JBoss Communications Diameter User Guide

by Bartosz Baranowski and Alexandre Mendonca

Preface v
1. Document Conventions v
1.1. Typographic Conventions v
1.2. Pull-quote Conventions vii
1.3. Notes and Warnings vii
2. Provide feedback to the authors! viii
1. Introduction to JBoss Communications Diameter 1
1.1. Message Format 1
1.2. Contents 4
2. JBoss Communications Diameter Stack 7
2.1. JBoss Communications Diameter Stack Design 7
2.1.1. JBoss Communications Diameter Stack Extensibility 7
2.1.2. JBoss Communications Diameter Stack Model 8
2.1.3. Application session factories 9
2.1.4. Session Replication 10
2.2. JBoss Communications Diameter Stack Setup 11
2.2.1. Pre-Install Requirements and Prerequisites 11
2.2.2. Source Code 11
2.3. JBoss Communications Diameter Stack Configuration 12
2.3.1. Cluster configuration 21
2.4. JBoss Communications Diameter Stack Source overview 23
2.4.1. Session Factory 24
2.4.2. Sessions 27
2.4.3. Application Session Factories 29
2.5. JBoss Communications Diameter Stack Validator 30
2.5.1. Validator Configuration 31
2.5.2. Validator Source Overview 38
3. JBoss Communications Diameter Multiplexer 41
3.1. JBoss Communications Diameter Multiplexer (MUX) Design 41
3.2. JBoss Communications Diameter Multiplexer (MUX) Setup 42
3.2.1. Pre-Install Requirements and Prerequisites 42
3.2.2. Source Code 43
3.3. JBoss Communications Diameter Multiplexer (MUX) Configuration 44
3.4. JBoss Communications Diameter Multiplexer (MUX) Source Overview 44
3.5. JBoss Communications Diameter Multiplexer (MUX) Dictionary 46
A. Revision History 51
Index 53

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 JAIN SLEE Example**, or contact the authors.

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

Diameter is a computer networking protocol for AAA (Authentication, Authorization and Accounting) defined in RFC 3588. It is a successor to RADIUS (and its name is a reference to it, a diameter is twice the radius). Diameter has been designed to overcome certain RADIUS limitations:

- No transport reliability and flexibility (Diameter uses TCP/SCTP instead of UDP)
- No security within protocol (Diameter supports IPSec (mandatory) and TLS (optional))
- Limited address space for AVPs (Diameter uses 32-bit address space instead of 8-bit)
- Only stateless mode is possible (Diameter supports both stateful and stateless modes)
- Static peers (Diameter offers dynamic discovery, using DNS SRV and NAPTR)
- No peer alignment capabilities (Diameter introduce capabilities negotiation)
- No support for transport layer failover (Diameter follows *RFC3539* [http://tools.ietf.org/html/rfc3539], which introduces proper procedures)
- Limited support for roaming (Diameter introduces mechanisms for secure and scalable roaming)
- No extension possible (Diameter allows extension - new commands and avps to be defined)

Diameter offers all capabilities of RADIUS protocol. Also as a logical successor, it is comptibile with RADIUS.

Diameter also allows to define extensions. Each extension is called "Application".

Each application may introduce new types of messages, AVP codes, and state machines. The Message and AVP codes are assigned by the IANA. Furthermore, each application has its own Application ID and Vendor ID that is used to distinguish between applications. In addition, application code is used to signal to other peers which operations are supported by connecting peer (Capabilities Exchange / Negotiation).

1.1. Message Format

Diameter is a byte based protocol. Each message has a fixed structure, which consists of two parts: header and payload.

Message header structure is common for all messages, it has a fixed length and content. It identifies message (by code, application and certain bit flags) in Diameter scope.

Payload is the message part built of carried AVPs. Its content differs for each command and application (however ALL define Session-Id AVP as mandatory).

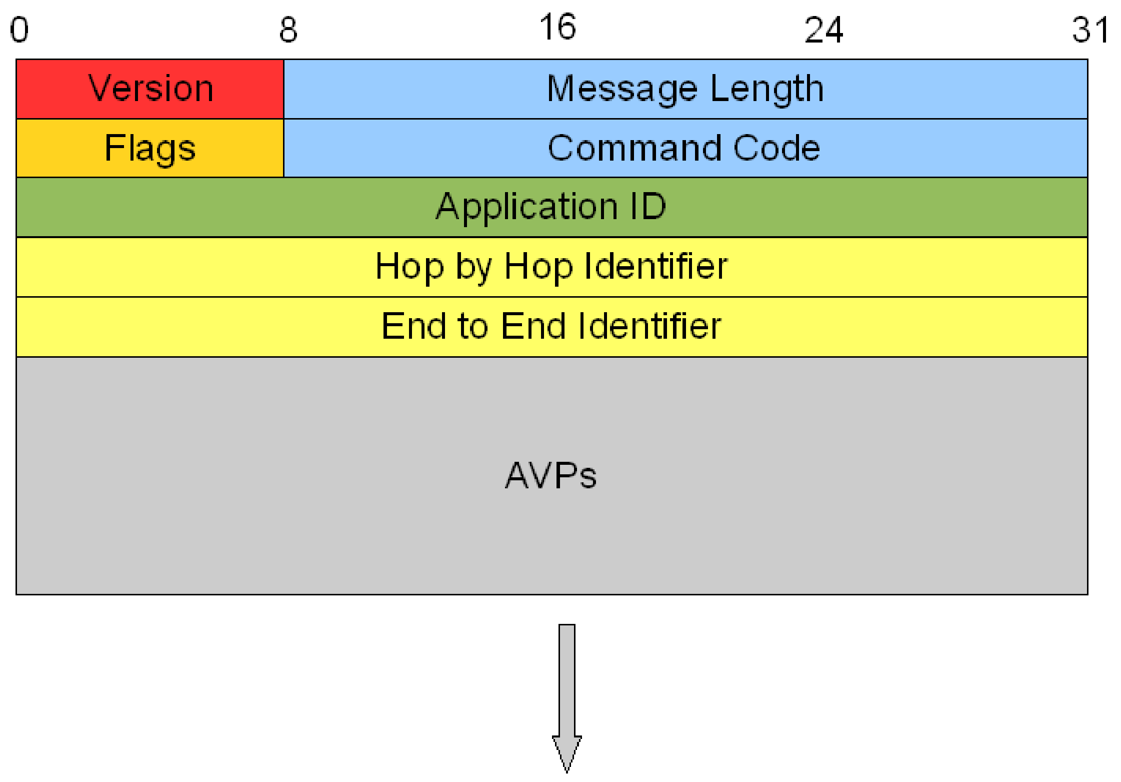

Figure 1.1. Diameter Message Structure

Header has following fields:

Message Headers

Version
: Indicates the Diameter protocol version. This value is always set to `1`.

