Configuring jboss-beans.xml

Table of Contents

Configuring Restcomm SS7 Service

Configuration is done through an XML descriptor file named *jboss-beans.xml* located at *$JBOSS_HOME/server/profile_name/deploy/restcomm-ss7-service/META-INF*, where `profile_name` is the name of the server profile. Default *jboss-beans.xml* contains only m3ua usage. Templates for usage of Dialogic boards or m3ua and Dialogic boards together can be found in the release binaries in folders: *ss7/template/META-INF-dialogic* and *ss7/template/META-INF-m3ua-dialogic*

Restcomm SS7 Layer 4 (`SCCP, ISUP`) leverages either of the following `MTP` layers to exchange signaling messages with remote signaling points:

- `M3UA`
- `dahdi`
- `dialogic`

Configuring M3UA

You must configure `M3UAManagement` if the underlying SS7 service will leverage `M3UA`. For more details on configuring `M3UAManagement`, please refer to [_managing_m3ua].

Scroll down to the section for M3UA Layer in the *jboss-beans.xml* file and define the properties to suit your requirements.

```
        <!-- ===================================================================== -->
        <!-- SCTP Properties                                                       -->
        <!-- Used by M3UA layer                                                    -->
        <!-- ===================================================================== -->
        <bean name="SCTPManagement"
class="org.mobicents.protocols.sctp.netty.NettySctpManagementImpl">
                <constructor>
                        <parameter>SCTPManagement</parameter>
                </constructor>
                <property name="persistDir">${jboss.server.data.dir}</property>
        </bean>

        <bean name="SCTPShellExecutor"
                class="org.restcomm.protocols.ss7.m3ua.impl.oam.SCTPShellExecutor">
            <property name="sctpManagements">
                <map keyClass="java.lang.String"
valueClass="org.mobicents.protocols.sctp.netty.NettySctpManagementImpl">
                    <entry>
                        <key>SCTPManagement</key>
                        <value>
                            <inject bean="SCTPManagement" />
                        </value>
                    </entry>
                </map>
            </property>
        </bean>
```

```
        <!-- ================================================================== -->
        <!-- M3UA -->
        <!-- M3UAManagement is managing the m3ua side commands -->
        <!-- ================================================================== -->
        <!-- -->
        <bean name="Mtp3UserPart"
class="org.restcomm.protocols.ss7.m3ua.impl.M3UAManagementImpl">
            <constructor>
                <parameter>Mtp3UserPart</parameter>
                <parameter>{this-platform}-jSS7</parameter>
            </constructor>
            <property name="persistDir">${jboss.server.data.dir}</property>
            <property name="maxSequenceNumber">256</property>
            <property name="maxAsForRoute">2</property>
            <property name="deliveryMessageThreadCount">1</property>
            <property name="routingLabelFormat">
                <inject bean="RoutingLabelFormat" />
            </property>
            <property name="transportManagement">
                <inject bean="SCTPManagement" />
            </property>
        </bean>

        <bean name="M3UAShellExecutor"
                class="org.restcomm.protocols.ss7.m3ua.impl.oam.M3UAShellExecutor">
            <property name="m3uaManagements">
                <map keyClass="java.lang.String"
valueClass="org.restcomm.protocols.ss7.m3ua.impl.M3UAManagementImpl">
                    <entry>
                        <key>Mtp3UserPart</key>
                        <value>
                            <inject bean="Mtp3UserPart" />
                        </value>
                    </entry>
                </map>
            </property>
        </bean>
```

`org.mobicents.protocols.sctp.netty.NettySctpManagementImpl`

This SCTP Management Bean takes a `String` as a constructor argument. The name is prepended to the name of the XML file created by the SCTP stack for persisting the state of SCTP resources. This XML file is stored in the path specified by the property `persistDir`. For example, in the above case, when Restcomm SS7 Service is started, a file named *SCTPManagement_sctp.xml* will be created at *$JBOSS_HOME/server/profile_name/data* directory. You can disable of using persistent configuration via using `org.mobicents.protocols.sctp.netty.NonPersistentNettySctpManagementImpl` class. The other properties of the Stack are defined below:

`org.restcomm.protocols.ss7.m3ua.impl.M3UAManagementImpl`

This M3UA Management Bean takes a `String` as a first constructor argument. The name is

prepended to the name of the XML file created by the M3UA stack for persisting the state of M3UA resources. The second constructor argument of M3UA Management Bean is also a `String`. This is a productName parameter.

