JBoss Communications
TCAP Stack User Guide

by Amit Bhayani, Bartosz Baranowski, and Oleg Kulikov

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts* [https://fedorahosted.org/liberation-fonts/] set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

`Mono-spaced Bold`

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

> To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

> Press **Enter** to execute the command.
>
> Press **Ctrl**+**Alt**+**F1** to switch to the first virtual terminal. Press **Ctrl**+**Alt**+**F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in `Mono-spaced Bold`. For example:

> File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

> Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).
>
> To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

`Mono-spaced Bold Italic` or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

> To connect to a remote machine using ssh, type `ssh` *`username@domain.name`* at a shell prompt. If the remote machine is `example.com` and your username on that machine is john, type `ssh john@example.com`.
>
> The `mount -o remount` *`file-system`* command remounts the named file system. For example, to remount the `/home` file system, the command is `mount -o remount /home`.
>
> To see the version of a currently installed package, use the `rpm -q` *`package`* command. It will return a result as follows: *`package-version-release`*.

Note the words in bold italics above username, domain.name, file-system, package, version and release. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

> When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as

a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules* (*MPMs*). Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in `Mono-spaced Roman` and presented thus:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Source-code listings are also set in `Mono-spaced Roman` but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
  public static void main(String args[])
      throws Exception
  {
    InitialContext iniCtx = new InitialContext();
    Object         ref    = iniCtx.lookup("EchoBean");
    EchoHome       home   = (EchoHome) ref;
    Echo           echo   = home.create();

    System.out.println("Created Echo");

    System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
  }

}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.

Note

A note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.

Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.

Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. Provide feedback to the authors!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in the the *Issue Tracker* [http://bugzilla.redhat.com/bugzilla/], against the product **JBoss Communications TCAP Stack** , or contact the authors.

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

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

Introduction to JBoss Communications TCAP Stack

Important

For better understanding of this chapter please read ITU-T Q.771 to Q.774.

TCAP supports the exchange of non-circuit related data between applications across the SS7 network using the SCCP connectionless service. Queries and responses sent between SSPs and SCPs are carried in TCAP messages. For example, an SSP sends a TCAP query to determine the routing number associated with a dialed 800/888 number and to check the personal identification number (PIN) of a calling card user. In mobile networks (IS-41 and GSM), TCAP carries Mobile Application Part (MAP) messages sent between mobile switches and databases to support user authentication, equipment identification, and roaming.

TCAP messages and parameters are refered as `primitives`. Primitives are exchanged within logical entity called dialog. There are two types of dialogs:

- Unstructured - short lived, it is created only to deliver single message (`Unstructured primitive`), after delivery dialog is invalidated.
- Structured - long lived, it is created when exchange starts and lives until it is terminated.

TCAP defines following message types:

- Unstructured - creates Unstructured dialog. Unstructured dialog is invalidated once this message is either sent or consumed. It is meant for messages which dont require answer or interaction.
- Begin - message sent to begin Structured dialog, it carries context information for dialog being created
- Continue - continues Structured dialog
- End - ends Structured dialog in graceful way
- Abort - aborts Structured dialog

TCAP messages convey two types of information:

- components - elements which carry information on requested operation, their status and result.
- dialog related - dialog context(application context name) and application specific information(user information)

Setup

2.1. Pre-Install Requirements and Prerequisites

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

2.1.1. Hardware Requirements

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

Note

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

2.1.2. Software Prerequisites

The Stack depends on:

- JBoss Communications ASN library
- JBoss Communications Stream library
- JBoss Communications M3UA library
- JBoss Communications MTP library
- JBoss Communications SCCP library

2.2. JBoss Communications TCAP Stack Source Code

2.2.1. Release Source Code Building

1. **Downloading the source code**

Important

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

Use SVN to checkout a specific release source, the base URL is ?, then add the specific release version, lets consider 1.0.0.BETA3.

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

2. **Building the source code**

Important

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

Use Maven to build the binaries.

```
[usr]$ cd tcap-1.0.0.BETA3
[usr]$ mvn install
```

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

2.2.2. Development Trunk Source Building

Similar process as for *Section 2.2.1, "Release Source Code Building"*, the only change is the SVN source code URL, which is NOT AVAILABLE.