Message Length
: Indicates the Diameter message length, including the header fields.

Flags
: Composed by eight bits, to provide information regarding message. First four bits in Flags octet have following meaning:

  - R = Message is a request `(1)` or an answer `(0)`;
  - P = Message is proxiable `(1)` and may be proxied, relayed or redirected, otherwise it must be processed locally `(0)`;
  - E = Message is an error message `(1)` or a regular message`(0)`;

- T = Message is being potentially re-transmitted `(1)` or being sent for the first time `(0)`

The last four bits are reserved for future use, and should be set to `0`.

Command Code
: Indicates the command associated with the message.

Application-ID
: Identifies application to which the message is applicable for. The application can be an authentication, accounting, or vendor specific application. The application-id in the header must be the same as what is contained in any relevant AVPs contained in the message.

Hop-by-Hop ID
: Unique ID that is used to match requests and answer. The Answer message must ensure this header field contains the same value present in the corresponding request. This is how answers are routed back to peer which sent message.

End-to-End ID
: Time-limited unique ID that is used to to detect duplicate messages. The ID must be unique for at least four minutes. The Answer message originator must ensure that this header contains the same value present in the corresponding request.

Message payload is built up from AVPs. Each AVP has similar structure: header and encoded data. Data can be simple (eg, integer, long) or complex (another encoded AVP).

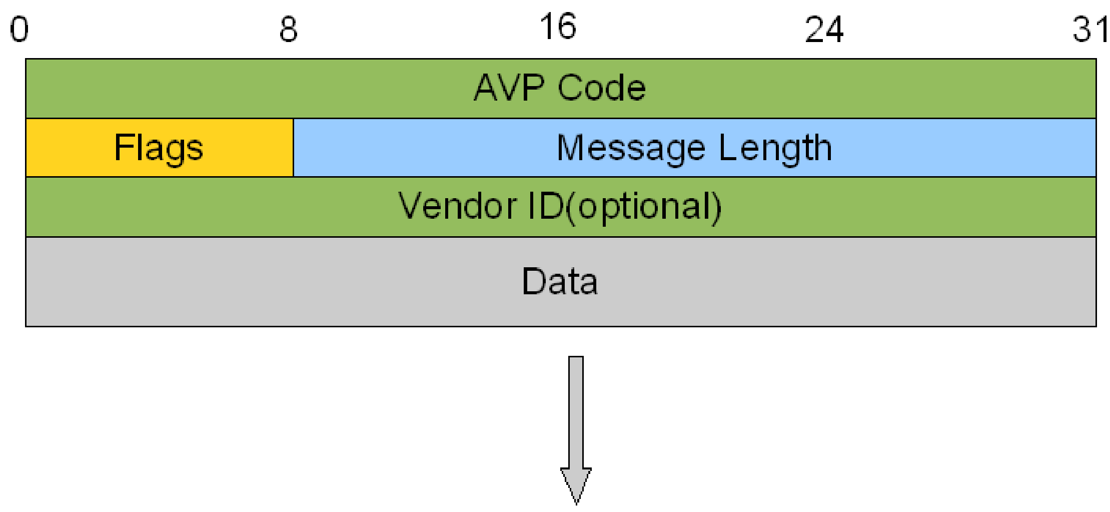

Figure 1.2. Payload Structure

Payload AVPs

AVP Code
: Uniquely identifies the attribute, by combining the specified code with the value contained within the Vendor-ID header field.

AVP numbers 1 to 255 are reserved for RADIUS backwards compatibility, and do not require the Vendor-ID header field. AVP numbers 256 and above are used exclusively for the Diameter protocol, and are allocated by IANA.

Flags

Bit flags, which specify how each attribute must be handled. Flags octet has following structure: V M P r r r r r.

Full description can be found in *Section 4.1 of RFC3588* [http://tools.ietf.org/html/rfc3588#section-4.1].

First three bits have the following meaning:

- V = If set, indicates that optional octets (Vendor-ID) is present in AVP header.
- M = If set, it indicates that receiveing peer must understand this AVP or send error answer.
- P = If set, it indicates the need for encryption for end-to-end security.

The last 5 bits are reserved for future use, and should be set to `0`.

AVP Length

Indicates the number of octets in the AVP, including the following information:

- AVP Code
- AVP Length
- AVP Flags
- Vendor-ID field (if present)
- AVP Data

Vendor-ID

Optional octet identifying AVP in application space. AVP code and AVP Vendor-ID create unique identifier for AVP.

1.2. Contents

JBoss Communications Diameter core is built on top of three basic components:

Stack

Extensible Diameter stack. It provides basic session support along with application specific sessions.

Multiplexer (MUX)

Diameter Stack multiplexer. Allows different listeners to share the same stack instance.

Dictionary

Diameter Message and AVP dictionary. Provides an API to access information about AVP: Flags, Type, assigned values, etc. Dictionary is embeded in the MUX.

JBoss Communications Diameter Stack

The JBoss Communications Diameter Stack is the core component of the presented Diameter solution. It performs all necessary tasks to allow user interact with Diameter network. It manages state of diameter peers and allows to route messages between them. For details please refer to *RFC 3588* [http://tools.ietf.org/html/rfc3588].

Currently the JBoss Communications Diameter Stack supports the following application sessions:

- Base
- Credit Control Application (CCA)
- Sh
- Ro
- Rf
- Cx/Dx

2.1. JBoss Communications Diameter Stack Design

2.1.1. JBoss Communications Diameter Stack Extensibility

JBoss Communications Diameter Stack has been designed to be extensible. In order to achieve that, two set of API are defined by the stack: one which defines basic contracts between user application and stack, second which defines contracts allowing for instance to inject custom objects into stack to perform certain tasks(`SessionFactory` for instance). `ISessionFactory` declares additional methods which allow developer to declare custom behaviour (custom application sessions, please refer to *Section 2.4.1, “Session Factory”* for more detailed information).