This XML file is stored in the path specified by the property `persistDir`. For example, in the above case, when Restcomm SS7 Service is started, a file named *Mtp3UserPart_m3ua1.xml* will be created at *$JBOSS_HOME/server/profile_name/data* directory. You can disable of using persistent configuration via using `org.restcomm.protocols.ss7.m3ua.impl.NonPersistentM3UAManagementImpl` class. The other properties of the Stack are defined below:

persistDir

As explained above

routingLabelFormat

The routing label format supported by this network. See Configuring MTP3 routing label for further details.

transportManagement

SCTPManagement mbean should be provided here.

other parameters

See [_m3ua_properties]

Configuring dahdi

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

The corresponding native libraries for `dahdi` (`libmobicents-dahdi-linux` library) must be compiled before stack configuring and put into `$JBOSS_HOME/bin/META-INF/lib/linux2/x86` if OS is 32 bit or `$JBOSS_HOME/bin/META-INF/lib/linux2/x64` if OS and JAVA is 64 bit.

See JSS7 Installation Guide for more details.

Libraries are compiled only for linux OS for 32-bit JAVA for now.

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

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

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

```
    <!-- ============================================== -->
    <!-- Linkset manager Service                        -->
    <!-- ============================================== -->
    <bean name="LinksetFactoryFactory"
class="org.restcomm.ss7.linkset.oam.LinksetFactoryFactory">
            <incallback method="addFactory" />
            <uncallback method="removeFactory" />
    </bean>

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

    <bean name="LinksetManager"
class="org.restcomm.ss7.linkset.oam.LinksetManagerImpl">
        <constructor>
            <parameter>LinksetManager</parameter>
        </constructor>
        <property name="scheduler">
            <inject bean="SS7Scheduler" />
        </property>
        <property name="linksetFactoryFactory">
            <inject bean="LinksetFactoryFactory" />
        </property>
        <property name="persistDir">${jboss.server.data.dir}</property>
    </bean>

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

When LinksetManagerImpl is started it looks for the file *linksetmanager.xml* containing serialized information about underlying linksets and links. The directory path is configurable by changing the value of the property `persistDir`.

linksetmanager.xml should never be edited by you manually. Always use the Shell Client to connect to the Stack and execute appropriate commands.

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

Configuring dialogic

`Dialogic` based MTP layer will only be used if you have installed Dialogic cards. `DialogicMtp3UserPart` communicates with Dialogic hardware. It is assumed here that MTP3 and MTP2 is leveraged from the Dialogic Stack either on-board or on-host.

The corresponding native libraries for `dialogic` (native lib `libgctjni.so` and gctApi library `gctApi.jar`) should be downloaded from the Dialogic site and copied : * `libgctjni.so` - to the folder `$JBOSS_HOME/bin/META-INF/lib/linux2/x86` if OS is 32 bit or `$JBOSS_HOME/bin/META-INF/lib/linux2/x64` if OS and JAVA is 64 bit. * `gctApi.jar` - to the folder `jboss-5.1.0.GA/server/default/deploy/Restcomm-ss7-service/lib`

See JSS7 Installation Guide for more details.

