Mobicents jSS7 Stack User Guide

Installation and User Guide

Amit Bhayani `<amit.bhayani (at) gmail.com>`
Bartosz Baranowski `<baranowb (at) gmail.com>`
Oleg Kulikov `<oleg.kulikoff (at) gmail.com>`

Mobicents jSS7 Stack User Guide: Installation and User Guide

by Amit Bhayani, Bartosz Baranowski, and Oleg Kulikov

Abstract

This Installation and User Guide documents information regarding the installation and use of Mobicents jSS7 Stack.

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/] 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://code.google.com/p/jss7/issues/list], against the product Mobicents jSS7 Stack , or contact the authors.

When submitting a bug report, be sure to mention the manual's identifier: SS7Stack_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.

Chapter 1. Introduction

1.1. Mobicents jSS7 Overview

Mobicents jSS7 (Java SS7) is the only Open Source Java based implementation of the SS7 protocol stack. It comes with JSLEE MAP and ISUP Resource Adaptors (RA) and enables legacy developers to build SS7 applications with ease. Developers only require an understanding of Resource Adaptors and focus on building applications rather than worry about the SS7 stack. Mobicents jSS7 is compatible with Dialogic boards, dahdi based cards (Diguim and Sangoma) and also has inbuilt support for SIGTRAN (M3UA). Mobicents jSS7 Stack also supports TeleStax SS7 Cards which has MTP2/3 support on-board and hence lower processing load on server hosting Mobicents jSS7 Stack

Since Mobicents jSS7 is Java based, it is cross-platform and can be installed and used on any Operating System that supports Java. The Open Source Software gives you the flexibility to understand the readily available source code and customise the product for your Enterprise needs.

1.2. Introduction to SS7

Important

Spaces were introduced in some tables and code listings to ensure proper page render.

1.2.1. SS7 Overview

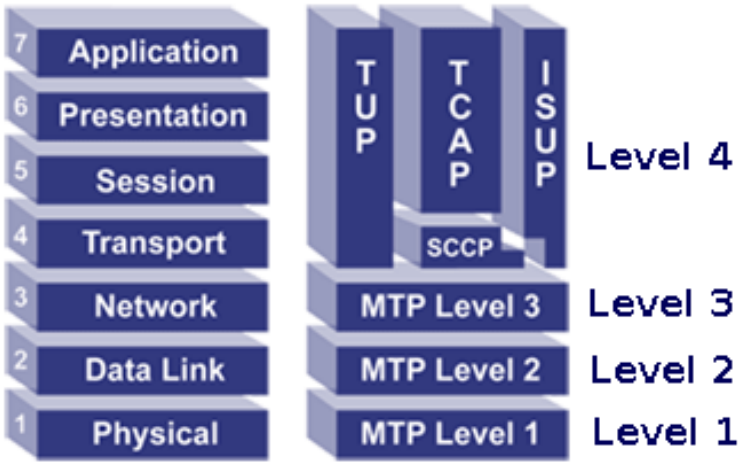
SS7 (Signalling System No.7) is a set of signaling protocols defined for information exchange in telephony. It is the global standard for telecommunications and is defined by the International Telecommunication Union (ITU) - Telecommunication Standardization Sector (ITU-T) [http://www.voip-info.org/wiki/view/ITU]. It is also commonly referred to as Common Channel Signaling System No. 7 (i.e., SS7 or C7).

In telephony (wireless and wireline), the information associated with a call must be exchanged between a telephone and the telephone exchange or between exchanges, transit nodes and other elements in the network. Information exchange is required to set up, control and tear down calls and this exchange of information is called Signaling. SS7 is the global standard that defines the procedures and protocol to be followed by network elements in the Public Switched Telephone Network (PSTN) in order to exchange information over a digital signaling network to effect wireless (cellular) and wireline call setup, route,control, monitor and terminate.

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]) standard used in Europe.

1.2.2. SS7 Stack Overview

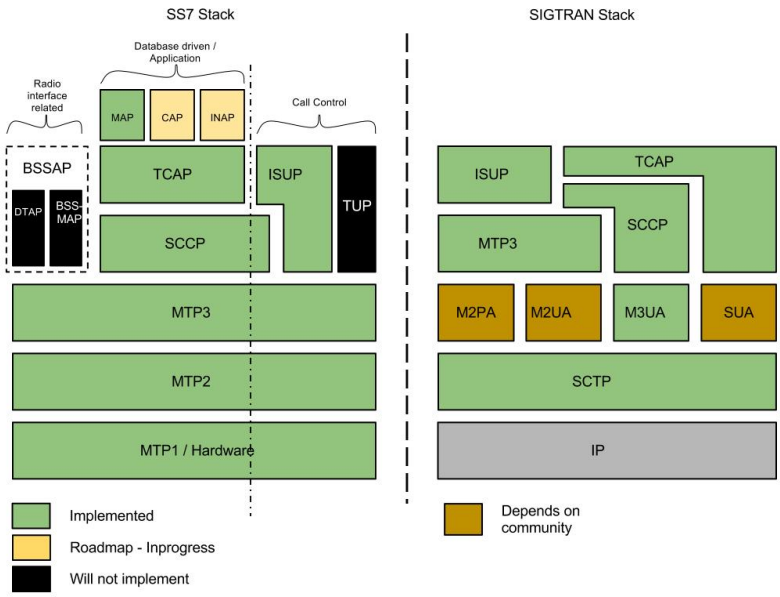
The hardware and software functions of the SS7 protocol are divided into functional abstractions called "levels". These levels map loosely to the Open Systems Interconnect (OSI) 7-layer model defined by the International Standards Organization (ISO) [http://www.iso.ch/].



SS7 Stack overview

Mobicents jSS7 Stack is a software based implementation of the SS7 protocol. It provides implementation for Level 2 and above in the SS7 protocol Stack. The Mobicents jSS7 Stack is a platform in the sense that it does not provide the application itself but rather allows users to build their own SS7 applications using Mobicents jSS7 Stack as a platform.

Below diagram shows various SS7 protocols implemented by Mobicents jSS7 Stack



Mobicents jSS7 Stack overview

1.2.3. 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 Time Division MultiplexingTDM. 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*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 x 64 kbit/s, where n = 120, 480, 1920, etc.

Chapter 2. The Basics of Mobicents jSS7 Stack

Warning

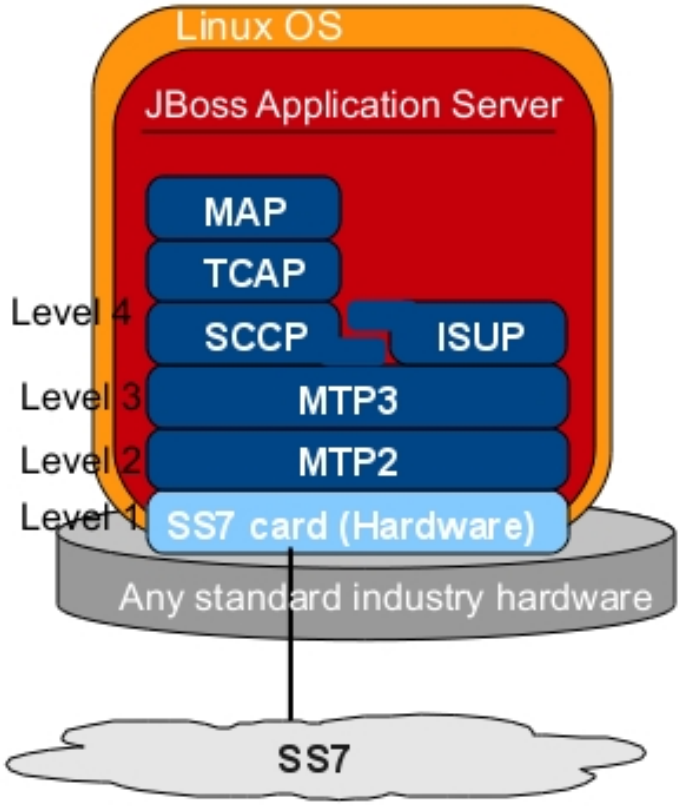
Be aware that Mobicents jSS7 Stack is subject to change since it is currently under active development!

2.1. Logical Divisions

The Mobicents jSS7 Stack is logically divided into two sections - lower and upper. The lower section provides implementation for SS7 Level 2 and Level 3. This section is therefore influenced by the type of SS7 hardware (Level 1) used. The upper section provides implementation for SS7 Level 4 and above. This logical division offers great flexibility where hardware is concerned. Irrespective of the type of hardware used in the lower section, Mobicents jSS7 Stack Level 4 and above remains the same.

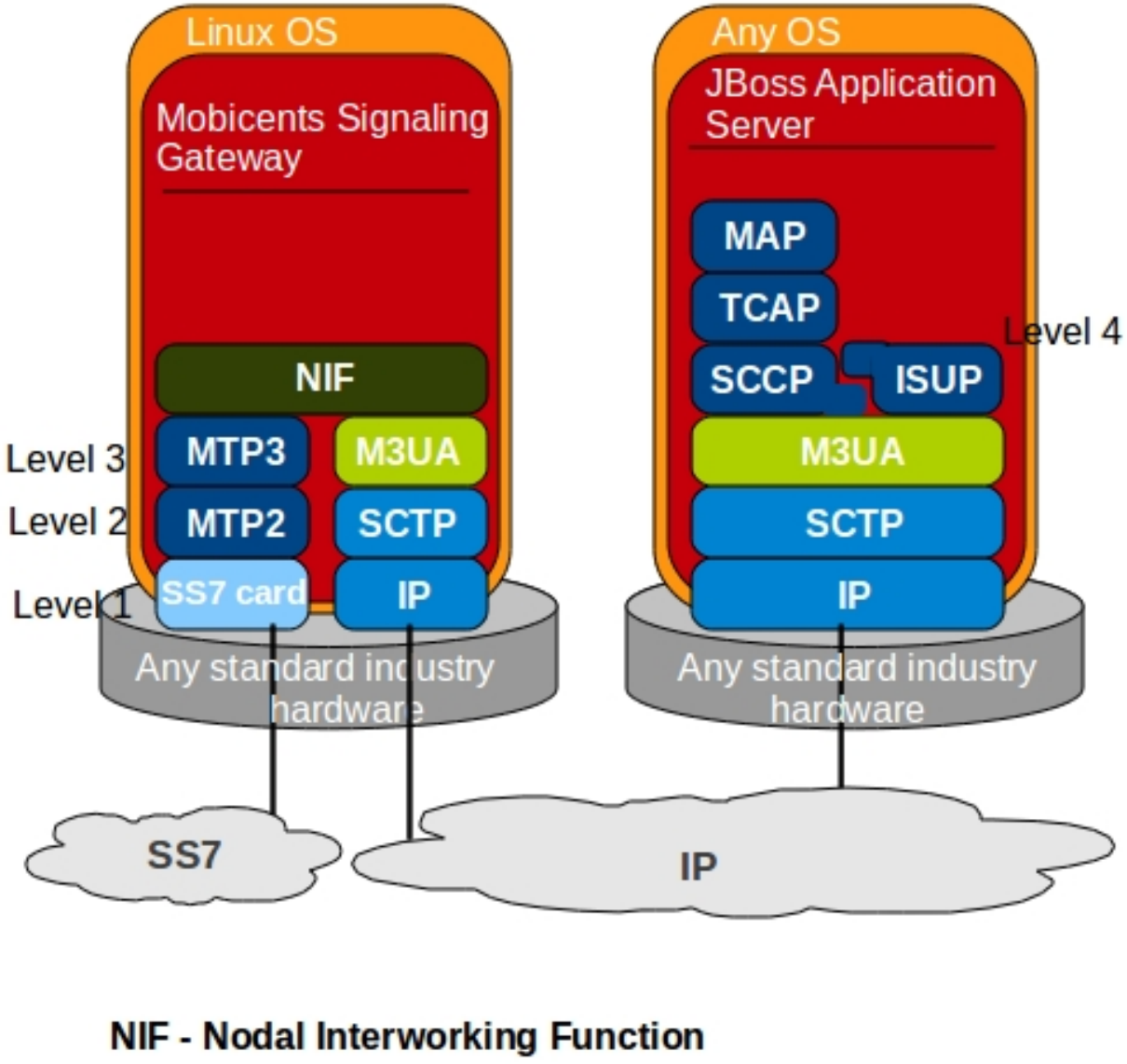
Mobicents jSS7 Stack gives you the flexibility to install and use the Levels 2,3 and 4 in the same JVM and machine where SS7 Hardware (Level 1) is installed. Alternately, you can also install Level 1,2 and 3 in one machine and install Level 4 on a separate machine. In the second scenario, `M3UA` over `SCTP` is used for communication between Level 3 and Level 4 (on different machines) and this is referred to as Mobicents Signaling Gateway. The figures below illustrate the above 2 scenarios.

Scenario 1: The complete Mobicents jSS7 Stack is installed in one machine.



Mobicents jSS7 Stack

Scenario 2: Mobicents Signaling Gateway - Level 3 and below are installed on one machine and Level 4 is installed on a different machine.



Mobicents Signaling Gateway

Important

If you use Mobicents M3UA stack, you must use JDK 7 to run the stack as well as to compile the source code. M3UA leverages Java SCTP which is available only in JDK 7.

2.2. Functional Blocks of Mobicents jSS7 Stack

Mobicents jSS7 Stack consists of the following functional blocks:

- Shell Management Client
- SS7 Service Elements
- Signaling Gateway

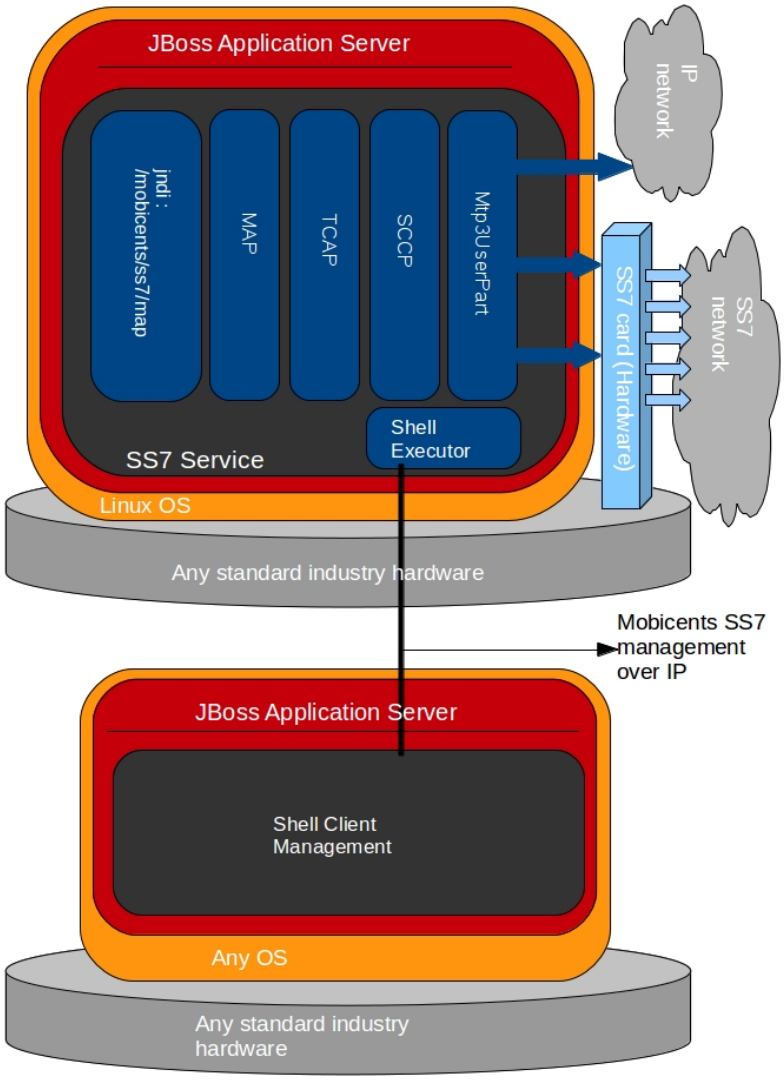
2.2.1. Shell Management client

`Shell` is a Command Line Interface (CLI) tool that will allow you to manage different aspects of Mobicents jSS7 Stack in an interactive manner. It connects to different instances of Mobicents jSS7 Stack which manages `Linksets`, `SCCP` resource, routing and `M3UA`. For detailed information please refer to: Chapter 5, *Shell Command Line*. Usually `Shell` will be invoked from remote machine(remote to `Linksets` and application protocols).

2.2.2. SS7 Service Elements

SS7 Service creates an instance of Mobicents MAP Stack and binds the instance to JNDI name `java:/mobicents/ss7/map`. SS7 Service is a JMX based service deployed in JBoss Application Server. It hides the underlying details like whether Level 4 and above are connected to the Mobicents Signaling Gateway via `M3UA` or if connected to the SS7 Hardware installed in the same machine as Level 4.

The figure below depicts the elements that are deployed as part of SS7 Service.



Mobicents jSS7 Stack Service Elements

SS7 Service Elements serve the following purposes:

Expose protocol access points:

Access points allow users to access lower layer protocols like `MAP` and interact with the `SS7` network through such protocols.

Expose management interface:

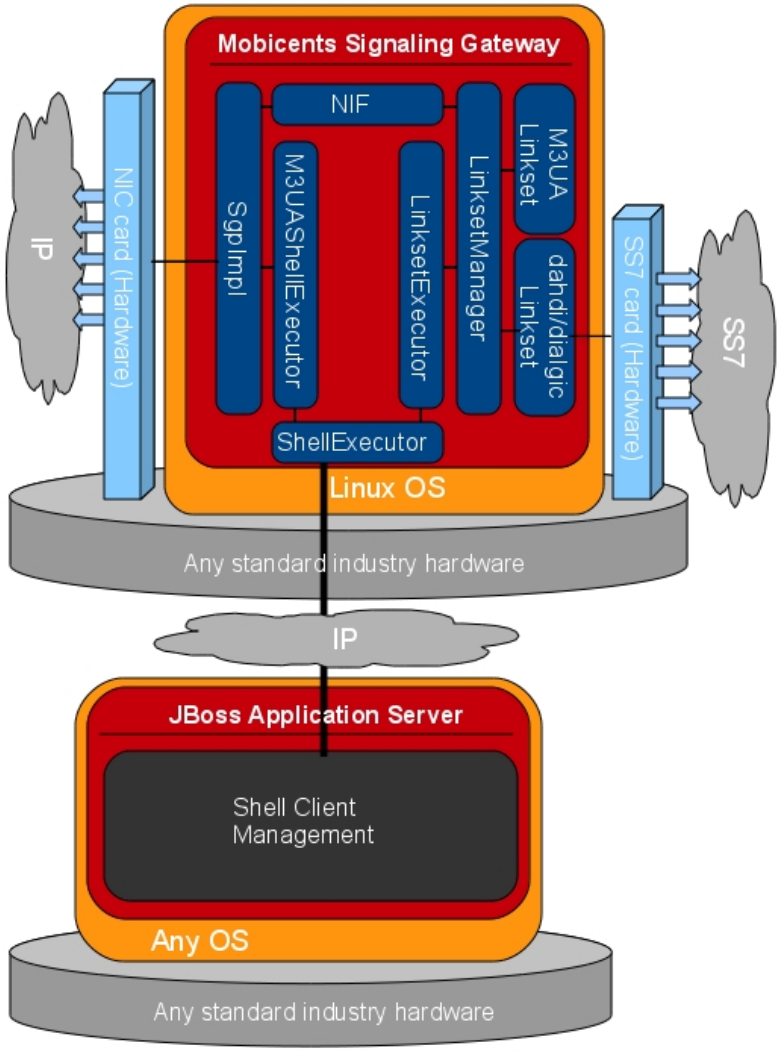
`Shell Executor` allows the `Shell` client to connect and issue commands.

Configuring the SS7 Service Elements is explained below in the section Section 3.5, “ Configuring Mobicents SS7 Service ”

2.2.3. Mobicents Signaling Gateway

Mobicents Signaling Gateway (SG) is a signaling agent that receives and sends Switched Circuit Network (SCN) native signaling at the edge of the IP network. Mobicents Signaling Gateway leverages MTP and Mobicents `M3UA` Stack explained in Section 2.2.3.1, “Mobicents M3UA Stack”.

The figure below shows the components included in Mobicents Signaling Gateway. Configuring the Signaling Gateway is explained in Section 3.8, “ Configuring Mobicents Signaling Gateway ”.



Mobicents Signaling Gateway Components

2.2.3.1. Mobicents M3UA Stack

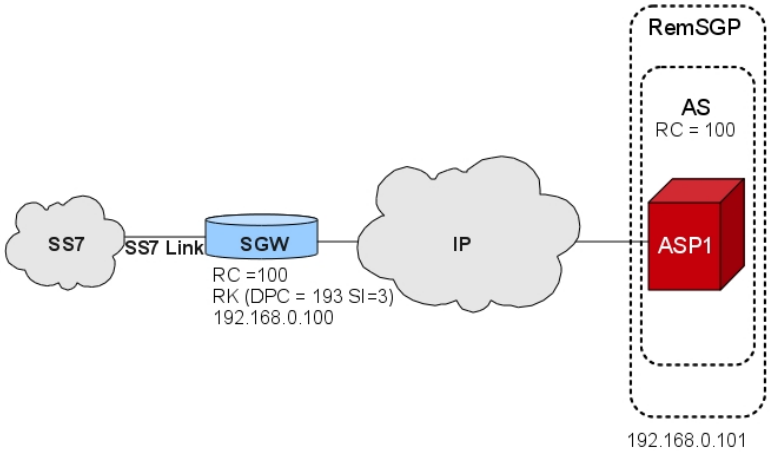
`M3UA` is a client-server protocol supporting the transport of any SS7 MTP3-User signalling (e.g. `ISUP` and `SCCP` messages) over IP. `M3UA` is defined by the IETF SIGTRAN working group in RFC 4666. Mobicents M3UA Stack can be used on the Application Server side or on the Signaling Gateway side or can also be used in peer-to-peer mode IPSP.

Note

Mobicents M3UA Stack uses Java SCTP layer which is only available from JDK 7 onwards.

2.2.3.1.1. Mobicents M3UA Stack on the Application Server side

The figure below demonstrates the basic functionality of the Mobicents M3UA Stack when configured as an Application Server (AS) that will communicate with an External Signaling Gateway.

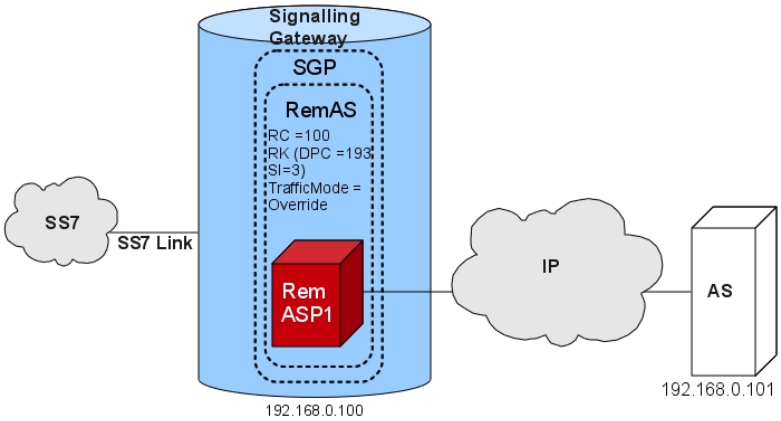


Mobicents M3UA Stack configured as an AS

To use M3UA Stack as an AS, the Routing Context (RC) may not be known and is optional. Refer to Section Section 3.5.1, “Configuring M3UA” for help in configuring M3UA Stack as an AS.

2.2.3.1.2. Mobicents M3UA Stack on the Signaling Gateway side

The figure below demonstrates the basic functionality of the Mobicents M3UA Stack when configured as a Signaling Gateway (SG). The Mobicents Signaling Gateway provides the Nodal Interworking Function (NIF) that allows SS7 Signaling (SCCP/ISUP) to be inter-worked into the M3UA/IP Network.



Mobicents M3UA Stack configured as a SG

Mobicents M3UA Stack used on the SG side will share common point code with a set of M3UA Application Server. M3UA stack on SG side can be configured as Loadbalance or Override traffic mode. It doesn't support Broadcast traffic mode. See Section Section 3.8, “ Configuring Mobicents Signaling Gateway ” for configuring M3UA Stack as SG. Mobicents M3UA Stack used on SG side doesn't support routing key management messages. The Routing Key should be provisioned statically using the management console.