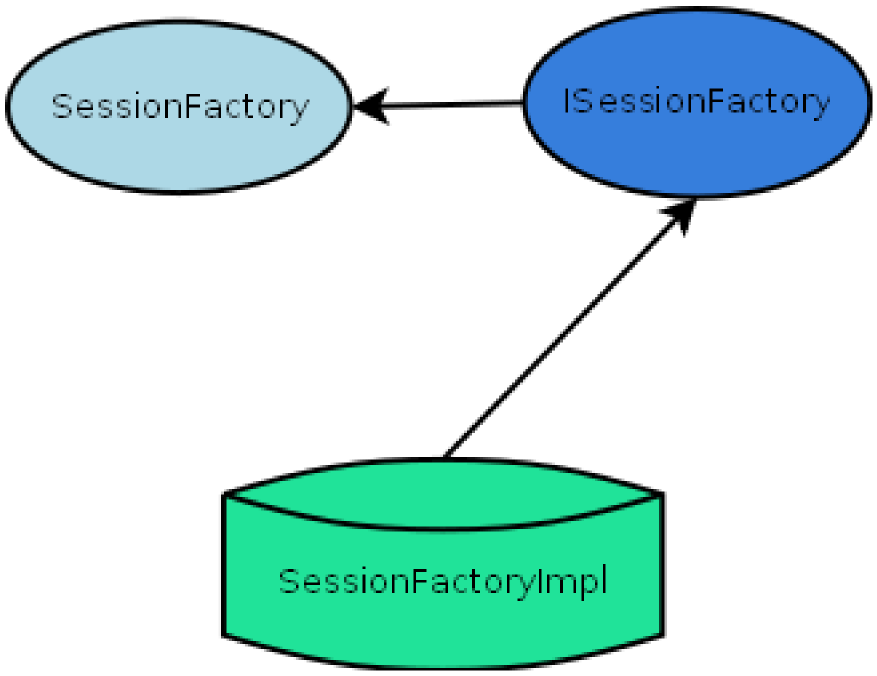

Figure 2.1. JBoss Communications Diameter Stack Extensibility Visualization

General pattern for interface declaration can be understood as follows: Interface `ComponentInterface` declares minimal set of methods for component to perform its task, and Interface `IComponentInterface` provides additional behavior methods. Please refer to java doc for list of interfaces and description of method contracts.

2.1.2. JBoss Communications Diameter Stack Model

JBoss Communications Diameter Stack performs the following tasks:

- Manage connections to remote peers
- Manage sessions objects
- Route messages on behalf of sessions
- Receive and deliver messages to assigned listeners (usually session object)

Sessions use stack and services it provides to communicate with remote peers. Application is only place which holds reference to sessions. It can be seen as follows:

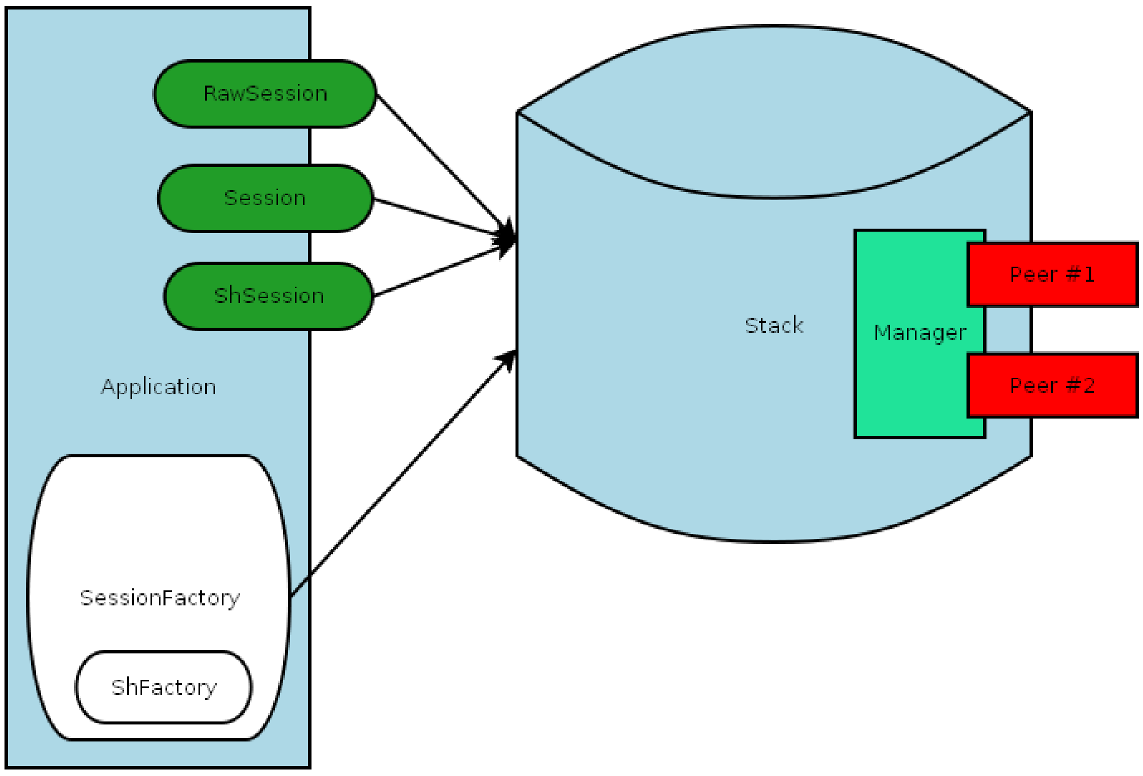

Figure 2.2. JBoss Communications Diameter Application and Stack Model

2.1.3. Application session factories

Application session factories perform two tasks:

- server stack as factory for sessions
- server session objects as holders for session related resources, like state change listener, event listeners, context

Generally application session factory and user application relation can be imagined as follows:

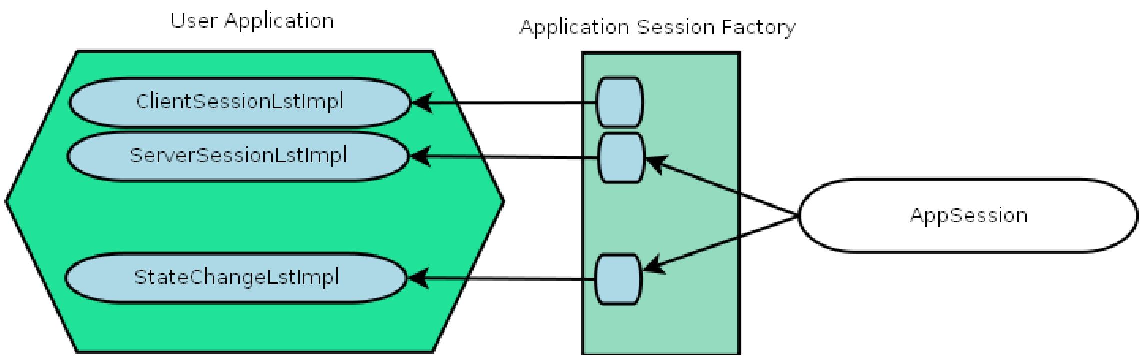

Figure 2.3. JBoss Communications Application session factory and user application

Note

Session context is callback interface defined by some sessions.