```
    <!-- ===================================================================== -->
    <!-- Dialogic Mtp3UserPart -->
    <!-- ===================================================================== -->
    <!---->
    <bean name="Mtp3UserPart"
class="org.restcomm.ss7.hardware.dialogic.DialogicMtp3UserPart">
        <constructor>
            <parameter>{this-platform}-jSS7</parameter>
        </constructor>
            <property name="sourceModuleId">61</property>
            <property name="destinationModuleId">34</property>
            <property name="routingLabelFormat">
                    <inject bean="RoutingLabelFormat" />
            </property>
    </bean>
```

This Dialogic Bean takes a `String` as a first constructor argument. This is a productName parameter.

The other properties of the Stack are defined below:

sourceModuleId

`sourceModuleId` is the id of source module and should match with the value configured in the file *system.txt* used by `dialogic` drivers. In the above example, 61 is assigned for mobicents process.

destinationModuleId

`destinationModuleId` is the id of destination module. In the above example, 34 is the id of Dialogic MTP3 module.

routingLabelFormat

The routing label format supported by this network. See Configuring MTP3 routing label for further details.

Configuring MTP3 routing label

MTP Level 3 routes messages based on the routing label in the signaling information field (SIF) of message signal units. The routing label is comprised of the destination point code (DPC), originating point code (OPC), and signaling link selection (SLS) field. Overtime different standards cameup with different routing label format. For example An ANSI routing label uses 7 octets; an ITU-T routing label uses 4 octets.

Restcomm jSS7 is flexible to configure the routing label as shown below.

```
        <!-- ================================================================== -->
        <!-- MTP3 Properties -->
        <!-- Define MTP3 routing label Format -->
        <!-- ================================================================== -->
        <bean name="RoutingLabelFormat"
class="org.restcomm.protocols.ss7.mtp.RoutingLabelFormat">
                <constructor
factoryClass="org.restcomm.protocols.ss7.mtp.RoutingLabelFormat"
                        factoryMethod="getInstance">
                        <parameter>ITU</parameter>
                </constructor>
        </bean>
```

Following table shows various routing formats supported

Table 1. Routing Format

Name	point code length	sls length
ITU	14-bits	4-bits
ANSI_Sls8Bit	24-bits	8-bits
ANSI_Sls5Bit	24-bits	5-bits
Japan_TTC_DDI	not supported yet	not supported yet
Japan_NTT	not supported yet	not supported yet
China	not supported yet	not supported yet

Configuring SCCP

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

```
    <!-- ================================================================== -->
    <!-- SCCP Service -->
    <!-- ================================================================== -->
    <!--<bean name="RuleComparator"
class="org.restcomm.protocols.ss7.sccp.impl.router.RuleByIdComparator"/>-->
      <bean name="RuleComparator"
class="org.restcomm.protocols.ss7.sccp.impl.router.RuleComparator"/>
      <bean name="RuleComparatorFactory"
class="org.restcomm.protocols.ss7.sccp.impl.router.RuleComparatorFactory">
            <constructor
factoryClass="org.restcomm.protocols.ss7.sccp.impl.router.RuleComparatorFactory"
                        factoryMethod="getInstance">
                    <parameter>RuleComparatorFactory</parameter>
            </constructor>
            <property name="ruleComparator">
                 <inject bean="RuleComparator"/>
```

```
            </property>
      </bean>
    <bean name="SccpStack" class="org.restcomm.protocols.ss7.sccp.impl.SccpStackImpl">
        <constructor>
            <parameter>
                <inject bean="SS7Scheduler" />
            </parameter>
            <parameter>SccpStack</parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
        <property name="mtp3UserParts">
            <map keyClass="java.lang.Integer"
valueClass="org.restcomm.protocols.ss7.mtp.Mtp3UserPart">
                <entry>
                    <key>1</key>
                        <value>
                                <inject bean="Mtp3UserPart" />
                        </value>
                </entry>
            </map>
        </property>
    </bean>

