Mobicents JAIN SLEE USSD Gateway Application User Guide

by Amit Bhayani and Bartosz Baranowski

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/mobicents/issues/list], against the product **Mobicents JAIN SLEE USSD Gateway Application**, or contact the authors.

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

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

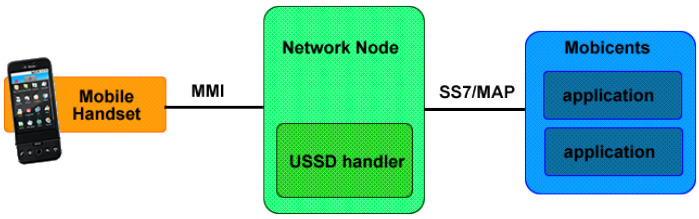
Introduction to Mobicents JAIN SLEE USSD Gateway Application

USSD stands for Unstructured Supplementary Service Data what is a capability of GSM mobile phone much like the Short Message Service (SMS). But there is a difference between USSD and SMS handling.

SMS uses `store and forward` method of message delivery. Short Message is delivered first to Sender's Short Message Service Center (SMSc) which will try to deliver the message to recipient. So SMS does not guarantee that message will be delivered instantly.

USSD information is sent from mobile handset directly to application platform handling service. So USSD suppose to establish a real time session between mobile handset and application handling the service. The concept of real time session is very useful for constructing an interactive menu driven application.

A user who is dialing USSD service number initiates dialog with USSD handling application deployed on the Mobicents Platform as depicted on the figure below. The "Network Node" depicted could be MSC, HLR or VLR. The Mobicents Platform integrates with "Network Node" using MAP protocol.



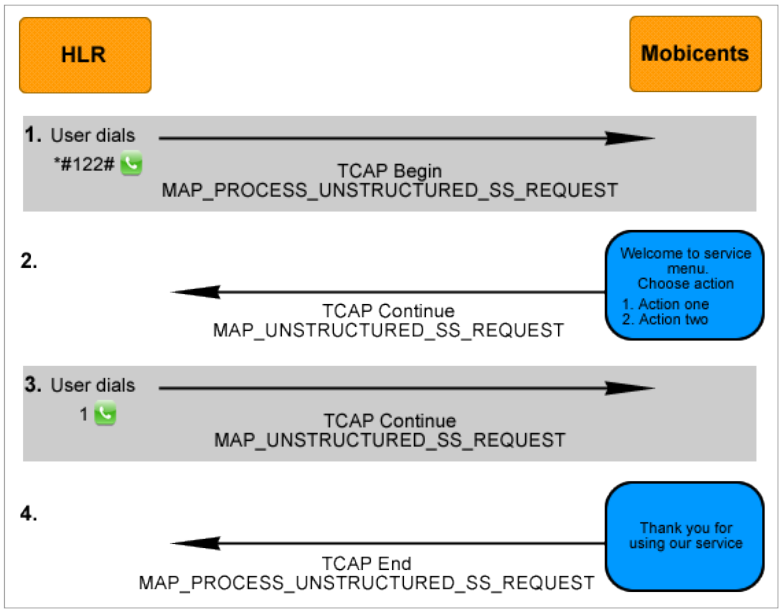
General interworking diagram

The detailed description of the allowed MMIs or phone number which user can dial is presented in `3GPP TS 22.090`. In the user's home network the following number range is defined for USSD services: *`1, 2 or 3 digits from the set (*, #) followed by 1X(Y), where X=any number 0-4, Y=any number 0-9, then, optionally "*" followed by any number of any characters, and concluding with # SEND`*

For example user can dial *#122# to reach a specific USSD service which is deployed in the home network. The application in its order can reply with menu.

One of the biggest benefits is that this service is always available even when user is currently in roaming.

Below diagram depicts typical MAP message flow for implementing data transfer between "Network Node" and Mobicents platform to implement menu driven application. For more information on mobile- (and network-) initiated USSD operations and the use of MAP USSD services, refer to [3GPPTS 24.090] in the References section.



Message flow

Mobile initiated USSD service starts when user dials USSD string *#122#.

- The Network sends TCAP Begin message with Component MAP_PROCESS_UNSTRUCTURED_SS_REQUEST to the Mobicents platform. The Mobicents platform invokes USSD application logic .
- Application request additional information from user (action one or action two) via MAP_UNSTRUCTURED_SS_REQUEST encapsulated in TCAP Continue message. At this time TCAP Dialogue starts.
- Application receives user's selection of the action.
- Application performs its logic and sends a response back to the user. At this time application do not want to get additional information from the user and it sends response using MAP_PROCESS_UNSTRUCTURED_SS_REQUEST and terminates TCAP dialogue.

1.1. USSD Gateway

Existing MSC, VLR, and HLR network elements are proprietary and run on non-standard operating environments located in trusted operator's zones that make it difficult to build and deploy new applications. Also, these network elements do not provide the tools and interfaces needed to access and retrieve data from content providers over Internet. The USSD Gateway connects to the MSC, VLR, or HLR and enables the flow of USSD messages to be extended to an open,

standards-based application server located in the IP network. The AS also provides the tools and interfaces to enable access to the content providers through the Internet.

Mobicents implementation of USSD Gateway is first and only open source USSD Gateway available as of today. The Mobicents USSD Gateway makes use of SIP and HTTP protocol between gateway and Value Added Service Modules or third party applications. Mobicents USSD Gateway receives the USSD request from subscriber handset/device via GSM Signaling network, these requests are translated to SIP or HTTP depending on the rules set by the user and then routed to corresponding Value Added Service (VAS) or 3rd party application. JBoss Drools is used to derive the protocol between Gateway and USSD Application and also the information of the server (for example IP, port etc) where these applications are deployed.

Setup

2.1. Pre-Install Requirements and Prerequisites

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

2.1.1. Hardware Requirements

The Application doesn't change the Mobicents JAIN SLEE Hardware Requirements, refer to Mobicents JAIN SLEE documentation for more information.

Note

Note that application makes use of Resource Adaptors - this implies that RAs requirements must be taken into consideration!

Also be aware that each Resource Adaptor may have some specific hardware requirements!

2.1.2. Software Prerequisites

The Application requires Mobicents JAIN SLEE properly set, with:

- SIP
- HTTP Client
- MAP

Resource Adaptors deployed.