Chapter 3. Installation and Running

3.1. Installing

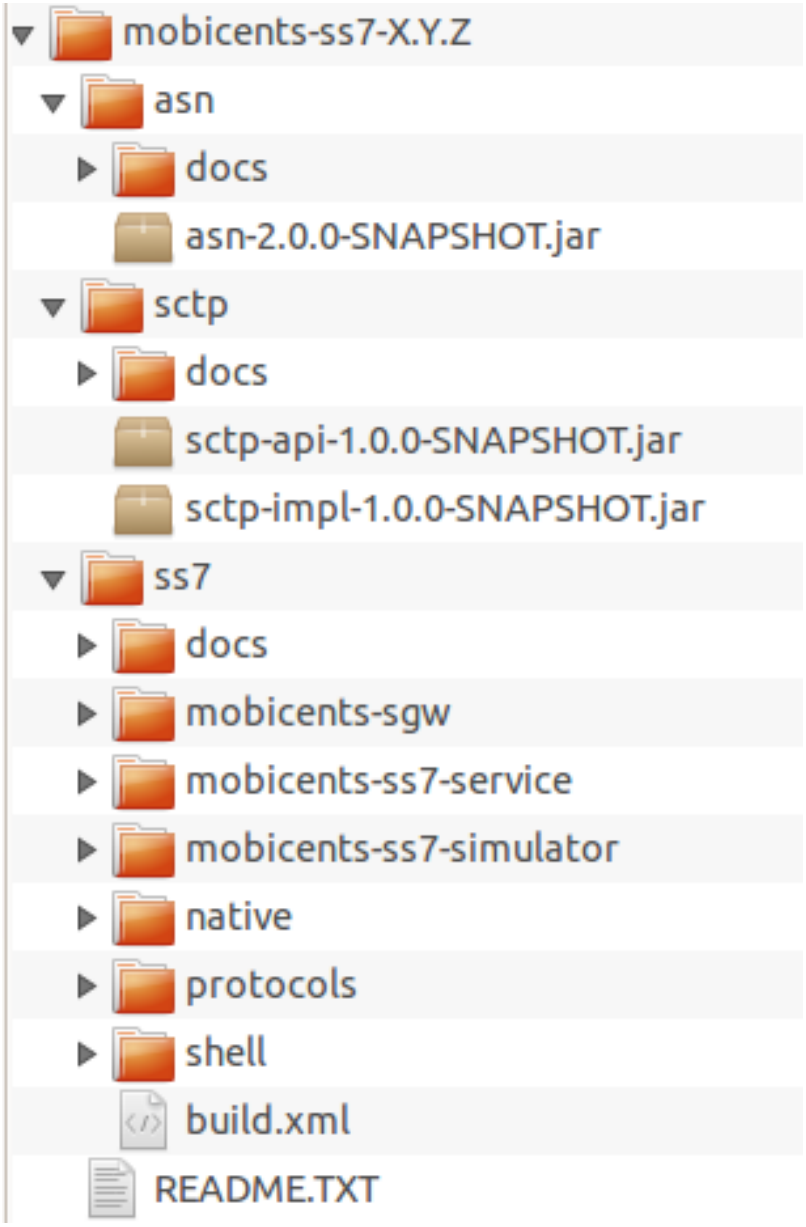
Mobicents SS7 stack at its core requires only Java if you are using only `M3UA`. However if you plan to use `Dahdi`, `Dialogic` or `Telestax` SS7 hardware, respective SS7 cards needs to be installed on the server along with native libraries.

A simple way to get started is to download and install binary. This will provide you with all the dependencies you need to get going. You can obtain binary release from http://sourceforge.net/projects/mobicents/files

3.1.1. Binary

The Mobicents jSS7 Stack binary is broken down into a few modules.

Binary release has following layout:



Mobicents jSS7 Stack binary layout.

Note

X.Y.Z in above layout is the respective release version of binary.

The following is a description of the important services and libraries that make up Mobicents jSS7 Stack

- `asn` : Abstract Syntax Notation One (ASN.1) library is used by various Mobicents jSS7 Stack protocols to encode/decode the structured data exchanged between Signaling Point over

networks. To know more about asn library refer to document included with asn. Applications using any of the Mobicents jSS7 Stack User Protocols may never need to call asn API directly, however it must be in classpath as Mobicents jSS7 Stack User Protocols refers this library.

- `sctp` : Stream Control Transmission Protocol (SCTP) Library is providing the convenient API's over Java SCTP. This library will be used only if M3UA layer is used.

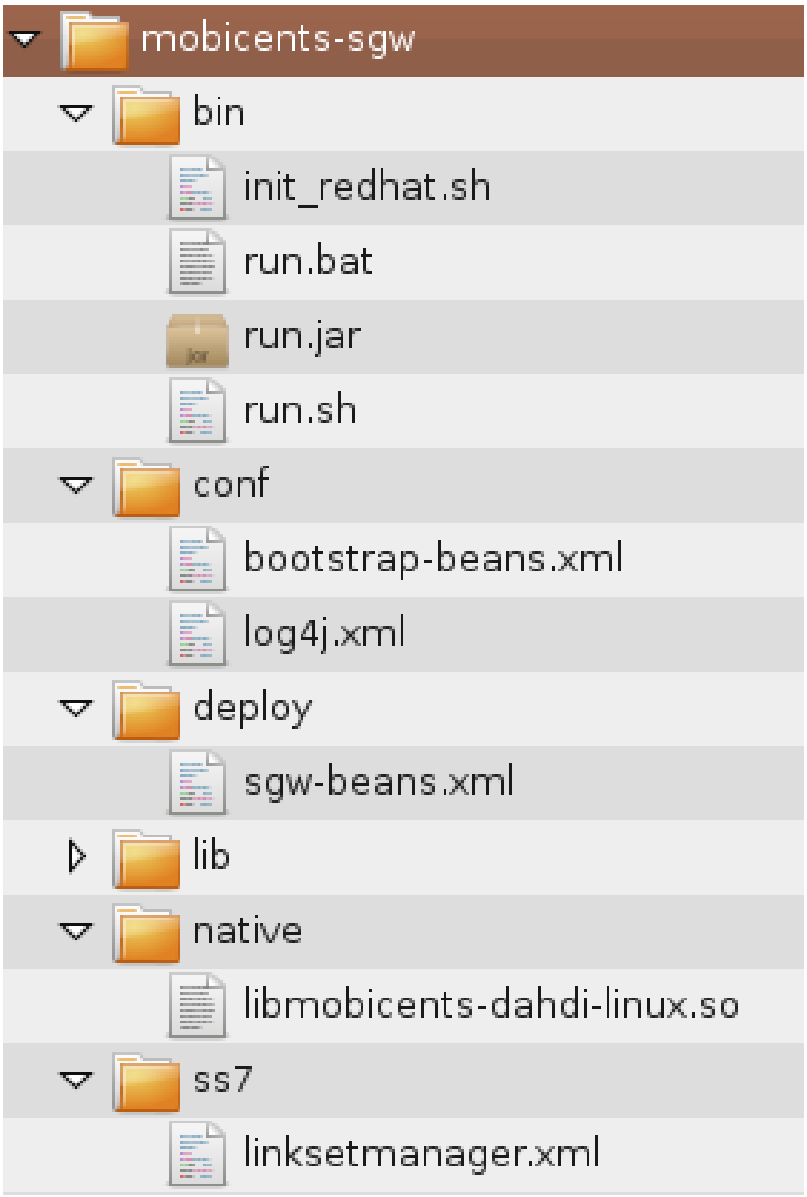
  To understand more about sctp, refer to documentation included in `sctp/docs`

- `ss7` : ss7 contains the core protocol libraries to be used by end application as well as service that is deployed in JBoss AS. The sub-modules included in ss7 are

  - `docs` : User guide for Mobicents jSS7 Stack

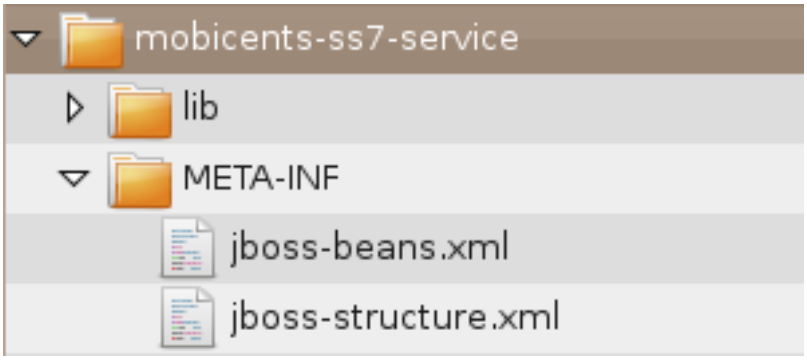
  - `mobicents-sgw` : Standalone Signaling Gateway as explained in section Section 2.2.3, “Mobicents Signaling Gateway”

    mobicents-sgw binary has following layout:



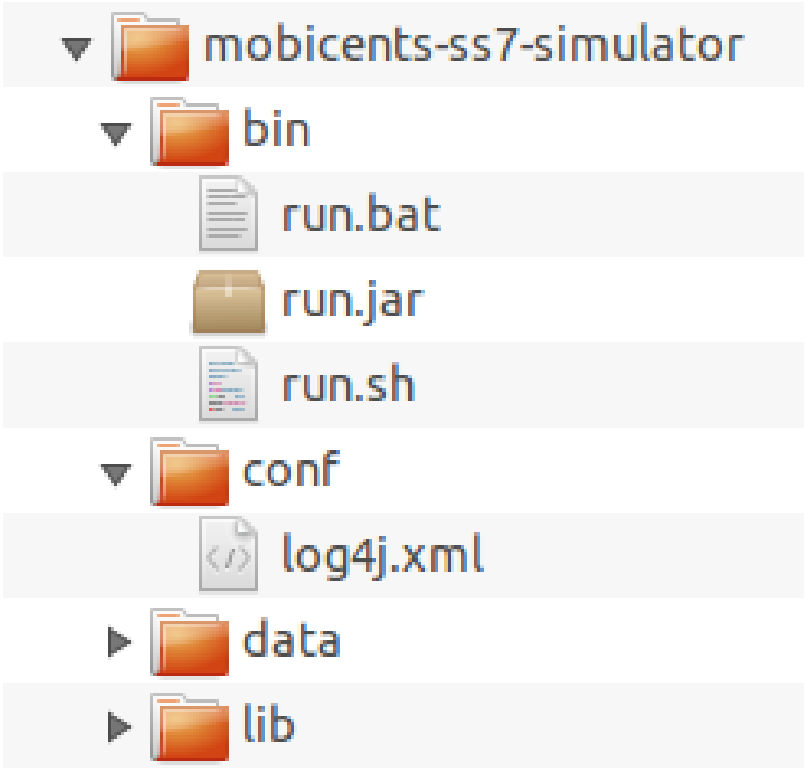
- `mobicents-ss7-service` : SS7 service is the core engine as explained in section Section 2.2.2, “SS7 Service Elements”

  mobicents-ss7-service binary has following layout:



- `mobicents-ss7-simulator` : SS7 Simulator is an application for testing SS7 stack and displaying its functionality. It is also a good example of how to use this stack.

  mobicents-ss7-simulator binary has following layout:



- `native` : native libraries component to interact with SS7 Card installed on server, runtime component. As of now native libraries are compiled only for linux OS. However if you plan to use `M3UA` there is no dependency on OS as everything is 100% java.

- `protocols` : The Mobicents jSS7 Stack User Protocols libraries. Your application would directly call the API's exposed by these libraries. Depending on application you may be either interested in `TCAP`, `MAP` or both or `ISUP` libraries

- `shell` : the Command Line Interface (CLI) module to manage the Mobicents jSS7 Stack. Refer Chapter 5, *Shell Command Line* to understand how to use shell

3.2. Mobicents SS7 Service

As the name indicates Mobicents SS7 Service is a deployable service that can be deployed in any container that supports `JMX` and exposes `JNDI`

Mobicents SS7 Service exposes convenient way of configuring SS7 stack via CLI commands. Service wraps SS7 level 4 i.e., MAP and lower layers and exposes it via JNDI such that layer above can do the look-up and use it in any application.

3.3. Installing Mobicents SS7 Service Binary

The SS7 Application depends on Mobicents SS7 Service and Mobicents SS7 Service must be installed before application can use it. The Mobicents SS7 Service binary requires that you have JBoss Application Server installed and JBOSS_HOME system property set. To know further details on setting JBOSS_HOME look Appendix B, *Setting the JBOSS_HOME Environment Variable*

Once JBOSS_HOME is properly set, use ant to deploy the mobicents-ss7-service, shell scripts and shell library.

Important

Ant 1.6 (or higher) is used to install the binary. Instructions for using Ant, including install, can be found at http://ant.apache.org/

```
[usr]$ cd ss7-2.0.0.BETA1/ss7
[usr]$ ant deploy
```

To undeploy these services

```
[usr]$ cd ss7-2.0.0.BETA1/ss7
[usr]$ ant undeploy
```

While above steps will deploy the necessary ss7 service and shell components, the `java.library.path` should be set to point the directory containing native component or should be copied to JBoss native library path manually. This step is only required if you are using the SS7 board on server.

3.4. Running Mobicents SS7 Service

Starting or stopping Mobicents SS7 Service is no different than starting or stopping JBoss Application Server

3.4.1. Starting

Once installed, you can run server by executing the run.sh (Unix) or run.bat (Microsoft Windows) startup scripts in the `<install_directory>/bin` directory (on Unix or Windows). If the service started properly you should see following lines in the Unix terminal or Command Prompt depending on your environment:

```
23:22:26,079 INFO  [LinksetManager] SS7 configuration file path /
home/abhayani/workarea/mobicents/jboss-5.1.0.GA/server/default/data/
linksetmanager.xml
23:22:26,141 INFO  [LinksetManager] Started LinksetManager
23:22:26,199 INFO  [SS7Service] Starting SCCP stack...
23:22:26,229 INFO  [SccpStackImpl] Starting ...
23:22:26,230 INFO  [RouterImpl] SCCP Router configuration file: /home/
abhayani/workarea/mobicents/jboss-5.1.0.GA/server/default/deploy/mobicents-
ss7-service/sccp-routing.txt
23:22:26,261 INFO  [SS7Service] SCCP stack Started. SccpProvider bound to
 java:/mobicents/ss7/sccp
23:22:26,261 INFO  [ShellExecutor] Starting SS7 management shell environment
23:22:26,270 INFO  [ShellExecutor] ShellExecutor listening
 at /127.0.0.1:3435
23:22:26,270 INFO  [SS7Service] [[[[[[[[[ Mobicents SS7 service
 started ]]]]]]]]]
```

If you have started ss7-2.0.0.BETA1 for the first time, SS7 is not configured. You need to use Shell Client to connect to ss7-2.0.0.BETA1 as defined in Chapter 5, *Shell Command Line* . With CLI you can configure how service interacts with SS7 network, that is you configure either installed SS7 card and its native library , or M3UA layer.

Once the configured, the state and configuration of SS7 is persisted which stands server re-start.

3.4.2. Stopping

You can shut down the server(s) by executing the **shutdown.sh -s** (Unix) or **shutdown.bat -s** (Microsoft Windows) scripts in the `<install_directory>/bin` directory (on Unix or Windows). Note that if you properly stop the server, you will see the following three lines as the last output in the Unix terminal or Command Prompt:

```
[Server] Shutdown complete
Halting VM
```

3.5. Configuring Mobicents SS7 Service

Configuration is done through an XML descriptor named `jboss-beans.xml` and is located at `$JBOSS_HOME/server/profile_name/deploy/mobicents-ss7-service/META-INF`, where profile_name is the server profile name.

The Mobicents SS7 Layer 4 (SCCP, ISUP) leverages either of following MTP layers to exchange signalling messages with remote signalling points

- M3UA

- `dahdi`
- `dialogic`

The ss7 service will be configured with either of these services.

3.5.1. Configuring M3UA

`M3UAManagement` is only needed if the underlying SS7 service will leverage `M3UA`. `M3UAManagement` configuration is further explained in ???

```
<bean name="SCTPManagement" class="org.mobicents.protocols.sctp.ManagementImpl">
    <constructor>
        <parameter>SCTPManagement</parameter>
    </constructor>
    <property name="persistDir">${jboss.server.data.dir}</property>
    <property name="singleThread">true</property>
    <property name="connectDelay">30000</property>
</bean>

<!-- ===================================================================== -->
<!-- M3UA -->
<!-- M3UAManagement is managing the m3ua side commands -->
<!-- ===================================================================== -->
<!-- -->
<bean name="Mtp3UserPart" class="org.mobicents.protocols.ss7.m3ua.impl.M3UAManagement">
    <constructor>
        <parameter>Mtp3UserPart</parameter>
    </constructor>
    <property name="persistDir">${jboss.server.data.dir}</property>
    <property name="transportManagement">
        <inject bean="SCTPManagement" />
    </property>
</bean>

<bean name="M3UAShellExecutor"
    class="org.mobicents.protocols.ss7.m3ua.impl.oam.M3UAShellExecutor">
    <property name="m3uaManagement">
        <inject bean="Mtp3UserPart" />
    </property>
    <property name="sctpManagement">
        <inject bean="SCTPManagement" />
    </property>
</bean>
```

`org.mobicents.protocols.sctp.ManagementImpl` takes `String` as constructor argument. The name is prepend to `xml` file created by SCTP stack for persisting state of SCTP resources. The `xml` is stored in path specified by `persistDir` property above.

For example in above case, when Mobicents SS7 Service is started file `SCTPManagement_sctp.xml` will be created at `$JBOSS_HOME/server/profile_name/data` directory

Stack has following properties:

singleThread
: SCTP stack is implemented such that 1 thread is dedicated for IO and by default 1 thread for calling application above SCTP. Its possible to configure multiple threads for calling applications above in which case `singleThread` should be set to false and set `workerThreads` to number of threads you want.

workerThreads
: Number of worker threads to call the application above SCTP. By default only one thread is used.

connectDelay
: If the SCTP Socket is client-side, connectDelay specifies the delay time in milliseconds before which connection with the server will attempted. This delay is necessary when there is network disruption and connection between client and server breaks so that SCTP stack doesn't continuously attempt to reconnect.

`org.mobicents.protocols.ss7.m3ua.impl.M3UAManagement` takes `String` as constructor argument. The name is prepend to `xml` file created by M3UA stack for persisting state of M3UA resources. The `xml` is stored in path specified by `persistDir` property above.

For example in above case, when Mobicents SS7 Service is started file `Mtp3UserPart_m3ua.xml` will be created at `$JBOSS_HOME/server/profile_name/data` directory

3.5.2. Configuring dahdi

`Dahdi` based MTP layer will only be used when you have installed dahdi based SS7 hardware (Sangoma or Diguim cards). `DahdiLinksetFactory` is responsible to create new instances of corresponding `DahdiLinkset` when instructed by `LinksetManager`.

- `DahdiLinksetFactory`

```
<bean name="DahdiLinksetFactory"
    class="org.mobicents.ss7.hardware.dahdi.oam.DahdiLinksetFactory">
</bean>
```

`LinksetFactoryFactory` is just a call-back class listening for new factories deployed and maintains Map of available factory name vs factory. You should never touch this bean.

3.5.2.1. Configuring LinksetManager

`LinksetManager` is responsible for managing `Linkset` and `Link`.

```
<!-- ============================================== -->
```

```
    <!-- Linkset manager Service                                              -->
    <!-- ============================================== -->
    <bean name="LinksetManager"
        class="org.mobicents.ss7.linkset.oam.LinksetManager">

        <property name="linksetFactoryFactory">
            <inject bean="LinksetFactoryFactory" />
        </property>
        <property name="persistDir">${jboss.server.data.dir}</property>
    </bean>

    <bean name="LinksetExecutor"
        class="org.mobicents.ss7.linkset.oam.LinksetExecutor">
        <property name="linksetManager">
            <inject bean="LinksetManager" />
        </property>
    </bean>
```

LinksetManager when started looks for file `linksetmanager.xml` containing serialized information about underlying linksets and links. The directory path is configurable by changing value of `persistDir` property.

Warning

`linksetmanager.xml` should never be edited by hand. Always use Shell Client to connect to Mobicents jSS7 Stack and execute commands.

`LinksetExecutor` accepts the `linkset` commands and executes necessary operations.

3.5.3. Configuring dialogic

`Dialogic` based MTP layer will only be used when you have installed Dialogic cards. `DialogicMtp3UserPart` communicates with Dialogic hardware. Its asumed here that MTP3 and MTP2 is leveraged from Dialogic stack either on-board or on-host.

```
    <!-- ================================================================= -->
    <!-- Dialogic Mtp3UserPart -->
    <!-- ================================================================= -->

  <bean name="Mtp3UserPart" class="org.mobicents.ss7.hardware.dialogic.DialogicMtp3UserPart">
        <property name="sourceModuleId">61</property> <property name="destinationModuleId">34</
property>
        </bean>
```

`sourceModuleId` is source module id and should match with configured in system.txt used by dialogic drivers. Here 61 is assigned for Mobicents process. `destinationModuleId` is destination module id. 34 is Dialogic MTP3 module id.

3.5.4. Configuring SCCP

As name suggests `SccpStack` initiates the SCCP stack routines.

```
    <!-- ===================================================================== -->
       <!-- SCCP Service -->
       <!-- ===================================================================== -->
    <bean name="SccpStack" class="org.mobicents.protocols.ss7.sccp.impl.SccpStackImpl">
        <constructor>
            <parameter>SccpStack</parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
        <property name="removeSpc">true</property>
        <property name="mtp3UserParts">

  <map keyClass="java.lang.Integer" valueClass="org.mobicents.protocols.ss7.mtp.Mtp3UserPart">
                <entry>
                    <key>1</key>
                    <value>
                        <inject bean="Mtp3UserPart" />
                    </value>
                </entry>
            </map>
        </property>
    </bean>

    <bean name="SccpExecutor"
        class="org.mobicents.protocols.ss7.sccp.impl.oam.SccpExecutor">
        <property name="sccpStack">
            <inject bean="SccpStack" />
        </property>
    </bean>
```

`org.mobicents.protocols.ss7.sccp.impl.SccpStackImpl` takes `String` as constructor argument. The name is prepend to `xml` file created by SCCP stack for persisting state of SCCP resources. The `xml` is stored in path specified by `persistDir` property above.

For example in above case, when Mobicents SS7 Service is started two file's `SccpStack_sccpresource.xml` and `SccpStack_sccprouter.xml` will be created at `$JBOSS_HOME/server/profile_name/data` directory

Stack has following properties:

persistDir
: As explained above

removeSpc
: After Global Title Translation, if the SCCP address includes the destination point code (DPC) however Address Indicator (AI) indicates route on Global Title and `removeSpc` is set to true, DPC will be removed from SCCP Address. The same rule applies for both calling as well as called party SCCP Address.

mtp3UserParts
: specifies SS7 Level 3 to be used as transport medium(be it SS7 card or M3UA). Mobicents jSS7 Stack SCCP allows configuring multiple MTP3 layers for same SCCP stack. This allows to have multiple local point-code and connecting to various networks while SCCP layer remains same

`SccpExecutor` accepts `sccp` commands and executes necessary operations

3.5.5. Configuring ShellExecutor

`ShellExecutor` is responsible for listening incoming commands. Received commands are executed on local resources to perform actions like creation and management of `SCCP` routing rule, management of `SCTP` and management of `M3UA` stack.

```
<!-- ============================================== -->
    <!-- Shell Service                                                  -->
    <!-- ============================================== -->
        <!-- Define Shell Executor -->
        <bean name="ShellExecutor" class="org.mobicents.ss7.ShellExecutor">
                <property name="address">${jboss.bind.address}</property>
                <property name="port">3435</property>

                <property name="sccpExecutor">
                        <inject bean="SccpExecutor" />
                </property>
                <property name="m3UAShellExecutor">
                        <inject bean="M3UAShellExecutor" />
                </property>
        </bean>
```

By default ShellExecutor listens at `jboss.bind.address` and port `3435`. You may set the `address` property to any valid IP address that your host is assigned. The shell commands are exchanged over TCP/IP.

Note

To understand JBoss bind options look at Installation_And_Getting_Started_Guide [http://docs.jboss.org/jbossas/docs/Installation_And_Getting_Started_Guide/5/html_single/index.html]

3.5.6. Configuring TCAP

`TcapStack` initiates the TCAP stack routines.

```
        <!-- ================================================================ -->
    <!-- TCAP Service -->
    <!-- ================================================================ -->
```

```
    <bean name="TcapStack" class="org.mobicents.protocols.ss7.tcap.TCAPStackImpl">
        <constructor>
            <parameter><inject bean="SccpStack" property="sccpProvider" /></parameter>
            <parameter>8</parameter>
        </constructor>
        <property name="dialogIdleTimeout">60000</property>
        <property name="invokeTimeout">30000</property>
        <property name="maxDialogs">25000</property>
    </bean>
```

`org.mobicents.protocols.ss7.tcap.TCAPStackImpl` takes `SccpStack` as constructor argument. TCAP uses passed SCCP stack. It also takes the sub system number (SSN) which is registered with passed SCCP stack.

Stack has following properties:

dialogIdleTimeout
: The TCAP Dialog idle timeout in milli-seconds

invokeTimeout
: Component invoke timeout value in milli-seconds

maxDialogs
: Maximum concurrent dialog's allowed at any given point of time. If application tries to create more dialog than this, exception is thrown

3.5.7. Configuring MAP

`MapStack` initiates the MAP stack routines.

```
        <!-- ===================================================================== -->
    <!-- MAP Service -->
    <!-- ===================================================================== -->
    <bean name="MapStack" class="org.mobicents.protocols.ss7.map.MAPStackImpl">
        <constructor>
            <parameter><inject bean="TcapStack" property="provider" /></parameter>
        </constructor>
    </bean>
```

`org.mobicents.protocols.ss7.map.MAPStackImpl` takes `TcapStack` as constructor argument. MAP uses passed TCAP stack.

3.5.8. Configuring SS7Service

`SS7Service` acts as core engine binding all the components together.

```
        <!-- ===================================================================== -->
    <!-- Mobicents SS7 Service -->
```

```
    <!-- ===================================================================== -->
    <bean name="SS7Service" class="org.mobicents.ss7.SS7Service">

  <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.mobicents.ss7:service=SS7Service",exposedInt
        </annotation>
        <property name="jndiName">java:/mobicents/ss7/map</property>
        <property name="stack">
            <inject bean="MapStack" property="MAPProvider" />
        </property>
    </bean>
```

SS7 service binds MapStack to JNDI `java:/mobicents/ss7/map`. The JNDI name can be configured to any valid JNDI name specific to your application.