2.1.4. Session Replication

JBoss Communications Diameter Stack supports replication of session data and state. Clustered stack instances can perform operations on session regardless of physical location. Logicaly clustered stack can be imagined as follows:

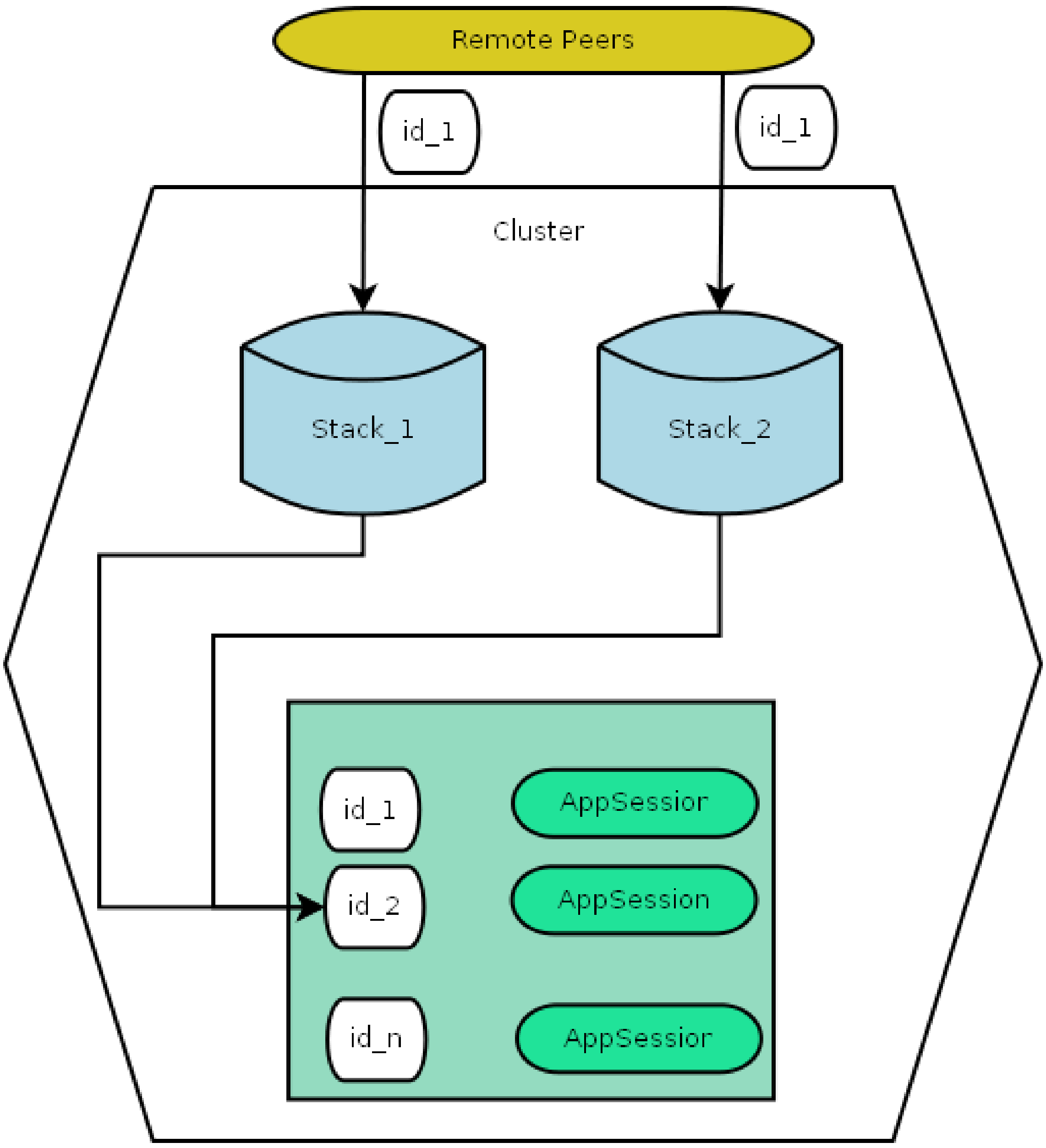

Figure 2.4. JBoss Communications Diameter Cluster

Stack replicates only non simple sessions. Thats because simple session do not hold state and can be simply recreated by application. Simple sessions include:

- RawSession
- Session

JBoss Communications Diameter Cluster replicates full state of sessions, however it does not replicate resources which are entirely local to stack instance, like session listeners. Local resources references are recreated once session is being prepared to be used in stack instance. Listeners (state and events) are fetched from respective session factory instance. See *Section 2.1.3, “Application session factories”* for more details.

2.2. JBoss Communications Diameter Stack Setup

2.2.1. Pre-Install Requirements and Prerequisites

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

2.2.1.1. Hardware Requirements

JBoss Communications Diameter Stack does not have any hardware requirements.

2.2.1.2. Software Prerequisites

JBoss Communications Diameter Stack has the following software dependencies:

- Pico Container
- slf4j

Clustered setup also requires following:

- JDiameter HA
- JBoss Cache

2.2.2. Source Code

This section provides instructions on how to obtain and build the JBoss Communications Diameter Stack from source code.

2.2.2.1. 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 1.3.1.FINAL.

```
[usr]$ svn co ??/1.3.1.FINAL jdiameter-1.3.1.FINAL
```

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 deployable unit binary.

```

    [usr]$ cd jdiameter-1.3.1.FINAL
    [usr]$ mvn install

```

   Once the process finishes you should have the JAR files deployed in maven archive.

2.2.2.2. Development Trunk Source Building

Similar process as for *Section 2.2.2.1, “Release Source Code Building”*, the only change is the SVN source code URL, which is ${THIS.JDIAMETER_TRUNK_SOURCE_CODE_URL}.

2.3. JBoss Communications Diameter Stack Configuration

The stack is initially configured by parsing an XML file. The top level structure of the file is described below. Further explanation of each child element, and the applicable attributes is provided later in this section.

```
<Configuration xmlns="http://www.jdiameter.org/jdiameter-server">

    <LocalPeer></LocalPeer>
    <Parameters></Parameters>
    <Network></Network>
    <Extensions></Extensions>

</Configuration>
```

```
<LocalPeer>
    <URI value="aaa://localhost:1812"/>
    <IPAddresses>
        <IPAddress value="127.0.0.1"/>
    </IPAddresses>

    <Realm value="mobicents.org"/>
    <VendorID value="193"/>
    <ProductName value="jDiameter"/>
    <FirmwareRevision value="1"/>

    <OverloadMonitor>
        <Entry index="1" lowThreshold="0.5" highThreshold="0.6">
            <ApplicationID>
                <VendorId value="193"/>
                <AuthApplId value="0"/>
                <AcctApplId value="19302"/>
            </ApplicationID>
        </Entry>
    </OverloadMonitor>
    <Applications>
        <ApplicationID>
            <VendorId value="193"/>
            <AuthApplId value="0"/>
            <AcctApplId value="19302"/>
        </ApplicationID>
    </Applications>
</LocalPeer>
```

The <LocalPeer> element contains parameters which affect the local Diameter peer. The available elements and attributes are listed for reference.