Note

Note MAP Resource Adaptor - has some specific software requirements! Please refer to MAP RA document in JSLEE Guide

2.2. Mobicents JAIN SLEE USSD Gateway Application Source Code

2.2.1. Release Source Code Building

1. **Downloading the source code**

Important

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

Use SVN to checkout a specific release source, the base URL is http://mobicents.googlecode.com/svn/tags/applications/ussdgateway, then add the specific release version, lets consider 1.0.0.BETA1.

```
[usr]$ svn co http://mobicents.googlecode.com/svn/tags/applications/ussdgateway/1.0.0.BETA1 slee-application-ussdgateway-1.0.0.BETA1
```

2. **Building the source code**

Important

Maven 2.0.9 (or higher) is used to build the release. Instructions for using Maven2, including install, can be found at *http://maven.apache.org*

Use Maven to build the binary.

```
[usr]$ cd slee-application-ussdgateway-1.0.0.BETA1
[usr]$ mvn install
```

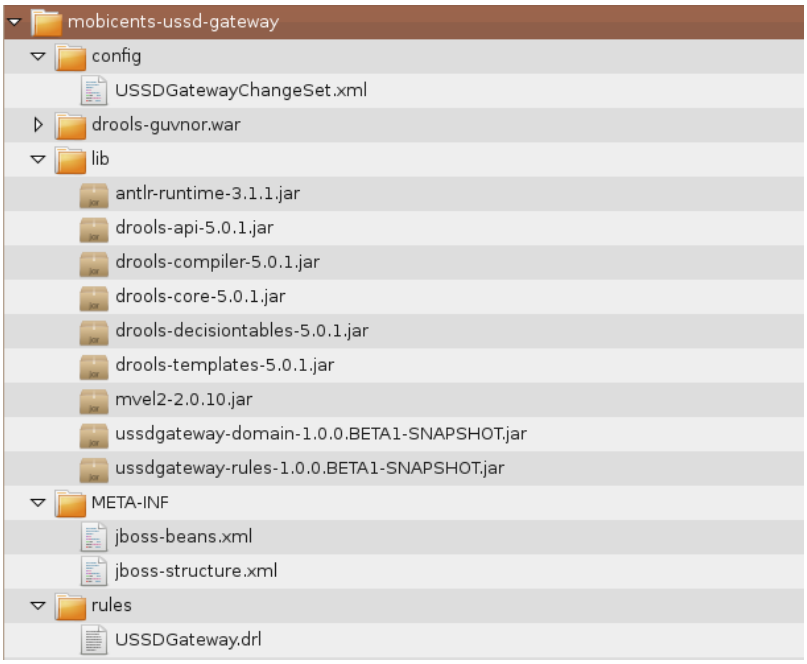
Once the process finishes you should have the `deployable-unit` jar file in the `target` directory, if Mobicents JAIN SLEE is installed and environment variable JBOSS_HOME is pointing to its underlying JBoss Application Server directory, then the deployable unit jar will also be deployed in the container.

2.2.2. Development Trunk Source Building

Similar process as for *Section 2.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is https://mobicents.googlecode.com/svn/trunk/applications/ussdgateway.

2.3. Folder structure of Mobicents JAIN SLEE USSD Gateway Application

Installing Mobicents USSD Gateway creates a mobicents-ussd-gateway directory that contains gateway configuration, libraries required for boot and running, example rules definition file (.drl) etc. You need to know your way around the distribution layout to locate the drools file's to add new rules. The figure "view of Mobicens USSD Gateway" illustrates the installation directory of the Gateway.



Mobicents USSD Gateway

2.4. Rule engine configuration

Important

USSD Gateway Application uses `Drools` as rule engine to perform decisions, it is important to understand *JBoss Drools* [http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-expert/html_single/]

Rule engine (`Drools`) is configured with `USSDGatewayChangeSet.xml` file. Its content alters how rule set is loaded and maintained within engine. There are two ways of maintaining rules:

locally
: rules are loaded from designated file. Configuration file should look as follows:

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
   xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
   xs:schemaLocation='http://drools.org/drools-5.0/change-set.xsd'>
   <add>
   (1)
      <resource
         source='file:/home/baranowb/servers/jboss-5.1.0.GA/server/default
            /deploy/mobicents-ussd-gateway/rules/'
         type='DRL' />
   </add>
</change-set>
```

(1) points to subdirectory in current application which is scanned for rule files.

remotely
: rules are managed by `Guvnor` Configuration file should look as follows:

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
   xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
   xs:schemaLocation='http://drools.org/drools-5.0/change-set.xsd'>
   <add>
   (1)
      <resource source='http://localhost:8080/drools-guvnor/
         org.drools.guvnor.Guvnor/package/ussdGateway/LATEST.drl' type='DRL' />
   </add>
</change-set>
```

(1) points to `Guvnors` latest rule file. Note that path after package `MUST` match your custom created package inside `Guvnor` .

2.5. Guvnor configuration

Important

USSD Gateway Application uses `Gunvor` to manage system wide rule set in consistent way, it is important to understand *Guvnor* [http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-guvnor/html_single/]

`Guvnor` is deployed along with USSD Gateway Application. To access it simply go to `http://<your server>/drools-guvnor/` . This will bring initial info screen or login screen - depends on configuration.

If you have not configured the security you can directly login without providing any user id or password.

2.5.1. Creating resources

Note

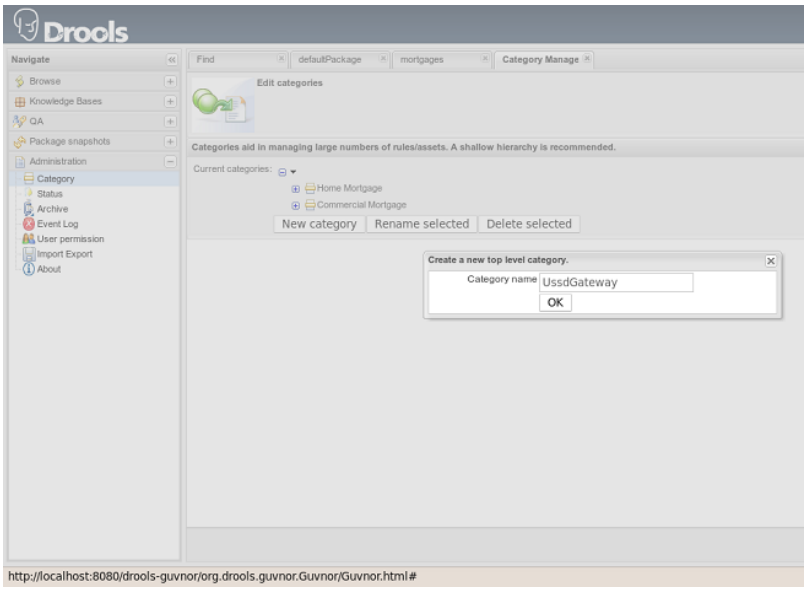
`Guvnor` requires upload for fact model and creation of some resources before it can perform its tasks.