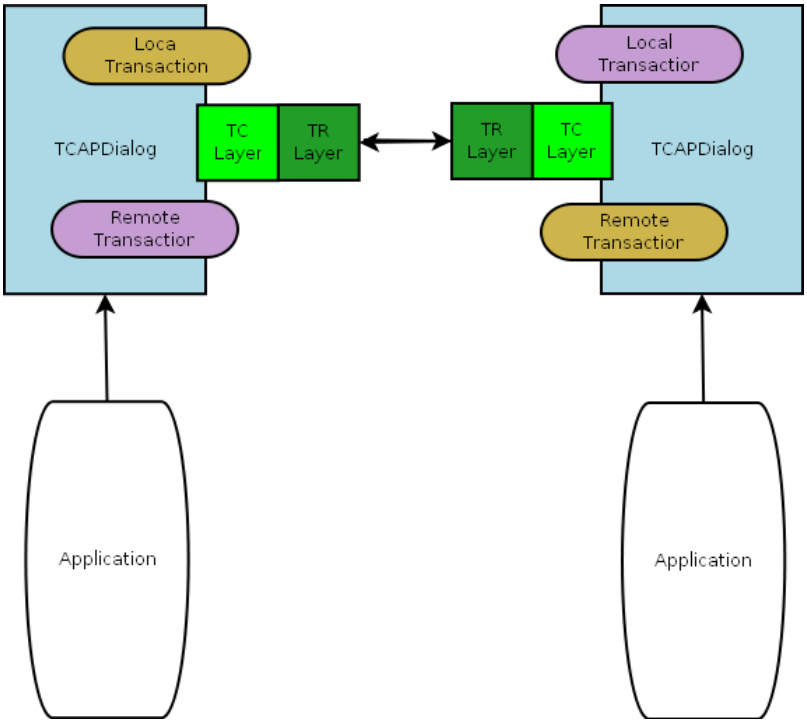
Design Overview

Important

Be aware, JBoss Communications TCAP Stack is subject to changes as it is under active development!

JBoss Communications TCAP Stack TCAP builds abstraction layer over protocol definition. It introduces simplified API in favor of JAIN TCAP

All interactions are handled by TCAP Dialog. Dialog agregates local and remote transactions into single reference. It maintains state associated with both ends and supervises proper encoding and content of exchanged primitives. Diagram below depicts logical structure of TCAP and application using it:



JBoss Communications TCAP Stack TCAP Dialog

Note

TC and TR layers are explained in ITU-T Q.77X specifications. This API does not expose those layers directly.

Protocol

4.1. API

4.1.1. TCAP Dialog overview

`TCAPDialog` is a class representing two TCAP transactions(remote and local transaction form logical dialog). Messages(primitives) are exchanged by means of this class. It is defined by following interface:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.api.TCAPException;
import org.mobicents.protocols.ss7.tcap.api.TCAPSendException;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCBeginRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCContinueRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCEndRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCUniRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCUserAbortRequest;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;
import org.mobicents.protocols.ss7.tcap.asn.comp.Component;


public interface Dialog {

    /**
     * returns this dialog ID. It MUST be unique at any given time in local
     * stack.
     *
     * @return
     */
    public Long getDialogId();

    /**
     * Gets local sccp address
     *
     * @return
     */
    public SccpAddress getLocalAddress();
```

```
/**
 * Gets remote sccp address
 *
 * @return
 */
public SccpAddress getRemoteAddress();

/**
 * Last sent/received ACN
 *
 * @return the acn
 */
public ApplicationContextName getApplicationContextName();

/**
 * Last sent/received UI
 *
 * @return the ui
 */
public UserInformation getUserInformation();

/**
 * returns new, unique for this dialog, invocation id to be used in
 * TC_INVOKE. If there is no free invoke id, it returns null. Invoke ID is
 * freed once operation using it is canceled, timeouts or simply returns
 * final value.
 *
 * @return
 */
public Long getNewInvokeId() throws TCAPException;

/**
 * Cancels INVOKE pending to be sent. It is equivalent to TC-U-CANCEL.
 *
 * @return <ul>
 *         <li><b>true</b> - if operation has been success and invoke id has been
 *          return to pool of available ids.</li>
 *         <li><b>false</b> -</li>
 *         </ul>
 * @throws TCAPException
 *             - thrown if passed invoke id is wrong
 */
public boolean cancelInvocation(Long invokeId) throws TCAPException;
```

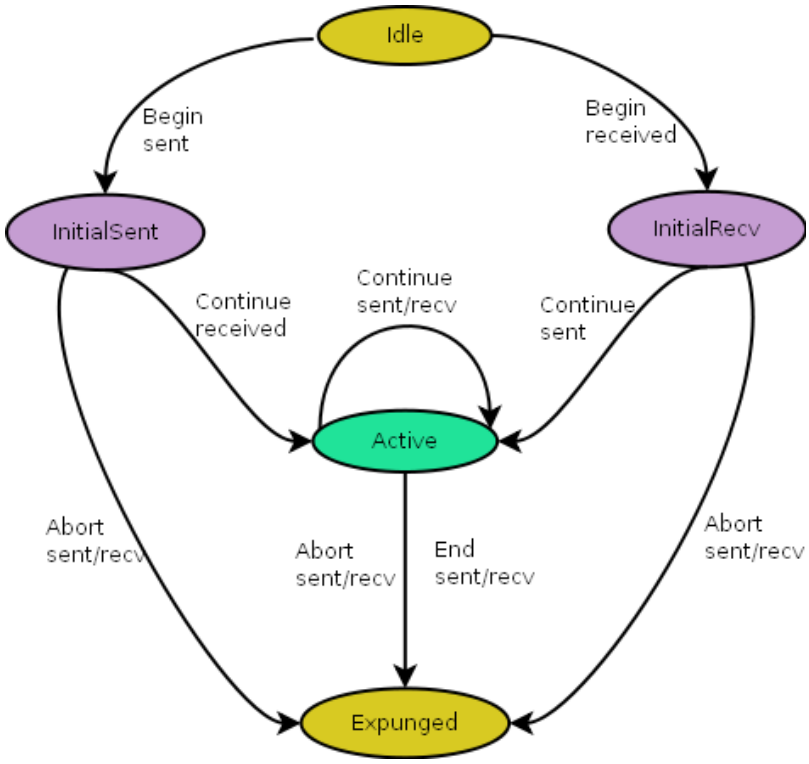
```
/**
 *
 * @return <ul>
 *         <li><b>true </b></li> - if dialog is established(at least one
 *         TC_CONTINUE has been sent/received.)
 *         <li><b>false</b></li> - no TC_CONTINUE sent/received
 *         </ul>
 */
public boolean isEstabilished();