3.6. Installing Mobicents Signaling Gateway Binary

Mobicents Signaling Gateway Binary doesn't require any additional steps. You may copy `mobicents-sgw` to any folder of your choice.

3.7. Running Mobicents Signaling Gateway

In the Linux terminal or Windows command prompt, the Mobicents Signaling Gateway has started successfully if the last line of output is similar to the following

```
15:51:18,247    INFO        [MainDeployer]    [[[[[[[[[    Mobicents    Signaling    Gateway:
 release.version=2.0.0.BETA1 Started ]]]]]]]]]
```

3.7.1. Starting Mobicents Signaling Gateway

Procedure 3.1. Running the Mobicents Signaling Gateway on Linux

1. Change the working directory to installation directory (the one in which the zip file's contents was extracted to)

   ```
   downloads]$ cd "mobicents-ss7-<version>/ss7/mobicents-sgw"
   ```

2. (Optional) Ensure that the `bin/run.sh` start script is executable.

   ```
   mobicents-sgw$ chmod +x bin/run.sh
   ```

3. Execute the `run.sh` Bourne shell script.

   ```
   mobicents-sgw$ ./bin/run.sh
   ```

Procedure 3.2. Running the Mobicents Signaling Gateway on Windows™

1. Using Windows Explorer, navigate to the `bin` subfolder in the installation directory.
2. The preferred way to start the Mobicents Signaling Gateway is from the Command Prompt. The command line interface displays details of the startup process, including any problems encountered during the startup process.

   Open the Command Prompt via the Start menu and navigate to the correct folder:

   ```
   C:\Users\<user>\My Downloads>cd "mobicents-ss7-<version>\ss7\mobicents-sgw"
   ```

3. Start the Gateway by executing one of the following files:

   - `run.bat` batch file:

     ```
     C:\Users\<user>\My Downloads\mms-standalone<version>>bin\run.bat
     ```

3.7.2. Start the Gateway With Alternate Configuration

Using `run.sh` without any arguments binds the gateway to `127.0.0.1`. To bind gateway to different ip, pass the ip address as value to `-b` command line option. For example to bind the server to `115.252.103.220`

```
mobicents-sgw$ ./bin/run.sh -b 115.252.103.220
```

3.7.3. Stopping

The only option to stop the gateway is by pressing `Ctrl c` and bringing down the JVM or kill the process.

3.8. Configuring Mobicents Signaling Gateway

Configuration is done through an XML descriptor named `sgw-beans.xml` and is located at `mobicents-sgw/deploy`,

3.8.1. Configuring M3AU (Signaling Gateway)

SGW will expose the SS7 signals received from legacy network to IP network over M3AU

```
<bean name="SCTPManagement" class="org.mobicents.protocols.sctp.ManagementImpl">
```

```
        <constructor>
            <parameter>SCTPManagement</parameter>
        </constructor>
        <property name="persistDir">${sgw.home.dir}/ss7</property>
        <property name="singleThread">true</property>
        <property name="connectDelay">30000</property>
    </bean>

    <bean name="Mtp3UserPart" class="org.mobicents.protocols.ss7.m3ua.impl.M3UAManagement">
        <constructor>
            <parameter>Mtp3UserPart</parameter>
        </constructor>
        <property name="persistDir">${sgw.home.dir}/ss7</property>
        <property name="transportManagement">
            <inject bean="SCTPManagement" />
        </property>
    </bean>

    <bean name="M3UAShellExecutor"
        class="org.mobicents.protocols.ss7.m3ua.impl.oam.M3UAShellExecutor">
        <property name="m3uaManagement">
            <inject bean="Mtp3UserPart" />
        </property>
        <property name="sctpManagement">
            <inject bean="SCTPManagement" />
        </property>
    </bean>
```

3.8.2. Configuring LinksetFactory

Concrete implementation of `LinksetFactory` is responsible to create new instances of corresponding `Linkset` when instructed by `LinksetManager`. Mobicents Signaling Gateway defines two linkset factories :

- `DahdiLinksetFactory`

```
    <bean name="DahdiLinksetFactory"
        class="org.mobicents.ss7.hardware.dahdi.oam.DahdiLinksetFactory">
    </bean>
```

- `DialogicLinksetFactory`

```
    <bean name="DialogicLinksetFactory"
        class="org.mobicents.ss7.hardware.dialogic.oam.DialogicLinksetFactory">
    </bean>
```

Its highly unlikely that you would require both the factories on same gateway. If you have `dahdi` based SS7 card installed, keep `DahdiLinksetFactory` and remove other. If you have `dialogic` based SS7 card installed, keep `DialogicLinksetFactory` and remove other.

`LinksetFactoryFactory` is just a call-back class listening for new factories deployed and maintains Map of available factory name vs factory. You should never touch this bean.

3.8.3. Configuring LinksetManager

`LinksetManager` is responsible for managing `Linkset` and `Link`.

```
<!-- ================================================ -->
    <!-- Linkset manager Service                                                  -->
    <!-- ================================================ -->
    <bean name="LinksetManager"
        class="org.mobicents.ss7.linkset.oam.LinksetManager">

        <property name="linksetFactoryFactory">
            <inject bean="LinksetFactoryFactory" />
        </property>
        <property name="persistDir">${linkset.persist.dir}</property>
    </bean>

    <bean name="LinksetExecutor"
        class="org.mobicents.ss7.linkset.oam.LinksetExecutor">
        <property name="linksetManager">
            <inject bean="LinksetManager" />
        </property>
    </bean>
```

LinksetManager when started looks for file `linksetmanager.xml` containing serialized information about underlying linksets and links. The directory path is configurable by changing value of `persistDir` property.

Warning

`linksetmanager.xml` should never be edited by hand. Always use Shell Client to connect to Mobicents Signaling Gateway and execute commands.

`LinksetExecutor` accepts the `linkset` commands and executes necessary operations.

3.8.4. Configuring ShellExecutor

`ShellExecutor` is responsible for listening to incoming command. Received commands are executed on local resources to perform actions like creation and management of `Linkset`, management of `M3UA` stack.

```
<!-- ================================================ -->
```

```
    <!-- Shell Service                                                    -->
    <!-- ============================================== -->
    <bean name="ShellExecutor"
        class="org.mobicents.ss7.sgw.ShellExecutor">
        <property name="address">${sgw.bind.address}</property>
        <property name="port">3436</property>
        <property name="linksetExecutor">
            <inject bean="LinksetExecutor" />
        </property>
        <property name="m3UAShellExecutor">
            <inject bean="M3UAShellExecutor" />
        </property>
    </bean>
```

By default ShellExecutor listens at `sgw.bind.address` and port `3436`. You may set the `address` property to any valid IP address that your host is assigned. The shell commands are exchanged over TCP/IP.

3.8.5. Configuring SignalingGateway

`SignalingGateway` acts as core engine binding all the components together.

```
        <!-- ============================================== -->
    <!-- Mobicents Signaling Gateway                                  -->
    <!-- ============================================== -->
    <bean name="SignalingGateway"
        class="org.mobicents.ss7.sgw.SignalingGateway">

        <property name="shellExecutor">
            <inject bean="ShellExecutor" />
        </property>

        <property name="nodalInterworkingFunction">
            <inject bean="NodalInterworkingFunction" />
        </property>

    </bean>
```

The `NodalInterworkingFunction` sits between the SS7 netwrok and IP network and routes messages to/from both the MTP3 and the M3UA layer, based on the SS7 DPC or DPC/SI address information

3.9. Setup from source

Mobicents jSS7 Stack is an open source project, instructions for building from source are part of the manual! Building from source means you can stay on top with the latest features. Whilst aspects of Mobicents jSS7 Stack are quite complicated, you may find ways to become contributors.

Mobicents jSS7 Stack works with JDK1.5 and above (If using M3UA, JDK1.7 and above). you will also need to have the following tools installed. Minimum requirement version numbers provided.

- `Git Client 1.6` : Instructions for using GIT, including install, can be found at http://git-scm.com/book
- `Subversion Client 1.4` : Instructions for using SVN, including install, can be found at http://subversion.tigris.org
- `Maven 2.0.9` : Instructions for using Maven, including install, can be found at http://maven.apache.org/
- `Ant 1.7.0` : Instructions for using Ant, including install, can be found at http://ant.apache.org

3.9.1. Release Source Code Building

1. **Downloading the source code**

   Use GIT to clone repository, the base URL is http://code.google.com/p/jss7, then to checkout specific release version(tag) use **git checkout tag_name**, lets consider release-2.0.0.BETA1.

```
[usr]$ git clone http://code.google.com/p/jss7
[usr]$ cd jss7
[usr]$ git checkout release-2.0.0.BETA1
```

   Browse the code online at http://code.google.com/p/jss7/source/browse/

2. **Building the source code**

   Now that we have the source the next step is to build and install the source. Mobicents jSS7 Stack uses Maven 2 to build the system. There are three profiles. Default one builds only java source. The other two profiles available "dahdilinux" and "dialogiclinux" additionaly compile native modules.

Note

Native modules are supported only for linux OS for now.

   Use "dahdilinux" profile if linux server on which this code is built already has dahdi module installed. Make sure you pass "include.zap" system property pointing to correct directory where dahdi is installed

```
[usr]$ cd jss7
[usr]$ mvn install -Pdahdilinux -Dinclude.zap=/usr/include/dahdi
```

```
```

Use "dialogiclinux" profile if linux server on which this code is built already has dialogic module installed. Make sure you pass "include.dialogic" and "include.dialogic.gctlib" system property pointing to correct directory where dialogic libraries are installed. include.dialogic.gctlib points to directory where `gctload` is present (generally /opt/dpklnx for linux OS)

```
[usr]$ cd jss7
[usr]$     mvn     install     -Pdialogclinux     -Dinclude.dialogic=/opt/dpklnx/INC     -
Dinclude.dialogic.gctlib=/opt/dpklnx
```

To build Mobicents jSS7 Stack without building any native libraries use

```
[usr]$ cd jss7
[usr]$ mvn install
```

Note

If you are using Mobicents jSS7 Stack without any native dependencies, Mobicents jSS7 Stack can run on any OS.

Use Ant to build the binary .

```
[usr]$ cd jss7/release
[usr]$ ant
```

3.9.2. Development Trunk Source Building

Similar process as for Section 3.9.1, “Release Source Code Building”, the only change is don't switch to specific tag.

Chapter 4. Hardware Setup

This chapter contains reference to configure hardware drivers for different types of `SS7` cards.

Mobicents jSS7 Stack supports `dahdi` based SS7 cards like `diguim` and `sangoma`. Generally `dahdi` based SS7 crads doesn't have MTP2/MTP3 support on board and relies on external software to provide these services.

Mobicents jSS7 Stack also supports `dialogic` based SS7 cards which has on board support for MTP2/MTP3

4.1. Sangoma

To install Sangoma cards visit the Sangoma wiki at http://wiki.sangoma.com/

4.2. Diguim

To install Diguim cards visit the Diguim site at http://www.digium.com/en/products/digital/

4.3. Dialogic

To install Dialogic cards visit the Dialogic site at http://www.dialogic.com/

Chapter 5. Shell Command Line

5.1. Introduction

Mobicents jSS7 Stack provides Shell client to manage configuration of jSS7 Stack Services. This chapter describes how to install and start client. Also it describes available commands and provides examples. To see examples of specific flow, to perform certain tasks, please refer to sections in chapter devoted to `Linksets`, `SCCP` or `M3UA` .

5.2. Starting

Shell client can be started with following command from `$JBOSS_HOME/bin` :

```
[$] ./ss7-run.sh
```

Once console starts, it will print following information:

```
=========================================================================

Mobicents SS7: release.version=1.0.0-SNAPSHOT
This is free software, with components licensed under the GNU General Public
 License
version 2 and other licenses. For further details visit http://mobicents.org
=========================================================================

mobicents>
```

The `ss7-run` script supports following options

```
Usage: SS7 [OPTIONS]
Valid Options
-v          Display version number and exit
-h          This help screen
```

Shell needs to connect to managed instance. Command to connect has following structure:

```
ss7 connect <IP> <PORT>
```

Example 5.1. Connec to remote machine

```
mobicents>ss7 connect 10.65.208.215 3435

mobicents(10.65.208.215:3435)>
```

Note

Host IP and port are optional, if not specified, shell will try to connect to `127.0.0.1:3435`

Command to disconnect has following structure:

```
ss7 discconnect
```

Example 5.2. Disconnect

```
mobicents(10.65.208.215:3435)>ss7 disconnect

Bye
mobicents>
```

5.3. Linkset Management

Linksets are managed by `linkset` command. It allows to perform following:

- create linkset
- delete linkset
- activate linkset
- deactivate linkset
- create link
- delete link
- activate link

- deactivate link
- list state of linksets and present links

5.3.1. Create Linkset

Linkset can be create by issuing command with following structure:

```
linkset create <linkset-type> opc <point-code> apc <point-code> ni <network-id> <linkset-name>
```

or in case of dialogic:

```
linkset create dialogic opc <point-code> apc <point-code> ni <network-id> srcmod <src-mode>
 destmod <dest-mode> <linkset-name>
```

or in case of M3UA:

```
linkset create m3ua opc <point-code> apc <point-code> ni <network-id> as <as-name> <linkset-name>
```

Where:

linkset-type
: refers to type of linkset to be created, ie. `dahdi` , `dialogic` or `m3ua` . Correct values depend on which linkset factories have been deployed.

point-code
: is simply `MTP` point - either local(`opc`) or remote(`dpc`)

ni
: is simply network identifier. It can have following values:

  0
  : International network

  1
  : Spare (for international use only)

  2
  : National network

  3
  : Reserved for national use

linkset-name
: simple string name, which identifies linkset

as-name
: Name of AS that M3UALinkset wrapps. Make sure that AS is already created as explained in Section 5.5.2.1, “Create AS”

Example 5.3. Linkset creation

```
mobicents(10.65.208.215:3435)>linkset create dahdi opc 1 apc 2 ni 0 linkset1
LinkSet successfully added
mobicents(10.65.208.215:3435)>linkset create dialogic opc 3 apc 4 ni 3
 srcmod 1 destmod 2 linkset2
LinkSet successfully added
```

5.3.2. Remove Linkset

Linkset can be deleted by issuing command with following structure:

```
linkset delete  <linkset-name>
```

Where:

linkset-name
: is name set during link creation

Example 5.4. Linkset Removal

```
mobicents(10.65.208.215:3435)>linkset delete linkset1
LinkSet successfully deleted
```

5.3.3. Activate Linkset

Linkset can be activated by issuing command with following structure:

```
linkset activate  <linkset-name>
```

Where:

linkset-name
: is name set during link creation

Example 5.5. Linkset Activation

```
mobicents(10.65.208.215)>linkset activate linkset1
LinkSet activated successfully
```

5.3.4. Deactivate Linkset

Linkset can be deactivated by issuing command with following structure:

```
linkset deactivate  <linkset-name>
```

Where:

linkset-name
: is name set during link creation

Example 5.6. Linkset Deactivateion

```
mobicents(10.65.208.215)>linkset deactivate linkset1
LinkSet deactivated successfully
```

5.3.5. Create Link

`Link` can be created in `Linkset` by issuing command with following structure:

```
linkset link create span <span-num> code <code-num> channel <channel-num> <linkset-name> <link-
name>
```

Where:

span-num
: integer number. It represents port number in card(indexed from 0).

code-num
: link code(sls assigned to this link).

channel-num
: integer number indicating time slot number(TDM time slot).

linkset-name
: is name set during link creation.

link-name
: name which identifies link in linkset.

Example 5.7.

```
mobicents(10.65.208.215:3435)>linkset link create span 1 code 1 channel 1
 linkset1 link1
Link successfully added
```

5.3.6. Remove Link

`Link` can be removed from in `Linkset` by issuing command with following structure:

```
linkset link delete <linkset-name> <link-name>
```

Where:

linkset-name
: is name set during link creation

link-name
: name which identifies link in linkset

Example 5.8. Link Removal

```
mobicents(10.65.208.215:3435)>linkset link delete linkset1 link1
Link successfully deleted
```

5.3.7. Activate Link

Link can be activated by issuing command with following structure:

```
linkset link activate <linkset-name> <link-name>
```

Where:

linkset-name
: is name set during link creation

link-name
: name which identifies link in linkset

Example 5.9. Link Activation

```
mobicents(10.65.208.215:3435)>linkset link activate linkset1 link1
Link activated successfully
```

5.3.8. Deactivate Link

Link can be deactivated by issuing command with following structure:

```
linkset link deactivate <linkset-name> <link-name>
```

Where:

linkset-name
: is name set during link creation

link-name
: name which identifies link in linkset

Example 5.10. Link Deactivateion

```
mobicents(10.65.208.215:3435)>linkset link deactivate linkset1 link1
Link deactivated successfully
```

5.3.9. Show status

Linkset and Link's status can be viewed by issuing command with following structure:

```
linkset show
```

Example 5.11. Linkset Status

```
mobicents(10.65.208.215:3435)>linkset show
linkset1      dahdi    opc=1           apc=2           ni=0
 state=UNAVAILABLE
    link1       span=1   channelId=1   code=1  state=UNAVAILABLE
```

The possible state of Linkset are

- UNAVAILABLE : Indicates the linkset does not have any “available” links and cannot transport traffic
- SHUTDOWN : Indicates the linkset has been shutdown in the configuration
- AVAILABLE : Indicates the linkset has at least one available link and can carry traffic

The possible state of Link are

- UNAVAILABLE : Indicates the link is not available to carry traffic. This can occur if the link is remotely or locally inhibited by a user. It can also be unavailable if MTP2 has not been able to successfully activate the link connection.
- SHUTDOWN : Indicates the link has been shutdown in the configuration.
- AVAILABLE : Indicates the link is active and able to transport traffic
- FAILED : A link is FAILED when the link is not shutdown but is unavailable at layer2 for some reason. For example Initial Alignment failed or the link test messages sent by MTP3 are not being acknowledged.

5.4. SCCP Management

SCCP provides connectionless and connection-oriented network services. This includes address(GTT) translation and routing, flow control segmentation and reassembly.

A global title is an address (e.g., a dialed 800 number, calling card number, or mobile subscriber identification number) which is translated by SCCP into a destination point code and subsystem number. A subsystem number uniquely identifies an application at the destination signaling point. SCCP is used as the transport layer for TCAP -based services

The first step for SCCP configuring is a service access points (sap) definition. This step is mandatory. Each SCCP stack can use one or more Mtp3UserPart (Refer Section 3.5.4, “Configuring SCCP” about Mtp3UserPart setting). A sap is a logical definition of the Mtp3UserPart (corresponded local SPC, network indicator (NI) and a set of destinations (remote SPC list)). The index of a sap must correspont to the index of the sap.

The second step is a definition of a list of available remote signalling pointcodes (SPC - rsp) and a list of available remote Sub-Systems (SNN - rss). This step is also mandatory. If routing only by GlobalTytle is used the remote Sub-Systems configuring is not required.

As SCCP acts as message router, it requires means to configure routing information. Rules (rule), primary (primary_add) and backup (backup_add) (if backup addresses are available) addresses should be configured.

If XUDT and LUDT messages are availavle in the SS7 network, user should config a set of long message rules (lmr) that will allow long messages. This step is not mandatory. If no long message rule is configured only UDT messages will be used.

The last step is also optional. A user can configure a set of concerned signaling point codes (csp). Each point code will be announced when local SCCP user becomes (un)available.

User can also configure general SCCP parameters (Refer Section 5.4.8, “General parameters” about general parameters setting).

5.4.1. Service access points and destinations management

SCCP service access points (sap) and destinations (dest) are managed by **sccp sap** and **sccp dest** commands. They allows to perform following:

- **sccp sap create**
- **sccp sap modify**
- **sccp sap delete**
- **sccp sap show**
- **sccp dest create**
- **sccp dest modify**
- **sccp dest delete**
- **sccp dest show**

5.4.1.1. Create service access point

Sap can be created by issuing command with following structure:

```
sccp sap create <id> <mtp3-id> <opc> <ni>
```

<id>
A unique number to identify this sap

<mtp3-id>
A Mtp3UserPart index that is used as an index of mtp3UserPart property for SccpStack bean. Refer to Section 3.5.4, “Configuring SCCP” for details. For each Mtp3UserPart a sap must be configured.

<opc>
Specifies the local signaling point code.

<ni>
Specifies the network indicator that forms the part of service information octet (SIO)

For each of sap one or more destinations should be configured. Refer to Section 5.4.1.5, “Create destination for service access point” for details.

Example 5.12. SCCP sap creation

```
mobicents(10.65.208.215:3435)>sccp sap create 1 1 101 2
mobicents(10.65.208.215:3435)>sccp sap create 2 2 102 2
```

5.4.1.2. Modify existing sap

Sap can be modified by issuing command with following structure:

```
sccp sap modify <id> <mtp3-id> <opc> <ni>
```

Meaning of parameters is the same.

5.4.1.3. Delete SCCP service access point

SCCP sap can be deleted by issuing command with following structure:

```
sccp sap delete <id>
```

Where:

<id>
is id set during sap creation

Example 5.13. SCCP sap Removal

```
mobicents(10.65.208.215:3435)>sccp sap delete 1
Service access point successfully removed
```

5.4.1.4. Show SCCP service access point

Saps can be viewed by issuing command with following structure:

```
sccp sap show <id>
```

Where:

<id>
id is optional. If passed only sap matching the id will be shown, else all the saps will be shown

5.4.1.5. Create destination for service access point

Destination can be created by issuing command with following structure:

```
sccp dest create <sap-id> <id> <first-dpc> <last-dpc> <first-sls> <last-sls> <sls-mask>
```

<sap-id>
An identifier of the sap for which the destination is being created

<id>
A number to identify this destination. The number must be unique for each sap.

<first-dpc>
Specifies the first value of the remote signaling point codes range.

<last-dpc>
Specifies the last value of the remote signaling point codes range. If destination specifies a single signalling point code, this value must be equal first-dpc

<first-sls>
Specifies the first value of the SLS range.

<last-sls>
Specifies the last value of the SLS range.

<sls-mask>
Specifies the mask value. SLS af a message will be exposed by bitwise AND operation with this mask before comparing with first-sls and last-sls valuers.

SLS value range is from 0 to 255. If the destination cover all possible SLS's use first-sls=0, last-sls=255, sls-mask=255

Example 5.14. SCCP destination creation

```
mobicents(10.65.208.215:3435)>sccp dest create 1 1 201 201 0 7 7
mobicents(10.65.208.215:3435)>sccp dest create 2 1 300 399 0 255 255
```

5.4.1.6. Modify existing destination for service access point

Destination can be modified by issuing command with following structure:

```
sccp dest modify <sap-id> <id> <first-dpc> <last-dpc> <first-sls> <last-sls> <sls-mask>
```

Meaning of parameters is the same.

5.4.1.7. Delete SCCP destination for service access point

SCCP destination can be deleted by issuing command with following structure:

```
sccp dest delete <sap-id> <id>
```

Where:

<sap-id>
: An identifier of the sap for which the destination has been created

<id>
: is id set during destination creation

Example 5.15. SCCP destination Removal

```
mobicents(10.65.208.215:3435)>sccp destination delete 1 1
Destination definition successfully deleted
```

5.4.1.8. Show SCCP destination for service access point

Destinations can be viewed by issuing command with following structure:

```
sccp dest show <sap-id> <id>
```

Where:

<sap-id>
An identifier of the sap for which the destination has been created

<id>
id is optional. If passed only destination matching the id will be shown, else all destinations of the saps will be shown

5.4.2. Rule Management

SCCP routing rules are managed by **sccp rule** command. It allows to perform following:

- **sccp rule create**
- **sccp rule modify**
- **sccp rule delete**
- **sccp rule show**

5.4.2.1. Create Rule

Rule can be created by issuing command with following structure:

```
sccp rule create <id> <mask> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
<nature-of-address-indicator>  <digits>  <ruleType>  <primary-address-id>  <backup-address-id>
 <loadsharing-algorithm>
```

This command should be specified after **primary_add** and **backup_add** are configured. Please refer Section 5.4.3, “Address Management” on how to configure **primary_add** and **backup_add**

<id>
A unique number to identify this rule

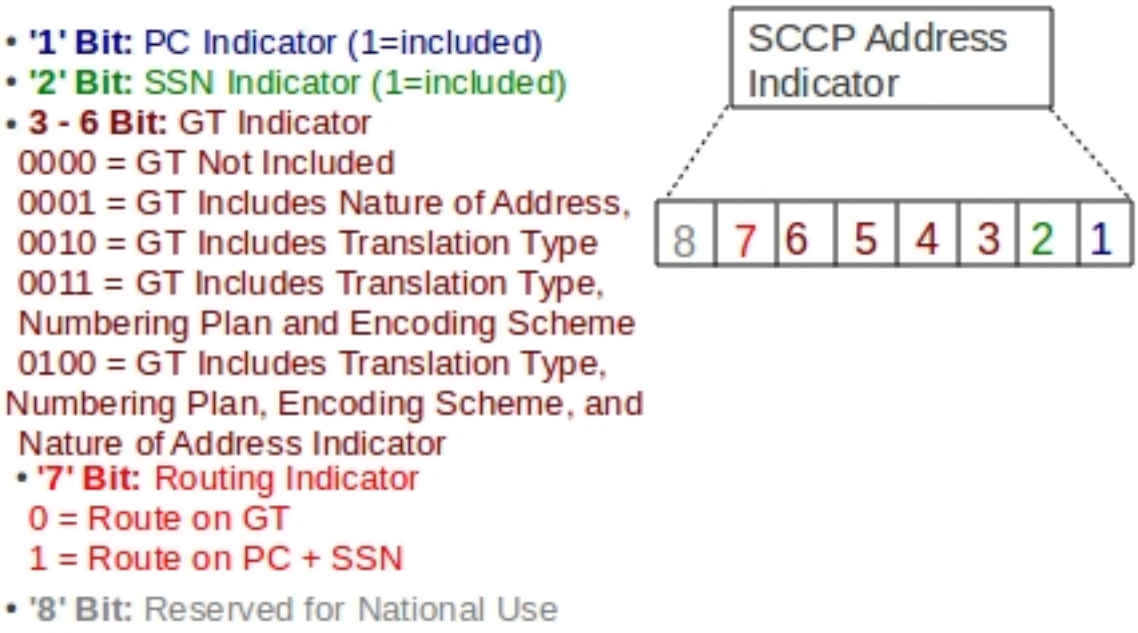
<mask>
mask defines which part of the originally dialed digits remains in the translated digits and which part is replaced by the digits from primary or backup address. mask is divided into sections by separator /. The number of sections in mask should be equal to sections in digits passed in this command and sections in primary or backup address

Table 5.1. mask definitions

Mnemonic	Function
-	Ignore
/	Separator used to split the mask into sections.
K	Keep the original dialed digits of this section into translated digits
R	Replace the original dialed digits of this section with same section from primary or backup address into translated digits

<address-indicator>

The address indicator is the first field in SCCP Party Address(called/calling) and is one octet in length. Its function is to indicate which information elements are present so that the address can be interpreted, in other words, it indicates the type of addressing information that is to be found in the address field. The addressing information from original global title is then compared with passed address information to match the rule.