<LocalPeer> Elements and Attributes

<URI>
: Specifies the URI for the local peer. URI has following format: "aaa://FQDN:port".

<IPAddresses>
: Contains one or more child <IPAddress> elements which contain a single, valid IP address for the local peer, stored in the *`value`* attribute of IPAddress.

<Realm>
: Specifies the realm of the local peer, using the *`value`* attribute.

<VendorID>
: Specifies a numeric identifier that corresponds to the vendor ID allocated by IANA.

<ProductName>
: Specifies the name of the local peer product name.

<FirmwareRevision>
: Specifies the version of the firmware.

<OverloadMonitor>
: Optional parent element containing child elements which specify settings relating to the Overload Monitor.

<Entry>
: Supports child elements of type <ApplicationID>, which specify the id of the tracked application(s). It also supports following properties:

  index
  : Defines the index of this overload monitor, so priorities/orders can be specified.

  lowThreshold
  : The low threshold for activation of the overload monitor.

  highThreshold
  : The high threshold for activation of the overload monitor.

<ApplicationID>
: Parent element containing child elements which specify information about the application. Child elements are: VendorId, AuthAppId and AcctAppId. Together they create an unique application identifier.

<VendorId>
: Specifies vendor id for application definition. It supports a single property: "value"

<AuthAppId>
: The Authentication Application ID for application definition. It supports a single property: "value"

<AcctApId>
: The Account Application ID for application defitniion. It supports a single property: "value"

<Applications>
: Contains a child element <ApplicationID>, which defines the list of default supported applications. It is used for server side. When stack is configured to accept incoming calls and there is empty list of preconfigured peers (server is configured to accept any connection).

```
<Parameters>

  <AcceptUndefinedPeer value="true"/>
  <DuplicateProtection value="true"/>
  <DuplicateTimer value="240000"/>
  <UseUriAsFqdn value="true"/> <!-- Needed for Ericsson SDK Emulator -->
  <QueueSize value="10000"/>
  <MessageTimeOut value="60000"/>
  <StopTimeOut value="10000"/>
  <CeaTimeOut value="10000"/>
  <IacTimeOut value="30000"/>
  <DwaTimeOut value="10000"/>
  <DpaTimeOut value="5000"/>
  <RecTimeOut value="10000"/>

  <Concurrent>
    <Entity name="ThreadGroup" size="64"/>
    <Entity name="ProcessingMessageTimer" size="1"/>
    <Entity name="DuplicationMessageTimer" size="1"/>
    <Entity name="RedirectMessageTimer" size="1"/>
    <Entity name="PeerOverloadTimer" size="1"/>
    <Entity name="ConnectionTimer" size="1"/>
    <Entity name="StatisticTimer" size="1"/>
    <Entity name="ApplicationSession" size="16"/>
  </Concurrent>

</Parameters>
```

The <Parameters> element contains elements which specify parameters for the Diameter stack. The available elements and attributes are listed for reference. If not specified otherwise each tag supports a single property "value" which indicates value of tag.

<Parameters> Elements and Attributes

<AcceptUndefinedPeer>
: Specifies whether the stack will accept connections from undefined peers. The default value is `false`.

<DuplicateProtection>
: Specifies whether duplicate message protection is enabled. The default value is `false`.

<DuplicateTimer>
: Specifies the time each duplicate message is valid for. The default, minimum value is `240000` (4 minutes in milliseconds)

<UseUriAsFqdn>
: Determines as URI should be used as FQDN, if set to `true` stack expects destination/origin host in format of "aaa://isdn.domain.com:3868" instead of proper "isdn.domain.com". Default value is `false`.

<QueueSize>
: Determines how many tasks peer state machine can have before rejecting next task. This queue contains FSM event and messaging.

<MessageTimeOut>
: Determines timeout for other than protocol FSM messages. Delay is in milliseconds.

<StopTimeOut>
: Determines how long stack waits for all resources to stop gracefully. Delays is in milliseconds.

<CeaTimeOut>
: Determines how long it takes for CER/CEA exchange to timeout if there is no response. Delay is in milliseconds.

<IacTimeOut>
: Determines how long to retry communication with a peer that stopped answering DWR messages. The delay is in milliseconds

<DwaTimeOut>
: Determines how long it takes for DWR/DWA exchange to timeout if there is no response. Delay is in milliseconds.

<DpaTimeOut>
: Determines how long it takes for DPR/DPA exchange to timeout if there is no response. Delay is in milliseconds.

<RecTimeOut>
: Determines how long it takes for reconnection procedure to timeout. Delay is in milliseconds.

<Concurrent />
: Controls thread pool sizes for different aspects of stack. Supports multiple *`Entity`* child elements. *`Entity`* elements configure thread groups

Entity element supports following properties:

name
: Specifies the name of the entity.

size
: Thread pool size of entity.

Default supported entities are as follows:

ThreadGroup
: Specifies maximum thread count in other entities

ProcessingMessageTimer
: Specifies thread count for message processing task.

DuplicationMessageTimer
: Specifies thread pool for identifying duplicate messages.

RedirectMessageTimer
: Specifies thread pool for redirecting messages, which do not need any further processing (ie, relaying).

PeerOverloadTimer
: Specifies thread pool for overload monitor tasks.

ConnectionTimer
: Specifies thread pool for managing tasks regarding peer connection FSM.

StatisticTimer
: Specifies thread pool for statistic gathering tasks.

ApplicationSession
: Specifies thread pool for managing invocation of application session FSM, which will invoke listeners.

```
<Network>

  <Peers>
    <!-- This peer is a server, if it's a client attempt_connect should be set to false -->
    <Peer name="aaa://127.0.0.1:3868" attempt_connect="true" rating="1"/>
  </Peers>

  <Realms>

 <Realm name="mobicents.org" peers="127.0.0.1" local_action="LOCAL" dynamic="false" exp_time="1">
      <ApplicationID>
        <VendorId value="193"/>
        <AuthApplId value="0"/>
```

```
                <AcctApplId value="19302"/>
            </ApplicationID>
        </Realm>
    </Realms>

</Network>
```

The <Network> element contains elements which specify parameters for external peers. The available elements and attributes are listed for reference.

<Network> Elements and Attributes

<Peers>
: Parent element containing child elements which specify external peers and the way they connect.

<Peer>
: Specifies the name of the external peer, whether the peer should be treated as a server or client, and what rating the peer has externally.

  Peer supports following properties:

  name
  : Specifies name of peer in for of URI. structure looks as follows "aaa://[fqdn|ip]:port", for example "aaa://192.168.1.1:3868"

  attempt_connect
  : Determines if stack should try to connect to this peer. This property accepts boolean values.

  rating
  : Specifies the rating of this peer, in order to achiever peer priorities/sorting.

<Realms>
: Parent element containing child elements which specify all realms that connect into the Diameter network.