/**
 *
 * @return <ul>
 *         <li><b>true </b></li> - if dialog is structured - its created
 *         with TC_BEGIN not TC_UNI
 *         <li><b>false</b></li> - otherwise
 *         </ul>
 */
public boolean isStructured();

// //////////////////
// Sender methods //
// //////////////////
/**
 * Schedules component for sending. All components on list are queued.
 Components are sent once message primitive is issued.
 *
 * @param componentRequest
 * @throws TCAPSendException
 */
public void sendComponent(Component componentRequest) throws TCAPSendException;
/**
 * Send initial primitive for Structured dialog.
 * @param event
 * @throws TCAPSendException - thrown if dialog is in bad state, ie.
  Being has already been sent or dialog has been removed.
 */
public void send(TCBeginRequest event) throws TCAPSendException;
/**
 * Sends intermediate primitive for Structured dialog.
 * @param event
 * @throws TCAPSendException - thrown if dialog is in bad state, ie.
  Begin has not been sent or dialog has been removed.
 */
```

```
    public void send(TCContinueRequest event) throws TCAPSendException;
    /**
     * Sends dialog end request.
     * @param event
     * @throws TCAPSendException - thrown if dialog is in bad state, ie.
      Begin has not been sent or dialog has been removed.
     */
    public void send(TCEndRequest event) throws TCAPSendException;
    /**
     * Sends Abort primitive with indication to user as source of termination.
     * @param event
     * @throws TCAPSendException
     */
    public void send(TCUserAbortRequest event) throws TCAPSendException;
    /**
     * Sends unstructured dialog primitive. After this method returns dialog
      is expunged from stack as its life cycle reaches end.
     * @param event
     * @throws TCAPSendException
     */
    public void send(TCUniRequest event) throws TCAPSendException;

    /**
     * Programmer hook to release.
     */
    public void release();

    /**
     * Resets timeout timer for particular operation.
     * @param invokeId
     * @throws TCAPException
     */
    public void resetTimer(Long invokeId) throws TCAPException;

    /**
     * Returns the state of this Dialog
     *
     * @return
     */
    public TRPseudoState getState();

}
```

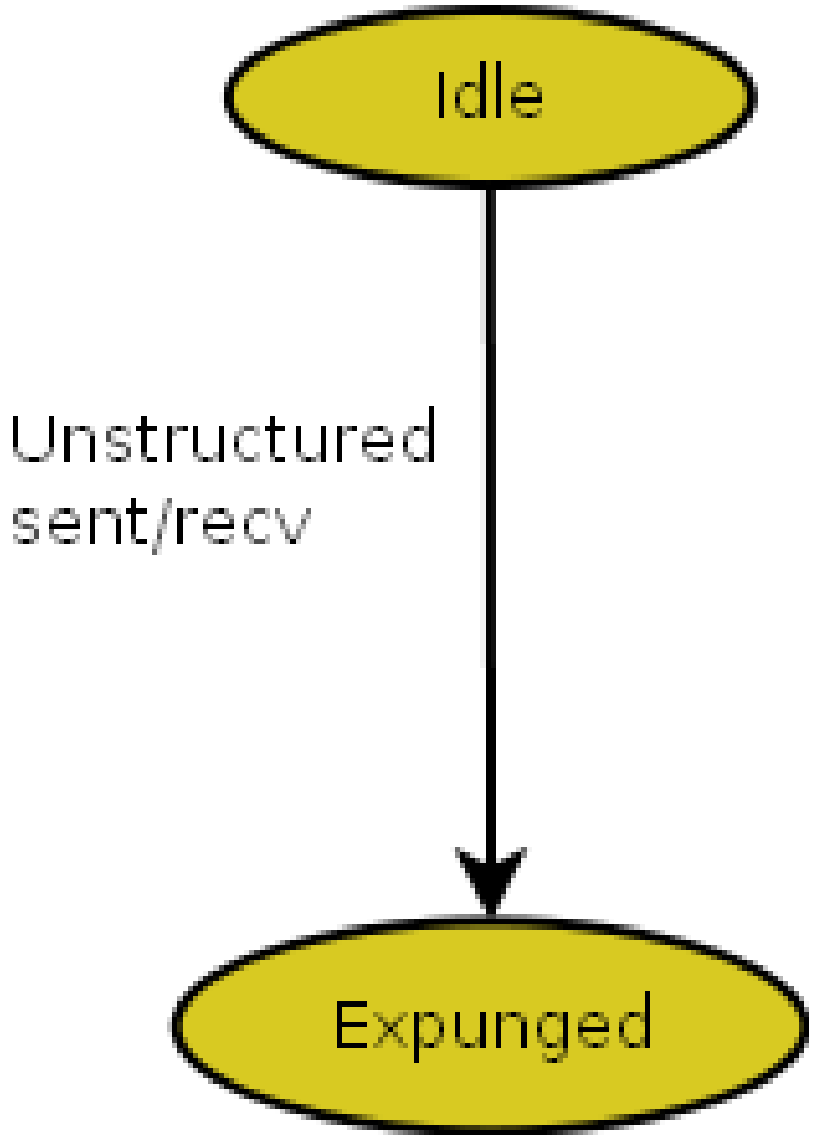
Important

`Application Context` and `User Information` and negotiation process is described Q.771. `Application Context` and `User Information` are user specific and identify capabilities of peer establishing dialog.

Each dialog has associated state machine which it follows. There are two types of dialogs, hence two machines are defined:



TCAP Structured Dialog FSM



TCAP Unstructured Dialog FSM

Each dialog has associated separate invoke id space. This space includes all integer numbers from set of <-128,127>. Invoke id is reference number for operation related components, please refer to *Section 4.1.2, "Operation primitives "* for more details.

4.1.2. Operation primitives

Note

In TCAP operation primitives are also refered as `Components` .

Operation pritmitives are sent as part of messages. Each primitive is scheduled in dialog and sent once user request message to be sent.

There are following primitives defined:

Invoke
: This element indicates that peer request some operation to be invoked. TCAP user populates it with specific to user `OperationCode` and `Parameter` .