SCCP Address Indicator

<point-code>
Point code. This is ignored if bit 0 of address-indicator is not set.

<subsystem-number>
Subsystem Number. This is ignored if bit 1 of address-indicator is not set.

<translation-type>
Translation type. This is ignored if GT Indicator is 0000 or 0001

Table 5.2. Translation Type Values

Value	Description
0	Unknown
1 to 63	International Service
64 to 127	Spare
128 to 254	National Network Specific
255	Reserved for Expansion

<numbering-plan>
The Number Plan (NP) field specifies the numbering plan that the address information follows. This is ignored if GT Indicator is 0000, 0001 or 0010

<nature-of-address-indicator>
The Nature of Address Indicator (NAI) field defines the address range for a specific numbering plan. This is only used if GT Indicator is 0100

<digits>
Specifies the string of digits divided into subsections using separator '/' depending on if mask contains separator. The dialed digits should match with theses digits as per rule specified bellow

Table 5.3. digit pattern

Value	Description
-	padding - ignored
*	wildcard - matches any number of digits
?	wildcard - matches exactly one digit
/	sparator used to split the digit pattern into sections. Each section can be processed

Value	Description
	differently as specified by mask parameter.

<ruleType>
Rule type. One of following values is possible.

Table 5.4. Rule type values

Value	Description
solitary	Only one (primary) address is used for routing (<backup-address-id> may be missed in this case).
dominant	Both (primary and backup) addresses are used and mandatory. If both of addresses are available the primary address is used for routing.
loadshared	Both (primary and backup) addresses are used and mandatory. If both of addresses are available either primary or backup address is used for routing. The <loadsharing-algorithm> should be configured in this case.

<primary-address-id>
Identifies the SCCP Address used as the primary translation

<backup-address-id>
Identifies the SCCP Address used as the backup translation in case if pointcode specified by primary address is not available. Backup address is used only dominant and loadshared address types

<loadsharing-algorithm>
This parameter is mandatory if <ruleType> parameter is "loadshared". Loadsharing algorithm is configired here. Possible values of the parameter:

Table 5.5. Rule type values

Value	Description
bit4	if((SLS & 0x10) == 0) <route to primary> else <route to backup> This algorithm is the best if all traffic is local (mobisents stack) originated
bit3	if((SLS & 0x08) == 0) <route to primary> else <route to backup> This algorithm can be used if not all traffic is local (mobisents stack) originated. But only 8 links in the both linksets is acceptable.

Example 5.16. SCCP Rule creation

```
mobicents(10.65.208.215:3435)>sccp rule create 1 R 71 2 8 0 0 3 123456789
 solitary 1
mobicents(10.65.208.215:3435)>sccp rule create 2 R 71 2 8 0 0 3 123456789
 dominant 1 1
mobicents(10.65.208.215:3435)>sccp rule create 2 R 71 2 8 0 0 3 123456789
 loadshared 1 1 bit4
```

5.4.2.2. Modify existing Rule

Rule can be modified by issuing command with following structure:

```
sccp rule modify <id> <mask> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
<nature-of-address-indicator> <digits> <ruleType> <primary-address-id> <backup-address-id>
```

Meaning of parameters is the same.

5.4.2.3. Delete SCCP Rule

SCCP Rule can be deleted by issuing command with following structure:

```
sccp rule delete  <id>
```

Where:

<id>
is id set during rule creation

Example 5.17. SCCP Rule Removal

```
mobicents(10.65.208.215:3435)>sccp rule delete 1
Rule successfully removed
```

5.4.2.4. Show SCCP Rule

Rule's can be viewed by issuing command with following structure:

```
sccp rule show <id>
```

Where:

<id>
id is optional. If passed only rule matching the id will be shown, else all the rules will be shown

5.4.3. Address Management

The command is used to define primary or backup address of translation. The global title address information of this command is combined with the global title being translated by examining the mask provided in the **sccp rule create** command.The syntax remains same except for primary address **sccp primary_add** is used and for backup address **sccp backup_add** is used

- **sccp primary_add create**

  sccp backup_add create
- **sccp primary_add modify**

  sccp backup_add modify
- **sccp primary_add delete**

  sccp backup_add delete
- **sccp primary_add show**

  sccp backup_add show

5.4.3.1. Create Address

Address can be created by issuing command with following structure:

- For primary address

```
sccp primary_add create <id> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
<nature-of-address-indicator> <digits>
```

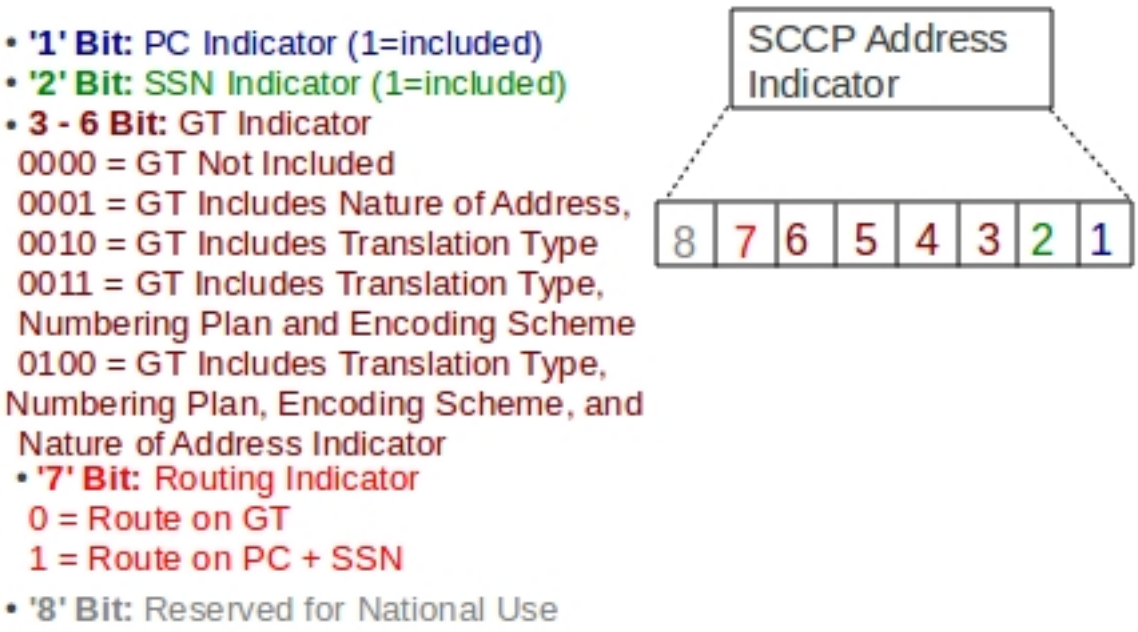
- For backup address

```
sccp backup_add create <id> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
```

```
  <nature-of-address-indicator> <digits>
```

<id>
A unique number to identify this address

<address-indicator>
The address indicator is the first field in SCCP Party Address(called/calling) and is one octet in length. Its function is to indicate which information elements are present so that the address can be interpreted, in other words, it indicates the type of addressing information that is to be found in the address field. The addressing information from original global title is then compared with passed address information to match the rule.



SCCP Address Indicator

<point-code>
Point code. This is ignored if bit 0 of address-indicator is not set.

<subsystem-number>
Subsystem Number. This is ignored if bit 1 of address-indicator is not set.

<translation-type>
Translation type. This is ignored if GT Indicator is 0000 or 0001

Table 5.6. Translation Type Values

Value	Description
0	Unknown
1 to 63	International Service

Value	Description
64 to 127	Spare
128 to 254	National Network Specific
255	Reserved for Expansion

<numbering-plan>
The Number Plan (NP) field specifies the numbering plan that the address information follows. This is ignored if GT Indicator is 0000, 0001 or 0010

<nature-of-address-indicator>
The Nature of Address Indicator (NAI) field defines the address range for a specific numbering plan. This is only used if GT Indicator is 0100

<digits>
The global title address information to translate to, specified as string of digits divided into subsections using separator '/' depending on if mask contains separator.

In addition the digits string can contain

Table 5.7. Address digit

Value	Description
-	padding - ignore
/	Separtor to split the digits into sections. Each section is processed differently as specified by the mask in **sccp rule create** command.

Example 5.18. SCCP Primary Address creation

```
mobicents(10.65.208.215:3435)>sccp primary_add create 1 71 2 8 0 0 3
 123456789
```

Example 5.19. SCCP Backup Address creation

```
mobicents(10.65.208.215:3435)>sccp backup_add create 1 71 3 8 0 0 3
 123456789
```

5.4.3.2. Modify existing Address

Address can be modified by issuing command with following structure:

- For primary address

```
sccp primary_add modify <id> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
<nature-of-address-indicator> <digits>
```

- For backup address

```
sccp backup_add modify <id> <address-indicator> <point-code> <subsystem-number> <translation-
type> <numbering-plan>
<nature-of-address-indicator> <digits>
```

Meaning of parameters is the same.

5.4.3.3. Delete Address

- For primary address

```
sccp primary_add delete <id>
```

- For backup address

```
sccp backup_add delete <id>
```

Where:

<id>
is id set during address creation

Example 5.20. Primary Address Removal

```
mobicents(10.65.208.215:3435)>sccp primary_add delete 1
Rule successfully removed
```

Example 5.21. Backup Address Removal

```
mobicents(10.65.208.215:3435)>sccp backup_add delete 1
Rule successfully removed
```

5.4.3.4. Show Address

Address's can be viewed by issuing command with following structure:

- For primary address

```
sccp primary_add show <id>
```

- For backup address

```
sccp backup_add show <id>
```

Where:

<id>
id is optional. If passed only address matching the id will be shown, else all the addresses will be shown

5.4.4. Remote Signaling Point Management

SCCP resources includes remote signaling point and remote subsystem. Each remote signaling point that SCCP can communicate with must be configured using **sccp rsp** command

- **sccp rsp create**
- **sccp rsp modify**
- **sccp rsp delete**
- **sccp rsp show**

5.4.4.1. Create Remote Signaling Point

Remote signaling point can be create by issuing command with following structure:

```
sccp rsp create <id> <remote-spc> <rspc-flag> <mask>
```

<id>
A unique number to identify this remote signaling point

<remote-spc>
The remote signaling point

<rspc-flag>
32 bit value. Not used for now. Reserved for future

<mask>
32 bit value. Not used for now. Reserved for future

Example 5.22. Remote Signalin Point creation

```
mobicents(10.65.208.215:3435)>sccp rsp create 1 6477 0 0
```

5.4.4.2. Modify existing Remote Signaling Point

Remote Signaling Point can be modified by issuing command with following structure:

```
sccp rsp modify <id> <remote-spc> <rspc-flag> <mask>
```

Meaning of parameters is the same.

5.4.4.3. Delete Remote Signaling Point

```
sccp rsp delete <id>
```

Where:

<id>
is id set during remote signaling point creation

Example 5.23. Remote Signaling Point removal

```
mobicents(10.65.208.215:3435)>sccp rsp delete 1
```

5.4.4.4. Show Remote Signaling Point/s

Remote signaling point can be viewed by issuing command with following structure:

```
sccp rsp show <id>
```

Where:

<id>
id is optional. If passed only remote signaling point matching the id will be shown, else all the addresses will be shown

5.4.5. Remote Sub-System Management

SCCP resources includes remote signaling point and remote subsytem. Each remote subsystem that SCCP can communicate with must be configured using **sccp rss** command

- **sccp rss create**
- **sccp rss modify**
- **sccp rss delete**
- **sccp rss show**

This command should be specified after remote signaling point is configured. Please refer Section 5.4.4, “Remote Signaling Point Management” on how to configure remote signaling point

5.4.5.1. Create Remote Sub-System

Remote subsystem can be created by issuing command with following structure:

```
sccp rss create <id> <remote-spc> <remote-ssn> <rss-flag> <mark-prohibited-when-spc-resuming>
```

<id>
A unique number to identify this remote subsystem

<remote-spc>
The remote signaling point where this remote susbsytem is deployed

<remote-ssn>
The remote subsystem number

<rss-flag>
32 bit value. Not used for now. Reserved for future

<mark-prohibited-when-spc-resuming>
This parameter is optional, its possible value is: prohibitedWhenSpcResuming. When this parameter is present configured subsystem is marked as prohibited when its corresponded signalling point code has been resumed.

Example 5.24. Remote Sub-System creation

```
mobicents(10.65.208.215:3435)>sccp rss create 1 6477 8 0
 prohibitedWhenSpcResuming
```

5.4.5.2. Modify existing Remote Sub-System

Remote Sub-System can be modified by issuing command with following structure:

```
sccp rss modify <id> <remote-spc> <remote-ssn> <rss-flag> <mark-prohibited-when-spc-resuming>
```

Meaning of parameters is the same.

5.4.5.3. Delete Remote Sub-System

```
sccp rss delete <id>
```

Where:

<id>
is id set during remote subsystem creation

Example 5.25. Remote Sub-System removal

```
mobicents(10.65.208.215:3435)>sccp rss delete 1
```

5.4.5.4. Show Remote Sub-System/s

Remote subsystem can be viewed by issuing command with following structure:

```
sccp rss show <id>
```

Where:

<id>
id is optional. If passed only remote subsystem matching the id will be shown, else all will be shown

5.4.6. Long message rules Management

Long message rules describe which message types (UDT/XUDT/LUDT) will be used for outgoing message encoding depends on dpc. If no long message rules is configured only UDT messages will be used.

- **sccp lmr create**
- **sccp lmr modify**
- **sccp lmr delete**
- **sccp lmr show**

5.4.6.1. Create Long message rule

Long message rule can be created by issuing command with following structure:

```
sccp lmr create <id> <first-spc> <last-spc> <long-message-rule-type>
```

<id>
A unique number to identify this Long message rule

<first-spc>
Specifies the first value of the remote signaling point codes range. (for which Long message rule will apply)

<last-spc>
Specifies the last value of the remote signaling point codes range. If Long message rule specifies a single signalling point code, this value must be equal first-spc.

<long-message-rule-type>
Specifies which message types will be used for the remote signaling point codes range. Possible values are: udt, xudt and ludt.

Example 5.26. Long message rule creation

```
mobicents(10.65.208.215:3435)>sccp lmr create 1 201 201 xudt
mobicents(10.65.208.215:3435)>sccp lmr create 2 230 239 udt
```

5.4.6.2. Modify existing Long message rule

Long message rule can be modified by issuing command with following structure:

```
sccp lmr modify <id> <first-spc> <last-spc> <long-message-rule-type>
```

Meaning of parameters is the same.

5.4.6.3. Delete Modify existing Long

```
sccp lmr delete <id>
```

Where:

<id>
: is id set during Long message rule creation

Example 5.27. Long message rule removal

```
mobicents(10.65.208.215:3435)>sccp lmr delete 1
```

5.4.6.4. Show Long message rule/s

Long message rule can be viewed by issuing command with following structure:

```
sccp lmr show <id>
```

Where:

<id>
: id is optional. If passed only Long message rule matching the id will be shown, else all the rules will be shown

5.4.7. Concerned signaling point codes Management

Concerned signaling point codes define a DPC list which will be noticed when local SSN is registered (SSA messages) or unregistered (SSP messages).

- **sccp csp create**
- **sccp csp modify**
- **sccp csp delete**
- **sccp csp show**

5.4.7.1. Create Concerned signaling point code

Concerned signaling point codes can be created by issuing command with following structure:

```
sccp csp create <id> <spc>
```

<id>
A unique number to identify this Concerned signaling point code

<spc>
Specifies the value of the remote signaling point code, which will be noticed.

Example 5.28. Concerned signaling point code creation

```
mobicents(10.65.208.215:3435)>sccp csp create 1 201
mobicents(10.65.208.215:3435)>sccp csp create 2 202
```

5.4.7.2. Modify existing Concerned signaling point code

Concerned signaling point code can be modified by issuing command with following structure:

```
sccp csp modify <id> <spc>
```

Meaning of parameters is the same.

5.4.7.3. Delete Concerned signaling point code

```
sccp csp delete <id>
```

Where:

<id>
is id set during Concerned signaling point code creation

Example 5.29. Concerned signaling point code removal

```
mobicents(10.65.208.215:3435)>sccp csp delete 1
```

5.4.7.4. Show Concerned signaling point code/s

Concerned signaling point code can be viewed by issuing command with following structure:

```
sccp csp show <id>
```

Where:

<id>
id is optional. If passed only Concerned signaling point code matching the id will be shown, else all the codes will be shown

5.4.8. General parameters

User can set several general parameters that influence the whole SCCP stack.

Table 5.8. SCCP general parameters

Mnemonic name	Function	Value range	Default value
zMarginXudtMessage	If the XUDT message data length greater this value, segmentation is processed. Otherwise no segmentation.	160 - 255	240
reassemblyTimerDelay	SCCP segmented message reassembling timeout (in milliseconds).	10000 - 20000	15000
maxDataMessage	Max available SCCP message data for all message types.	2560 - 3952	2560
removeSpc	Remove PC from calledPartyAddress when sending to MTP3. After Global Title Translation, if the SCCP	true or false	true

Mnemonic name	Function	Value range	Default value
	address includes the destination point code (DPC) however Address Indicator (AI) indicates route on Global Title and removeSpc is set to true, DPC will be removed from SCCP Address. The same rule applies for both calling as well as called party SCCP Address.		
sstTimerDuration_Min	Min (starting) SST sending interval (in milliseconds).	5000 - 10000	10000
sstTimerDuration_Max	Max (after increasing) SST sending interval (in milliseconds).	600000 - 1200000	600000
sstTimerDuration_IncreaseFactor	Multiplication factor of SST sending interval (next interval will be greater the current by sstTimerDuration_IncreaseFactor). This value has type "double".	1 - 4	1.5

5.4.8.1. General parameters setting

General parameter can be set by issuing command with following structure:

```
sccp set <parameter-name> <parameter-value>
```

<parameter-name>
A mnemonic name of a parameter.

<parameter-value>
A value of a parameter.

Example 5.30. General parameters setting

```
mobicents(10.65.208.215:3435)>sccp set zMarginXudtMessage 230
mobicents(10.65.208.215:3435)>sccp set removeSpc false
```

5.4.8.2. General parameters getting

General parameter can be got by issuing command with following structure:

```
sccp get <parameter-name>
```

<parameter-name>
A mnemonic name of a parameter. This parameter is optional. If a mnemonic name is absent all parameter values will be returned.

Example 5.31. General parameters getting

```
mobicents(10.65.208.215:3435)>sccp get zMarginXudtMessage
mobicents(10.65.208.215:3435)>sccp get
```

5.5. M3UA Management

M3UA stack is also responsible to manage the SCTP Associations.

5.5.1. M3UA Management - SCTP

M3UA - SCTP is managed by `sctp` command. It allows to perform following:

- **sctp server create**
- **sctp server destroy**
- **sctp server start**
- **sctp server stop**
- **sctp server show**
- **sctp association create**
- **sctp association destroy**
- **sctp association show**

5.5.1.1. Create SCTP Server

SCTP Server can be created by issuing command with following structure:

```
sctp server create <server-name> <host-ip> <host-port> <socket-type>
```

Where:

server-name
: Unique name assigned to the server.

host-ip
: The host ip address where underlying SCTP server socket will bind. For SCTP multi-home support, you can pass comma separated ip addresses that this server socket will bind to. If the primary ip address becomes unavailable, it will automatically fall back to secondary address. For socket-type TCP, comma separated values will be ignored and only first value (primary address) will be used

host-port
: The host port where underlying SCTP server socket will bind

socket-type
: This is optional. If not passed default is SCTP else specify as TCP.

Example 5.32. SCTP Server creation

```
mobicents(127.0.0.1:3436)>sctp server create TestServer 127.0.0.1 2905
Successfully added Server=TestServer
```

Example 5.33. SCTP Server creation with multi-home

```
mobicents(127.0.0.1:3436)>sctp server create TestServer
 10.2.50.145,10.2.50.146 2905
Successfully added Server=TestServer
```

Example 5.34. TCP Server creation

```
mobicents(127.0.0.1:3436)>sctp server create TestServerTCP 10.2.50.145 2906
 TCP
Successfully added Server=TestServerTCP
```

5.5.1.2. Destroy SCTP Server

SCTP Server can be destroyed by issuing command with following structure:

```
sctp server destroy <server-name>
```

Where:

server-name
: Unique name of the server to be destroyed. Make sure server is stopped before destroying.

Example 5.35. Destroy SCTP Server

```
mobicents(127.0.0.1:3436)>sctp server destroy TestServer
Successfully removed Server=TestServer
```

5.5.1.3. Start SCTP Server

SCTP Server can be started by issuing command with following structure:

```
sctp server start <server-name>
```

Where:

server-name
: Unique name of the server to be started. The underlying SCTP server socket is bound to ip:port configured at creation time.

Example 5.36. Start SCTP Server

```
mobicents(127.0.0.1:3436)>sctp server start TestServer
Successfully started Server=TestServer
```

5.5.1.4. Stop SCTP Server

SCTP Server can be stopped by issuing command with following structure:

```
sctp server stop <server-name>
```

Where:

server-name
: Unique name of the server to be stopped. The underlying socket is closed at this point and all resource are released.

Example 5.37. Stop SCTP Server

```
mobicents(127.0.0.1:3436)>sctp server stop TestServer
Successfully stopped Server=TestServer
```

5.5.1.5. Show SCTP Server

SCTP Server's configuration can be viewed by issuing command with following structure:

```
sctp server show
```

Example 5.38. Show SCTP Server

```
mobicents(local)>sctp server show

SERVER TCP name=TestServerTCP started=false hostIp=10.2.50.145 hostPort=2906
Associations:

SERVER SCTP name=TestServer started=false hostIp=10.2.50.145 hostPort=2905
 secondaryHost=10.2.50.146
Associations:
```

5.5.1.6. Create SCTP Association

Association can be created by issuing command with following structure:

```
sctp association create <assoc-name> <CLIENT | SERVER> <server-name> <peer-ip> <peer-port> <host-
ip> <host-port> <socket-type>
```

Where:

assoc-name
: Unique name of the association

CLIENT | SERVER
: If this association is client side or server side. If its client side, it will initiate the connection to peer and bind's to host-ip:host-port trying to connect to peer-ip:peer-port. For SCTP multi-home support, you can pass comma separated ip addresses that this association will bind

to. If the primary ip address becomes unavailable, it will automatically fall back to secondary address. For socket-type TCP, comma separated values will be ignored and only first value (primary address) will be used

If its server side, it waits for peer to initiate connection. The connection request will be accepted from peer-ip:peer-port. host-ip and host-port is not required, even if passed it will be ignored

server-name
: If this association is server side, server-name must be passed to associate with server. Server with server-name should have already been created by using command Section 5.5.1.1, “Create SCTP Server”

  If this association is client side, server-name shouldn't be passed.

socket-type
: This is optional. If not passed default is SCTP else specify as TCP. If association is of SERVER type, the socket-type should match with one specified while creating server.