In case `Guvnor` has not been used(it is a new repository) you will get a message asking if you would you like to install a sample repository? Its upto you to install the sample repository. If you say yes, you would get sample repository which you can refer to have better understanding of Guvnor

Once you log-in follow the bellow steps:

1. **Create a category specific to USSD gateway.**

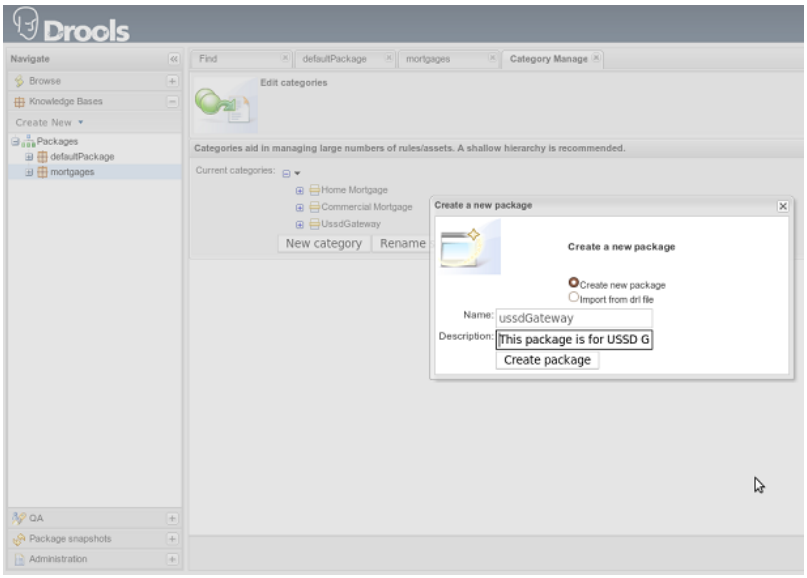
   Go to **Administration > Category > New Category** . Enter Category name as `UssdGateway` .



Guvnor category

2. **Create package for fact model.**

   Rules need a fact model (object model) to work off, so next you will want to go to the Package management feature. Go to **Knowledge Bases > Create New > New Package** . Type `ussdGateway` (note that this name `MUST` match package in `USSDGatewayChangeSet.xml` file).

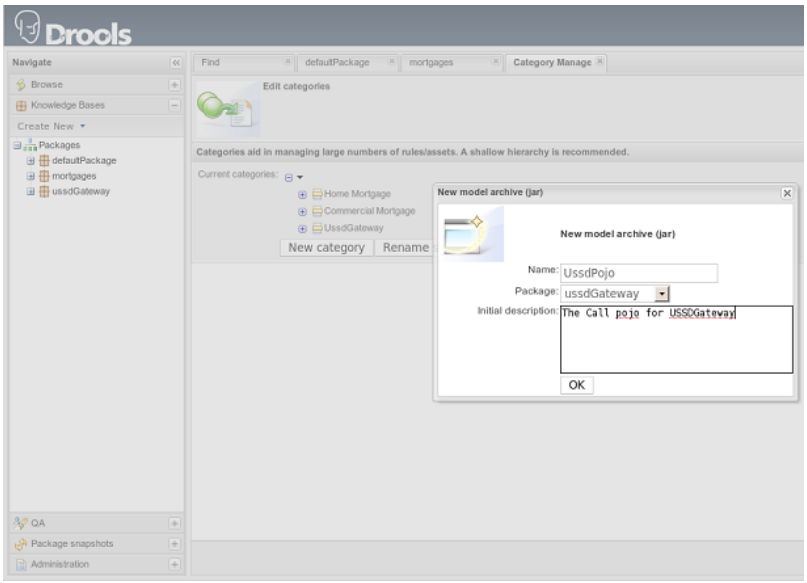


Guvnor package

3. **Upload fact model.**

   To upload a model, use ussdgateway-domain-x.y.z.jar which has the fact model (Call.java API) that you will be using in your rules. When you are in the model editor screen, you can upload a jar file, choose the package name from the list that you created in the previous step.

Go to **Knowledge Base > Create New > Upload POJO Model Jar** . On the screen enter name as `UssdPojo` , select package `ussdGateway` and add the description, click **Ok** .



Guvnor fact model upload

Browse in newly open window and point to `${JBOSS.HOME}/server/default/deploy/mobicents-ussd-gateway/lib/ussdgateway-domain-x.y.z.jar` .

4. **Edit your package configuration.**

   Now edit your package configuration (you just created) to import the fact types you just uploaded (add import statements), and save the changes. Go to Knowledge Bases and click on `ussdGateway` package. Click on **Save and validate configuration** button.

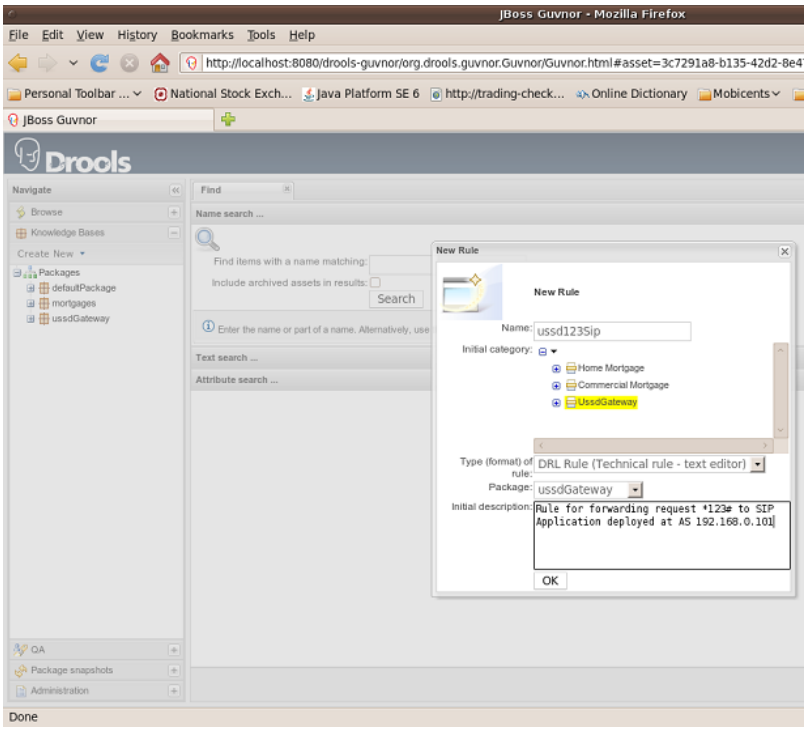
This concludes configuration of `Guvnor` . Note that this has to be done only once.