```
public interface Invoke extends Component{


    public InvokeClass getInvokeClass();

    /**
     *
     * @param invokeClass
     */
    public void setInvokeClass(InvokeClass invokeClass);

    /**
     * @return the invokeTimeout
     */
    public long getInvokeTimeout();

    /**
     * Sets timeout for this invoke operation in miliseconds. If no indication
     * on operation status is received, before this value passes, operation
     * timesout.
     *
     * @param invokeTimeout
     *            the invokeTimeout to set
     */
    public void setInvokeTimeout(long invokeTimeout);

    // optional
    public void setLinkedId(Long i);

    public Long getLinkedId();

    // mandatory
    public void setOperationCode(OperationCode i);

    public OperationCode getOperationCode();

    // optional
```

```
    public void setParameter(Parameter p);

    public Parameter getParameter();
}

package org.mobicents.protocols.ss7.tcap.api.tc.component;

/**
 * Class of invoke type, ref Q.771 2.3.1.3.
 * <ul>
 * <li>Class 1 # Both success and failure are reported.</li>
 * <li>Class 2 # Only failure is reported.</li>
 * <li>Class 3 # Only success is reported.</li>
 * <li>Class 4 # Neither success, nor failure is reported.</li>
 * <ul>
 *
 */
public enum InvokeClass {

    Class1, Class2, Class3, Class4;
}
```

ReturnResult

This element indicates partial result of `Invoke` . TCAP user populates it with specific to user `OperationCode` and `Parameter` .

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

public interface ReturnResult extends ... {

    //opt all
    public void setOperationCode(OperationCode oc);
    public OperationCode getOperationCode();

    public void setParameter(Parameter p);
    public Parameter getParameter();

}
```

ReturnResultLast

This element indicates final result of `Invoke` . After this element is consumed `InvokeId` associated with its operation is set free. TCAP user populates it with specific to user `OperationCode` and `Parameter` .

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

public interface ReturnResult extends ... {

   //opt all
   public void setOperationCode(OperationCode oc);
   public OperationCode getOperationCode();

   public void setParameter(Parameter p);
   public Parameter getParameter();

}
```

Reject

This element indicates that peer rejected `Invoke` . It contains `Problem` which indiactes cause of rejection.

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

import org.mobicents.protocols.asn.Tag;

public interface Reject extends Component {

   public Problem getProblem();
   public void setProblem(Problem p);

}
```

Listed primitives can be scheduled to send with dialog `sendComponent` method. As such each extends super interface for components:

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

import org.mobicents.protocols.asn.Tag;
import org.mobicents.protocols.ss7.tcap.asn.Encodable;

public interface Component extends Encodable{

   //this is doubled by each interface,
   public void setInvokeId(Long i);
   public Long getInvokeId();

   public ComponentType getType();

}
```

Additionaly TCAP Stack TCAP defines elements to convey user specific information inside operation primitives:

OperationCode
: contains information on user specific operation - identifies this operation by code in user context.

```
import org.mobicents.protocols.asn.Tag;
import org.mobicents.protocols.ss7.tcap.asn.Encodable;

public interface OperationCode extends Encodable{

   public void setOperationType(OperationCodeType t);
   public OperationCodeType getOperationType();

   public void setCode(Long i);
   public Long getCode();