<Realm>
: Child element containing attributes and elements which describe different realms configured for the Core. It supports <ApplicationID> child elements, which define applications supported.

  Realm supports following parameters:

  peers
  : Comma separated list of peers, each peer is represented by IP Address or FQDN.

  local_action
  : Determines the action the Local Peer will play on the specified realm: Act as a LOCAL peer

dynamic
: Specifies if this realm is dynamic, ie, peers connecting to local peer with this realm name, will be added to the realm peer list if not present already.

exp_time
: The expire time for a peer belonging to this realm to be removed from it, if no connection is available.

An example configuration file for a server supporting the CCA, Sh and Ro Applications:

```
<?xml version="1.0"?>
<Configuration xmlns="http://www.jdiameter.org/jdiameter-server">

  <LocalPeer>
    <URI value="aaa://127.0.0.1:3868" />
    <Realm value="mobicents.org" />
    <VendorID value="193" />
    <ProductName value="jDiameter" />
    <FirmwareRevision value="1" />
    <OverloadMonitor>
      <Entry index="1" lowThreshold="0.5" highThreshold="0.6">
        <ApplicationID>
          <VendorId value="193" />
          <AuthApplId value="0" />
          <AcctApplId value="19302" />
        </ApplicationID>
      </Entry>
    </OverloadMonitor>
  </LocalPeer>

  <Parameters>
    <AcceptUndefinedPeer value="true" />
    <DuplicateProtection value="true" />
    <DuplicateTimer value="240000" />
    <UseUriAsFqdn value="false" /> <!-- Needed for Ericsson Emulator (set to true) -->
    <QueueSize value="10000" />
    <MessageTimeOut value="60000" />
    <StopTimeOut value="10000" />
    <CeaTimeOut value="10000" />
    <IacTimeOut value="30000" />
    <DwaTimeOut value="10000" />
    <DpaTimeOut value="5000" />
    <RecTimeOut value="10000" />
    <Concurrent>
      <Entity name="ThreadGroup" size="64"/>
```

```
            <Entity name="ProcessingMessageTimer" size="1"/>
            <Entity name="DuplicationMessageTimer" size="1"/>
            <Entity name="RedirectMessageTimer" size="1"/>
            <Entity name="PeerOverloadTimer" size="1"/>
            <Entity name="ConnectionTimer" size="1"/>
            <Entity name="StatisticTimer" size="1"/>
            <Entity name="ApplicationSession" size="16"/>
        </Concurrent>
    </Parameters>

    <Network>
        <Peers>
            <Peer name="aaa://127.0.0.1:1218" attempt_connect="false" rating="1" />
        </Peers>
        <Realms>
            <!-- CCA -->
            <Realm name="mobicents.org" peers="127.0.0.1" local_action="LOCAL"
                dynamic="false" exp_time="1">
                <ApplicationID>
                    <VendorId value="0" />
                    <AuthApplId value="4" />
                    <AcctApplId value="0" />
                </ApplicationID>
            </Realm>

            <!-- Sh -->
            <Realm name="mobicents.org" peers="127.0.0.1" local_action="LOCAL"
                dynamic="false" exp_time="1">
                <ApplicationID>
                    <VendorId value="10415" />
                    <AuthApplId value="16777217" />
                    <AcctApplId value="0" />
                </ApplicationID>
            </Realm>

            <!-- Ro -->
            <Realm name="mobicents.org" peers="127.0.0.1" local_action="LOCAL"
                dynamic="false" exp_time="1">
                <ApplicationID>
                    <VendorId value="10415" />
                    <AuthApplId value="4" />
                    <AcctApplId value="0" />
                </ApplicationID>
            </Realm>
        </Realms>
    </Network>

    <Extensions />

  </Configuration>
```

2.3.1. Cluster configuration

To enable stack clsuter mode you need following:

- Add the following entries in `Parameters` section of `jdiameter-config.xml`:

```
<SessionDatasource>org.mobicents.diameter.impl.                ha.data.ReplicatedData</
SessionDatasource>
<TimerFacility>org.mobicents.diameter.impl.ha.            timer.ReplicatedTimerFacilityImpl</
TimerFacility>
```

- Proper `JBoss Cache` configuration file: `jdiameter-jbc.xml` (located under `config` directory). For instance following content is enough:

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

<jbosscache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:jboss:jbosscache-core:config:3.0">

  <locking isolationLevel="REPEATABLE_READ"
    lockParentForChildInsertRemove="false" lockAcquisitionTimeout="20000"
    nodeLockingScheme="mvcc" writeSkewCheck="false" concurrencyLevel="500" />

  <jmxStatistics enabled="false" />

  <startup regionsInactiveOnStartup="false" />
  <shutdown hookBehavior="DEFAULT" />
  <listeners asyncPoolSize="1" asyncQueueSize="100000" />
```

```
<invocationBatching enabled="false" />

<serialization objectInputStreamPoolSize="12"
   objectOutputStreamPoolSize="14" version="3.0.0"
   marshallerClass="org.jboss.cache.marshall.CacheMarshaller300"
   useLazyDeserialization="false" useRegionBasedMarshalling="false" />

<clustering mode="replication" clusterName="DiameterCluster">

   <async useReplQueue="true" replQueueInterval="1000"
      replQueueMaxElements="500" serializationExecutorPoolSize="20"
      serializationExecutorQueueSize="5000000"/>

   <jgroupsConfig>
      <UDP
         mcast_addr="${jgroups.udp.mcast_addr:228.10.10.10}"
         mcast_port="${jgroups.udp.mcast_port:18811}"
         discard_incompatible_packets="true"
         max_bundle_size="60000"
         max_bundle_timeout="30"
         ip_ttl="${jgroups.udp.ip_ttl:2}"
         enable_bundling="true"
         thread_pool.enabled="true"
         thread_pool.min_threads="1"
         thread_pool.max_threads="25"
         thread_pool.keep_alive_time="5000"
         thread_pool.queue_enabled="false"
         thread_pool.queue_max_size="100"
         thread_pool.rejection_policy="Run"
         oob_thread_pool.enabled="true"
         oob_thread_pool.min_threads="1"
         oob_thread_pool.max_threads="8"
         oob_thread_pool.keep_alive_time="5000"
         oob_thread_pool.queue_enabled="false"
         oob_thread_pool.queue_max_size="100"
         oob_thread_pool.rejection_policy="Run"/>

      <PING timeout="2000"
         num_initial_members="3"/>
      <MERGE2 max_interval="30000"
         min_interval="10000"/>
      <FD_SOCK/>
      <FD timeout="10000" max_tries="5" />
      <VERIFY_SUSPECT timeout="1500"  />
```