2.5.2. Creating rules

`Guvnor` allows to create rules and edit previously existing ones. Changes done with `Guvnor` are automaticly propagated to all clients. To create rule follow procedure below:

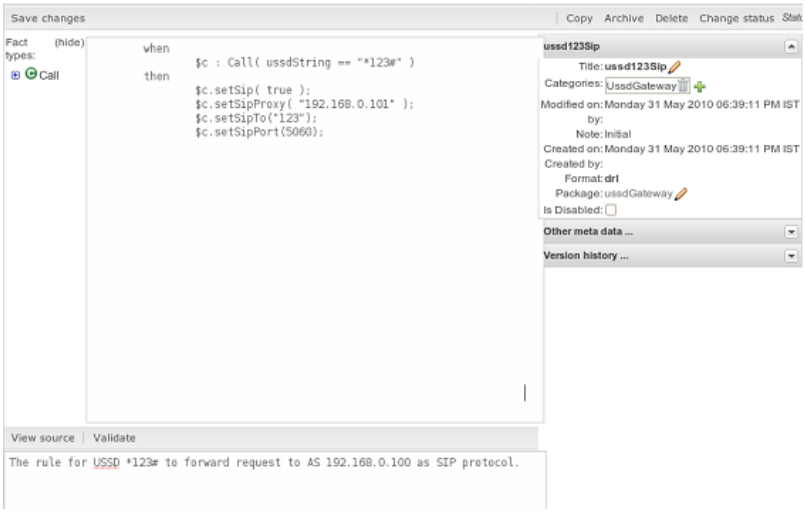
1. **Create rule.**

   Go to **Knowledge Bases> Create New > New Rule**. Enter Name as `ussd123Sip`, click on `UssdGateway` Initial Category. Select **DRL Rule (Technical rule - text editor)**, actually you can use any editor here that you are comfortable with. Select `ussdGateway` as package. Enter description and click `Ok`.



Guvnor new rule

2. **Edit rule.**



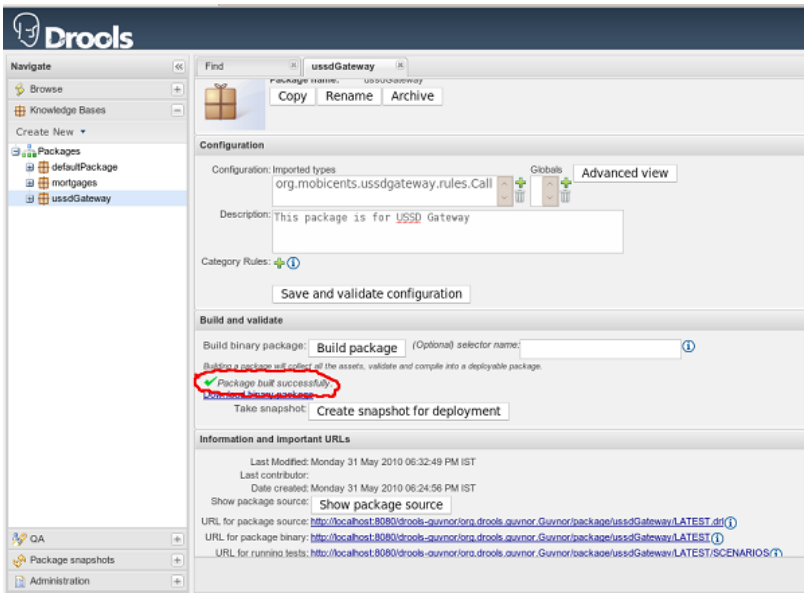
Guvnor edit rule

3. **Accept rule.**

   Click on **Validate** to validate the Rules you just defined. Once done with rule editing, you can check in the changes (save) by clicking on **Save Changes**

4. **Rebuild and validate package**

After you have edited some rules in **ussdGateway** package, you can click on the **ussdGateway** package, open the package, and build the whole package.



Guvnor new rule

Design Overview

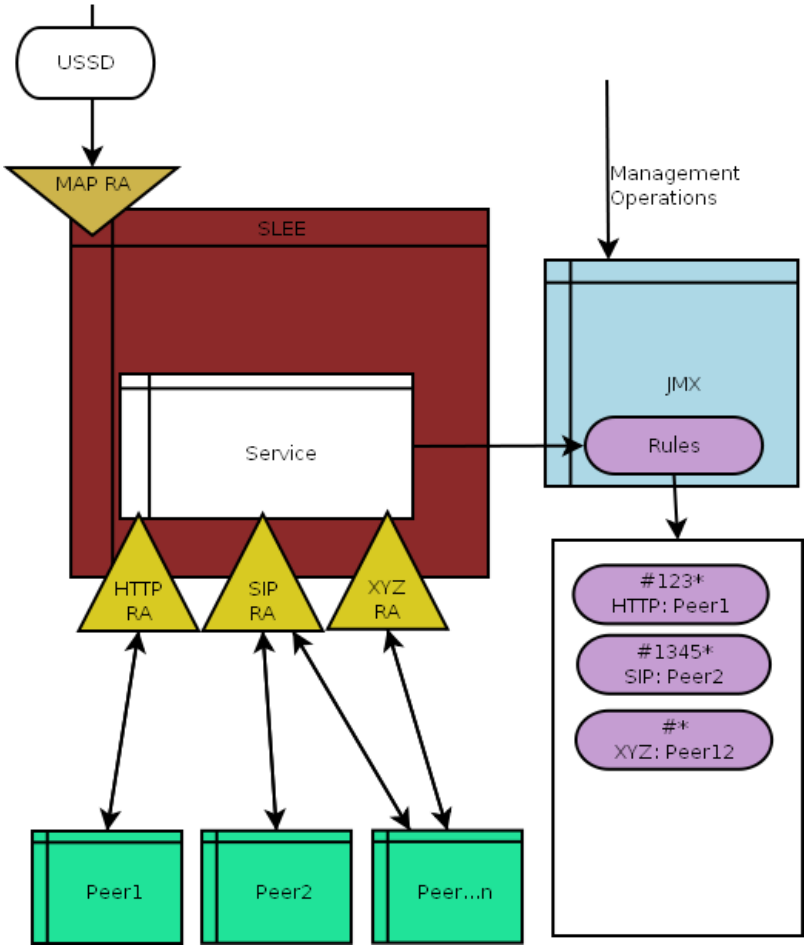
USSD Gateway Application is JAIN SLEE 1.1 Application. It is capable of forwarding USSD messages to desired peer. Application can be divided into logical modules:

rules
: this module is responsible for configuration. Based on user defined rules file, it chooses peer and means of establishing session with it.

jmx
: this module is responsible only for managing and exposing rules engine to SLEE Service

SLEE
: this module is consumer of rules. Based on rules output, it is responsible for relying USSD messages between originator and consumer peer.