Example 5.39. Create CLIENT SCTP Association

```
mobicents(local)>sctp association create Assoc1 CLIENT 192.168.56.101 2905
 192.168.56.1,192.168.56.1 2905
Successfully added client Associtaion=Assoc1
```

Example 5.40. Create SERVER SCTP Association

```
mobicents(192.168.56.1:3436)>sctp association create Assoc2 SERVER
 TestServer 192.168.56.1 2905
Successfully added server Association=TestServer
```

5.5.1.7. Destroy SCTP Association

Association can be destroyed by issuing command with following structure:

```
sctp association destroy <assoc-name>
```

Where:

assoc-name
: Unique name of the association to be destroyed

Example 5.41. Destroy SCTP Association

```
mobicents(192.168.56.1:3436)>sctp association destroy Assoc1
Successfully removed association=Assoc1
```

5.5.1.8. Show SCTP Association

Configuration of Association can be viewed by issuing command with following structure:

```
sctp association show
```

Example 5.42. Show SCTP Association

```
mobicents(local)>sctp association show

ASSOCIATION SCTP name=Assoc1 started=false peerIp=192.168.56.101
 peerPort=2905 hostIp=192.168.56.1 hostPort2905 type=CLIENT
 secondaryHost=192.168.56.1

ASSOCIATION SCTP name=Assoc2 started=false peerIp=192.168.56.1 peerPort=2905
 server=TestServer type=SERVER
```

5.5.2. M3UA Management

M3UA is managed by `m3ua` command. It allows to perform following:

- **m3ua as create**
- **m3ua as destroy**
- **m3ua as show**
- **m3ua asp create**
- **m3ua asp destroy**
- **m3ua asp show**
- **m3ua asp start**
- **m3ua asp stop**

- **m3ua as add**
- **m3ua as remove**
- **m3ua route add**
- **m3ua route remove**
- **m3ua route show**

5.5.2.1. Create AS

Application Server (AS) can be created by issuing command with following structure:

```
m3ua as create <as-name> <AS | SGW | IPSP> mode <SE | DE> ipspType <client | server> rc <routing-context> traffic-mode <traffic mode> <network-appearance>
```

Where:

as-name
: simple string name, which identifies AS. Make sure this is unique. This is mandatory parameter

AS | SGW | IPSP
: Specify if this is of type AS or SGW or IPSP. This is mandatory parameter

SE | DE
: Specify if the single or double exchange of ASP State Maintenance (ASPSM) and ASP Traffic Maintenance (ASPTM) messages should be performed. This is mandatory parameter.

client | server
: If As if of type IPSP, speicfy here if its client or server type.

routing-context
: refers to Routing Context already configured on M3UA stack on SGW side. This is optional parameter. If no Routing Context is passed, Application Server Process (assigned to this AS) may not be configured to process signalling traffic related to more than one Application Server, over a single SCTP Association

  Also if ASP is configured to process siganlling traffic from always one AS, irrespective of received messages have routing context set or not, it will always be delieverd to AS for further processing. However if ASP is configured to process siganlling traffic related to more than one AS over a single SCTP Association and signalling message is received without RC, it drops the message and sends back Error message. Respective log4j error is also logged.

traffic-mode
: Traffic mode for ASP's. By default its loadshare. Mobicents M3UA only supports loadshare and override, broadcast is not supported.

network-appearance
: The Network Appearance is a M3UA local reference shared by SG and AS (typically an integer) that, together with an Signaling Point Code, uniquely identifies an SS7 node by indicating the specific SS7 network to which it belongs. It can be used to distinguish between signalling traffic associated with different networks being sent between the SG and the ASP over a common SCTP association. This is optional.

Example 5.43. AS (IPSP) creation

```
mobicents(127.0.0.1:3435)>m3ua as create AS1 IPSP mode DE ipspType server rc
 1 traffic-mode loadshare
Successfully created AS name=AS1
```

Example 5.44. AS creation

```
mobicents(127.0.0.1:3435)>m3ua as create AS2 AS mode SE rc 100 traffic-mode
 loadshare network-appearance 12
Successfully created AS name=AS2
```

5.5.2.2. Destroy AS

Application Server (AS) can be destroyed by issuing command with following structure:

```
m3ua as destroy <as-name>
```

Where:

as-name
: Simple string name, which identifies AS. Make sure AS is in state INACTIVE and all the ASP's are unassigned before destroying

Example 5.45. Destroy AS

```
mobicents(127.0.0.1:3435)>m3ua as destroy AS1
Successfully destroyed AS name=AS1
```

5.5.2.3. Show AS

Application Server configured can viewed by issuing command with following structure:

```
m3ua as show
```

Example 5.46. Show AS

```
mobicents(local)>m3ua as show

AS name=AS2 functionality=AS mode=SE rc=[100] trMode=2 defaultTrMode=2 na=12
 peerFSMState=DOWN
Assigned to :


AS name=AS1 functionality=IPSP mode=DE ipspType=SERVER rc=[1] trMode=2
 defaultTrMode=2 localFSMState=DOWN peerFSMState=DOWN
Assigned to :
```

5.5.2.4. Create ASP

Application Server Process (ASP) can be created by issuing command with following structure:

```
m3ua asp create <asp-name> <sctp-association>
```

Where:

asp-name
: Name of this ASP. It should be unique

sctp-association
: name of SCTP Association

Example 5.47. ASP creation

```
mobicents(127.0.0.1:3435)>m3ua asp create ASP1 Assoc1
Successfully created AS name=ASP1
```

5.5.2.5. Destroy ASP

ASP can be destroyed by issuing command with following structure:

```
m3ua asp destroy <asp-name>
```

Where:

asp-name
: Name of this ASP to be destroyed. Make sure ASP is stopped before destroying

Example 5.48. Destroy ASP

```
mobicents(127.0.0.1:3435)>m3ua asp destroy ASP1
Successfully destroyed ASP name=ASP1
```

5.5.2.6. Show ASP

ASP configured can be viewed by issuing command with following structure:

```
m3ua asp show
```

Example 5.49. Show ASP

```
mobicents(local)>m3ua asp show

ASP name=ASP1 sctpAssoc=Assoc1 started=false
Assigned to :
```

5.5.2.7. Start ASP

Application Server Process (ASP) can be started with following structure

```
m3ua asp start <asp-name>
```

Where:

asp name
: name of ASP created earlier. Make sure ASP you are trying to start is assigned to at least one AS

Example 5.50. Start ASP

```
mobicents(127.0.0.1:3435)>m3ua asp start ASP1
```

```
Successfully started ASP name=ASP1
```

5.5.2.8. Stop ASP

Application Server Process (ASP) can be stopped with following structure

```
m3ua asp stop <asp-name>
```

Where:

asp name
: name of ASP started earlier.

Example 5.51. Stop ASP

```
mobicents(127.0.0.1:3435)>m3ua asp stop ASP1
Successfully stopped ASP name=ASP1
```

5.5.2.9. Add ASP to AS

Application Server Process (ASP) can be assigned to Application Server (AS) with following structure

```
m3ua as add <as-name> <asp-name>
```

Where:

as name
: name of AS created earlier

asp name
: name of ASP created earlier

Note

Mobicents M3UA supports configuring ASP to process signalling traffic related to more than one Application Server, over a single SCTP Association. However you need to make sure that all the AS's that ASP is shared with has Routing Context (unique) configured.

Example 5.52. Add ASP to AS

```
mobicents(127.0.0.1:3435)>m3ua as add AS1 ASP1
Successfully added ASP name=ASP1 to AS name=AS1
```

5.5.2.10. Remove ASP from AS

Application Server Process (ASP) can be unassigned from Application Server (AS) with following structure

```
m3ua as remove <as-name> <asp-name>
```

Where:

as name
: name of AS

asp name
: name of ASP

Example 5.53. Remove ASP from AS

```
mobicents(127.0.0.1:3435)>m3ua as remove AS1 ASP1
Successfully removed ASP name=ASP1 from AS name=AS1
```

5.5.2.11. Add Route

Configure the destination point code that message will be routed to

```
m3ua route add <as-name> <dpc> <opc> <si>
```

Where:

as name
: name of AS created earlier

dpc
: Destination point code

opc
: Originating point code

si
: Service Indicator

Example 5.54. Add Route

```
mobicents(127.0.0.1:3435)>m3ua route add AS1 2 -1 -1
```

5.5.2.12. Remove Route

Remove the As configured for the destination point code

```
m3ua route remove <as-name> <dpc> <opc> <si>
```

Where:

as name
: name of AS assigned to route message for this dpc

dpc
: Destination point code

opc
: Originating point code

si
: Service Indicator

Example 5.55. Remove Route

```
mobicents(127.0.0.1:3435)>m3ua route remove AS1 2 -1 -1
```

5.5.2.13. Show Route

Show all the routes configured

```
m3ua route show
```

Example 5.56. Show Route

```
mobicents(local)>m3ua route show

2:-1:-1        AS1,AS2,
```

Chapter 6. ISUP

ISUP(ISDN User Part or ISUP) is part of SS7 which is used to establish telephone calls and manage call switches(exchanges). Exchanges are connected via E1 or T1 trunks. Each trunk is divided by means of TDM into time slots. Each time slot is distinguished as circuit. Circuits (identified by code) are used as medium to transmit voice data between user equipment (or exchanges if more than one is involved).

ISUP allows not only to setup a call, but to exchange information about exchange state and its resources(circuits).

Note

Mobicents ISUP is based on ITU-T Q.76X series of documents.

6.1. ISUP Configuration

Mobicents ISUP stack is configured with simple properties. Currently following properties are supported:

Table 6.1. ISUP Configuration options

Name	Default value	Value range	Description
ni	None, must be provided	0-3	Sets value of network indicator that should be used by stack.
localspc	None, must be provided	0 - (2^14)-1	Sets local signaling point code. It will be used as OPC for outgoing signaling units.
t1	4s	4s - 15s	Sets T1 value. Started when REL is sent. See A.1/Q.764
t5	5 min.	5min - 15 min	Sets T5 value. Started when initial REL is sent. See A.1/Q.764
t7	20s	20s -30s	Sets T7 value. (Re)Started when Address Message is sent. See A.1/Q.764

Name	Default value	Value range	Description
t12	15s	15s - 60s	Sets T12 value. Started when BLO is sent. See A.1/Q.764
t13	5min	5min - 15min	Sets T13 value. Started when initial BLO is sent. See A.1/Q.764
t14	5s	15s - 60s	Sets T14 value. Started when UBL is sent. See A.1/Q.764
t15	5min	5min - 15min	Sets T15 value. Started when initial UBL is sent. See A.1/Q.764
t16	5s	15s - 60s	Sets T16 value. Started when RSC is sent. See A.1/Q.764
t17	5min	5min - 15min	Sets T17 value. Started when initial RSC is sent. See A.1/Q.764
t18	5s	15s - 60s	Sets T18 value. Started when CGB is sent. See A.1/Q.764
t19	5min	5min - 15min	Sets T19 value. Started when initial CGB is sent. See A.1/Q.764
t20	5s	15s - 60s	Sets T20 value. Started when CGU is sent. See A.1/Q.764
t21	5min	5min - 15min	Sets T21 value. Started when initial CGU is sent. See A.1/Q.764
t22	5s	15s - 60s	Sets T22 value. Started when GRS is sent. See A.1/Q.764
t23	5min	5min - 15min	Sets T23 value. Started when initial

Name	Default value	Value range	Description
			GRS is sent. See A.1/Q.764
t28	10s	10s	Sets T28 value. Started when CQM is sent. See A.1/Q.764
t33	12s	12s - 15s	Sets T33 value. Started when INR is sent. See A.1/Q.764

Note that before start user must provide two interfaces to stack:

Mtp3UserPart
: implementation of transport layer which should be used by stack

CircuitManager
: circuit manager implementation. This interface stores information on mapping between `CIC`(Circuit Identification Code) and `DPC`(Destination Point Code) used as destination for outgoing messages.

6.2. ISUP Usage

The `org.mobicents.protocols.ss7.isup.ISUPStack` interface defines the methods required to represent ISUP Protocol Stack. ISUPStack exposes `org.mobicents.protocols.ss7.isup.ISUPProvider`. This interface defines the methods that will be used by any registered ISUP User application implementing the `org.mobicents.protocols.ss7.isup.ISUPListener` to listen ISUP events(messages and timeouts).

6.3. ISUP Example

Below is simple example of stack usage:

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.mobicents.protocols.ss7.isup.ISUPEvent;
import org.mobicents.protocols.ss7.isup.ISUPListener;
import org.mobicents.protocols.ss7.isup.ISUPProvider;
import org.mobicents.protocols.ss7.isup.ISUPStack;
import org.mobicents.protocols.ss7.isup.ISUPTimeoutEvent;
import org.mobicents.protocols.ss7.isup.ParameterException;
import org.mobicents.protocols.ss7.isup.impl.ISUPStackImpl;
import org.mobicents.protocols.ss7.isup.message.ISUPMessage;
```

```
import org.mobicents.ss7.linkset.oam.Layer4;
import org.mobicents.ss7.linkset.oam.Linkset;

public class ISUPTest implements ISUPListener
{

    protected ISUPStack stack;
    protected ISUPProvider provider;

    protected Linkset isupLinkSet;


    public void setUp() throws Exception {

        this.isupLinkSet = ....; //same linksets as in SS7Service
        this.stack = new ISUPStackImpl();
        this.stack.configure(getSpecificConfig());
        this.provider = this.stack.getIsupProvider();
        this.provider.addListener(this);
        Mtp3UserPart userPart = // create with proper factory, dahdii, dialogi, m3ua
        this.stack.setMtp3UserPart(userPart);
        CircuitManagerImpl circuitManager = new CircuitManagerImpl();
        circuitManager.addCircuit(1, 431613); // CIC - 1, DPC for it - 431613


        this.stack.setCircuitManager(circuitManager);
        this.stack.start();

    }


    public void onEvent(ISUPEvent event) {
        ISUPMessage msg = event.getMessage();
        switch(msg.getCircuitIdentificationCode().getCIC())
        {
            case AddressCompleteMessage._COMMAND_CODE:
            //only complete
            break;
            case ConnectedMessage._COMMAND_CODE:
            case AnswerMessage._COMMAND_CODE:
            //we are good to go
            ConnectedNumber cn = (ConnectedNumber)msg.getParameter(ConnectedNumber._PARAMETER_CODE);
            //do something
            break;
            case ReleaseMessage._COMMAND_CODE:
            //remote end does not want to talk
            RealeaseCompleteMessage rlc = provider.getMessageFactory().createRLC();
            rlc.setCircuitIdentificationCode(msg.getCircuitIdentificationCode());
            rlc.setCauseIndicators(   ((ReleaseComplete)msg).getCauseIndicators());
            provider.sendMessage(rlc);

        }

    }

    public void onTimeout(ISUPTimeoutEvent event) {
        switch(event.getTimerId())
        {
```

```
            case ISUPTimeoutEvent.T1:
                //do something
                break;
            case ISUPTimeoutEvent.T7:
                //do even more
                break;
        }

    }

}
```

Chapter 7. SCCP

The Signaling Connection Control Part (SCCP) is defined in ITU-T Recommendations Q.711-Q.716. SCCP sits on top of Message Transfer Part 3 (MTP3) in the SS7 protocol stack. The SCCP provides additional network layer functions to provide transfer of noncircuit-related (NCR) signaling information, application management procedures and alternative, more flexible methods of routing.

7.1. Routing Management

SCCP provides a routing function that allows signaling messages to be routed to a signaling point based on dialed digits, for example. This capability is known as Global Title Translation (GTT), which translates what is known as a global title (for example, dialed digits for a toll free number) into a signaling point code and a subsystem number so that it can be processed at the correct application.

Routing rules are configured using the Command Line Interface as explained Section 5.4, “SCCP Management”

7.1.1. GTT Configuration

GTT is performed in two stages. First is matching the rule and second is actual translation.

For matching the rule, the called party address global title digits are matched with <digits> configured in **sccp rule create** Section 5.4.2.1, “Create Rule” command above. Once the digits match actual translation is done

Matching rule

As explained in **sccp rule create** Section 5.4.2.1, “Create Rule” command the <digits> can be divided into sections using the "/" separate character. Each section defines set of digits to be matched. Wild card * can be used to match any digits and ? can be used to match exatcly one digit

For example Rule is to match starting 4 digits (should be 1234) and doesn't care for rest; the <digits> in the command will be 1234/*. If the Rule is such that starting 3 digits should be 123, doesn't care for other three digits but last two digits should be 78; the <digits> in the command will be 123/???/78. If digit to digit matching is needed the the <digits> in the command will be exact digits to be matched without sections.

Translation

For translation each section in <mask> defined in **sccp rule create** command defines how replacement operation is performed. If <mask> defines K, the originally dialed digits are kept and if <mask> defines R the digits from primary address or back address are used. The primary/backup address should always define the point code and the translated address will

always have this point code. If the primary/backup address defines the subsystem number the translated address will also have this subsystem number. The address-indicator of translated address is always from primary/backup address. See bellow examples

Example 1 : Remove the Global Title and add PC and SSN

Element	Address Indicator	PC	SSN	TT	NP	NAI	Digits
Dialed Address	0 0 0 1 0 0 0 0			1			123456789
Rule Address	0 0 0 1 0 0 0 0			1			123456789
Rule mask							R
Primary Address	0 1 0 0 0 0 1 1	123	8				-
Translated Address	0 1 0 0 0 0 1 1	123	8				

GTT - Example 1

Example 2 : Partial match

Match a eight digit number starting "800", followed by any four digits, then "9". If the translated digits is not null and if the primary/backup address has no Global Title, the Global Title from dialed address is kept with new translated digits.

Element	Address Indicator	PC	SSN	TT	NP	NAI	Digits
Dialed Address	0 0 0 1 0 0 0 0			1			80012349
Rule Address	0 0 0 1 0 0 0 0			1			800/????/9
Rule mask							R/K/R
Primary Address	0 0 0 0 0 0 0 1	123					123/----/4
Translated Address	0 0 0 1 0 0 0 1	123		1			12312344

GTT - Example 2

Example 3 : Partial match

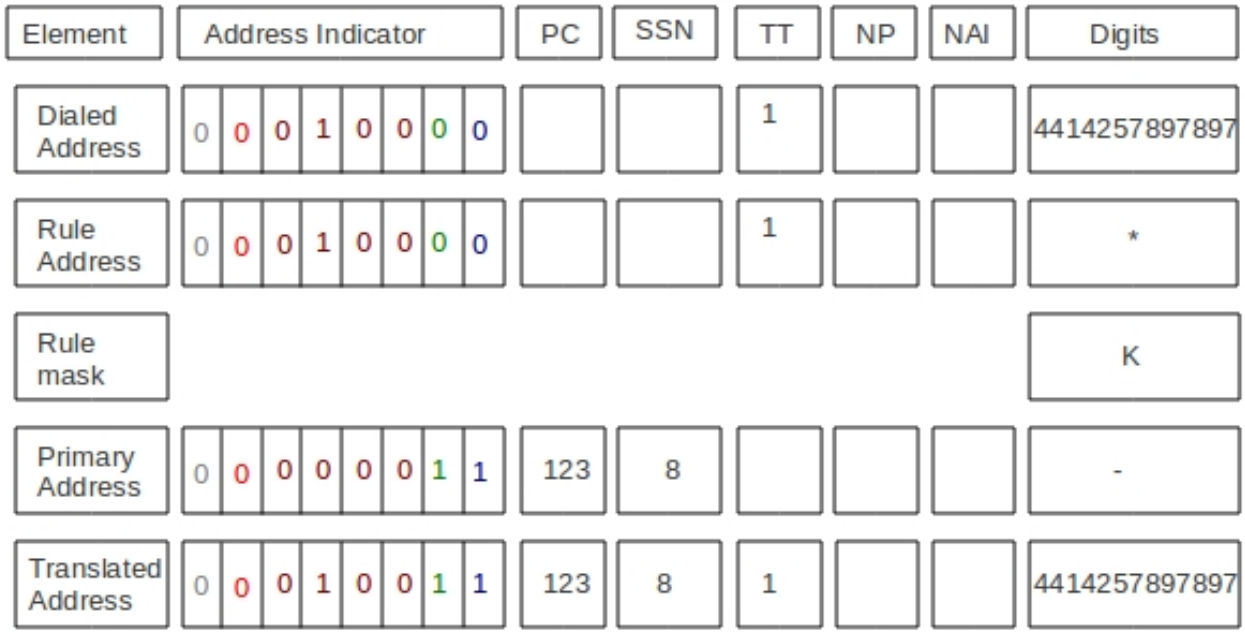
Match "800800", followed by any digits Remove the first six digits. Keep any following digits in the Input. Add a PC(123) and SSN (8).

Element	Address Indicator	PC	SSN	TT	NP	NAI	Digits
Dialed Address	0 0 0 1 0 0 0 0			1			80080012345
Rule Address	0 0 0 1 0 0 0 0			1			800800/*
Rule mask							R/K
Primary Address	0 0 0 0 0 0 1 1	123	8				-/-
Translated Address	0 0 0 1 0 0 1 1	123	8	1			12345

GTT - Example 3

Example 4 : Partial match

Match any digits keep the digits in the and add a PC(123) and SSN (8). If the translated digits is not null and if the primary/backup address has no Global Title, the Global Title from dialed address is kept with new translated digits.



GTT - Example 4

7.2. SCCP Usage

The instance of `org.mobicents.protocols.ss7.sccp.SccpStack` acts as starting point. All the sccp messages sent by SCCP User Part are routed as per the rule configured in Router

Note

The term SCCP User Part refers to the applications that use SCCP's services.

The SCCP User Part gets handle to `SccpStack` by doing JNDI look-up as explained in Section 7.3, “Access Point”

`SccpStack` exposes `org.mobicents.protocols.ss7.sccp.SccpProvider` that interacts directly with SccpStack. This interface defines the methods that will be used by SCCP User Part to send `org.mobicents.protocols.ss7.sccp.message.SccpMessage` and register `org.mobicents.protocols.ss7.sccp.SccpListener`'s to listen for incoming SCCP messages.

SCCP User Part registers SccpListener for specific local subsystem number. For every incoming SccpMessage, if the called subsystem matches with this local subsystem, the corresponding SccpListner is called.

SccpProvider also exposes `org.mobicents.protocols.ss7.sccp.message.MessageFactory` and `org.mobicents.protocols.ss7.sccp.parameter.ParameterFactory` to create new SccpMessage. For transfer data via connectionless service `org.mobicents.protocols.ss7.sccp.message.SccpDataMessage` is used. (This class use UDT, XUDT, LUDT SCCP message type for message transfer.)

7.3. Access Point

`SS7 Service` provides user with access point to `SCCP` protocol/stack.

To get handle to `SccpStack` do the JNDI look-up passing the JNDI name configured in SS7 service as explained in Section 3.5.8, “Configuring SS7Service”

```
        private static SccpProvider getSccpProvider() throws NamingException {

            // no arg is ok, if we run in JBoss
            InitialContext ctx = new InitialContext();
            try {
                String providerJndiName = "/mobicents/ss7/sccp";
                return ((SccpStack) ctx.lookup(providerJndiName)).getSccpProvider();

            } finally {
                ctx.close();
            }
        }
```

7.4. SCCP User Part Example

Below is SCCP User Part example listening for incoming SCCP message and sending back new message

```
public class Test implements SccpListener {

    private SccpProvider sccpProvider;
    private SccpAddress localAddress;
    private int localSsn = 8;

    private static SccpProvider getSccpProvider() throws NamingException {

        // no arg is ok, if we run in JBoss
        InitialContext ctx = new InitialContext();
        try {
            String providerJndiName = "/mobicents/ss7/sccp";
            return ((SccpStack) ctx.lookup(providerJndiName)).getSccpProvider();
        } finally {
            ctx.close();
        }
```

```
    }

    public void start() throws Exception {

        this.sccpProvider = getSccpProvider();

        int translationType = 0;

        GlobalTitle gt = GlobalTitle.getInstance(translationType,
                NumberingPlan.ISDN_MOBILE, NatureOfAddress.NATIONAL, "1234");

      localAddress = new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_GLOBAL_TITLE, -1, gt, 0);

        this.sccpProvider.registerSccpListener(this.localSsn, this);
    }

    public void stop() {
        this.sccpProvider.deregisterSccpListener(this.localSsn);
    }

    public void onMessage(SccpDataMessage message) {

        localAddress = message.getCalledPartyAddress();
        SccpAddress remoteAddress = message.getCallingPartyAddress();

        // now decode content
        byte[] data = message.getData();

        // processing a request
        byte[] answerData = new byte[10];
        // put custom executing code here and fill answerData

        HopCounter hc = this.sccpProvider.getParameterFactory().createHopCounter(5);

        SccpDataMessage sccpAnswer = this.sccpProvider.getMessageFactory().createDataMessageClass1(remoteAddress,
                message.getSls(), localSsn, false, hc, null);

        try {
            this.sccpProvider.send(sccpAnswer);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void onNotice(SccpNoticeMessage message) {
    }

    public void onCoordRequest(int dpc, int ssn, int multiplicityIndicator) {
    }

    public void onCoordResponse(int dpc, int ssn, int multiplicityIndicator) {
    }

    public void onState(int dpc, int ssn, boolean inService, int multiplicityIndicator) {
    }

    public void onPcState(int dpc, SignallingPointStatus status, int restrictedImportanceLevel, RemoteSccpStatus
    }
```

```
}
```

Chapter 8. TCAP

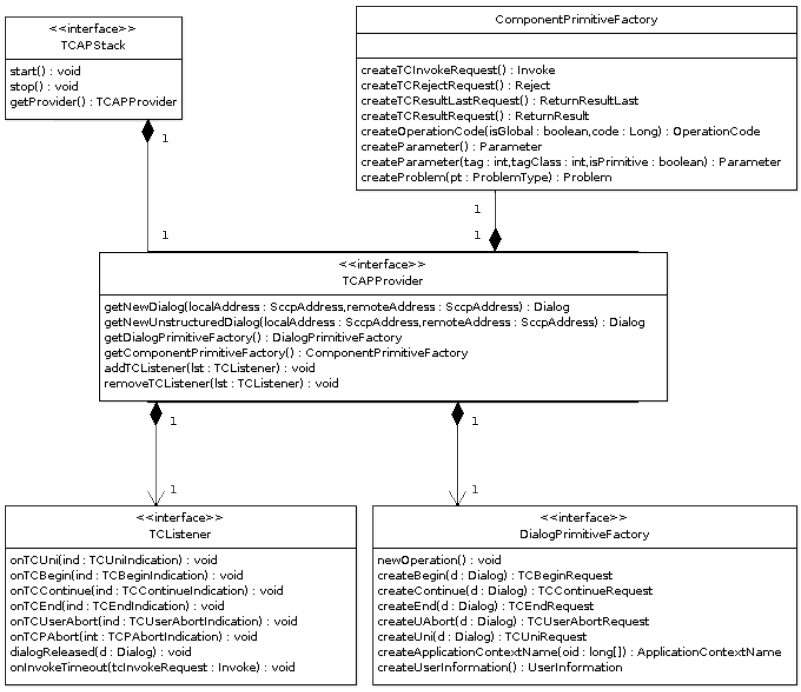
The Transaction Capabilities Application Part (TCAP) is defined in ITU-T Recommendations Q.771-Q.775. TCAP allows services at network nodes to communicate with each other using an agreed-upon set of data elements. Its primary purpose is to facilitate multiple concurrent dialogs between the same sub-systems on the same machines, using Transaction IDs to differentiate these, similar to the way TCP ports facilitate multiplexing connections between the same IP addresses on the Internet.

8.1. Mobicents jSS7 Stack TCAP Usage

The `org.mobicents.protocols.ss7.tcap.api.TCAPStack` interface defines the methods required to represent TCAP Protocol Stack. TCAPStack exposes `org.mobicents.protocols.ss7.tcap.api.TCAPProvider` that interacts directly with TCAPStack. TCAPProvider defines methods that will be used by TCAP User Part to create new `org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog` to be sent across network. TCAP User Part also allows to register`org.mobicents.protocols.ss7.tcap.api.TCListener` to listen TCAP messages.

TCAPProvider also exposes `org.mobicents.protocols.ss7.tcap.api.DialogPrimitiveFactory` to create dialog primitives and `org.mobicents.protocols.ss7.tcap.api.ComponentPrimitiveFactory` to create components. Components are a means of invoking an operation at a remote node

The UML Class Diagram looks like



Mobicents jSS7 Stack TCAP Class Diagram

The `org.mobicents.protocols.ss7.tcap.TCAPStackImpl` is concrete implementation of `TCAPStack`. The TCAP User Part creates instance of `TCAPStackImpl` passing the reference of `SccpProvider` and new instance of `SccpAddress` representing address to which bind listener. The `TCAP` stack creates internaly Mobicents `MAP` Stack implementation. Passed `SccpAddress` is used to match against incoming messages destination address.