```
            <BARRIER />
            <pbcast.NAKACK
               use_mcast_xmit="false" gc_lag="0"
               retransmit_timeout="300,600,1200,2400,4800"
               discard_delivered_msgs="true"/>
            <UNICAST timeout="300,600,1200,2400,3600"/>
            <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
               max_bytes="400000"/>
            <VIEW_SYNC avg_send_interval="60000"   />
            <pbcast.GMS print_local_addr="true" join_timeout="3000"
               view_bundling="true"/>
            <FC max_credits="20000000"
               min_threshold="0.10"/>
            <FRAG2 frag_size="60000"  />
            <pbcast.STATE_TRANSFER  />
         </jgroupsConfig>
      </clustering>

   </jbosscache>
```

2.4. JBoss Communications Diameter Stack Source overview

JBoss Communications Diameter stack is built with the following basic components:

Session Factory
: Session Factory governs creation of sessions - raw and specific application sessions.

Raw and Application Sessions
: Sessions govern stateful message routing between peers. Specific application sessions consume different type of messages and act differently based on data present.

Stack
: Stack governs all necessary components which allow to establish connection and communicate with remote peers.

Note

For more detailed information please refer to Javadoc and simple examples which can be found at *SVN Testsuite Trunk* [${THIS.TESTSUITE_TRUNK_SOURCE_CODE_URL}].

2.4.1. Session Factory

`SessionFactory` provides stack user access to session objects. It manages registered application session factories, in order to allow creation of specific application sessions. An Session Factory instance can be obtained from stack with use of `getSessionFactory()` method. Base `SessionFactory` interface is defined as follows:

```
package org.jdiameter.api;

import org.jdiameter.api.app.AppSession;

public interface SessionFactory {

    RawSession getNewRawSession() throws InternalException;

    Session getNewSession() throws InternalException;

    Session getNewSession(String sessionId) throws InternalException;

    <T extends AppSession> T getNewAppSession(ApplicationId applicationId,
        Class<? extends AppSession> userSession) throws InternalException;

    <T extends AppSession> T getNewAppSession(String sessionId, ApplicationId
        applicationId, Class<? extends AppSession> userSession) throws InternalException;
}
```

However, since stack is extensible, it is safe to cast `SessionFactory` object to this interface:

```
package org.jdiameter.client.api;


public interface ISessionFactory extends SessionFactory {

    <T extends AppSession> T getNewAppSession(String sessionId,
        ApplicationId applicationId, java.lang.Class<? extends AppSession>
        aClass, Object... args) throws InternalException;

    void registerAppFacory(Class<? extends AppSession> sessionClass,
        IAppSessionFactory factory);

    void unRegisterAppFacory(Class<? extends AppSession> sessionClass);
    IConcurrentFactory getConcurrentFactory();

}
```

`RawSession getNewRawSession() throws InternalException;`

This method creates `RawSession`. Raw sessions are meant as handles for code performing part of routing decision on stacks behalf, such as rely agents for instance.

`Session getNewSession() throws InternalException;`

This method creates session which acts as endpoint for peer communication(for given session ID). It declares method which work with `Request` and `Answer` objects. Session created with this method has autogenerated ID. It should be considered as client session.

`Session getNewSession(String sessionId) throws InternalException;`

As above, however created session has Id equal to passsed as argument. Created session should be considered as server session.

`<T extends AppSession> T getNewAppSession(ApplicationId applicationId, Class<? extends AppSession> userSession) throws InternalException;`

This method creates new specific application session, identified by application Id and class of session passed. Session Id is generated by implementation. New application session should be considered as client session. It is safe to type cast return value to class passed as argument. This method delegates call to specific application session factory.

`<T extends AppSession> T getNewAppSession(String sessionId, ApplicationId applicationId, Class<? extends AppSession> userSession) throws InternalException;`

As above, however session Id is equal to argument passed. New session should be considered as server session.

`<T extends AppSession> T getNewAppSession(String sessionId, ApplicationId applicationId, java.lang.Class<? extends AppSession> aClass, Object... args) throws InternalException;`

As above, however it allows to pass some additional argument. Passed values are implementation specifc.

`void registerAppFacory(Class<? extends AppSession> sessionClass, IAppSessionFactory factory);`

registers *`factory`* for certain *`sessionClass`*. This factory will receive delegated call when ever `getNewAppSession` method is called with application class matching one from register method.

`void unRegisterAppFacory(Class<? extends AppSession> sessionClass);`

removes application session factory registered for *`sessionClass`*.

Example 2.1. SessionFactory use example

```
class Test implements EventListener<Request, Answer>
{

....
public void test(){
    Stack stack = new StackImpl();
   XMLConfiguration config = new XMLConfiguration(new FileInputStream(new File(configFile));

    SessionFactory sessionFactory = stack.init(config);
    stack.start();
    //perferctly legal, both factories are the same.
    sessionFactor = stack.getSessionFactory();
    Session session = sessionFactory.getNewSession();
    session.setRequestListener(this);
    Request r = session.createRequest(308,ApplicationId.createByAuth(100L,10101L),
        "mobicents.org","aaa://uas.fancyapp.mobicents.org");

    //add avps specific for app
    session.send(r,this);
    }
}
```

Example 2.2. SessionFactory use example

```
class Test implements EventListener<Request, Answer>
{
    Stack stack = new StackImpl();
   XMLConfiguration config = new XMLConfiguration(new FileInputStream(new File(configFile));

    ISessionFactory sessionFactory = (ISessionFactory)stack.init(config);
    stack.start();
    //perferctly legal, both factories are the same.
    sessionFactor = (ISessionFactory)stack.getSessionFactory();
    sessionFactory.registerAppFacory(ClientShSession.class, new ShClientSessionFactory(this));

    //our implementation of factory does not require any parameters
    ClientShSession session = (ClientShSession) sessionFactory.getNewAppSession(null, null
        , ClientShSession.class, null);

    ...
    session.sendUserDataRequest(udr);
}
```

2.4.2. Sessions

`RawSession`, `Session` and `ApplicationSession` provide means of disspatching and receiving messages. Specific implementation of `ApplicationSession` may provide non standard methods.

`RawSession` and `Session` life span is controlled entirelly by application. However `ApplicationSession` life time depends on implemented state machine.

`RawSession` is defined as follows:

```
public interface BaseSession extends Wrapper, Serializable {

    long getCreationTime();

    long getLastAccessedTime();

    boolean isValid();

    Future<Message> send(Message message) throws InternalException,
        IllegalDiameterStateException, RouteException, OverloadException;

    Future<Message> send(Message message, long timeOut, TimeUnit timeUnit)
        throws InternalException, IllegalDiameterStateException, RouteException, OverloadException;

    void release();
}

public interface RawSession extends BaseSession {

    Message createMessage(int commandCode, ApplicationId applicationId, Avp... avp);

    Message createMessage(int commandCode, ApplicationId applicationId,
        long hopByHopIdentifier, long endToEndIdentifier, Avp... avp);