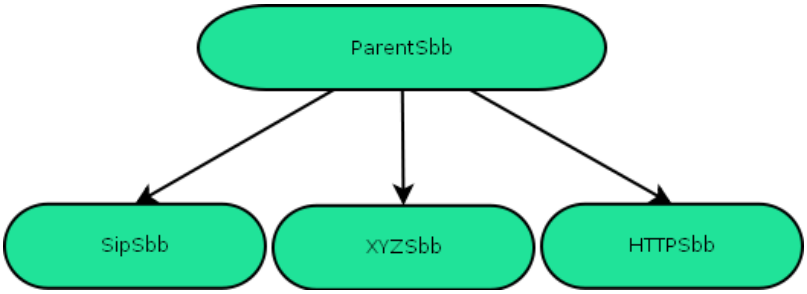
Following diagram depicts top design overview:



USSD Gateway Design overview

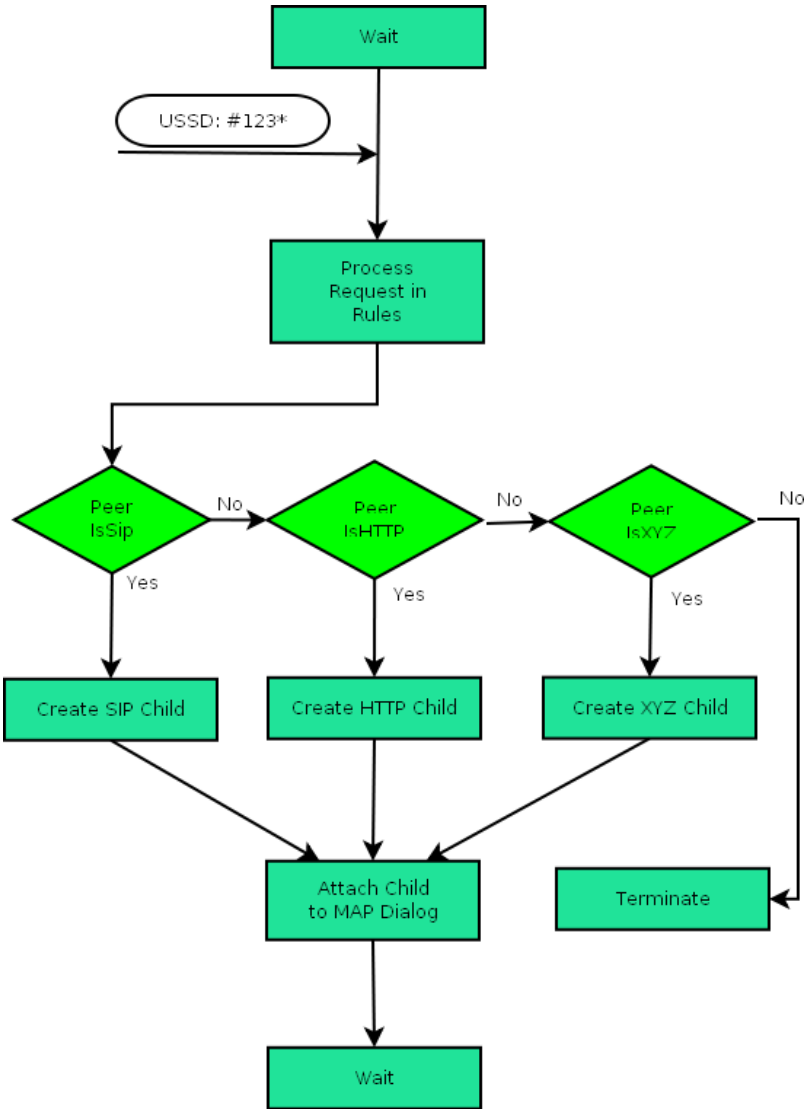
SLEE Service consists of two basic elements: parent and children SBBs. Parent SBB is root SBB in USSD Gateway Application service. Children serve logic to proxy USSD data to desired peer by means of specific protocol.

Overview of relation is depicted on diagram below:



USSD Gateway Design Service SBB relation overview

Parent receives initial USSD message within MAP message. It consults configured rules to make decision on fate of received message. Subsequent messages are handled by children. Flow for both operations is depicted on diagram below:



USSD Gateway SBB Flow diagram

Important

Currently gateway supports following protocols for proxying:

- SIP
- HTTP

Source Code Overview

Important

To obtain the application's complete source code please refer to *Section 2.2, "Mobicents JAIN SLEE USSD Gateway Application Source Code"*.

This chapter explains how components perform their tasks. For more detailed explanation of JSLEE related source code and xml descriptors, please refer to simple examples, like `sip-wakeup`. Also for detailed description of `rules` source code please refer to `Drools` documentation

4.1. Rules Source

USSD Gateway Application makes use of `Drools` as rules engine.