    <bean name="SccpExecutor"
class="org.restcomm.protocols.ss7.sccp.impl.oam.SccpExecutor">
        <property name="sccpStacks">
            <map keyClass="java.lang.String"
valueClass="org.restcomm.protocols.ss7.sccp.impl.SccpStackImpl">
                <entry>
                    <key>SccpStack</key>
                    <value>
                        <inject bean="SccpStack" />
                    </value>
                </entry>
            </map>
        </property>
    </bean>
```

`org.restcomm.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 Restcomm SS7 Service is started 3 file's *SccpStack_management2.xml*, *SccpStack_sccpresource2.xml* and *SccpStack_sccprouter2.xml* will be created at *$JBOSS_HOME/server/profile_name/data* directory You can disable of using persistent configuration via using `org.restcomm.protocols.ss7.sccp.impl.NonPersistentSccpStackImpl` class. Stack has following properties:

persistDir

As explained above

mtp3UserParts

specifies SS7 Level 3 to be used as transport medium(be it SS7 card or M3UA). Restcomm jSS7 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

For sorting SCCP rules by ID we need to set `RuleComparator` to `org.restcomm.protocols.ss7.sccp.impl.router.RuleByIdComparator`. By default it sorts by calledDigits and callingDigits patterns.

Configuring TCAP

`TcapStack` initiates the TCAP stack routines. Respective TCAP stack beans are instantiated for each `MAP`, `CAP` Service. If you are using either one, feel free to delete the other.

```
    <!-- ================================================================== -->
    <!-- TCAP Service -->
    <!-- ================================================================== -->
    <bean name="TcapStackMap" class="org.restcomm.protocols.ss7.tcap.TCAPStackImpl">
        <constructor>
            <parameter>TcapStackMap</parameter>
            <parameter>
                <inject bean="SccpStack" property="sccpProvider" />
            </parameter>
            <parameter>8</parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
        <property name="previewMode">false</property>
    </bean>

    <bean name="TcapStackCap" class="org.restcomm.protocols.ss7.tcap.TCAPStackImpl">
        <constructor>
            <parameter>TcapStackCap</parameter>
                <parameter>
                    <inject bean="SccpStack" property="sccpProvider" />
                </parameter>
                <parameter>146</parameter>
            </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
        <property name="previewMode">false</property>
    </bean>

    <bean name="TcapStack" class="org.restcomm.protocols.ss7.tcap.TCAPStackImpl">
        <constructor>
            <parameter>TcapStack</parameter>
            <parameter>
                <inject bean="SccpStack" property="sccpProvider" />
            </parameter>
```

```
            <parameter>9</parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
        <property name="previewMode">false</property>
    </bean>

    <bean name="TcapExecutor" class=
"org.restcomm.protocols.ss7.tcap.oam.TCAPExecutor">
        <property name="tcapStacks">
            <map keyClass="java.lang.String"
valueClass="org.restcomm.protocols.ss7.tcap.TCAPStackImpl">
                <entry>
                    <key>TcapStackMap</key>
                    <value>
                        <inject bean="TcapStackMap" />
                    </value>
                </entry>
                <entry>
                    <key>TcapStackCap</key>
                    <value>
                        <inject bean="TcapStackCap" />
                    </value>
                </entry>
                <entry>
                    <key>TcapStack</key>
                    <value>
                        <inject bean="TcapStack" />
                    </value>
                </entry>
            </map>
        </property>
    </bean>
```

`org.restcomm.protocols.ss7.tcap.TCAPStackImpl` takes `String` as a first constructor argument. The name is prepend to `xml` file created by TCAP stack for persisting state of TCAP resources. The `xml` is stored in path specified by `persistDir` property above.

For example in above case, when Restcomm SS7 Service is started 3 file's *TcapStack_management.xml*, *TcapStack_managementMap.xml* and *TcapStack_managementCap.xml* will be created at *$JBOSS_HOME/server/profile_name/data* directory. Then `org.restcomm.protocols.ss7.tcap.TCAPStackImpl` takes `SccpStack` as second constructor argument. TCAP uses passed SCCP stack. Constructor also takes the sub system number (SSN) which is registered with passed SCCP stack (this is the third parameter). You can disable of using persistent configuration via using `org.restcomm.protocols.ss7.tcap.NonPersistentTCAPStackImpl` class. TCAP Stack has following configurable properties:

persistDir

As explained above

previewMode: public void setPreviewMode(boolean val);

PreviewMode is needed for special processing mode. By default TCAP is not set in PreviewMode. When PreviewMode set in TCAP level:

- Stack only listens for incoming messages and does not send anything. The methods `send()`, `close()`, `sendComponent()` and other such methods do nothing.
- A TCAP Dialog is temporary. TCAP Dialog is discarded after any incoming message like TC-BEGIN or TC-CONTINUE has been processed.
- For any incoming messages (including TC-CONTINUE, TC-END, TC-ABORT) a new TCAP Dialog is created (and then deleted).
- There are no timers and timeouts.

`TcapExecutor` accepts `tcap` commands and executes necessary operations

Configuring ShellExecutor

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