```
              SccpProvider sccpProvider = getSccpProvider(); //JNDI lookup of SCCP Stack and
get Provider
            SccpAddress localAddress createLocalAddress();

            TCAPStack tcapStack = new TCAPStackImpl(sccpPprovider, localAddress);


            ...

            private SccpAddress createLocalAddress()
            {

             return new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, 1, null, 8);
            }
```

The reference to `SccpProvider` is received from `SccpStack`. To get handle to `SccpStack` do the JNDI look-up passing the JNDI name configured in SS7 service as explained in Section 7.3, “Access Point”

The TCAP User Part should register the concrete implementation of `TCListener` with `TCAPProvider` to listen for incoming TCAP messages.

```
    public class Client implements TCListener{
            .....
            tcapProvider = tcapStack.getProvider();
            tcapProvider.addTCListener(this);
            ....
    }
```

The TCAP User Part leverages TCAPProvider to create new Dialog. The component's between the nodes are exchanged within this Dialog

```
        clientDialog = this.tcapProvider.getNewDialog(thisAddress, remoteAddress);
```

The TCAP User Part leverages ComponentPrimitiveFactory to create new components. These components are sent usig the dialog

```
        //create some INVOKE
        Invoke invoke = cpFactory.createTCInvokeRequest();
        invoke.setInvokeId(this.clientDialog.getNewInvokeId());
        OperationCode oc = cpFactory.createOperationCode();
        oc.setLocalOperationCode(12L);
        invoke.setOperationCode(oc);
        //no parameter
        this.clientDialog.sendComponent(invoke);
```

8.2. Mobicents jSS7 Stack TCAP User Part Example

Below is TCAP User Part example. This example creates dialog and exchanges messages withing structured dialog. Refer to source for function calls:

```
public class Client implements TCListener{
    //encoded Application Context Name
    public static final long[] _ACN_ = new long[] { 0, 4, 0, 0, 1, 0, 19, 2 };

    private TCAPStack stack;
    private SccpAddress thisAddress;
    private SccpAddress remoteAddress;

    private TCAPProvider tcapProvider;
    private Dialog clientDialog;

    Client(SccpProvider sccpPprovider, SccpAddress thisAddress,SccpAddress remoteAddress) {
        super();
          this.stack = new TCAPStackImpl(sccpPprovider,thisAddress); //pass address, so stack
 can register in SCCP
        this.runningTestCase = runningTestCase;
        this.thisAddress = thisAddress;
        this.remoteAddress = remoteAddress;
        this.tcapProvider = this.stack.getProvider();
        this.tcapProvider.addTCListener(this);
    }

    private static SccpProvider getSccpProvider() throws NamingException {

        // no arg is ok, if we run in JBoss
        InitialContext ctx = new InitialContext();
        try {
            String providerJndiName = "/mobicents/ss7/sccp";
```

```
            return ((SccpStack) ctx.lookup(providerJndiName)).getSccpProvider();
        } finally {
            ctx.close();
        }
    }

    public void start() throws TCAPException, TCAPSendException {
        clientDialog = this.tcapProvider.getNewDialog(thisAddress, remoteAddress);
        ComponentPrimitiveFactory cpFactory = this.tcapProvider.getComponentPrimitiveFactory();

        //create some INVOKE
        Invoke invoke = cpFactory.createTCInvokeRequest();
        invoke.setInvokeId(this.clientDialog.getNewInvokeId());

        OperationCode oc = cpFactory.createOperationCode();
        oc.setLocalOperationCode(12L);
        invoke.setOperationCode(oc);
        //no parameter
        this.clientDialog.sendComponent(invoke);

        ApplicationContextName acn = this.tcapProvider.getDialogPrimitiveFactory()
            .createApplicationContextName(_ACN_);
        //UI is optional!
        TCBeginRequest tcbr = this.tcapProvider.getDialogPrimitiveFactory().createBegin(this.clientDialog);
        tcbr.setApplicationContextName(acn);
        this.clientDialog.send(tcbr);
    }

    public void onDialogReleased(Dialog d)
    {
        d.keepAlive();
    }


    public void onInvokeTimeout(Invoke tcInvokeRequest)
    {
    }


    public void onDialogTimeout(Dialog d)
    {
    }

    public void onTCBegin(TCBeginIndication ind) {


    }

    public void onTCContinue(TCContinueIndication ind) {

        //send end
        TCEndRequest end = this.tcapProvider.getDialogPrimitiveFactory().createEnd(ind.getDialog());
        end.setTermination(TerminationType.Basic);
        try {
            ind.getDialog().send(end);

        } catch (TCAPSendException e) {
            throw new RuntimeException(e);
        }
```

```
    }

    public void onTCEnd(TCEndIndication ind) {
        //should not happen, in this scenario, we send data.
    }

    public void onTCUni(TCUniIndication ind) {
        //not going to happen

    }


    public void onTCPAbort(TCPAbortIndication ind) {
        // TODO Auto-generated method stub

    }

    public void onTCUserAbort(TCUserAbortIndication ind) {
        // TODO Auto-generated method stub

    }


    public static void main(String[] args)
    {

        SccpAddress localAddress = new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, 1, null, 8);


        SccpAddress remoteAddress = new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, 2, null, 8);
        Client c = new Client(getSccpProvider(),localAddress,remoteAddress);
    }

}
```

Chapter 9. MAP

Mobile application part (MAP) is the protocol that is used to allow the GSM network nodes within the Network Switching Subsystem (NSS) to communicate with each other to provide services, such as roaming capability, text messaging (SMS), Unstructured Supplementary Service Data (USSD) and subscriber authentication. MAP provides an application layer on which to build the services that support a GSM network. This application layer provides a standardized set of services. MAP uses the services of the SS7 network, specifically the Signaling Connection Control Part (SCCP) and the Transaction Capabilities Application Part (TCAP)

Important

For better understanding of this chapter please read GSM 09.02.

Note

Mobicents jSS7 Stack MAP has implementation for USSD, SMS and Location Management Service (LMS) Messages only. Any contribution to implement other messages are welcome. We will provide you all the help that you may need initially.

9.1. jSS7 Stack MAP

The `org.mobicents.protocols.ss7.map.api.MAPStack` interface defines the methods required to represent MAP Protocol Stack. MAPStack exposes `org.mobicents.protocols.ss7.map.api.MAPProvider` that interacts directly with MAPStack. This interface defines the methods that will be used by any registered MAP User application. MAP user application must implement interfaces to listen MAP messages and dialogue and component handling primitives. One interface is `org.mobicents.protocols.ss7.map.api.MAPDialogListener` (listening associated with dialog events). Others are one or more interfaces for listening associated with component events based on `org.mobicents.protocols.ss7.map.api.MAPServiceListener interface.` Each MAP-User interested in listening messages specific to MAP Service implements specific MAPServiceListener. Below there is a list of implemented services:

Each MAP-User interested in listening messages specific to MAP Service implements specific `MAPServiceListener`.

- MAP-User interested only in USSD messages implements `org.mobicents.protocols.ss7.map.api.service.supplementary.MAPServiceSupplementaryListener`
- MAP-User interested only in SMS messages implements `org.mobicents.protocols.ss7.map.api.service.sms.MAPServiceSmsListener`

- MAP-User interested only in LMS messages implements `org.mobicents.protocols.ss7.map.api.service.lsm.MAPServiceLsmListener`

MAP-User interested in all the services must implement all the service listener class.

The `org.mobicents.protocols.ss7.map.MAPStackImpl` is concrete implementation of `MAPStack`. The MAP User application creates an instance of `MAPStackImpl` passing the reference of `SccpProvider` and Sub System Number and then start it. All incoming messages are checked for destination SSN, if it matches with the one registered with this MAPStackImpl the corresponding listener is called else the peer receives error.

```
SccpProvider sccpProvider = getSccpProvider(); //JNDI lookup of SCCP Stack and get Provider
        MAPStackImpl mapStack = new MAPStackImpl(sccpProvider, 8);
        mapStack.start();
        ...
```

The reference to `SccpProvider` is received from SccpStack. To get handle to `SccpStack` do the JNDI look-up passing the JNDI name configured in SS7 service as explained in Section 7.3, “Access Point”.

The MAP User application should register the concrete implementation of `MAPDialogListener` with `MAPProvider` to listen for incoming MAP Dialog and MAP Primitive messages. The MAP User application should register the concrete implementation of `MAPServiceListener` with corresponding `MAPServiceBase` to listen for incoming MAP Service messages. Following MAPServiceBase are exposed by `MAPProvider`.

- For LSM service `org.mobicents.protocols.ss7.map.api.service.lsm.MAPServiceLsm`
- For SMS service `org.mobicents.protocols.ss7.map.api.service.sms.MAPServiceSms`
- For USSD service `org.mobicents.protocols.ss7.map.api.service.supplementary.MAPServiceSupplementary`

```
public class MAPExample implements MAPDialogListener, MAPServiceSupplementaryListener {
        ....
        mapProvider = mapStack.getMAPProvider();
        mapProvider.addMAPDialogListener(this);
        mapProvider.getMAPServiceSupplementary().addMAPServiceListener(this);
        ....
}
```

Before any MAP specific service can be used, the corresponding service should be activated.

```
        ....
        // Make the supplementary service activated
        mapProvider.getMAPServiceSupplementary().acivate();
        ....
```

The MAP User Application leverages `MAPParameterFactory` to create instances of `USSDString`, `ISDNAddressString` and many other primitives that are used by MAP services.

```
        MapParameterFactory paramFact = mapProvider.getMapServiceFactory();
        USSDString ussdString = paramFact.createUSSDString("*125*+31628839999#",
                null);
        ISDNAddressString msisdn = this. paramFact.createISDNAddressString(
                AddressNature.international_number, NumberingPlan.ISDN,
                "31628838002");
```

9.2. jSS7 Stack Sending a MAP request (processUnstructuredSS-Request as an example)

For sending a MAP request we need at the client side:

- Create a new MAP Dialog

```
// First create Dialog
SccpAddress origAddress = createLocalAddress();
ISDNAddressString origReference = client.getMAPProvider().getMAPParameterFactory().
    createISDNAddressString(AddressNature.international_number, NumberingPlan.land_mobile, "31628968300");
SccpAddress destAddress = createRemoteAddress();
ISDNAddressString destReference = client.getMAPProvider().getMAPParameterFactory().
    createISDNAddressString(AddressNature.international_number, NumberingPlan.land_mobile, "204208300008002");

currentMapDialog = mapProvider.getMAPServiceSupplementary().
    createNewDialog(MAPApplicationContext.getInstance(MAPApplicationContextName.networkUnstructuredSsContext,
    MAPApplicationContextVersion.version2), origAddress,
    destReference, remoteAddress, destReference);
```

- Add an Invoke component (processUnstructuredSS-Request message)

```
// The dataCodingScheme is still byte, as I am not exactly getting how
// to encode/decode this.
byte ussdDataCodingScheme = 0x0f;
// The Charset is null, here we let system use default Charset (UTF-7 as
// explained in GSM 03.38. However if MAP User wants, it can set its own
// impl of Charset
USSDString ussdString = paramFact.createUSSDString(ussdMessage, null);
ISDNAddressString msisdn = client.getMAPProvider().getMAPParameterFactory().
```

```
createISDNAddressString(AddressNature.international_number, NumberingPlan.ISDN, "31628838002");
currentMapDialog.addProcessUnstructuredSSRequest(ussdDataCodingScheme, ussdString, alertingPattern, msisdn);
```

- Send a TC-Begin message to the server peer

```
currentMapDialog.send();
```

- Wait for a response from the server

At the server side when receiving TC-Begin message the following sequence of events occurs:

```
void MAPDialogListener. onDialogRequest(MAPDialog mapDialog, AddressString destReference,
    AddressString origReference, MAPExtensionContainer extensionContainer);
```

This is the request for MAP Dialog processing. A MAP user can reject the Dialog by invoking mapDialog.refuse() method.

Then the incoming primitives corresponded events (one or more) occur. In this case it is

```
void MAPServiceSupplementaryListener.onProcessUnstructuredSSRequest(ProcessUnstructuredSSRequest procUnstrReqInd)
```

When processing component-depended messages you can add response components. In this case it is processUnstructuredSS-Response as an example:

```
        USSDString ussdString = ind.getUSSDString();
        String request = ussdString.getString();

        // processing USSD request
        String response = "Your balans is 100$";

        // The dataCodingScheme is still byte, as I am not exactly getting how
        // to encode/decode this.
        byte ussdDataCodingScheme = 0x0f;
        USSDString ussdResponse = paramFact.createUSSDString(response, null);

        try {
            mapDialog.addProcessUnstructuredSSResponse(ind.getInvokeId(), ussdDataCodingScheme, ussdResponse);
        } catch (MAPException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

If the preparing the response spends much time we should return the control and prepare the answer in your thread.

If error or reject primitives are included into a TCAP message following events occur:

```
public void onErrorComponent(MAPDialog mapDialog, Long invokeId, MAPErrorMessage mapErrorMessage);
public void onProviderErrorComponent(MAPDialog mapDialog, Long invokeId, MAPProviderError providerError);
public void onRejectComponent(MAPDialog mapDialog, Long invokeId, Problem problem);
```

After all incoming components have been processing onDialogDelimiter(MAPDialog mapDialog); event is invoking. If all response components have been prepared we can tell the stack to send response:

- `currentMapDialog.mapDialog.close(false);` - to send TC-END
- `currentMapDialog .mapDialog.send();` - to send TC-CONTINUE
- `currentMapDialog.mapDialog.close(true);` - sends TC-END without any components (prearrangedEnd)

If all response components have been prepared we should return the control and will send a response when all components are ready.

In the error case you can terminate a MAP dialog in any method by invoking

- `currentMapDialog.abort(mapUserAbortChoice);` - sends TC-U-ABORT primitive

If no local actions or no response from a remote size for a long time timeouts occur and the following methods are invoked:

- `MAPDialogListener.onDialogTimeout(MAPDialog mapDialog);`
- `MAPServiceListener.onInvokeTimeout(MAPDialog mapDialog, Long invokeId);`

In `onDialogTimeout()` method you can invoke `mapDialog.keepAlive();` to prevent closing a Dialog. For preventing an Invoke timeout you should invoke `resetInvokeTimer(Long invokeId);` (before onInvokeTimeout() occured).

9.3. jSS7 Stack MAP Usage

The complete example looks like

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

import javax.naming.InitialContext;
import javax.naming.NamingException;
import org.mobicents.protocols.ss7.indicator.RoutingIndicator;
import org.mobicents.protocols.ss7.map.api.primitives.AddressNature;
import org.mobicents.protocols.ss7.map.api.primitives.ISDNAddressString;
```

```
import org.mobicents.protocols.ss7.map.api.primitives.NumberingPlan;
import org.mobicents.protocols.ss7.sccp.SccpStack;
import org.mobicents.protocols.ss7.sccp.SccpProvider;
import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;

public class Example {

    private static SccpProvider getSccpProvider() throws NamingException {
        // no arg is ok, if we run in JBoss

        InitialContext ctx = new InitialContext();
        try {
            String providerJndiName = "/mobicents/ss7/sccp";
            return ((SccpStack) ctx.lookup(providerJndiName)).getSccpProvider();
        } finally {
            ctx.close();
        }
    }

    private static SccpAddress createLocalAddress() {
        return new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, 1, null, 8);
    }

    private static SccpAddress createRemoteAddress() {
        return new SccpAddress(RoutingIndicator.ROUTING_BASED_ON_DPC_AND_SSN, 2, null, 8);
    }


    public static void startUssdClient() throws Exception {
        SccpProvider sccpProvider = getSccpProvider(); // JNDI lookup of SCCP
        UssdClientExample client = new UssdClientExample(sccpProvider, 8);

        client.start();

        SccpAddress origAddress = createLocalAddress();
        ISDNAddressString origReference = client.getMAPProvider().getMAPParameterFactory()
                .createISDNAddressString(AddressNature.international_number,
                NumberingPlan.land_mobile, "31628968300");
        SccpAddress destAddress = createRemoteAddress();
        ISDNAddressString destReference = client.getMAPProvider().getMAPParameterFactory()
                .createISDNAddressString(AddressNature.international_number,
                NumberingPlan.land_mobile, "204208300008002");

        ISDNAddressString msisdn = client.getMAPProvider().getMAPParameterFactory()
                .createISDNAddressString(AddressNature.international_number,
                NumberingPlan.ISDN, "31628838002");
        client.sendProcessUssdRequest(origAddress, origReference, destAddress, destReference,
                "*123#", null, msisdn);

        // wait for answer
        Thread.sleep(600000);

        client.stop();
    }

    public static void startUssdServer() throws Exception {
        SccpProvider sccpProvider = getSccpProvider(); // JNDI lookup of SCCP
        UssdServerExample server = new UssdServerExample(sccpProvider, 8);
```

```
        server.start();

        // wait for a request
        Thread.sleep(600000);

        server.stop();
    }
}
```

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

import org.mobicents.protocols.ss7.map.api.MAPApplicationContext;
import org.mobicents.protocols.ss7.map.api.MAPApplicationContextName;
import org.mobicents.protocols.ss7.map.api.MAPApplicationContextVersion;
import org.mobicents.protocols.ss7.map.api.MAPDialog;
import org.mobicents.protocols.ss7.map.api.MAPDialogListener;
import org.mobicents.protocols.ss7.map.api.MAPException;
import org.mobicents.protocols.ss7.map.api.MAPMessage;
import org.mobicents.protocols.ss7.map.api.MAPParameterFactory;
import org.mobicents.protocols.ss7.map.api.MAPProvider;
import org.mobicents.protocols.ss7.map.api.MAPStack;
import org.mobicents.protocols.ss7.map.api.dialog.MAPAbortProviderReason;
import org.mobicents.protocols.ss7.map.api.dialog.MAPAbortSource;
import org.mobicents.protocols.ss7.map.api.dialog.MAPNoticeProblemDiagnostic;
import org.mobicents.protocols.ss7.map.api.dialog.MAPProviderError;
import org.mobicents.protocols.ss7.map.api.dialog.MAPRefuseReason;
import org.mobicents.protocols.ss7.map.api.dialog.MAPUserAbortChoice;
import org.mobicents.protocols.ss7.map.api.errors.MAPErrorMessage;
import org.mobicents.protocols.ss7.map.api.primitives.AddressString;
import org.mobicents.protocols.ss7.map.api.primitives.AlertingPattern;
import org.mobicents.protocols.ss7.map.api.primitives.IMSI;
import org.mobicents.protocols.ss7.map.api.primitives.ISDNAddressString;
import org.mobicents.protocols.ss7.map.api.primitives.MAPExtensionContainer;
import org.mobicents.protocols.ss7.map.api.primitives.USSDString;
import org.mobicents.protocols.ss7.map.api.service.supplementary.MAPDialogSupplementary;
import org.mobicents.protocols.ss7.map.api.service.supplementary.MAPServiceSupplementaryListener;
import org.mobicents.protocols.ss7.map.api.service.supplementary.ProcessUnstructuredSSRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.ProcessUnstructuredSSResponse;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSNotifyRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSNotifyResponse;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSResponse;
import org.mobicents.protocols.ss7.sccp.SccpProvider;
import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.comp.Problem;

public class UssdClientExample implements MAPDialogListener, MAPServiceSupplementaryListener {

    private MAPStack mapStack;
    private MAPProvider mapProvider;
    private MAPParameterFactory paramFact;
    private MAPDialogSupplementary currentMapDialog;
```

```
    public UssdClientExample(SccpProvider sccpPprovider, int ssn) {
        mapStack = new MAPStackImpl(sccpPprovider, ssn);
        mapProvider = mapStack.getMAPProvider();
        paramFact = mapProvider.getMAPParameterFactory();

        mapProvider.addMAPDialogListener(this);
        mapProvider.getMAPServiceSupplementary().addMAPServiceListener(this);
    }

    public MAPProvider getMAPProvider() {
        return mapProvider;
    }

    public void start() {
        mapStack.start();

        // Make the supplementary service activated
        mapProvider.getMAPServiceSupplementary().acivate();

        currentMapDialog = null;
    }

    public void stop() {
        mapStack.stop();
    }

    public void sendProcessUssdRequest(SccpAddress origAddress, AddressString origReference,
            SccpAddress remoteAddress, AddressString destReference,
            String ussdMessage, AlertingPattern alertingPattern, ISDNAddressString msisdn)
            throws MAPException {
        // First create Dialog
        currentMapDialog = mapProvider.getMAPServiceSupplementary().createNewDialog(
                MAPApplicationContext.getInstance(MAPApplicationContextName.networkUnstructuredSsContext,
                MAPApplicationContextVersion.version2), origAddress,
                destReference, remoteAddress, destReference);

        // The dataCodingScheme is still byte, as I am not exactly getting how
        // to encode/decode this.
        byte ussdDataCodingScheme = 0x0f;
        // The Charset is null, here we let system use default Charset (UTF-7 as
        // explained in GSM 03.38. However if MAP User wants, it can set its own
        // impl of Charset
        USSDString ussdString = paramFact.createUSSDString(ussdMessage, null);

        currentMapDialog.addProcessUnstructuredSSRequest(ussdDataCodingScheme, ussdString,
            alertingPattern, msisdn);
        // This will initiate the TC-BEGIN with INVOKE component
        currentMapDialog.send();
    }

    @Override
    public void onProcessUnstructuredSSResponse(ProcessUnstructuredSSResponse ind) {
        if (currentMapDialog == ind.getMAPDialog()) {
            USSDString ussdString = ind.getUSSDString();
            String response = ussdString.getString();
            // processing USSD response
        }
    }
```

```
    @Override
    public void onErrorComponent(MAPDialog mapDialog, Long invokeId, MAPErrorMessage mapErrorMessage) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onProviderErrorComponent(MAPDialog mapDialog, Long invokeId, MAPProviderError providerError) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onRejectComponent(MAPDialog mapDialog, Long invokeId, Problem problem) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onInvokeTimeout(MAPDialog mapDialog, Long invokeId) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onMAPMessage(MAPMessage mapMessage) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onProcessUnstructuredSSRequest(ProcessUnstructuredSSRequest procUnstrReqInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSRequest(UnstructuredSSRequest unstrReqInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSResponse(UnstructuredSSResponse unstrResInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSNotifyRequest(UnstructuredSSNotifyRequest unstrNotifyInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSNotifyResponse(UnstructuredSSNotifyResponse unstrNotifyInd) {
        // TODO Auto-generated method stub
```

```
    }

    @Override
    public void onDialogDelimiter(MAPDialog mapDialog) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRequest(MAPDialog mapDialog, AddressString destReference,
            AddressString origReference, MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRequestEricsson(MAPDialog mapDialog, AddressString destReference,
            AddressString origReference, IMSI eriImsi, AddressString eriVlrNo) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogAccept(MAPDialog mapDialog, MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogReject(MAPDialog mapDialog, MAPRefuseReason refuseReason,
         MAPProviderError providerError, ApplicationContextName alternativeApplicationContext,
            MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogUserAbort(MAPDialog mapDialog, MAPUserAbortChoice userReason,
        MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogProviderAbort(MAPDialog mapDialog, MAPAbortProviderReason abortProviderReason,
        MAPAbortSource abortSource,
            MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogClose(MAPDialog mapDialog) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogNotice(MAPDialog mapDialog, MAPNoticeProblemDiagnostic noticeProblemDiagnostic) {
```

```
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRelease(MAPDialog mapDialog) {
        currentMapDialog = null;
    }

    @Override
    public void onDialogTimeout(MAPDialog mapDialog) {
        // TODO Auto-generated method stub

    }
}
```