Engine is configured with `DRL` files. `DRL` file contains set of rules which perform operations on facts passed into engine. USSD Gateway Application `DRL` file defines rules to match initial USSD string to set of values identifying protocol and address of peer to which messages should be forwarded. Rule file name is `USSDGateway.drl`. File content looks as follows:

```
package org.mobicents.ussdgateway.rules
(1)
import org.mobicents.ussdgateway.rules.Call;
(2)
rule "USSDGateway1"
  (3)
  when
    $c : Call( ussdString == "*123#" )
  then(4)
    $c.setSip( true );
    $c.setSipProxy( "192.168.0.101" );
    $c.setSipTo("123");
    $c.setSipPort(5060);
(5)
end
```

(1) import of fact POJO

2. definition of rule
3. condition to enter rule clause. It accesses fact property `ussdString` and matches it against `#123*` value, if it matches engine jumps to `then` part
4. rule part which sets defined SIP peer as destination for messages
5. end of `USSDGateway1` rule

Rules are fed with facts on which engine performs matching operations(in general). Facts are simple POJO classes. This applications fact looks as follows:

```
package org.mobicents.ussdgateway.rules;

import java.io.Serializable;

/**
 * Acts as Fact for Rules
 * @author amit bhayani
 *
 */
public class Call implements Serializable {
    //Initial string, its like #123*
    private String ussdString;

    private boolean isSip;
    private boolean isHttp;
    private boolean isSmpp;

    private String sipProxy;
    private String sipTo;
    private int sipPort;

    //to be used with other protocols
    private String genericUrl;

    public Call(String ussdString){
        this.ussdString = ussdString;
    }

    public String getUssdString() {
        return ussdString;
    }

    public boolean isSip() {
        return isSip;
```

```
    }

    public void setSip(boolean isSip) {
        this.isSip = isSip;
    }

    public boolean isHttp() {
        return isHttp;
    }

    public void setHttp(boolean isHttp) {
        this.isHttp = isHttp;
    }

    public boolean isSmpp() {
        return isSmpp;
    }

    public void setSmpp(boolean isSmpp) {
        this.isSmpp = isSmpp;
    }

    public String getSipProxy() {
        return sipProxy;
    }

    public void setSipProxy(String sipProxy) {
        this.sipProxy = sipProxy;
    }

    public String getSipTo() {
        return sipTo;
    }

    public void setSipTo(String sipTo) {
        this.sipTo = sipTo;
    }

    public int getSipPort() {
        return sipPort;
    }

    public void setSipPort(int sipPort) {
        this.sipPort = sipPort;
```

```
    }

    /**
     * @return the genericUrl
     */
    public String getGenericUrl() {
        return genericUrl;
    }

    /**
     * @param genericUrl the genericUrl to set
     */
    public void setGenericUrl(String genericUrl) {
        this.genericUrl = genericUrl;
    }


}
```

4.2. JMX Source

JMX part of USSD Gateway Application is responsible for initiating rules engine and exposing management methods. Single JMX bean is defined with XML descriptor file `jboss-beans.xml`. Descriptor content looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:jboss:bean-deployer:2.0 bean-deployer_2_0.xsd"
    xmlns="urn:jboss:bean-deployer:2.0">

    <bean name="RulesService" class="org.mobicents.ussdgateway.rules.RulesService">
        <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(
            name="org.mobicents.ussdgateway:service=RulesService"
            ,exposedInterface=org.mobicents.ussdgateway.rules.RulesServiceMBean.class
            ,registerDirectly=true)
        </annotation>
    </bean>
</deployment>
```

JMX interface is defined as follows:

```
package org.mobicents.ussdgateway.rules;

import javax.naming.NamingException;

/**
 *
 * @author amit bhayani
 *
 */
public interface RulesServiceMBean  extends org.jboss.system.ServiceMBean {
    (1)
    String getJndiName();
    void setJndiName(String jndiName) throws NamingException;

    (2)
    void startService() throws Exception;
    void stopService() throws Exception;

}
```

It defines following:

1. management method. This method controls JNDI name under which rule engine is made available to SLEE Service
2. regular lifecycle methods

Implementation of above interface fulfills defined contracts in following way:

void startService() throws Exception

```
public void startService() throws Exception {
(1)
        setupRule();
```

```
(2)
        rebind();

        this.logger.info("Started Rules Service");
}

private void setupRule() {
(3)
        Resource resource = ResourceFactory.newUrlResource(CHANGESET_FILE_PATH);
(4)
        kagent = KnowledgeAgentFactory.newKnowledgeAgent("UssdGatewayAgent");
        kagent.applyChangeSet(resource);

        ResourceFactory.getResourceChangeNotifierService().start();
        ResourceFactory.getResourceChangeScannerService().start();


}
private void rebind() throws NamingException {

        InitialContext rootCtx = new InitialContext();
        // Get the parent context into which we are to bind
        Name fullName = rootCtx.getNameParser("").parse(jndiName);
        System.out.println("fullName=" + fullName);
        Name parentName = fullName;
        if (fullName.size() > 1)
           parentName = fullName.getPrefix(fullName.size() - 1);
        else
           parentName = new CompositeName();
        Context parentCtx = createContext(rootCtx, parentName);
        Name atomName = fullName.getSuffix(fullName.size() - 1);
        String atom = atomName.get(0);
        NonSerializableFactory.rebind(parentCtx, atom, kagent);
}

private static Context createContext(Context rootContext, Name name)
           throws NamingException {
        Context subctx = rootContext;
        for (int n = 0; n < name.size(); n++) {
           String atom = name.get(n);
           try {
              Object obj = subctx.lookup(atom);
              subctx = (Context) obj;
           } catch (NamingException e) { // No binding exists, create a
```

```
            // subcontext
            subctx = subctx.createSubcontext(atom);
        }
    }

    return subctx;
}
```

1. load rules into engine
2. bind rule engine to specified JNDI name
3. load `ChangeSet` as rule resource
4. create rule engine and initiate it

void stopService() throws Exception

```
public void stopService() throws Exception {
    (1)
    unbind(jndiName);
}
```

1. unbind rule engine from JNDI

void setJndiName(String jndiName) throws NamingException

```
public void setJndiName(String jndiName) throws NamingException {
    (1)
    String oldName = this.jndiName;
    this.jndiName = jndiName;
    (2)
    if (getState() == STARTED) {
        unbind(oldName);
        try {
            rebind();
        } catch (Exception e) {
            NamingException ne = new NamingException(
```

```
                    "Failed to update jndiName");
                ne.setRootCause(e);
                throw ne;
            }
        }

    }
```

1. set property and retain old value
1. rebind rule engine in JNDI in case bean is running

JMX bean loads change set file `USSDGatewayChangeSet.xml`. This file controls how rules set is loaded and maintained by rule engine. Please refer to *Section 2.4, "Rule engine configuration"* for details.

4.3. SLEE Service Source

4.3.1. Service descriptor

Mobicents USSD Gateway Application is build with single SLEE service. Service is defined with single XML descriptor. Descriptor file name is `callsbb-service.xml` . It contains following information:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE service-xml PUBLIC "-//Sun Microsystems, Inc.//DTD JAIN SLEE Service 1.1//EN"
                             "http://java.sun.com/dtd/slee-service-xml_1_1.dtd">
<service-xml>
   <service> 1
      <service-name>mobicents-ussdgateway</service-name>
      <service-vendor>org.mobicents</service-vendor>
      <service-version>1.0</service-version>
      <root-sbb> 2
         <sbb-name>ParentSbb</sbb-name>
         <sbb-vendor>org.mobicents</sbb-vendor>
         <sbb-version>1.0</sbb-version>
      </root-sbb> 3
      <default-priority>50</default-priority>
   </service>
</service-xml>
```

1. definition of service ID
2. definition of root SBB . SBB is referenced by its ID
3. definition of services priority - it affects order of event routing in container

4.3.2. SLEE Library

Since there is no standard defined for communication between USSD gateway and service providing peers, Mobicents USSD Gateway Application defines its own standard for encoding.