}
```

Parameter

container for user specific (encoded) parameter/s. Its content should be decoded in context to user context and `OperationCode` value.

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

import org.mobicents.protocols.asn.Tag;
import org.mobicents.protocols.ss7.tcap.asn.Encodable;

/**
 * This class embeds Parameter/Parameters. It can be either primitive or
 * constructed. It supports two types of encoding - content as byte[] - either
 * simple or complex - depends on how TC-User sets parameters, or content as
 * Parameter[] - in this case encoding is done as constructed element. It
 * supports following cases:
 * <ul>
 * <li><br>
 * byte[] as primitive -
 *
 * <pre>
 * Parameter p = ....
 *  p.setTag(0x01);
 *  p.setTagClass({@link Tag.CLASS_APPLICATION});
 *  p.setData(new byte[]{ENCODED});
 *  p.setPrimitive(true);
 * </pre>
 *
 * </li>
 * <li><br>
 * byte[] as constructed
 *
 * <pre>
 * Parameter p = ....
 *  p.setTag(0x01);
 *  p.setTagClass({@link Tag.CLASS_APPLICATION});
 *  p.setData(new byte[]{ENCODED});
 *  p.setPrimitive(false);
 * </pre>
 *
 * </li>
 * <li><br>
 * As constructed, with Parameter[] -
 *
```

```
 * <pre>
 * Parameter[] params = ....;
 *  Parameter p = ....
 *  p.setTag(0x01);
 *  p.setTagClass({@link Tag.CLASS_APPLICATION});
 *  //This sets primitive explicitly to false, it must be constructed type.
 *  p.setParameters(params);
 * </pre>
 *
 * </li>
 * </ul>
 * Note that on read only byte[] is filled! In case TC-USER makes call to
 * {@link #getParameters()} method - it triggers parsing to array, so it is
 * perfectly legal to obtain byte[], rather than Parameter[].
 *
 */
public interface Parameter extends Encodable {


    public byte[] getData();
    /**
     * Sets content as raw byte[]. invocation parameter must be encoded by user.
     * @param b
     */
    public void setData(byte[] b);

    /**
     * Determine if this parameter is of primitive type - not constructed.
     * @return
     */
    public boolean isPrimitive();

    public void setPrimitive(boolean b);

    /**
     * @return the tag
     */
    public int getTag();

    /**
     * @param tag
     *            the tag to set
     */
    public void setTag(int tag);
```

```
    /**
     * @return the tagClass
     */
    public int getTagClass();

    /**
     * @param tagClass
     *            the tagClass to set
     */
    public void setTagClass(int tagClass);
    /**
     * Return decoded raw byte[] data.
     * @return
     */
    public Parameter[] getParameters();
    /**
     * Sets Parameter[]. Automaticly marks this object as constructed - {@link #isPrimitive()} will return false.
     * @param paramss
     */
    public void setParameters(Parameter[] paramss);

}
```

Problem

contains information on problem which occurred. It is part of `Reject` component.

```
package org.mobicents.protocols.ss7.tcap.asn.comp;

import org.mobicents.protocols.ss7.tcap.asn.Encodable;

public interface Problem extends Encodable {


    public ProblemType getType();
    public void setType(ProblemType t);

    public void setGeneralProblemType(GeneralProblemType t);
    public GeneralProblemType getGeneralProblemType();
```

```
    public void setInvokeProblemType(InvokeProblemType t);
    public InvokeProblemType getInvokeProblemType();

    public void setReturnErrorProblemType(ReturnErrorProblemType t);
    public ReturnErrorProblemType getReturnErrorProblemType();

    public void setReturnResultProblemType(ReturnResultProblemType t);
    public ReturnResultProblemType getReturnResultProblemType();


}
```

Operation primitives are created by factory defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api;


import org.mobicents.protocols.ss7.tcap.asn.comp.Invoke;
import org.mobicents.protocols.ss7.tcap.asn.comp.OperationCode;
import org.mobicents.protocols.ss7.tcap.asn.comp.Parameter;
import org.mobicents.protocols.ss7.tcap.asn.comp.Problem;
import org.mobicents.protocols.ss7.tcap.asn.comp.ProblemType;
import org.mobicents.protocols.ss7.tcap.asn.comp.Reject;
import org.mobicents.protocols.ss7.tcap.asn.comp.ReturnResult;
import org.mobicents.protocols.ss7.tcap.asn.comp.ReturnResultLast;

public interface ComponentPrimitiveFactory {

    public Invoke createTCInvokeRequest();

    public Reject createTCRejectRequest();

    public ReturnResultLast createTCResultLastRequest();

    public ReturnResult createTCResultRequest();

    public OperationCode createOperationCode(boolean isGlobal, Long code);

    public Parameter createParameter();
```

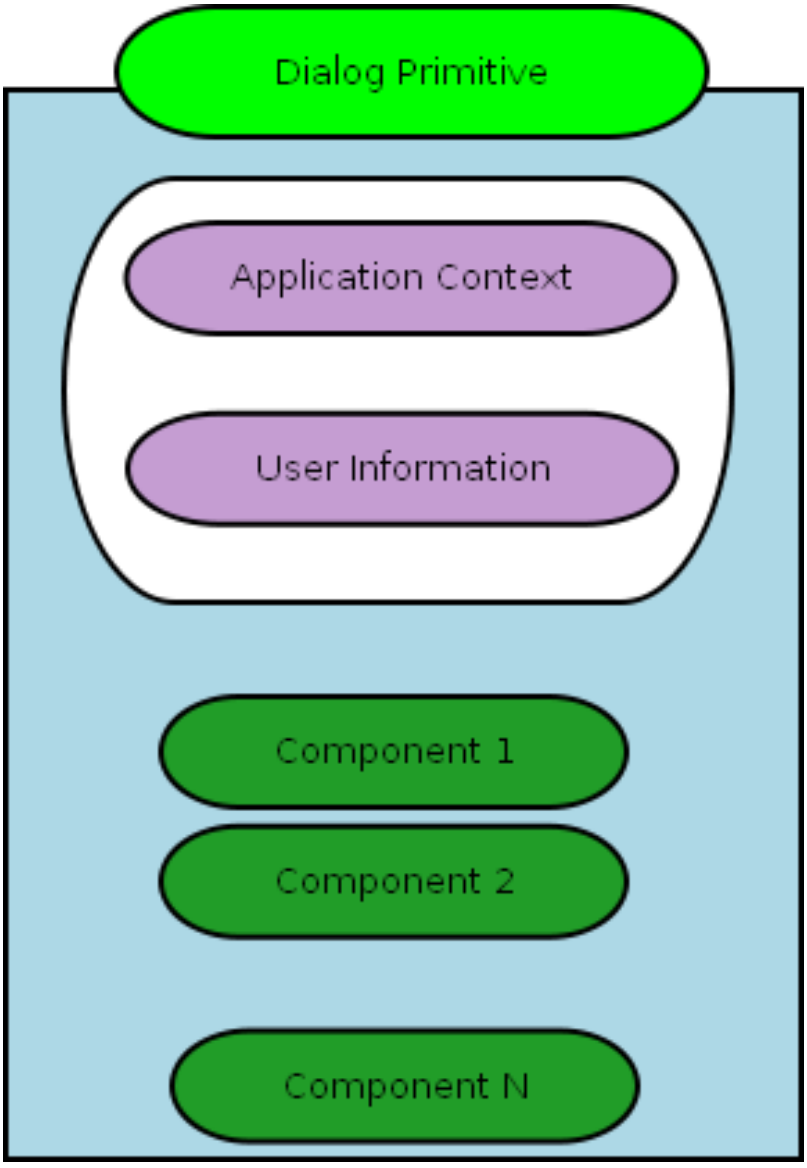
```
    public Parameter createParameter(int tag, int tagClass, boolean isPrimitive);

