

Mobicents MTP 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 |
| 1. Introduction to Mobicents MTP Library | 1 |
| 1.1. Time Division Multiplexing | 1 |
| 1.2. MTP Introduction | 2 |
| 2. Setup | 3 |
| 2.1. Pre-Install Requirements and Prerequisites | 3 |
| 2.1.1. Hardware Requirements | 3 |
| 2.1.2. Software Prerequisites | 3 |
| 2.2. Mobicents MTP Library Source Code | 3 |
| 2.2.1. Release Source Code Building | 3 |
| 2.2.2. Development Trunk Source Building | 4 |
| 3. Design Overview | 5 |
| 3.1. Supported hardware | 5 |
| 3.2. Provider abstraction | 6 |
| 4. Protocol | 9 |
| 4.1. API Overview | 9 |
| 4.2. Configuration | 14 |
| 4.2.1. M3UA Provider | 14 |
| 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://code.google.com/p/mobicents/issues/list) [http://code.google.com/p/mobicents/issues/list], against the product **Mobicents MTP Library** , or contact the authors.

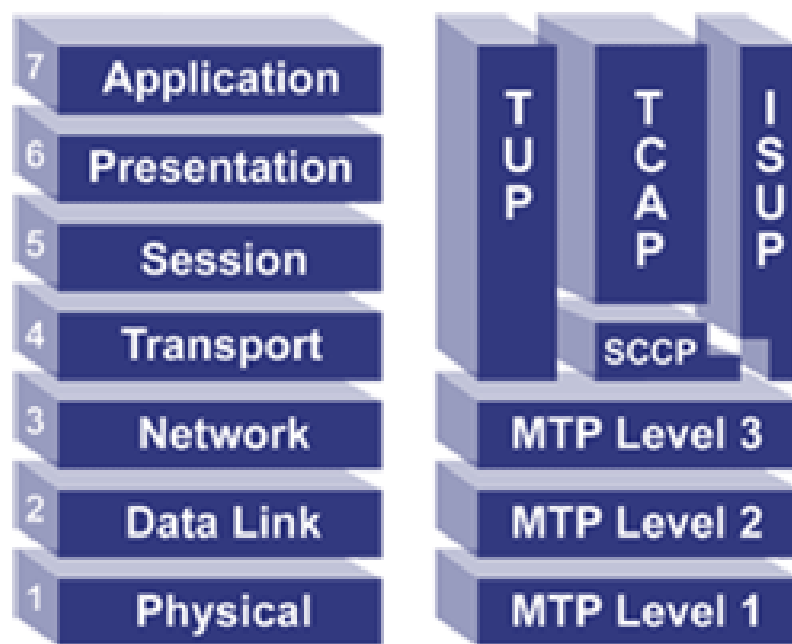
When submitting a bug report, be sure to mention the manual's identifier: MTPLibrary_User_Guide

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 Mobicents MTP Library

Common Channel Signaling System No. 7 (i.e., SS7 or C7) is a global standard for telecommunications defined by the [International Telecommunication Union \(ITU\) Telecommunication Standardization Sector \(ITU-T\)](http://www.voip-info.org/wiki/view/ITU) [http://www.voip-info.org/wiki/view/ITU] . The standard defines the procedures and protocol by which network elements in the public switched telephone network (PSTN)) exchange information over a digital signaling network to effect wireless (cellular) and wireline call setup, routing and control. The ITU definition of SS7 allows for national variants such as the American National Standards Institute (ANSI) and Bell Communications Research (Telcordia Technologies) standards used in North America and the European Telecommunications Standards Institute ([ETSI](http://www.voip-info.org/wiki/view/ETSI) [http://www.voip-info.org/wiki/view/ETSI]) standard used in Europe.

The hardware and software functions of the SS7 protocol are divided into functional abstractions called "levels". These levels map loosely to the Open Systems Interconnect (OSI) 7-layer model defined by the [International Standards Organization \(ISO\)](http://www.iso.ch/) [http://www.iso.ch/] .