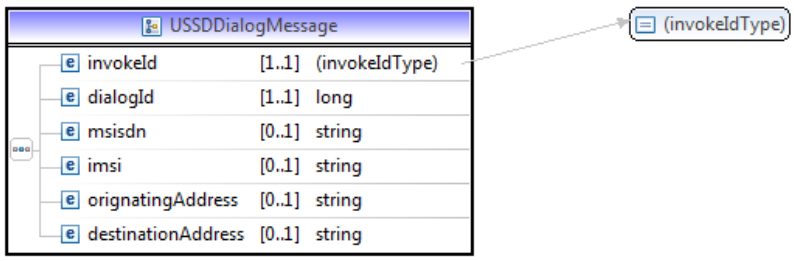
This SLEE library contains all classes required to properly encode and decode messages exchanged between Mobicents USSD Gateway and service providing peers. Library itself is built on top of JAXB framework.

Note

Library is SLEE component, in other environments it is possible that it wont work properly.

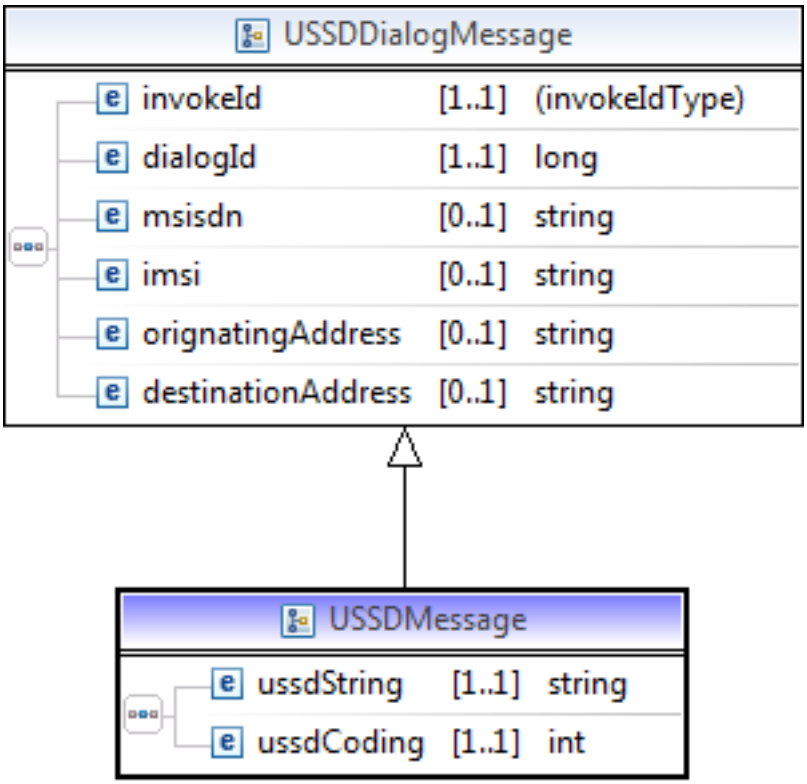
Standard is defined with single XSD file `ussd.xsd`. This file defines three types of encoded data:

USSDDialogMessage
: This structure is base for all messages exchanged between Gateway and application server. It conveys parameter which help to identify dialog and resources it addresses.



USSD Dialog Message

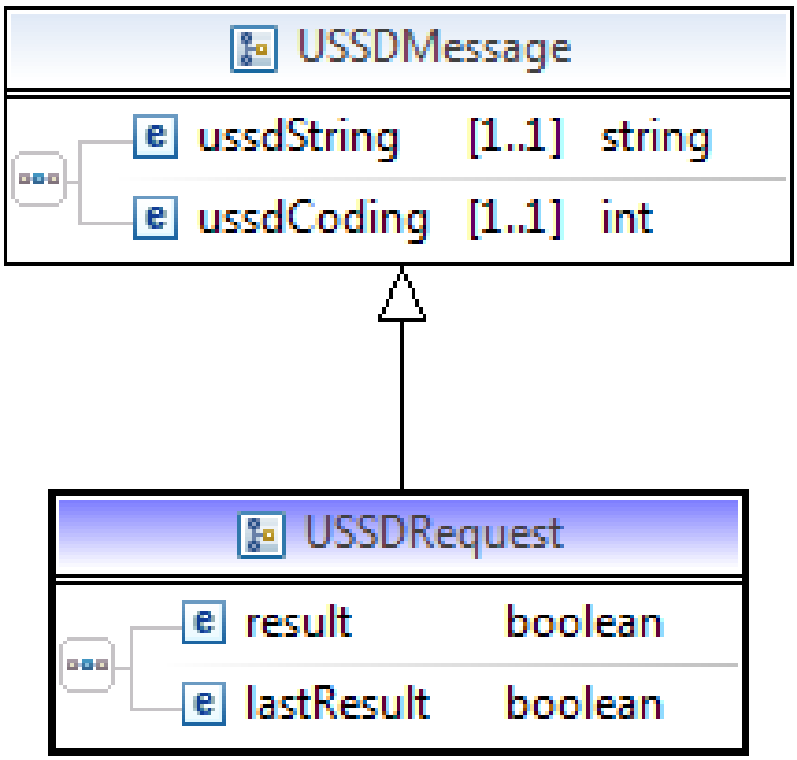
USSDMessage
: This structure is base for all USSD messages exchanged between Gateway and application server. It adds USSD specific data.