    public Problem createProblem(ProblemType pt);
}
```

4.1.3. Dialog primitives

Dialog primitives are in other words messages exchanged. Primitives are passed to dialog object which forms proper message from following:

- primitive and information passed in it
- components scheduled in dialog

Message formed by dialog, can be logicaly depicted as follows:



TCAP message structure

Each component eligible for passing to dialog is a child of following interface:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog;


public interface DialogRequest {
   /**
    * Return dialog for this indication
    * @return
    */
   public Dialog getDialog();
   /**
    * Determine type: Begin, Continue...
    */
   public EventType getType();

}
```

Each component eligible for passing as indication to listener is a child of following interface:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog;
import org.mobicents.protocols.ss7.tcap.asn.comp.Component;

public interface DialogIndication {


   /**
    * Return dialog for this indication
    * @return
    */
   public Dialog getDialog();
   /**
    * get components if present, if there are none, it will return null;
    * @return
```

```
     */
    public Component[] getComponents();

    public EventType getType();

    public Byte getQos();
}
```

TCAP defines primitives :

TCBegin

Exchanged as initial message in structured dialog. Starts negotiation of `Application Context Name` and `User Information` . Its interface is defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCBeginRequest extends DialogRequest {
    /**
     * Sets QOS optional parameter. Its passed to lower layer
     * @param b
     */
    public void setQOS(Byte b) throws IllegalArgumentException;
    public Byte getQOS();

    //only getter, since we send via Dialog object, ID is ensured to be present.


    /**
     * Destination address. If this address is different than one
     * in dialog, this value will overwrite dialog value.
     */
    public SccpAddress getDestinationAddress();
    public void setDestinationAddress(SccpAddress dest);
    /**
     * Origin address. If this address is different than one in dialog,
     * this value will overwrite dialog value.
     */
```

```
    public SccpAddress getOriginatingAddress();
    public void setOriginatingAddress(SccpAddress dest);

    /**
     * Application context name for this dialog.
     * @return
     */
    public ApplicationContextName getApplicationContextName();
    public void setApplicationContextName(ApplicationContextName acn);
    /**
     * User information for this dialog.
     * @return
     */
    public UserInformation getUserInformation();
    public void setUserInformation(UserInformation acn);


}
```

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCBeginIndication extends DialogIndication {

    public Byte getQOS();

    public ApplicationContextName getApplicationContextName();

    public UserInformation getUserInformation();

    public SccpAddress getDestinationAddress();

    public SccpAddress getOriginatingAddress();

}
```

TCContinue

Exchanged as intermediate message until dialog reaches its end. Its interface defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCContinueRequest extends DialogRequest {

    /**
     * Sets QOS optional parameter. Its passed to lower layer
     *
     * @param b
     */
    public void setQOS(Byte b) throws IllegalArgumentException;

    public Byte getQOS();

    /**
     * Sets origin address. This parameter is used only in first TCContinue,
     * sent as response to TCBegin. This parameter, if set, changes local peer
     * address(remote end will send request to value set by this method).
     *
     * @return
     */
    public SccpAddress getOriginatingAddress();

    public void setOriginatingAddress(SccpAddress dest);

    /**
     * Application context name for this dialog.
     *
     * @return
     */
    public ApplicationContextName getApplicationContextName();
```

```
    public void setApplicationContextName(ApplicationContextName acn);

    /**
     * User information for this dialog.
     *
     * @return
     */
    public UserInformation getUserInformation();

    public void setUserInformation(UserInformation acn);

}
```

```
import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCContinueIndication extends DialogIndication {

    public Byte getQOS();

    public ApplicationContextName getApplicationContextName();

    public UserInformation getUserInformation();

    public SccpAddress getOriginatingAddress();
}
```

TCEnd

Sent to terminate dialog(and two transactions associated with it). Its interface is defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;
```