```
    <!-- ================================================================== -->
    <!-- Shell Service -->
    <!-- ================================================================== -->
    <!-- Define Shell Executor -->
    <bean name="ShellExecutor"
class="com.mobicents.ss7.management.console.ShellServer">
        <constructor>
            <parameter>
                <inject bean="SS7Scheduler" />
            </parameter>
            <parameter>
                <list class="javolution.util.FastList"
elementClass="org.restcomm.ss7.management.console.ShellExecutor">
                    <inject bean="SccpExecutor" />
                    <inject bean="M3UAShellExecutor" />
                    <inject bean="SCTPShellExecutor" />
                    <inject bean="TcapExecutor" />
                    <!-- <inject bean="LinksetExecutor" /> -->
                </list>
            </parameter>
        </constructor>

        <property name="address">${jboss.bind.address}</property>
        <property name="port">3435</property>
        <property name="securityDomain">java:/jaas/jmx-console</property>
    </bean>
```

By default ShellExecutor listens at `jboss.bind.address` and port `3435`. (This is used when you use CLI access after running of ss7-cli command). You may set the `address` property to any valid IP address that your host is assigned. The shell commands are exchanged over TCP/IP.

To understand JBoss bind options look at Installation_And_Getting_Started_Guide

`SCTPShellExecutor` and `M3UAShellExecutor` is declared only if MTP layer `M3UA` is used. If `dialogic` MTP layer is used these beans are not decalred and should be removed from `FastList` too. For `dahdi` need to declare `LinksetExecutor` bean and add in `FastList` above.

Configuring MAP

`MapStack` initiates the MAP stack routines.

```
    <!-- ================================================================== -->
    <!-- MAP Service -->
    <!-- ================================================================== -->
    <bean name="MapStack" class="org.restcomm.protocols.ss7.map.MAPStackImpl">
        <constructor>
            <parameter>MapStack</parameter>
            <parameter>
                <inject bean="TcapStackMap" property="provider" />
            </parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
    </bean>
```

`org.restcomm.protocols.ss7.tcap.MAPStackImpl` takes `String` as a first constructor argument. The name is prepend to `xml` file created by MAP stack for persisting state of MAP resources. The `xml` is stored in path specified by `persistDir` property above. The second constructor argument is `TcapStack`. MAP uses passed TCAP stack.

Feel free to delete declaration of this bean if your service is not consuming MAP messages.

Configuring CAP

`CapStack` initiates the CAP stack routines.