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

import org.mobicents.protocols.ss7.map.api.MAPDialog;
import org.mobicents.protocols.ss7.map.api.MAPDialogListener;
import org.mobicents.protocols.ss7.map.api.MAPException;
import org.mobicents.protocols.ss7.map.api.MAPMessage;
import org.mobicents.protocols.ss7.map.api.MAPParameterFactory;
import org.mobicents.protocols.ss7.map.api.MAPProvider;
import org.mobicents.protocols.ss7.map.api.MAPStack;
import org.mobicents.protocols.ss7.map.api.dialog.MAPAbortProviderReason;
import org.mobicents.protocols.ss7.map.api.dialog.MAPAbortSource;
import org.mobicents.protocols.ss7.map.api.dialog.MAPNoticeProblemDiagnostic;
import org.mobicents.protocols.ss7.map.api.dialog.MAPProviderError;
import org.mobicents.protocols.ss7.map.api.dialog.MAPRefuseReason;
import org.mobicents.protocols.ss7.map.api.dialog.MAPUserAbortChoice;
import org.mobicents.protocols.ss7.map.api.errors.MAPErrorMessage;
import org.mobicents.protocols.ss7.map.api.primitives.AddressString;
import org.mobicents.protocols.ss7.map.api.primitives.IMSI;
import org.mobicents.protocols.ss7.map.api.primitives.MAPExtensionContainer;
import org.mobicents.protocols.ss7.map.api.primitives.USSDString;
import org.mobicents.protocols.ss7.map.api.service.supplementary.MAPDialogSupplementary;
import org.mobicents.protocols.ss7.map.api.service.supplementary.MAPServiceSupplementaryListener;
import org.mobicents.protocols.ss7.map.api.service.supplementary.ProcessUnstructuredSSRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.ProcessUnstructuredSSResponse;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSNotifyRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSNotifyResponse;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSRequest;
import org.mobicents.protocols.ss7.map.api.service.supplementary.UnstructuredSSResponse;
import org.mobicents.protocols.ss7.sccp.SccpProvider;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.comp.Problem;

public class UssdServerExample implements MAPDialogListener, MAPServiceSupplementaryListener {

    private MAPStack mapStack;
    private MAPProvider mapProvider;
    private MAPParameterFactory paramFact;
```

```
    private MAPDialogSupplementary currentMapDialog;

    public UssdServerExample(SccpProvider sccpPprovider, int ssn) {
        mapStack = new MAPStackImpl(sccpPprovider, ssn);
        mapProvider = mapStack.getMAPProvider();
        paramFact = mapProvider.getMAPParameterFactory();

        mapProvider.addMAPDialogListener(this);
        mapProvider.getMAPServiceSupplementary().addMAPServiceListener(this);
    }

    public MAPProvider getMAPProvider() {
        return mapProvider;
    }

    public void start() {
        mapStack.start();

        // Make the supplementary service activated
        mapProvider.getMAPServiceSupplementary().acivate();

        currentMapDialog = null;
    }

    public void stop() {
        mapStack.stop();
    }

    @Override
    public void onProcessUnstructuredSSRequest(ProcessUnstructuredSSRequest ind) {

        USSDString ussdString = ind.getUSSDString();
        String request = ussdString.getString();

        // processing USSD request
        String response = "Your balans is 100$";

        // The dataCodingScheme is still byte, as I am not exactly getting how
        // to encode/decode this.
        byte ussdDataCodingScheme = 0x0f;
        USSDString ussdResponse = paramFact.createUSSDString(response, null);

        try {
            currentMapDialog.addProcessUnstructuredSSResponse(ind.getInvokeId(),
                ussdDataCodingScheme, ussdResponse);
        } catch (MAPException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public void onDialogDelimiter(MAPDialog mapDialog) {
        // This will initiate the TC-END with ReturnResultLast component
        try {
            currentMapDialog.send();
        } catch (MAPException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        }
    }


    @Override
    public void onErrorComponent(MAPDialog mapDialog, Long invokeId,
            MAPErrorMessage mapErrorMessage) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onProviderErrorComponent(MAPDialog mapDialog, Long invokeId,
            MAPProviderError providerError) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onRejectComponent(MAPDialog mapDialog, Long invokeId, Problem problem) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onInvokeTimeout(MAPDialog mapDialog, Long invokeId) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onMAPMessage(MAPMessage mapMessage) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onProcessUnstructuredSSResponse(ProcessUnstructuredSSResponse ind) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSRequest(UnstructuredSSRequest unstrReqInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSResponse(UnstructuredSSResponse unstrResInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onUnstructuredSSNotifyRequest(UnstructuredSSNotifyRequest unstrNotifyInd) {
        // TODO Auto-generated method stub

    }
```

```
    @Override
    public void onUnstructuredSSNotifyResponse(UnstructuredSSNotifyResponse unstrNotifyInd) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRequest(MAPDialog mapDialog, AddressString destReference,
            AddressString origReference, MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRequestEricsson(MAPDialog mapDialog, AddressString destReference,
            AddressString origReference, IMSI eriImsi, AddressString eriVlrNo) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogAccept(MAPDialog mapDialog, MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogReject(MAPDialog mapDialog, MAPRefuseReason refuseReason,
            MAPProviderError providerError,
            ApplicationContextName alternativeApplicationContext,
            MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogUserAbort(MAPDialog mapDialog, MAPUserAbortChoice userReason,
            MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogProviderAbort(MAPDialog mapDialog, MAPAbortProviderReason abortProviderReason,
            MAPAbortSource abortSource,
            MAPExtensionContainer extensionContainer) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogClose(MAPDialog mapDialog) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogNotice(MAPDialog mapDialog, MAPNoticeProblemDiagnostic noticeProblemDiagnostic) {
```

```
        // TODO Auto-generated method stub

    }

    @Override
    public void onDialogRelease(MAPDialog mapDialog) {
        currentMapDialog = null;
    }

    @Override
    public void onDialogTimeout(MAPDialog mapDialog) {
        // TODO Auto-generated method stub

    }
}
```

Chapter 10. SS7 Simulator

SS7 Simulator is an application for testing SS7 stack and displaying its functionality. It is also a good example of how to use this stack.

SS7 Simulator contains three layers of SS7 stack components and one testing task layer which presents the concrete testing task.

Layer 1 presents MTP-3 corresponded stack. It can be one of:

- M3UA
- DialogicCard
- DahdiCard (not yet implemented)

Layer 2 presents :

- SCCP
- ISUP (not yet implemented)

Layer 3 presents :

- TCAP+MAP
- TCAP+CAP (not yet implemented)
- TCAP+INAP (not yet implemented)

Testing task layer can present one of the following testing task:

- USSD client test
- USSD server test
- SMCS SMS delivering test (not yet implemented)
- Dialogic MTU interconnection test (not yet implemented)
- Dialogic MTR interconnection test (not yet implemented)

A user can select for testing layers that he needs. Some layers demand correct low layers. For example TCAP+MAP layer demands SCCP as layer 2. Each layer can be separately configured depending on user testing tasks. Configuring of layers covers only major options (does not cover all possible SS7 stack options).

10.1. SS7 Simulator configuring

For Simulator compiling the following library is needed: com.sun.jdmk:jmxtools:jar:1.2.1. In order to download it manually:

- Go to: http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-plat-419418.html#7657-jmx-1.2.1-oth-JPR&usg=AFQjCNGOqUaCB1ULVG7cMULio9u70MKocA
- Select "Java Management Extension (JMX) 1.2.1"
- Press "Accept License Agreement"
- Download jmx-1_2_1-ri.zip file
- Extract lib/jmxtools:jar
- You can just put this file as ".m2/repository/com/sun/jdmk/jmxtools/1.2.1/jmxtools-1.2.1.jar" file into maven repository (if ".m2/repository" is a root of this repository) or register it.

After compiling Simulator binaries can be found in the folder simulator/bootstrap/target/simulator-ss7. Use "run.bat" or "run.sh" command file for running Simulator (from a folder simulator/bootstrap/target/simulator-ss7/bin).

You can run several instances of Simulator from one folder. Each of this instances can have its own configuration options. Configuration options are saved into a xml configuration file. Each running instance has its own name (called "host name") and the name of configuration file depends on this host name. For example if the host name is "a1", the name of the configuration file will be "a1_simulator.xml". Each running Simulator instance must have different host name in order to each running Simulator instance can have different options.

Before running a test all layers must be configured. After test running a user can perform some actions depending on the test. Results of a test are emitted as "notifications". Notifications are displayed in GUI interface of Simulator and Jconsole application (also GUI interface) and also are written into a file "<host name>.log" (for example "a1.log") and into a console.

You can run and manage SS7 Simulator locally or remotely. If you are running Simulator locally you are using a GUI interface. If you are running Simulator remotely you can use RMI access (via a GUI interface) and HTML interface (using a HTML Browser).

10.1.1. Running SS7 Simulator locally

In order to run SS7 Simulator locally you must run a command "run gui" (from a folder simulator/bootstrap/target/simulator-ss7/bin). You can pass the only parameter which is the "host name". Possible examples (host name is "a1"):

```
run gui -na1
run gui --name=a1
```

The GUI application form "Connecting to a testerHost ..." will appear. You have to fill "Host name" field (if it is empty), select "Create a local testerHost" option and press "Start" button. The local Simulator host will be run.

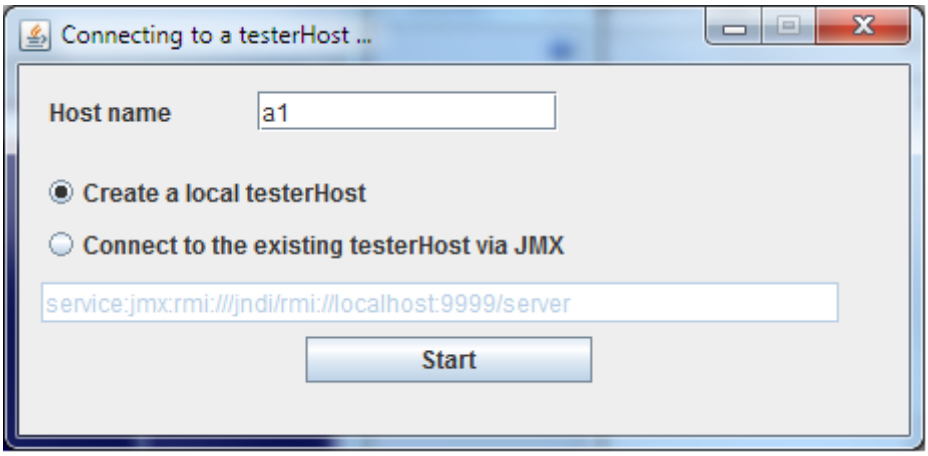


Figure 10.1. Launching Simulator in a local mode

The form "SS7 Simulator: <host name>-local" appears. The next step is selecting modules for layers 1-3 and a testing task and configuring modules (by pressing buttons "..."). Button "Save" is used for saving options to the disk, button "Reload" is used for reloading options from a disk. When layer configuring you can press one of two buttons "Load default values for side A" and "Load default values for side B" for loading default values for testing. These default values can be used if you are using SS7 Simulators in the one computer host as the side A and the side B for interaction.

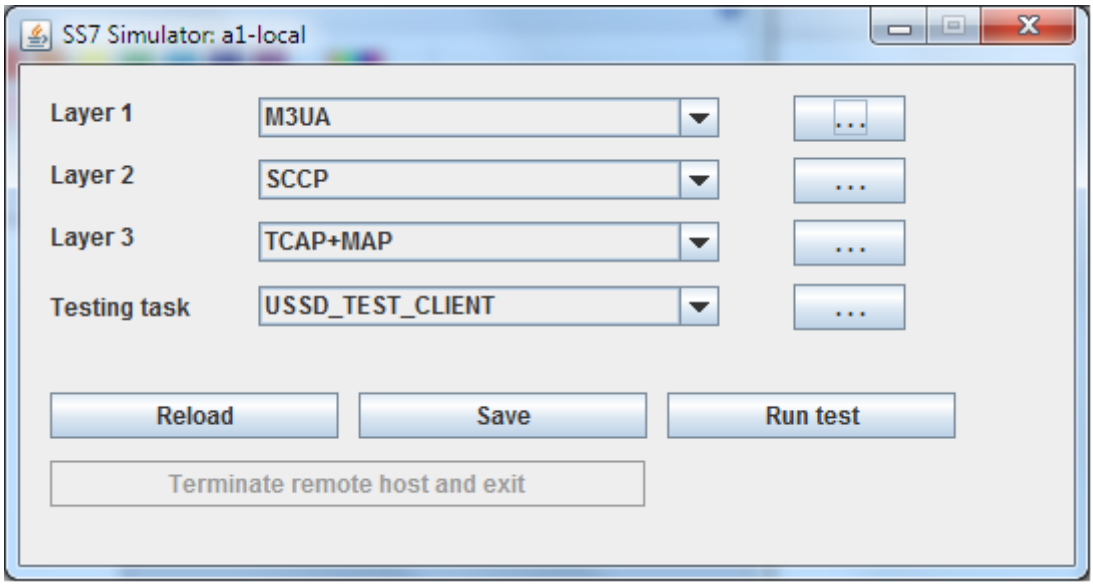


Figure 10.2. The main form of the Simulator

After configuring press button "Run test". The form for testing will be displayed. In the upper part of this form the information of layers and test state is displayed. This information is refreshed every 5 seconds and can be refreshed manually by "Refresh state" button pressing. After pressing "Start" button all modules will start and will be ready for test, after pressing "Stop" button all modules will

stop. In the central part of the form notifications from testing modules are displayed. In the lower part of the form test depended information are allocated. Details of this part is displayed in the corresponded test definition. See Section 10.2, “SS7 Simulator test cases ”

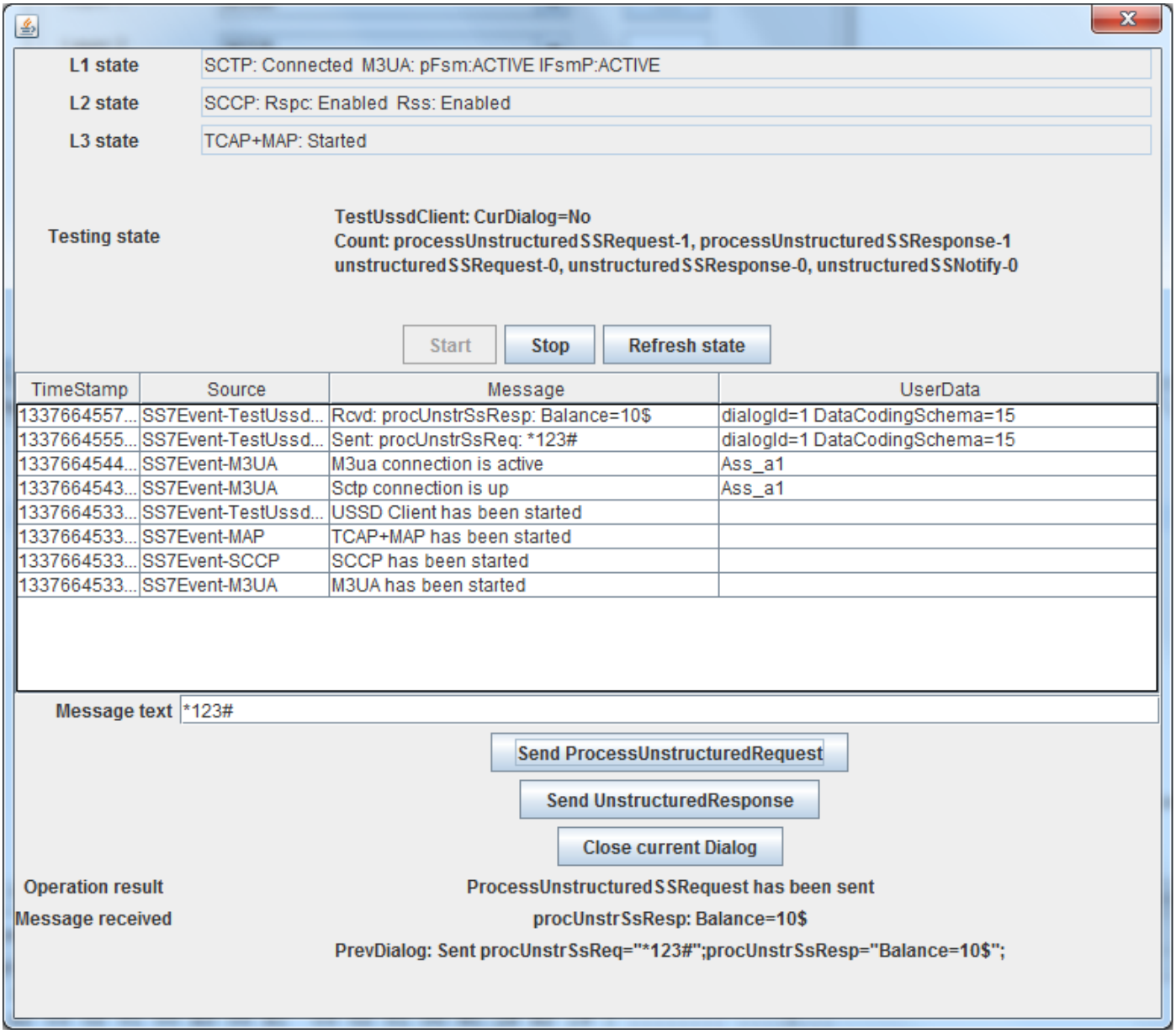


Figure 10.3. The example of the Simulator testing form

10.1.2. Running SS7 Simulator remotely

Now there are two interfaces for managing: RMI and HTML adapters. Both two styles of management can be run at the same time.

For launching Simulator remotely we should firstly run a tester host. The format of launching a tester host is:

```
run core -na1 -t8001 -r9999
```

or

```
run core --name=a1 --http=8001 --rmi=9999
```

Options "-n" (or "--name=") defines a host name ("a1"). Options "-r" (or "--rmi=") defines a port for rmi requests listerning (9999). Options "-t" (or "--http=") defines a port for html requests listerning (8001). You can use only rmi connector or only http connector or both.

10.1.2.1. RMI interface

The best client for management via RMI interface is the GUI interface of Simulator. It can be launched by the same way as we launch Simulator locally:

```
run gui
```

After launching we should select the option "Connect to the existing testerHost via JMX" and in the default RMI connection string replace "localhost" with the proper IP address of the tester host computer and the "9999" with the proper RMI port (the port from option "-r" or "--rmi=") (if needed). After connecting the management will be the same as if the application is run locally (see Section 10.1.1, “Running SS7 Simulator locally ”).

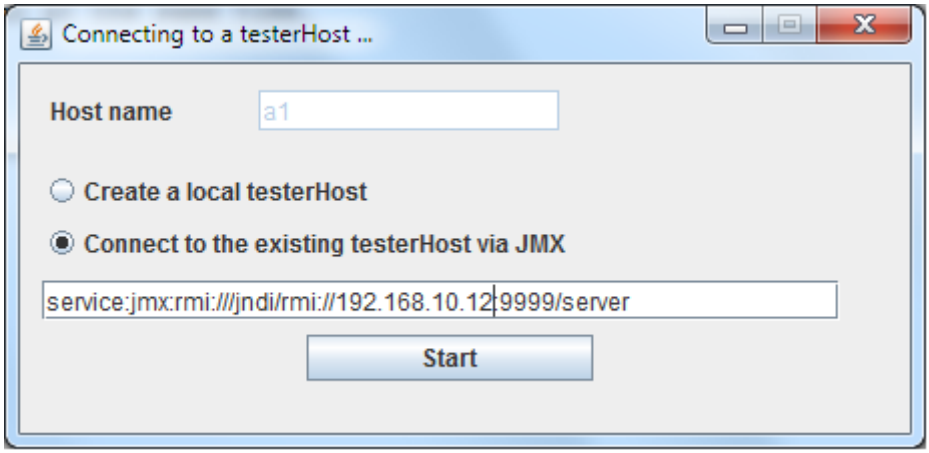


Figure 10.4. Launching Simulator GUI for a remote mode

"Jconsole" application can be also used as the client. But this application is less convenient.

10.1.2.2. HTPL interface

HTML management is less convenient than RMI management. But it can be used if RMI is not acceptable (for example if we are behind a proxy). For paramters configuring and luanching tests we can use any html browser. Use "http://<IP address>:<port>" as an URL. The port is defined in "-t" or "--http=" options.

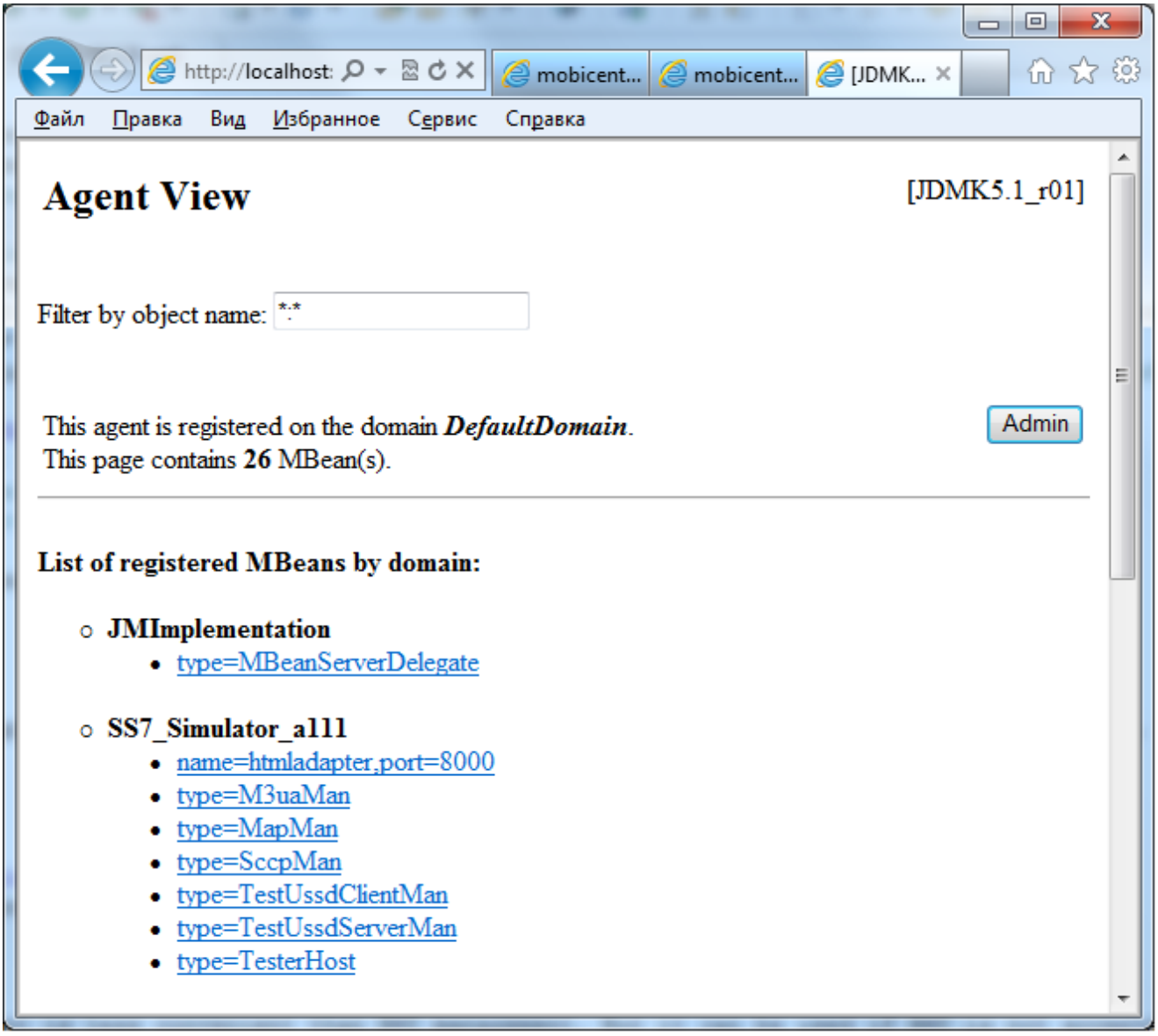


Figure 10.5. Managing of a Simulator by a Browser (HTMP connector)

"TesterHost" is a main MBean in which we can select the mode for test working, start/stop testing (buttons "Start"/"Stop"), quit a tester host (button "quit"). Use other beans for setting options for layers and test tasks.

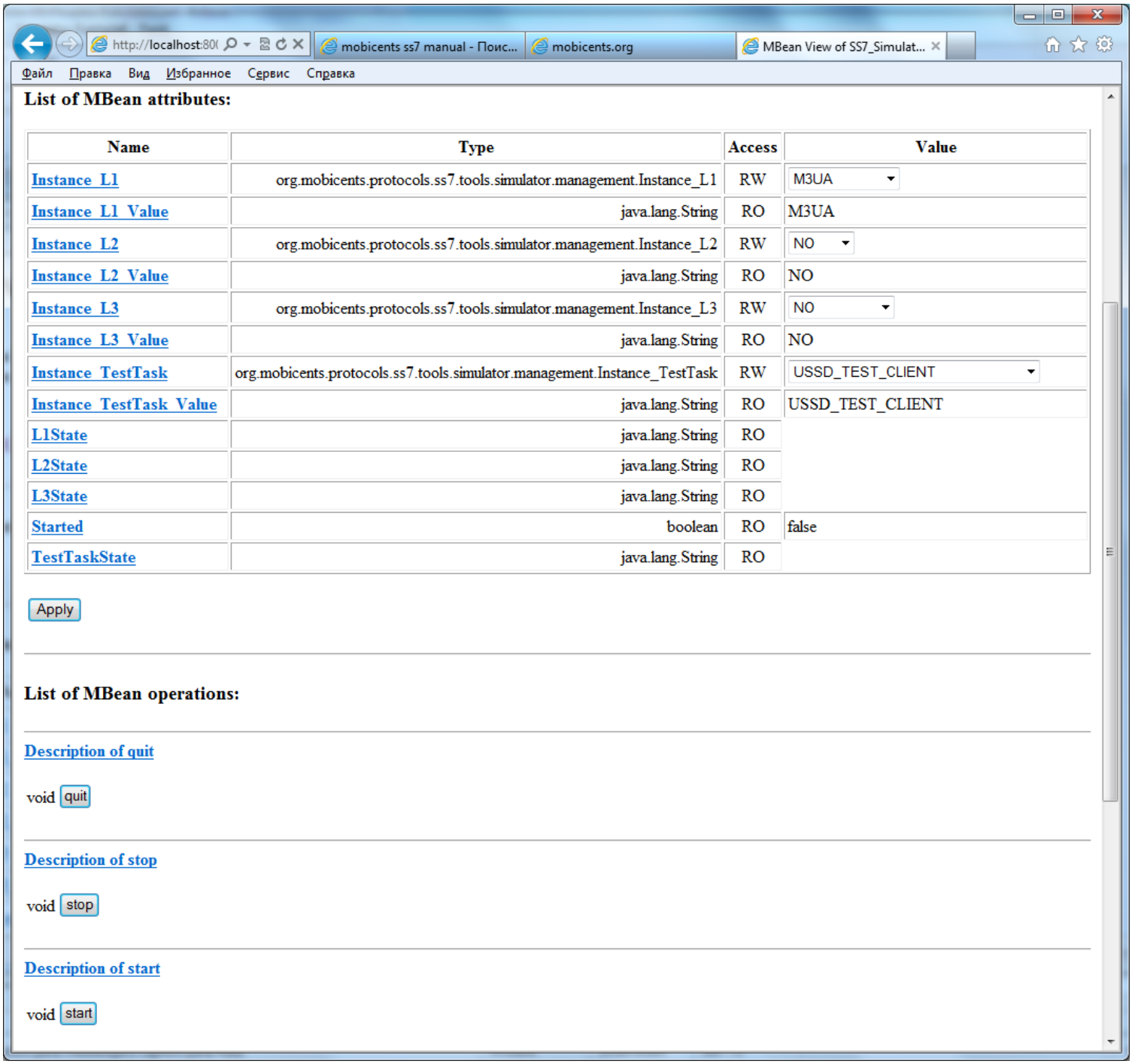


Figure 10.6. TesterHost bean

Results of Simulator working can be found at the console (at the server) or in the log file (file name is like "a1.log").

10.2. SS7 Simulator test cases

Several test cases are implemened now.

10.2.1. USSD Client

UssdClient test can perform following tasks:

- Sending a ProcessUnstructuredSs request, waiting for the answer and displaying the answer
- Sending a UnstructuredSs response as an answer for UnstructuredSs request
- Receiving UnstructuredSs notify and displaying it
- Spectial case for load testing: Sending to the Ussd server a flow of ProcessUnstructuredSs requests without stopping (and receiving responses).

For working in the manual mode select the option "Manual operation", for auto sending messages - the option "Auto sending ProcessUnstructuredSSRequest". You can send ProcessUnstructuredSs request and UnstructuredSs response only manually by inserting a message text and pressing buttons. For working in the auto mode you should define the string of auto processUnstructuredSs request and the count of maximum active MAP dialogs (default value is 10). The more dialogs is defined the more messages per second will be sent. Msisdn, data coding schema and alerting pattern values should be also configured before test starting. For the auto mode we recomment to check the option "One notification for 100 dialogs" for preventing too many notifications when load testing.

You can send ProcessUnstructuredSs request and UnstructuredSs response manually by inserting a message text and pressing buttons. You can not send a new ProcessUnstructiredSs request till the response for previos request has been received (or till dialog timeout). You can also manually close the current dialog by pressing "Close current dialog" button.

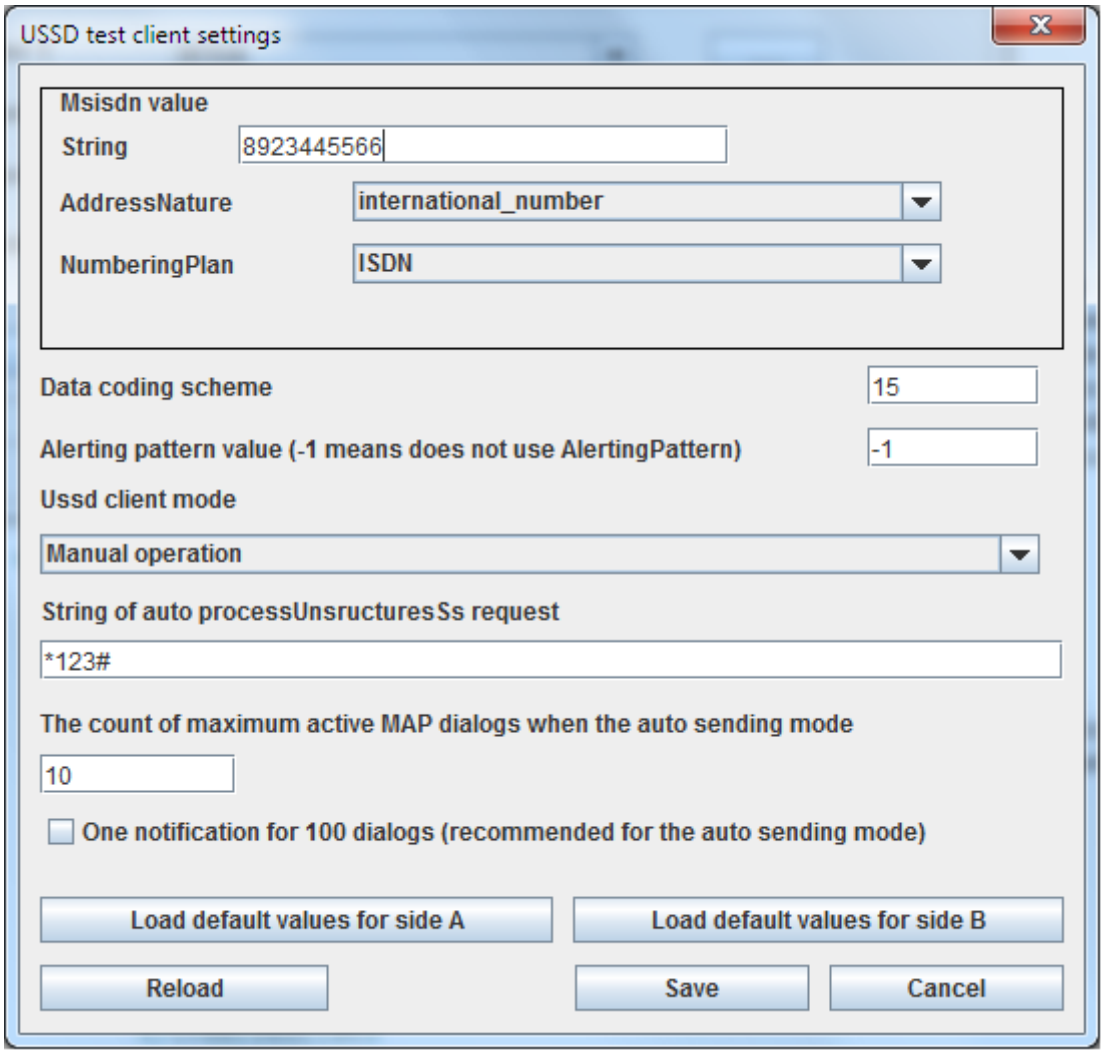


Figure 10.7. Ussd Client option form

10.2.2. USSD Server

UssdServer test can work in manual and automatic modes. In the manual mode a user manually inserts the response message value and sends ProcessUnstructuredSs response (or UnstructuredSs request) to the Ussd client. In the automatic mode UssdServer automatically sends a predefined message as an ProcessUnstructuredSs response ("Auto sending ProcessUnstructuredSsResponse" option) or automatically sends a predefined message as an UnstructuredSs request and then sends a predefined message ProcessUnstructuredSs response ("Auto sending Unstructured_SS_Request then after response sending ProcessUnstructuredSsResponse" option). Use "Auto sending ProcessUnstructuredSsResponse" option for UssdServer if UssdClient is used for a load test ("Auto sending ProcessUnstructuredSSRequest"). For load tests we recommend to check the option "One notification for 100 dialogs" for preventing too many notifications when load testing.

UssdServer test can perform followng tasks:

- Sending a ProcessUnstructiredSs response / UnstructiredSs request in automatic or manual modes. Ussd Client must initiate a dialog using ProcessUnstructiredSs request.
- Manually sending an UnstructuredSsNotify.

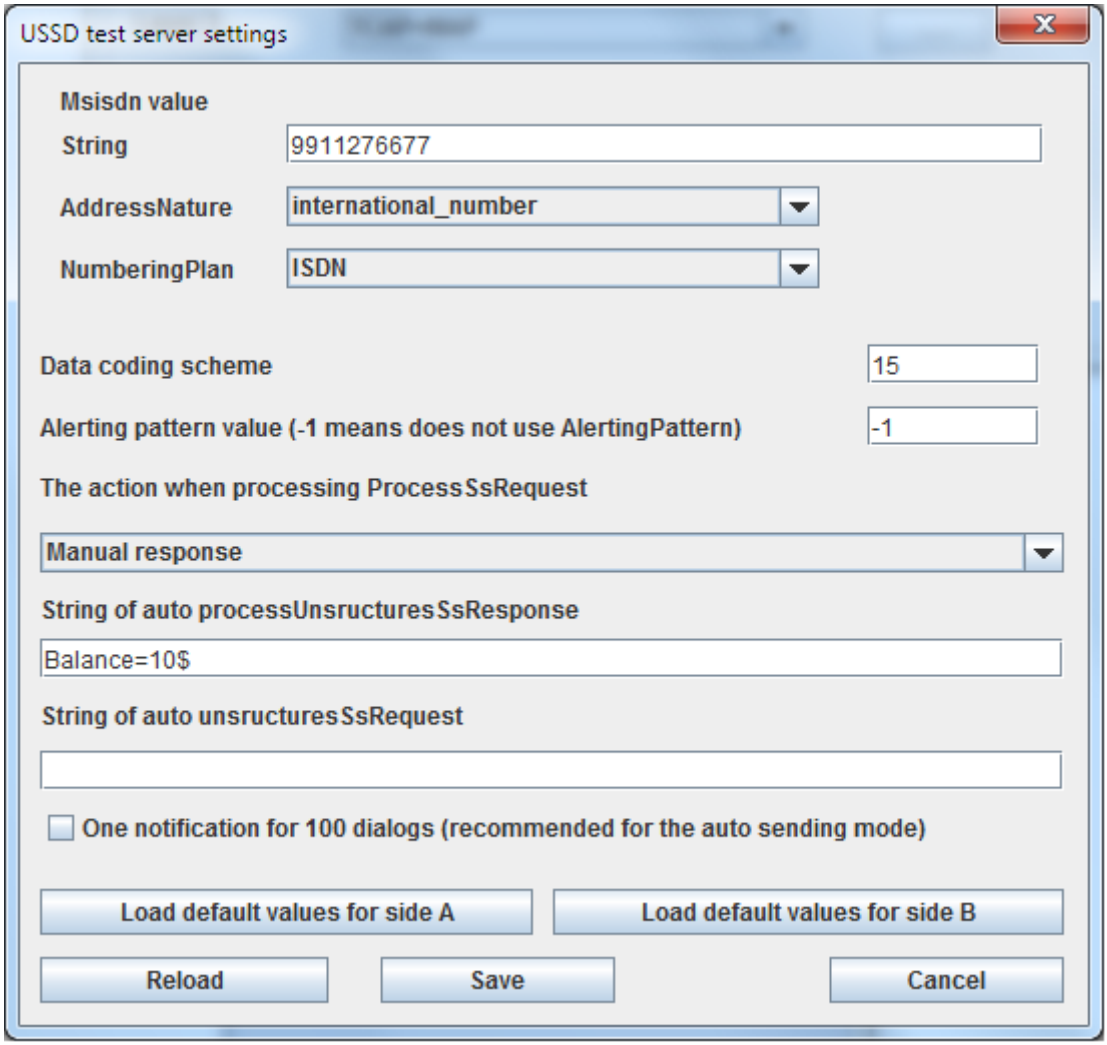


Figure 10.8. Ussd Server option form

Appendix A. Java Development Kit (JDK): Installing, Configuring and Running

The Mobicents Platform is written in Java; therefore, before running any Mobicents server, you must have a working Java Runtime Environment (JRE) or Java Development Kit (JDK) installed on your system. In addition, the JRE or JDK you are using to run Mobicents must be version 5 or higher[1].

Should I Install the JRE or JDK? Although you can run Mobicents servers using the Java Runtime Environment, we assume that most users are developers interested in developing Java-based, Mobicents-driven solutions. Therefore, in this guide we take the tact of showing how to install the full Java Development Kit.

Should I Install the 32-Bit or the 64-Bit JDK, and Does It Matter? Briefly stated: if you are running on a 64-Bit Linux or Windows platform, you should consider installing and running the 64-bit JDK over the 32-bit one. Here are some heuristics for determining whether you would rather run the 64-bit Java Virtual Machine (JVM) over its 32-bit cousin for your application:

- Wider datapath: the pipe between RAM and CPU is doubled, which improves the performance of memory-bound applications when using a 64-bit JVM.
- 64-bit memory addressing gives virtually unlimited (1 exabyte) heap allocation. However large heaps affect garbage collection.
- Applications that run with more than 1.5 GB of RAM (including free space for garbage collection optimization) should utilize the 64-bit JVM.
- Applications that run on a 32-bit JVM and do not require more than minimal heap sizes will gain nothing from a 64-bit JVM. Barring memory issues, 64-bit hardware with the same relative clock speed and architecture is not likely to run Java applications faster than their 32-bit cousin.

Note that the following instructions detail how to download and install the 32-bit JDK, although the steps are nearly identical for installing the 64-bit version.

Downloading. You can download the Sun JDK 5.0 (Java 2 Development Kit) from Sun's website: http://java.sun.com/javase/downloads/index_jdk5.jsp. Click on the Download link next to "JDK 5.0 Update <*x*>" (where <*x*> is the latest minor version release number). On the next page, select your language and platform (both architecture—whether 32- or 64-bit—and operating

[1] At this point in time, it is possible to run most Mobicents servers, such as the JAIN SLEE, using a Java 6 JRE or JDK. Be aware, however, that presently the XML Document Management Server does not run on Java 6. We suggest checking the Mobicents web site, forums or discussion pages if you need to inquire about the status of running the XML Document Management Server with Java 6.

system), read and agree to the `Java Development Kit 5.0 License Agreement`, and proceed to the download page.

The Sun website will present two download alternatives to you: one is an RPM inside a self-extracting file (for example, `jdk-1_5_0_16-linux-i586-rpm.bin`), and the other is merely a self-extracting file (e.g. `jdk-1_5_0_16-linux-i586.bin`). If you are installing the JDK on Red Hat Enterprise Linux, Fedora, or another RPM-based Linux system, we suggest that you download the self-extracting file containing the RPM package, which will set up and use the SysV service scripts in addition to installing the JDK. We also suggest installing the self-extracting RPM file if you will be running Mobicents in a production environment.

Installing. The following procedures detail how to install the Java Development Kit on both Linux and Windows.

Procedure A.1. Installing the JDK on Linux

- Regardless of which file you downloaded, you can install it on Linux by simply making sure the file is executable and then running it:

```
~]$ chmod +x "jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
~]$ ./"jdk-1_5_0_<minor_version>-linux-<architecture>-rpm.bin"
```

You Installed Using the Non-RPM Installer, but Want the SysV Service Scripts

If you download the non-RPM self-extracting file (and installed it), and you are running on an RPM-based system, you can still set up the SysV service scripts by downloading and installing one of the `-compat` packages from the JPackage project. Remember to download the `-compat` package which corresponds correctly to the minor release number of the JDK you installed. The compat packages are available from ftp://jpackage.hmdc.harvard.edu/JPackage/1.7/generic/RPMS.non-free/.

Important

You do not need to install a `-compat` package in addition to the JDK if you installed the self-extracting RPM file! The `-compat` package merely performs the same SysV service script set up that the RPM version of the JDK installer does.

Procedure A.2. Installing the JDK on Windows

- Using Explorer, simply double-click the downloaded self-extracting installer and follow the instructions to install the JDK.

Configuring. Configuring your system for the JDK consists in two tasks: setting the `JAVA_HOME` environment variable, and ensuring that the system is using the proper JDK (or JRE) using the **alternatives** command. Setting `JAVA_HOME` usually overrides the values for **java**, **javac** and **java_sdk_1.5.0** in **alternatives**, but we will set them all just to be safe and consistent.

Setting the `JAVA_HOME` Environment Variable on Generic Linux

After installing the JDK, you must ensure that the `JAVA_HOME` environment variable exists and points to the location of your JDK installation.

Setting the `JAVA_HOME` Environment Variable on Linux. You can determine whether `JAVA_HOME` is set on your system by **echo**ing it on the command line:

```
~]$ echo $JAVA_HOME
```

If `JAVA_HOME` is not set already, then you must set its value to the location of the JDK installation on your system. You can do this by adding two lines to your personal `~/.bashrc` configuration file. Open `~/.bashrc` (or create it if it doesn't exist) and add a line similar to the following one anywhere inside the file:

```
export JAVA_HOME="/usr/lib/jvm/jdk1.5.0_<version>"
```

You should also set this environment variable for any other users who will be running Mobicents (any environment variables **export**ed from `~/.bashrc` files are local to that user).

Setting `java`, `javac` and `java_sdk_1.5.0` Using the **alternatives** command

Selecting the Correct System JVM on Linux using alternatives . On systems with the **alternatives** command, including Red Hat Enterprise Linux and Fedora, you can easily choose which JDK (or JRE) installation you wish to use, as well as which **java** and **javac** executables should be run when called.

As the root user, call **/usr/sbin/alternatives** with the `--config java` option to select between JDKs and JREs installed on your system:

```
root@localhost ~]$ /usr/sbin/alternatives --config java

There are 3 programs which provide 'java'.

  Selection    Command
-----------------------------------------------
   1           /usr/lib/jvm/jre-1.5.0-gcj/bin/java
   2           /usr/lib/jvm/jre-1.6.0-sun/bin/java
*+ 3         /usr/lib/jvm/jre-1.5.0-sun/bin/java


Enter to keep the current selection[+], or type selection number:
```

In our case, we want to use the Sun JDK, version 5, that we downloaded and installed, to run the **java** executable. In the **alternatives** information printout above, a plus (+) next to a number indicates the one currently being used. As per **alternatives**' instructions, pressing **Enter** will simply keep the current JVM, or you can enter the number corresponding to the JVM you would prefer to use.

Repeat the procedure above for the **javac** command and the `java_sdk_1.5.0` environment variable, *as the root user*:

```
~]$ /usr/sbin/alternatives --config javac
```

```
~]$ /usr/sbin/alternatives --config java_sdk_1.5.0
```

Setting the `JAVA_HOME` Environment Variable on Windows

For information on how to set environment variables in Windows, refer to http://support.microsoft.com/kb/931715.

Testing. Finally, to make sure that you are using the correct JDK or Java version (5 or higher), and that the java executable is in your `PATH`, run the **java `-version`** command in the terminal from your home directory:

```
~]$ java -version
java version "1.5.0_16"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_16-b03)
Java HotSpot(TM) Client VM (build 1.5.0_16-b03, mixed mode, sharing)
```

Uninstalling. There is usually no reason (other than space concerns) to remove a particular JDK from your system, given that you can switch between JDKs and JREs easily using **alternatives**, and/or by setting `JAVA_HOME`.

Uninstalling the JDK on Linux. On RPM-based systems, you can uninstall the JDK using the **yum `remove <jdk_rpm_name>`** command.

Uninstalling the JDK on Windows. On Windows systems, check the JDK entry in the `Start` menu for an uninstall command, or use `Add/Remove Programs`.

Appendix B. Setting the JBOSS_HOME Environment Variable

The Mobicents Platform (Mobicents) is built on top of the JBoss Application Server. You do not need to set the `JBOSS_HOME` environment variable to run any of the Mobicents Platform servers *unless* `JBOSS_HOME` is *already* set.

The best way to know for sure whether `JBOSS_HOME` was set previously or not is to perform a simple check which may save you time and frustration.

Checking to See If JBOSS_HOME is Set on Unix. At the command line, **echo `$JBOSS_HOME`** to see if it is currently defined in your environment:

```
~]$ echo $JBOSS_HOME
```

The Mobicents Platform and most Mobicents servers are built on top of the JBoss Application Server (JBoss Application Server). When the Mobicents Platform or Mobicents servers are built *from source*, then `JBOSS_HOME` *must* be set, because the Mobicents files are installed into (or "over top of" if you prefer) a clean JBoss Application Server installation, and the build process assumes that the location pointed to by the `JBOSS_HOME` environment variable at the time of building is the JBoss Application Server installation into which you want it to install the Mobicents files.

This guide does not detail building the Mobicents Platform or any Mobicents servers from source. It is nevertheless useful to understand the role played by JBoss AS and `JBOSS_HOME` in the Mobicents ecosystem.

The immediately-following section considers whether you need to set `JBOSS_HOME` at all and, if so, when. The subsequent sections detail how to set `JBOSS_HOME` on Unix and Windows

Important

Even if you fall into the category below of *not needing* to set `JBOSS_HOME`, you may want to for various reasons anyway. Also, even if you are instructed that you do *not need* to set `JBOSS_HOME`, it is good practice nonetheless to check and make sure that `JBOSS_HOME` actually *isn't* set or defined on your system for some reason. This can save you both time and frustration.

You *DO NOT NEED* to set `JBOSS_HOME` if...

- ...you have installed the Mobicents Platform binary distribution.

- ...you have installed a Mobicents server binary distribution *which bundles JBoss Application Server.*

You *MUST* set `JBOSS_HOME` if...

- ...you are installing the Mobicents Platform or any of the Mobicents servers *from source*.
- ...you are installing the Mobicents Platform binary distribution, or one of the Mobicents server binary distributions, which *do not* bundle JBoss Application Server.

Naturally, if you installed the Mobicents Platform or one of the Mobicents server binary releases which *do not* bundle JBoss Application Server, yet requires it to run, then you should install before setting `JBOSS_HOME` or proceeding with anything else.

Setting the JBOSS_HOME Environment Variable on Unix. The `JBOSS_HOME` environment variable must point to the directory which contains all of the files for the Mobicents Platform or individual Mobicents server that you installed. As another hint, this topmost directory contains a `bin` subdirectory.

Setting `JBOSS_HOME` in your personal `~/.bashrc` startup script carries the advantage of retaining effect over reboots. Each time you log in, the environment variable is sure to be set for you, as a user. On Unix, it is possible to set `JBOSS_HOME` as a system-wide environment variable, by defining it in `/etc/bashrc`, but this method is neither recommended nor detailed in these instructions.

Procedure B.1. To Set JBOSS_HOME on Unix...

1. Open the `~/.bashrc` startup script, which is a hidden file in your home directory, in a text editor, and insert the following line on its own line while substituting for the actual install location on your system:

```
export JBOSS_HOME="/home/<username>/<path>/<to>/<install_directory>"
```

2. Save and close the `.bashrc` startup script.
3. You should **source** the `.bashrc` script to force your change to take effect, so that `JBOSS_HOME` becomes set for the current session[1].

```
~]$ source ~/.bashrc
```

4. Finally, ensure that `JBOSS_HOME` is set in the current session, and actually points to the correct location:

[1] Note that any other terminals which were opened prior to your having altered `.bashrc` will need to **source** `~/.bashrc` as well should they require access to `JBOSS_HOME`.

Note

The command line usage below is based upon a binary installation of the Mobicents Platform. In this sample output, `JBOSS_HOME` has been set correctly to the *`topmost_directory`* of the Mobicents installation. Note that if you are installing one of the standalone Mobicents servers (with JBoss AS bundled!), then `JBOSS_HOME` would point to the *`topmost_directory`* of your server installation.

```
~]$ echo $JBOSS_HOME
/home/silas/<path>/<to>/<install_directory>
```

Setting the JBOSS_HOME Environment Variable on Windows. The `JBOSS_HOME` environment variable must point to the directory which contains all of the files for the Mobicents Platform or individual Mobicents server that you installed. As another hint, this topmost directory contains a `bin` subdirectory.

For information on how to set environment variables in recent versions of Windows, refer to http://support.microsoft.com/kb/931715.

Appendix C. Revision History

Revision History		
Revision 1.0	Wed June 2 2010	BartoszBaranowski
Creation of the Mobicents jSS7 Stack User Guide.		
Revision 1.1	Tue Dec 21 2010	AmitBhayani
Creation of the Mobicents jSS7 Stack User Guide.		

Index

F