```
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;


public interface TCEndRequest extends DialogRequest {
    /**
     * Sets QOS optional parameter. Its passed to SCCP layer?
     *
     * @param b
     */
    public void setQOS(Byte b) throws IllegalArgumentException;

    public Byte getQOS();

    /**
     * Application context name for this dialog.
     *
     * @return
     */
    public ApplicationContextName getApplicationContextName();

    public void setApplicationContextName(ApplicationContextName acn);

    /**
     * User information for this dialog.
     *
     * @return
     */
    public UserInformation getUserInformation();

    public void setUserInformation(UserInformation acn);

    /**
     * Type of termination. See values of
     * {@link org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TerminationType
     * TerminationType} enum.
     *
     * @param t
     */
    public void setTermination(TerminationType t);

    public TerminationType getTerminationType();
}
```

```
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCEndIndication extends DialogIndication {

    public Byte getQOS();

    // parts from DialogPortion, if present
    public ApplicationContextName getApplicationContextName();

    public UserInformation getUserInformation();

}
```

TCUserAbort

Sent to abort dialog beeing created or ongoign one. Its interface is defined as follows:

```
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

/**
 * <pre>
 * -- NOTE # When the Abort Message is generated
 *      by the Transaction sublayer, a p-Abort Cause must be
 * -- present.The u-abortCause may be generated by the component sublayer
 *      in which case it is an ABRT
 * -- APDU, or by the TC-User in which case it could be either an ABRT
 *      APDU or data in some user-defined
 * -- abstract syntax.
 * </pre>
 *
 */
public interface TCUserAbortRequest extends DialogRequest {

    public void setQOS(Byte b) throws IllegalArgumentException;
```

```
    public Byte getQOS();

    public ApplicationContextName getApplicationContextName();

    public void setApplicationContextName(ApplicationContextName acn);

    public UserInformation getUserInformation();

    public void setUserInformation(UserInformation acn);

}
```

```
import org.mobicents.protocols.ss7.tcap.asn.AbortSource;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

/**
 * <pre>
 * -- NOTE # When the Abort Message is generated by the
 *         Transaction sublayer, a p-Abort Cause must be
 * -- present.The u-abortCause may be generated by the
 *      component sublayer in which case it is an ABRT
 * -- APDU, or by the TC-User in which case it could be
 *      either an ABRT APDU or data in some user-defined
 * -- abstract syntax.
 * </pre>
 *
 */
public interface TCUserAbortIndication extends DialogIndication {


    public UserInformation getUserInformation();

    public AbortSource getAbortSource();

}
```

TCPAbort

Sent to abort dialog beeing created or ongoign one. It is sent by provider, it indicates problem on transport or stack layer. Its interface is defined as follows:

```
import org.mobicents.protocols.ss7.tcap.asn.comp.PAbortCauseType;

/**
 * <pre>
 * -- NOTE # When the Abort Message is generated by the
 *         Transaction sublayer, a p-Abort Cause must be
 * -- present.The u-abortCause may be generated by the
 *         component sublayer in which case it is an ABRT
 * -- APDU, or by the TC-User in which case it could be
 *      either an ABRT APDU or data in some user-defined
 * -- abstract syntax.
 * </pre>
 *
 */
public interface TCPAbortIndication extends DialogIndication{

    //mandatory
    public PAbortCauseType getPAbortCause();
}
```

TCUni

Only message available for unstructured dialog. Its interface is defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api.tc.dialog.events;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

public interface TCUniRequest extends DialogRequest {

    /**
     * Sets QOS optional parameter. Its passed to lower layer
```

```
     *
     * @param b
     */
    public void setQOS(Byte b) throws IllegalArgumentException;

    public Byte getQOS();

    /**
     * Destination address. If this address is different than one in dialog,
     * this value will overwrite dialog value.
     */
    public SccpAddress getDestinationAddress();

    public void setDestinationAddress(SccpAddress dest);

    /**
     * Origin address. If this address is different than one in dialog, this
     * value will overwrite dialog value.
     */
    public SccpAddress getOriginatingAddress();

    public void setOriginatingAddress(SccpAddress dest);

    /**
     * Application context name for this dialog.
     *
     * @return
     */
    public ApplicationContextName getApplicationContextName();

    public void setApplicationContextName(ApplicationContextName acn);

    /**
     * User information for this dialog.
     *
     * @return
     */
    public UserInformation getUserInformation();

    public void setUserInformation(UserInformation acn);

}
```

```
import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;

/**
 * @author baranowb
 *
 */
public interface TCUniIndication extends DialogIndication {

    public Byte getQOS();

    // parts from DialogPortion, if present
    public ApplicationContextName getApplicationContextName();

    public UserInformation getUserInformation();

    public SccpAddress getDestinationAddress();

    public SccpAddress getOriginatingAddress();
}
```

Dialog primitives are created(as components) by means of factory. Factory is defined in following way:

```
package org.mobicents.protocols.ss7.tcap.api;

import org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCBeginRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCContinueRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCEndRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCUniRequest;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.TCUserAbortRequest;
import org.mobicents.protocols.ss7.tcap.asn.ApplicationContextName;
import org.mobicents.protocols.ss7.tcap.asn.UserInformation;
```

```
public interface DialogPrimitiveFactory {

    public TCBeginRequest createBegin(Dialog d);

    public TCContinueRequest createContinue(Dialog d);

    public TCEndRequest createEnd(Dialog d);

    public TCUserAbortRequest createUAbort(Dialog d);

    public TCUniRequest createUni(Dialog d);

    public ApplicationContextName createApplicationContextName(long[] oid);

    public UserInformation createUserInformation();

}
```

4.1.4. Stack

TCAP is part of SS7 protocol stack. AS such it relies on SCCP as means of transport. To create TCAP stack you require properly configured SCCP layer. Please refer to *Section 4.2, "Configuration"* for details.