```
    <!-- ================================================================== -->
    <!-- CAP Service -->
    <!-- ================================================================== -->
    <bean name="CapStack" class="org.restcomm.protocols.ss7.cap.CAPStackImpl">
        <constructor>
            <parameter>CapStack</parameter>
            <parameter>
                <inject bean="TcapStackCap" property="provider" />
            </parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
    </bean>
```

`org.restcomm.protocols.ss7.tcap.CAPStackImpl` takes `String` as a first constructor argument. The name is prepend to `xml` file created by CAP stack for persisting state of CAP resources. The `xml` is stored in path specified by `persistDir` property above. The second constructor argument is `TcapStack`. CAP uses passed TCAP stack.

Feel free to delete declaration of this bean if your service is not consuming CAP messages.

Configuring ISUP

`IsupStack` initiates the ISUP stack routines.

```
    <!-- ================================================================== -->
    <!-- ISUP Service -->
    <!-- ================================================================== -->
    <bean name="CircuitManager"
 class="org.restcomm.protocols.ss7.isup.impl.CircuitManagerImpl">
    </bean>

    <bean name="IsupStack" class="org.restcomm.protocols.ss7.isup.impl.ISUPStackImpl">
        <constructor>
            <parameter>
                <inject bean="SS7Scheduler" />
            </parameter>
            <parameter>22234</parameter>
            <parameter>2</parameter>
        </constructor>
        <property name="mtp3UserPart">
            <inject bean="Mtp3UserPart" />
        </property>
        <property name="circuitManager">
            <inject bean="CircuitManager" />
        </property>
    </bean>
```

`org.restcomm.protocols.ss7.isup.impl.ISUPStackImpl` takes `SS7Scheduler`, local signaling pointcode

and network indicator as constructor argument.

Stack has following properties:

mtp3UserPart

specifies SS7 Level 3 to be used as transport medium (be it SS7 card or M3UA).

circuitManager

CIC management bean

Feel free to delete declaration of this bean if your service is not consuming ISUP messages.

Configuring SS7Service

SS7Service acts as core engine binding all the components together.

```
    <!-- ===================================================================== -->
    <!-- RestComm SS7 Service -->
    <!-- ===================================================================== -->
    <bean name="TCAPSS7Service" class="org.restcomm.ss7.SS7Service">
        <constructor><parameter>TCAP</parameter></constructor>

<annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.restcomm.ss7:servi
ce=TCAPSS7Service",exposedInterface=org.restcomm.ss7.SS7ServiceMBean.class,registerDir
ectly=true)
        </annotation>
        <property name="jndiName">java:/restcomm/ss7/tcap</property>
        <property name="stack">
            <inject bean="TcapStack" property="provider" />
        </property>
    </bean>
    <bean name="MAPSS7Service" class="org.restcomm.ss7.SS7Service">
        <constructor><parameter>MAP</parameter></constructor>

<annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.restcomm.ss7:servi
ce=MAPSS7Service",exposedInterface=org.restcomm.ss7.SS7ServiceMBean.class,registerDire
ctly=true)
        </annotation>
        <property name="jndiName">java:/restcomm/ss7/map</property>
        <property name="stack">
            <inject bean="MapStack" property="MAPProvider" />
        </property>
    </bean>
    <bean name="CAPSS7Service" class="org.restcomm.ss7.SS7Service">
        <constructor><parameter>CAP</parameter></constructor>

<annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.restcomm.ss7:servi
ce=CAPSS7Service",exposedInterface=org.restcomm.ss7.SS7ServiceMBean.class,registerDire
ctly=true)
        </annotation>
```

```
        <property name="jndiName">java:/restcomm/ss7/cap</property>
        <property name="stack">
            <inject bean="CapStack" property="CAPProvider" />
        </property>
    </bean>
    <bean name="ISUPSS7Service" class="org.restcomm.ss7.SS7Service">
        <constructor><parameter>ISUP</parameter></constructor>

<annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.restcomm.ss7:servi
ce=ISUPSS7Service",exposedInterface=org.restcomm.ss7.SS7ServiceMBean.class,registerDir
ectly=true)
        </annotation>
        <property name="jndiName">java:/restcomm/ss7/isup</property>
        <property name="stack">
            <inject bean="IsupStack" property="isupProvider" />
        </property>
    </bean>
```

TCAPSS7Service binds TcapStack to JNDI `java:/restcomm/ss7/tcap`.

MAPSS7Service binds MapStack to JNDI `java:/restcomm/ss7/map`.

CAPSS7Service binds CapStack to JNDI `java:/restcomm/ss7/cap`.

ISUPSS7Service binds IsupStack to JNDI `java:/restcomm/ss7/isup`.

The JNDI name can be configured to any valid JNDI name specific to your application.

Feel free to delete service that your application is not using.

Configuring jSS7 Management Service

jSS7 Managemenet Service provides some extra functionality for stack management including jmx access to stacks, performance (statistics) and alarm management.

```
    <bean name="Ss7Management"
        class="org.restcomm.protocols.ss7.oam.common.jmxss7.Ss7Management">

<annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name="org.restcomm.ss7:servi
ce=Ss7Management",exposedInterface=org.restcomm.protocols.ss7.oam.common.jmxss7.Ss7Man
agementMBean.class,registerDirectly=true)</annotation>
        <property name="agentId">jboss</property>
    </bean>

    <bean name="RestcommAlarmManagement"
        class="org.restcomm.protocols.ss7.oam.common.alarm.AlarmProvider">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
```

```
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
        </constructor>
    </bean>

    <bean name="RestcommStatisticManagement"

class="org.restcomm.protocols.ss7.oam.common.statistics.CounterProviderManagement">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
        </constructor>
        <property name="persistDir">${jboss.server.data.dir}</property>
    </bean>

    <bean name="RestcommSctpManagement"
        class="org.restcomm.protocols.ss7.oam.common.sctp.SctpManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="SCTPManagement" />
            </parameter>
        </constructor>
    </bean>

    <bean name="RestcommM3uaManagement"
        class="org.restcomm.protocols.ss7.oam.common.m3ua.M3uaManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="Mtp3UserPart" />
            </parameter>
        </constructor>
    </bean>

    <bean name="RestcommSccpManagement"
        class="org.restcomm.protocols.ss7.oam.common.sccp.SccpManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="SccpStack" />
            </parameter>
        </constructor>
```

```
    </bean>

    <bean name="RestcommTcapManagement"
        class="org.restcomm.protocols.ss7.oam.common.tcap.TcapManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="TcapStack" />
            </parameter>
        </constructor>
    </bean>

    <bean name="RestcommTcapMapManagement"
        class="org.restcomm.protocols.ss7.oam.common.tcap.TcapManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="TcapStackMap" />
            </parameter>
        </constructor>
    </bean>

    <bean name="RestcommTcapCapManagement"
        class="org.restcomm.protocols.ss7.oam.common.tcap.TcapManagementJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="TcapStackCap" />
            </parameter>
        </constructor>
    </bean>

<!--
    <bean name="RestcommLinksetManagement"
        class="org.restcomm.protocols.ss7.oam.common.linkset.LinksetManagerJmx">
        <constructor>
            <parameter>
                <inject bean="Ss7Management" />
            </parameter>
            <parameter>
                <inject bean="LinksetManager" />
            </parameter>
        </constructor>
    </bean>
-->
```

Configuring Restcomm Signaling Gateway

Configuration is done through an XML descriptor named *sgw-beans.xml* and is located at *restcomm-ss7-sgw/deploy*

Before Restcomm Signaling Gateway is configured the corresponding native libraries for `dahdi` or `dialogic` should be copied to `restcomm-ss7-sgw/native/32` or `restcomm-ss7-sgw/native/64` folders and gctApi library to `restcomm-ss7-sgw/lib` folder. See more details for where to get native libraries in Configuring dahdi and Configuring dialogic.

Configuring M3UA (Signaling Gateway)

SGW will expose the SS7 signals received from legacy network to IP network over M3UA