USSD Message

USSDRequest

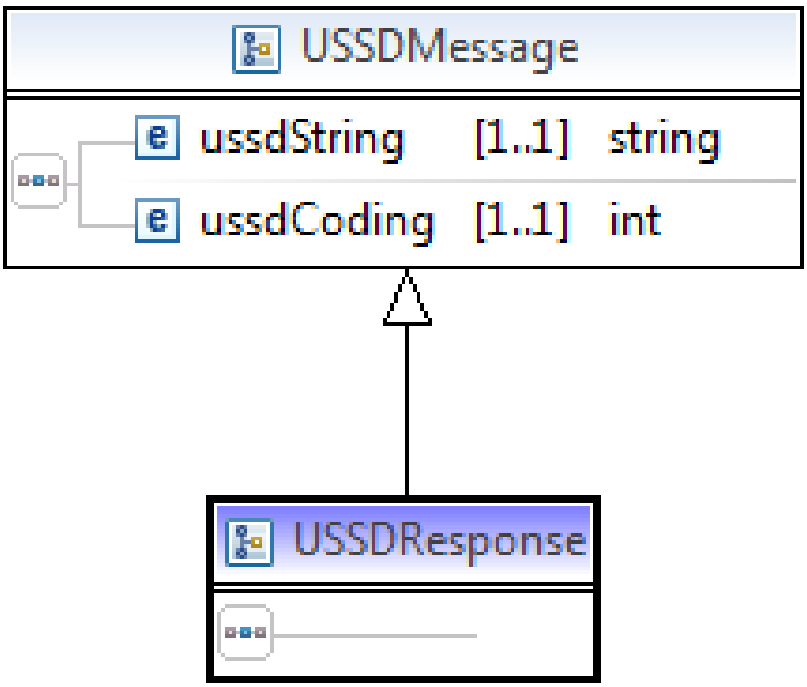
This structure conveys data sent from between peers, for instance: `#123*` or just simply `3` as subsequent request. It has fields to indicate relevant metadata to conveyed string.



USSD Request

USSDResponse

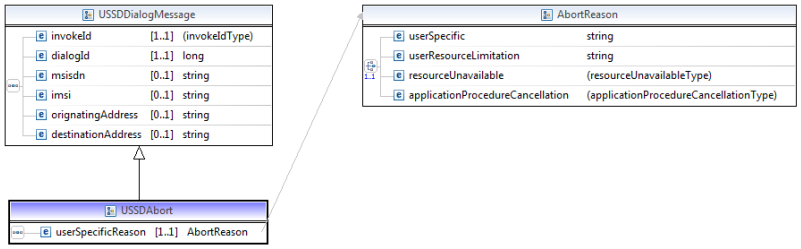
This structure indicates last message exchanged in dialog. It is standard USSD Message, however semantically it ends dialog. Its invoke ID MUST match invoke id from dialog initiating request.



USSD Response

USSDAbort

`Abort` is used to indicate some sort of error. Its content `MUST` provide clue on error origination.



USSD Abort

Error details are encoded within abort reason as one of its values(exlusive):

- userSpecific - simple string explaining application level error
- userResourceLimitation - simple string explaining application level out of resource error
- resourceUnavailable - enumeration type. It can have one of following values:
  - shortTermResourceLimitation
  - longTermResourceLimitation
- applicationProcedureCancellation - enumeration type. It can have one of following values:
  - handoverCancellation
  - radioChannelRelease
  - networkPathRelease
  - callRelease
  - associatedProcedureFailure
  - tandemDialogueRelease
  - remoteOperationsFailure

Library classes are plain JAXB POJOs. They are marshalled to XML form in standard way:

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
```

```
import org.mobicents.ussdgateway.ObjectFactory;
import org.mobicents.ussdgateway.USSDAbort;
import org.mobicents.ussdgateway.USSDRequest;
import org.mobicents.ussdgateway.USSDResponse;



JAXBContext jAXBContext = JAXBContext.newInstance("org.mobicents.ussdgateway");
ObjectFactory objectFactory = new ObjectFactory();


USSDRequest req = this.objectFactory.createUSSDRequest();
req.setInvokeId(1);
req.setUssdCoding(0xFF);
req.setUssdString("#112*");
ByteArrayOutputStream bos = new ByteArrayOutputStream();

JAXBElement<USSDRequest> jxb = this.objectFactory.createRequest(req);
jAXBContext.createMarshaller().marshal(jxb, bos);

String xmlRequest = new String(bos.toByteArray());

Unmarshaller um = this.jAXBContext.createUnmarshaller();
JAXBElement o = (JAXBElement) um
        .unmarshal(new ByteArrayInputStream(xmlRequest));
if (o.getDeclaredType().equals(USSDRequest.class)) {
    //do something
} else if (o.getDeclaredType().equals(USSDResponse.class)) {
    //do something
} else if (o.getDeclaredType().equals(USSDAbort.class)) {
    //do something
}
```

Note

InvokeId has constaraint on its value `<-128,127>`

In SLEE components library is referenced as follows:

```
<library-ref>
    <library-name>library-ussdgateway</library-name>
    <library-vendor>org.mobicents</library-vendor>
    <library-version>2.0</library-version>
</library-ref>
```

Traces and Alarms

5.1. Tracers

USSD Gateway USSD Gateway Application creates following tracers:

Table 5.1. USSD Gateway Application Tracer and Log Categories

Sbb	Tracer name	LOG4J category
ParentSbb	USSD-Parent	javax.slee.SbbNotification[service=ServiceID[name=UssdGatewayService,vendor=org.mobicents,version=1.0],sbb=SbbID[name=ParentSbb,vendor=org.mobicents,version=1.0]].USSD-Parent
SipSbb	USSD-CHILD-SipSbb	javax.slee.SbbNotification[service=ServiceID[name=UssdGatewayService,vendor=org.mobicents,version=1.0],sbb=SbbID[name=SipSbb,vendor=org.mobicents,version=1.0]].USSD-CHILD-SipSbb
HttpClientSbb	USSD-CHILD-HttpClientSbb	javax.slee.SbbNotification[service=ServiceID[name=UssdGatewayService,vendor=org.mobicents,version=1.0],sbb=SbbID[name=HttpClientSbb,vendor=org.mobicents,version=1.0]].USSD-CHILD-HttpClientSbb

Important

Spaces where introduced in `LOG4J category` column values, to correctly render the table. Please remove them when using copy/paste.

5.2. Alarms

Mobicents USSD Gateway Application does not rise any alarms.

Appendix A. Revision History

Revision History

Revision 1.0 Wed June 2 2010 BartoszBaranowski

Creation of the Mobicents JAIN SLEE USSD Gateway Application User Guide.

Index

F