JBoss Communications TCAP is defined by provider and `stack` interfaces. Interfaces are defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api;

public interface TCAPStack {

    /**
     * Returns stack provider.
     * @return
     */
    public TCAPProvider getProvider();
    /**
     * Stops this stack and transport layer(SCCP)
     */
    public void stop();
    /**
```

```
     * Start stack and transport layer(SCCP)
     * @throws IllegalStateException - if stack is already running or not configured
     * @throws StartFailedException
     */
    public void start() throws IllegalStateException, StartFailedException;

}
```

```
package org.mobicents.protocols.ss7.tcap.api;

import org.mobicents.protocols.ss7.sccp.parameter.SccpAddress;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog;

public interface TCAPProvider {

    /**
     * Create new structured dialog.
     * @param localAddress - desired local address
     * @param remoteAddress - initial remote address, it can change after first TCContinue.
     * @return
     */
    public Dialog getNewDialog(SccpAddress localAddress, SccpAddress remoteAddress) throws TCAPException;
    /**
     * Create new unstructured dialog.
     * @param localAddress
     * @param remoteAddress
     * @return
     * @throws TCAPException
     */
    public Dialog getNewUnstructuredDialog(SccpAddress localAddress, SccpAddress remoteAddress) throws TCA

    ///////////////
    // Factories //
    ///////////////

    public DialogPrimitiveFactory getDialogPrimitiveFactory();
    public ComponentPrimitiveFactory getComponentPrimitiveFactory();

    ///////////////
    // Listeners //
```

```
    ///////////////

    public void addTCListener(TCListener lst);

    public void removeTCListener(TCListener lst);
}
```

Provider allows user to access stack facitlities, create dialogs and register as listener for incoming messages. Listener declares set of callbacks methods. It is defined as follows:

```
package org.mobicents.protocols.ss7.tcap.api;

import org.mobicents.protocols.ss7.tcap.api.tc.dialog.Dialog;
import org.mobicents.protocols.ss7.tcap.api.tc.dialog.events.*;
import org.mobicents.protocols.ss7.tcap.asn.comp.Invoke;

public interface TCListener {

    // dialog handlers
    /**
     * Invoked for TC_UNI. See Q.771 3.1.2.2.2.1
     */
    public void onTCUni(TCUniIndication ind);

    /**
     * Invoked for TC_BEGIN. See Q.771 3.1.2.2.2.1
     */
    public void onTCBegin(TCBeginIndication ind);

    /**
     * Invoked for TC_CONTINUE dialog primitive. See Q.771
     * 3.1.2.2.2.2/3.1.2.2.2.3
     *
     * @param ind
     */
    public void onTCContinue(TCContinueIndication ind);

    /**
     * Invoked for TC_END dialog primitive. See Q.771 3.1.2.2.2.4
     *
     * @param ind
```

```
     */
    public void onTCEnd(TCEndIndication ind);

    /**
     * Invoked for TC-U-Abort primitive(P-Abort-Cause is present.). See Q.771
     * 3.1.2.2.2.4
     *
     * @param ind
     */
    public void onTCUserAbort(TCUserAbortIndication ind);

    /**
     * Invoked when dialog has been terminated by some unpredicatable
     * environment cause. See Q.771 3.1.4.2
     *
     * @param ind
     */
    public void onTCPAbort(TCPAbortIndication ind);

    /**
     * Called once dialog is released. It is invoked once primitives are
     * delivered. Indicates that stack has no reference, and dialog object is
     * considered invalid.
     *
     * @param d
     */
    public void dialogReleased(Dialog d);

    /**
     *
     * @param tcInvokeRequest
     */
    public void onInvokeTimeout(Invoke tcInvokeRequest);

}
```

4.2. Configuration

TCAP layer does not require any config. However it requires properly set SCCP and its sublayers. Please refer to `SCCP User Guide` for supported configuration options.

4.2.1. Dependencies

TCAP depends on following:

- MTP
- SCCP

4.3. Example

```
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);
      this.runningTestCase = runningTestCase;
      this.thisAddress = thisAddress;
      this.remoteAddress = remoteAddress;
      this.tcapProvider = this.stack.getProvider();
      this.tcapProvider.addTCListener(this);
   }

   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());
```

```
        invoke.setOperationCode(cpFactory.createOperationCode(true,new Long(12)));
        //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 dialogReleased(Dialog d) {


    }

    public void onInvokeTimeout(Invoke tcInvokeRequest) {

    }

    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

    }


}
```

Appendix A. Revision History

Revision History

Revision 1.0 Wed June 2 2010 BartoszBaranowski

Creation of the JBoss Communications TCAP Stack User Guide.

Revision 1.1 Thu Oct 7 2010 BartoszBaranowski

Creation of the JBoss Communications TCAP Stack User Guide.

Index

F