SS7 Stack overview

1.1. Time Division Multiplexing

In circuit switched networks such as the Public Switched Telephone Network (PSTN) there exists the need to transmit multiple subscribers' calls along the same transmission medium. To accomplish this, network designers make use of TDM. TDM allows switches to create channels, also known as tributaries, within a transmission stream. A standard DS0 voice signal has a data

bit rate of 64 kbit/s, determined using Nyquist's sampling criterion. TDM takes frames of the voice signals and multiplexes them into a TDM frame which runs at a higher bandwidth. So if the TDM frame consists of n voice frames, the bandwidth will be $n \times 64$ kbit/s. Each voice sample timeslot in the TDM frame is called a channel. In European systems, TDM frames contain 30 digital voice channels, and in American systems, they contain 24 channels. Both standards also contain extra bits (or bit timeslots) for signalling (SS7) and synchronisation bits. Multiplexing more than 24 or 30 digital voice channels is called higher order multiplexing. Higher order multiplexing is accomplished by multiplexing the standard TDM frames. For example, a European 120 channel TDM frame is formed by multiplexing four standard 30 channel TDM frames. At each higher order multiplex, four TDM frames from the immediate lower order are combined, creating multiplexes with a bandwidth of $n \times 64$ kbit/s, where $n = 120, 480, 1920$, etc.

1.2. MTP Introduction

The Message Transfer Part (MTP) is divided into three levels. The lowest level, MTP Level 1, is equivalent to the OSI Physical Layer. MTP Level 1 defines the physical, electrical, and functional characteristics of the digital signaling link. Physical interfaces defined include E-1 (2048 kb/s; 32 64 kb/s channels), DS-1 (1544 kb/s; 24 64kb/s channels), V.35 (64 kb/s), DS-0 (64 kb/s), and DS-0A (56 kb/s). MTP Level 2 ensures accurate end-to-end transmission of a message across a signaling link. Level 2 implements flow control, message sequence validation, and error checking. When an error occurs on a signaling link, the message (or set of messages) is retransmitted. MTP Level 2 is equivalent to the OSI Data Link Layer. MTP Level 3 provides message routing between signaling points in the SS7 network. MTP Level 3 re-routes traffic away from failed links and signaling points and controls traffic when congestion occurs. MTP Level 3 is equivalent to the OSI Network Layer.

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 Mobicents 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:

- Mobicents `Stream` library
- Mobicents `M3UA` library

2.2. Mobicents MTP 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 `http://mobicents.googlecode.com/svn/tags/protocols/mtp`, then add the specific release version, lets consider 1.0.0.BETA5.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/protocols/mtp/1.0.0.BETA5 mtp-1.0.0.BETA5
```

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 mtp-1.0.0.BETA5  
[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 <http://mobicents.googlecode.com/svn/trunk/protocols/mtp>.

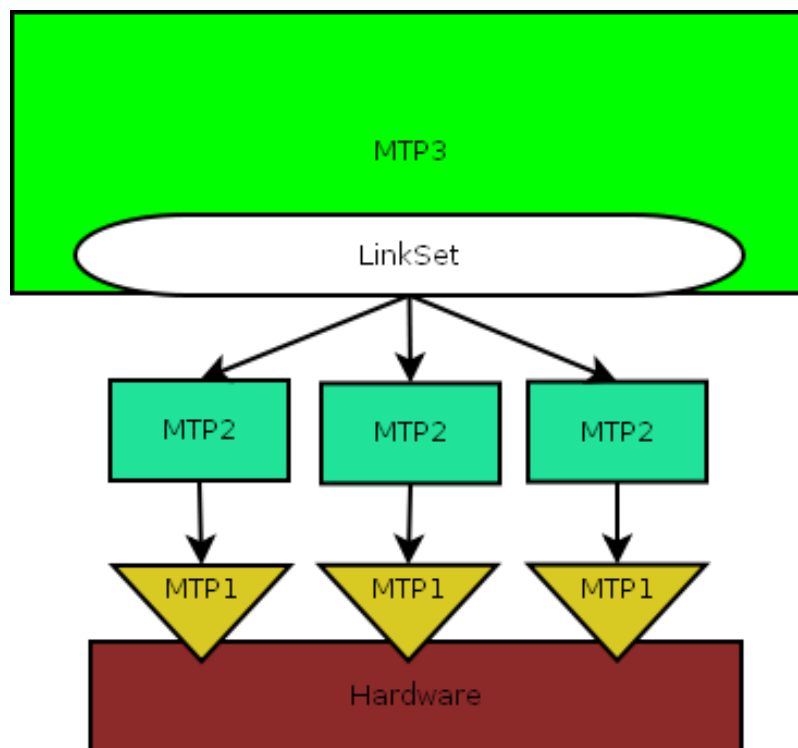
Design Overview



Important

Be aware, Mobicents MTP Library is subject to changes as it is under active development!

This module builds layer on top of hardware signaling devices. It allows top level protocols to use its API regardless of used device. Top overview of logical components is depicted on diagram below:



Mobicents MTP Library general design

3.1. Supported hardware

There is variety of SS7 hardware. Depending on driver, it provides different level of abstraction. Mobicents MTP Library supports following:

Intel SS7 family board

Dialogic® SS7 boards are designed to meet the needs of telecommunications equipment manufacturers, systems integrators, and service providers deploying solutions worldwide. Two families of SS7 products are available to enable affordable, high-performance, signaling applications.

Dialogic cards include hardware MTP layer 1 and 2.

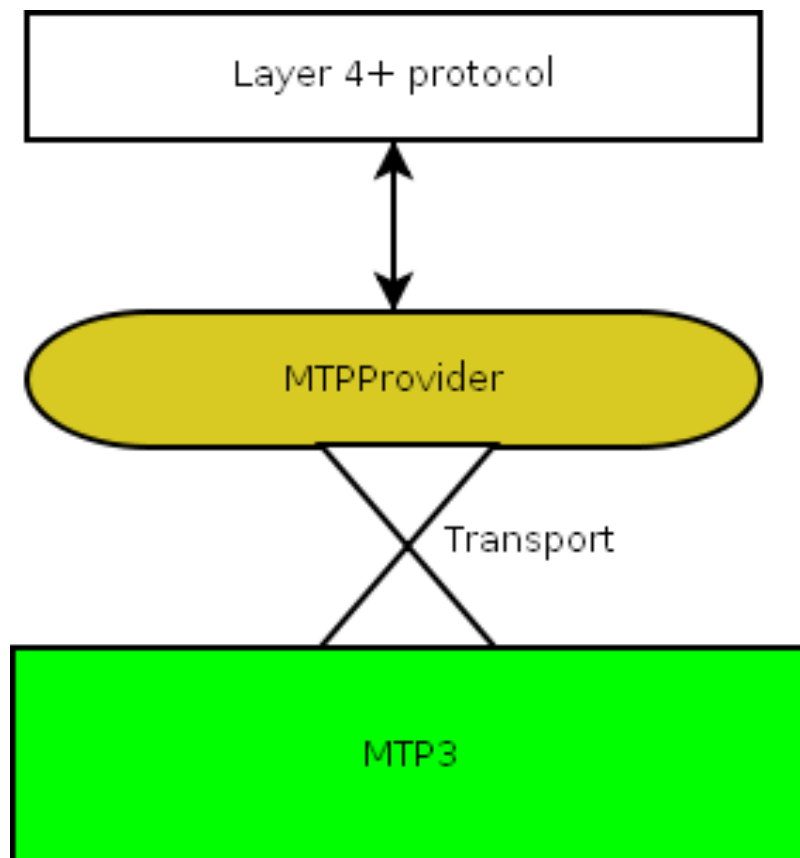
Zap compatible board

There are hardware TDM devices which share common driver suite called Zaptel Telephony Driver Suite (Zaptel). Most devices sold by Digium are members of the Zaptel family of hardware devices.

Zaptel cards provide only streaming capabilities. Each card requires full setup of MTP layers.

3.2. Provider abstraction

Mobicents MTP Library builds abstraction on MTP source with `MtpProvider`. Upper layer stacks depend on provider as means of receiving and sending `MSU`. Diagram below depicts general design:



Mobicents MTP Library Provider abstraction

Stack user provides proper implementation of `MTP3` (MSU source and sink) which can communicate with `MtpProvider`. Depending on condition and demand user also provides provider implementation.

Mobicents MTP Library provides stack user with tools to create MTP3 source and provider:

- `Mtp` classes for ZAPtel driver

- MtpProvider interface abstraction
- Implementation of MtpProvider over Mobicents Stream library

Protocol

4.1. API Overview

MTP layer is built with several components. Following list those that may be directly involved in creating application on top of this stack:

org.mobicens.proocols.ss7.mtp.Mtp1

This interface is implemented by classes directly interacting with SS7 hardware. It declares utility methods to open/close underlying implementations and generic read/write methods. It is decalred as follows:

```
package org.mobicens.proocols.ss7.mtp;

import java.io.IOException;

public interface Mtp1 {
    /**
     * Gets the code of this channel.
     *
     * @return the code of this channel.
     */
    public int getCode();

    /**
     * Set MTP2 layer serving this MTP1
     *
     * @param link
     */
    public void setLink(Mtp2 link);

    /**
     * Get MTP2 latyer serving this MTP1
     *
     * @return
     */
    public Mtp2 getLink();

    /**
     * Reads up to buffer.length bytes from layer 1.
     *
     */
}
```

```
* @param buffer
*      reader buffer
* @return the number of actually read bytes.
*/
public int read(byte[] buffer) throws IOException;

/**
* Writes data to layer 1.
*
* @param buffer
*      the buffer containing data to write.
* @param bytesToWrite
*/
public void write(byte[] buffer, int bytesToWrite) throws IOException;

/**
* Open message transfer part layer 1.
*/
public void open() throws IOException;

/**
* Close message transfer part layer 1.
*/
public void close();

}
```

org.mobicenss7.mtp.Mtp2

This is concrete implementation of MTP2 layer. It requires `Mtp1`. It declares following methods, relevant to configuration process:

- `public void setLayer1(Mtp1 layer1)` - sets concrete MTP1 serving for this link
- `public void setLayer3(Mtp2Listener layer3)` - sets listener which receives call backs from this layer (actually itsMTP3)

Mtp2 declares single method for data send operation: `public boolean queue(byte[] msg)`. This method requires properly formed Mtp3 message.

org.mobicenss7.mtp.Mtp3

This is concrete implementation of MTP3 layer. It implements state machine and encoding/decoding rules. It declares following methods, relevant from user point:

- `public void setSelectorFactory(SelectorFactory selectorFactory)` - sets selector factory for Mtp1

- `public void setOpc(int opc)` - sets local point code
- `public void setDpc(int dpc)` - set remote point code
- `public void setLinks(List<Mtp2> channels)` - sets list of Mtp2 links. Each channel should have Mtp1 assigned.
- `public void setUserPart(MtpUser mtpUser)` - sets concrete implementation of MtpUser interface. It will be called back to inform about events in this layer.
- `public boolean send(byte[] msg)` - sends passed message down the stream. It expects well formed MTP3 message.

MtpUser is defined in following way:

```
package org.mobicens.protocols.ss7.mtp;

public interface MtpUser {

    /**
     * Callback method from lower layers MTP3-. This is called once MTP3
     * determines that link is stable and is able to send/receive messages
     * properly. This method should be called only once. Every linkup event.
     */
    public void linkUp();

    /**
     * Callback method from MTP3 layer, informs upper layers that link is not
     * operable.
     */
    public void linkDown();

    /**
     * Callback from Layer4+. It expects properly encoded MTP3 message. It forwards data to MTP3
     * @param msgBuff
     */
    public void receive(byte[] msgBuff);

    public void setMtp3(Mtp3 mtp);
}
```

org.mobicens.protocols.ss7.mtp.RoutingLabel

This is utility class used by other layers. It extracts routing label from MTP3 MSU and performs operation to create label ready to be used in answers.

org.mobicens.protocols.ss7.mtp.provider.MtpProvider

This interface defines contract with upper layer protocol stacks. Its concrete implementation is created with `MtpProviderFactory` class. It is defined as follows:

```
package org.mobicens.protocols.ss7.mtp.provider;

import java.io.IOException;
import java.util.Properties;
import org.mobicens.protocols.ConfigurationException;
import org.mobicens.protocols.StartFailedException;

public interface MtpProvider {

    /**
     * Assigns originated point code
     *
     * @param opc the originated point code
     */
    public void setOriginalPointCode(int opc);

    /**
     * Assigns destination point code.
     *
     * @param dpc destination point code in decimal format.
     */
    public void setAdjacentPointCode(int dpc);

    /**
     * @return the dpc
     */
    public int getAdjacentPointCode();

    /**
     * @return the opc
     */
    public int getOriginalPointCode();

    /**
     * Sets listener for MTP callbacks. If null is passed internal refence is
```

```

    * cleared.
    *
    * @param lst
    */
    public void setMtpListener(MtpListener lst);

    /**
     * Passes argument to MTP layers for processing. Passed buffer should not be
     * reused after passing to this method!
     *
     * @param msu
     * @throws IOException
     *         - when IO can not be performed, ie, link is not up.
     * @return
     */
    public void send(byte[] msu) throws IOException;

    /**
     * Starts this provider implementation. Depending on internal it can start
     * local MTP process, or M3UA layer.
     *
     * @throws IOException
     * @throws StartFailedException
     */
    public void start() throws StartFailedException;

    /**
     * Stops this provider. This call clears all references, ie. listener is
     * cleared as {@link #setMtpListener(MtpListener)} with null argument.
     */
    public void stop();

    /**
     * Method which configures implementation. Depending on implementation
     * different properties are supported. However each property starts with
     * "mtp." prefix.
     *
     * @param p
     */
    public void configure(Properties p) throws ConfigurationException;

    /**
     * Checks if link is up;
     *

```

```

* @return
*/
public boolean isLinkUp();
}

```

`org.mobicens.proocols.ss7.mtp.provider.MtpListener`

This interface defines callback methods which are called from `MtpProvider` concrete class.

`org.mobicens.proocols.ss7.mtp.provider.MtpProviderFactory`

Factory class for concrete implementation of `MtpProvider`. Create method accepts `java.util.Properties`. It expects `mtp.driver` property to contain either full class name of concrete implementation of provider or name of supported providers.

4.2. Configuration

MTP Providers support different properties based on implementation. However each property must start with `mtp.` prefix.

4.2.1. M3UA Provider

M3UA Provider is provider which enables higher layer to be integrated with M3 User Adaptation layer. Implementation class of this provider is `org.mobicens.proocols.ss7.mtp.provider.m3ua.Provider`. It is based on Mobicents M3UA Library.

It supports following configuration properties:

Table 4.1. M3UAProvider configuration properties

| Property name | Description |
|---------------------------------|---|
| <code>mtp.address.remote</code> | Address of remote end of data link. It expects data in format: <code>IP:Port</code> . |
| <code>mtp.address.local</code> | As above. It points to local address to which data link is bound. |
| <code>mtp.apc</code> | Indicates adjacent point code (dpc for originating messages). |
| <code>mtp.opc</code> | Indicates originating point code. |

Example properties file which will configure this provider look as follows:

```
mtp.address.remote=127.0.0.1:3434
```

```
mtp.address.local=127.0.0.1:3435  
mtp.opc=13159  
mtp.apc=350
```

Appendix A. Revision History

Revision History

| | | |
|---|-----------------|-------------------|
| Revision 1.0 | Wed June 2 2010 | BartoszBaranowski |
| Creation of the Mobicents MTP Library User Guide. | | |
| Revision 1.1 | Thu Oct 6 2010 | BartoszBaranowski |
| Creation of the Mobicents MTP Library User Guide. | | |

Index

F

feedback, viii

