this.parameterFactory.createCircuitIdentificationCode();
        cic.setCIC(12);
        iam.setCircuitIdentificationCode(cic);

        ctx = this.provider.createClientTransaction(iam);

        ctx.sendRequest();
    }

    /*
     * (non-Javadoc)
     *
     * @see
```

```
 * org.mobicents.protocols.ss7.isup.ISUPListener#onMessage(org.mobicents
 * .protocols.ss7.isup.message.ISUPMessage)
 */
public void onMessage(ISUPMessage message) {
    switch (message.getMessageType().getCode()) {
    case AddressCompleteMessage.MESSAGE_CODE:

        //ACM is send back as first

        break;
    case AnswerMessage.MESSAGE_CODE:
        //second, this terminates transaction
        break;
    default:

    }

}

/*
 * (non-Javadoc)
 *
 * @see
 * org.mobicents.protocols.ss7.isup.ISUPListener#onTransactionEnded(org.
 * mobicents.protocols.ss7.isup.ISUPClientTransaction)
 */
public void onTransactionEnded(ISUPClientTransaction tx) {

}

/*
 * (non-Javadoc)
 *
 * @see
 * org.mobicents.protocols.ss7.isup.ISUPListener#onTransactionEnded(org.
 * mobicents.protocols.ss7.isup.ISUPServerTransaction)
 */
public void onTransactionEnded(ISUPServerTransaction tx) {


}

/*
 * (non-Javadoc)
```

```
     *
     * @see
     * org.mobicents.protocols.ss7.isup.ISUPListener#onTransactionTimeout(org
     * .mobicents.protocols.ss7.isup.ISUPClientTransaction)
     */
    public void onTransactionTimeout(ISUPClientTransaction tx) {

    }

    /*
     * (non-Javadoc)
     *
     * @see
     * org.mobicents.protocols.ss7.isup.ISUPListener#onTransactionTimeout(org
     * .mobicents.protocols.ss7.isup.ISUPServerTransaction)
     */
    public void onTransactionTimeout(ISUPServerTransaction tx) {

    }

    /*
     * (non-Javadoc)
     *
     * @see org.mobicents.protocols.ss7.isup.ISUPListener#onTransportDown()
     */
    public void onTransportDown() {
        // TODO Auto-generated method stub

    }

    /*
     * (non-Javadoc)
     *
     * @see org.mobicents.protocols.ss7.isup.ISUPListener#onTransportUp()
     */
    public void onTransportUp() {
        // TODO Auto-generated method stub

    }



}
```

Appendix A. Revision History

Revision History
Revision 1.0 Wed June 2 2010 BartoszBaranowski
Creation of the JBoss Communications ISUP Stack User Guide.

Index

F