```
    <bean name="SCTPManagement"
class="org.mobicents.protocols.sctp.netty.NettySctpManagementImpl">
        <constructor>
            <parameter>SCTPManagement</parameter>
        </constructor>
        <property name="persistDir">${sgw.home.dir}/ss7</property>
    </bean>

    <bean name="SCTPShellExecutor"
class="org.restcomm.protocols.ss7.m3ua.impl.oam.SCTPShellExecutor">
        <property name="sctpManagements">
            <map keyClass="java.lang.String"
valueClass="org.mobicents.protocols.sctp.ManagementImpl">
                <entry>
                    <key>SCTPManagement</key>
                    <value>
                        <inject bean="SCTPManagement" />
                    </value>
                </entry>
            </map>
        </property>
    </bean>

    <bean name="Mtp3UserPart"
class="org.restcomm.protocols.ss7.m3ua.impl.M3UAManagementImpl">
        <constructor>
            <parameter>Mtp3UserPart</parameter>
            <parameter>Restcomm-jSS7</parameter>
        </constructor>
        <property name="persistDir">${sgw.home.dir}/ss7</property>
        <property name="transportManagement">
            <inject bean="SCTPManagement" />
        </property>
    </bean>

    <bean name="M3UAShellExecutor"
class="org.restcomm.protocols.ss7.m3ua.impl.oam.M3UAShellExecutor">
        <property name="m3uaManagements">
            <map keyClass="java.lang.String"
valueClass="org.restcomm.protocols.ss7.m3ua.impl.M3UAManagementImpl">
                <entry>
                    <key>Mtp3UserPart</key>
                    <value>
                        <inject bean="Mtp3UserPart" />
                    </value>
                </entry>
            </map>
        </property>
    </bean>
```

Configuring LinksetFactory

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

- `DahdiLinksetFactory`

```
    <bean name="DahdiLinksetFactory"
        class="org.restcomm.ss7.hardware.dahdi.oam.DahdiLinksetFactory">
        <property name="scheduler">
            <inject bean="Scheduler" />
        </property>
    </bean>
```

- `DialogicLinksetFactory`

```
    <bean name="DialogicLinksetFactory"
        class="org.restcomm.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.

Configuring LinksetManager

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

```
    <!-- ============================================== -->
    <!-- Linkset manager Service                                                -->
    <!-- ============================================== -->
    <bean name="LinksetManager"
class="org.restcomm.ss7.linkset.oam.LinksetManagerImpl">
        <constructor>
            <parameter>LinksetManager</parameter>
        </constructor>
        <property name="scheduler">
            <inject bean="Scheduler" />
        </property>

        <property name="linksetFactoryFactory">
            <inject bean="LinksetFactoryFactory" />
        </property>
        <property name="persistDir">${sgw.home.dir}/ss7</property>
    </bean>

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

LinksetManagerImpl 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.

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

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

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.restcomm.ss7.management.console.ShellServer">
        <constructor>
            <parameter>
                <inject bean="Scheduler" />
            </parameter>
            <parameter>
                <list class="javolution.util.FastList"
elementClass="org.restcomm.ss7.management.console.ShellExecutor">
                    <inject bean="M3UAShellExecutor" />
                    <inject bean="SCTPShellExecutor" />
                    <inject bean="LinksetExecutor" />
                </list>
            </parameter>
        </constructor>

        <property name="address">${sgw.bind.address}</property>
        <property name="port">3435</property>
    </bean>
```

By default ShellExecutor listens at `sgw.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.

Configuring SignalingGateway

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

```
    <!-- ================================================ -->
    <!-- mobicents Signaling Gateway                      -->
    <!-- ================================================ -->
    <bean name="SignalingGateway" class="org.restcomm.ss7.sgw.SignalingGateway">
        <property name="scheduler">
            <inject bean="Scheduler" />
        </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