    Message createMessage(Message message, boolean copyAvps);

    void send(Message message, EventListener<Message, Message> listener)
        throws InternalException, IllegalDiameterStateException, RouteException, OverloadException;

    void send(Message message, EventListener<Message, Message> listener,
        long timeOut, TimeUnit timeUnit) throws InternalException,
    IllegalDiameterStateException, RouteException, OverloadException;
}
```

`long getCreationTime();`
: Returns time stamp of this session creation

`long getLastAccessedTime();`
: Returns the time stamp indicating last send or receive operation.

`boolean isValid();`
: Returns `true` when this session is still valid (ie, `release()` has not been called)

`void release();`
: Application calls this method to inform that session should free any associated resource - it shall not be used anymore

`Future<Message> send(Message message)`
: Sends message in async mode. Provided `Future` reference provides means of accessing answer once its received

`void send(Message message, EventListener<Message, Message> listener, long timeOut, TimeUnit timeUnit)`
: As above. Allows to specif time out value for send operation

`Message createMessage(int commandCode, ApplicationId applicationId, Avp... avp);`
: Creates a Diameter message. It should be explicitly set either as request or answer. Passed parameters are used to build message

`Message createMessage(int commandCode, ApplicationId applicationId, long hopByHopIdentifier, long endToEndIdentifier, Avp... avp);`
: Same as above, however it allow to also set Hop-by-Hop and End-to-End Identifiers in message header. This method should be used to create answers

`Message createMessage(Message message, boolean copyAvps);`
: Clones message and returns created object. The copyAvps parameter defines if basic AVPs (Session, Route and Proxy information) should be copied to the new object.

`void send(Message message, EventListener<Message, Message> listener)`
: Sends message, answer will be delivered to the specified listener

`void send(Message message, EventListener<Message, Message> listener, long timeOut, TimeUnit timeUnit)`
: As above, it allows to pass answer wait timeout

`Session` defines similar methods, with exactly the same purpose:

```
public interface Session extends BaseSession {
    String getSessionId();

    void setRequestListener(NetworkReqListener listener);

    Request createRequest(int commandCode, ApplicationId appId, String destRealm);

    Request createRequest(int commandCode, ApplicationId appId, String destRealm, String destHost);

    Request createRequest(Request prevRequest);

    void send(Message message, EventListener<Request, Answer> listener)
        throws InternalException, IllegalDiameterStateException, RouteException, OverloadException;

    void send(Message message, EventListener<Request, Answer> listener, long timeOut,
        TimeUnit timeUnit) throws InternalException, IllegalDiameterStateException,
        RouteException, OverloadException;
}
```

2.4.3. Application Session Factories

In table below you can find session factories provided by current implementatiotion, along with short description:

Table 2.1. Application Factories

Factory class	Application type & id	Application	Reference
org.jdiameter.impl.app.acc. AccSessionFactoryImpl	AccountingId[0:3]	Acc	*RFC3588*
org.jdiameter.impl.app.auth. AuthSessionFactoryImpl	Specific	Auth	*RFC3588*
org.jdiameter.impl.app.cca. CCASessionFactoryImpl	AuthId[0:4]	CCA	*RFC4006*
org.jdiameter.impl.app.sh. ShSessionFactoryImpl	AuthId[10415:16777217]	Sh	*TS.29328*, *TS.29329*
org.jdiameter.impl.app.cxdx. CxDxSessionFactoryImpl	AuthId[13019:16777216]	Cx	*TS.29228*, *TS.29229*
org.jdiameter.impl.app.cxdx. CxDxSessionFactoryImpl	AuthId[10415:16777216]	Dx	*TS.29228*, *TS.29229*
org.jdiameter.impl.app.acc. AccSessionFactoryImpl	AccountingId[10415:3]	Rf	*TS.32240*
org.jdiameter.impl.app.cca. CCASessionFactoryImpl	AuthId[10415:4]	Ro	*TS.32240*

Note

There is no specific factory for Ro and Rf, those application reuse respective session and session factories.

Note

Application id contains two numbers - [VendorId:ApplicationId].

Important

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

2.5. JBoss Communications Diameter Stack Validator

Validator is one of the Stack features. The primary purpose of the Validator is to detect malformed messages, such as an Answer message containing a Destination-Host Attribute Value Pair (AVP).

The Validator is capable of validating multi-leveled, grouped AVPs, excluding the following content types:

- URI, or Identifier types.
- Enumerated types against defined values.

That is, it is capable of checking structural integrity, not content.

Validator performs the following checks:

Index
: Checks if AVPs are in correct place, eg, *`Session-Id`* must always be encoded before any other AVP.

Multiplicity
: Checks if message AVPs occurs the correct number of times, eg, if Session-Id is present one and only one time.

Validator is called by stack implementation. It is invoked after message is received and before it is dispatched to remote peer. Note that latter means that if peer does not exist in local peer table validator is not called, as stack fails before calling it.

2.5.1. Validator Configuration

Validator is configured with a single XML file. This file contains the structure definition for messages and AVPs.

Upon creation of Diameter Stack, the validator is initialized. It performs initialization by looking up `dictionary.xml` file in classpath.

Note

Configuration file contains more data than Validator use to build its data base. It is caused by the fact that Dictionary uses the same file to configure itself. It reuses AVP definitions, extended with some information like AVP type and flags.

Configuration file structure is the following:

```
<dictonary>
      <validator enabled="true|false" sendLevel="OFF|MESSAGE|ALL" receiveLevel="OFF|
MESSAGE|ALL" />
  <vendor name="" vendor-id="" code=""/>
  <typedefn type-name="" type-parent=""/>
  <application id="" name="">
    <avp ...>
      <type type-name=""/>
      <enum name="" code=""/>
      <grouped>
        <gavp name=""/>
      <grouped/>
    <avp/>
    <command name="" code="" request="true|false"/>
    <avp ...>
      <type type-name=""/>
      <enum name="" code=""/>
      <grouped>
        <gavp name=""/>
      <grouped/>
    <avp/>
  <application>
<dictionary/>
```

<dictionary>. The root element, which contains the child elements comprising the validator and dictionary components. This element does not support any attributes.

<validator>. Specifies whether message validation is activated for sent and received stack messages. The element supports the following optional attributes:

enabled
: Specifies whether the validator is activated or deactivated. If not specified, the validator is deactivated.

sendLevel
: Determines the validation level for message sent by the stack instance. Values determine if sent messages are not validated at all (`OFF`), only message level AVPs are checked (`MESSAGE`) or all AVPs are in message are checked (`ALL`).

receiveLevel
: Determines the validation level for message received by the stack instance. Values determine if sent messages are not validated at all (`OFF`), only message level AVPs are checked (`MESSAGE`) or all AVPs are in message are checked (`ALL`).

<vendor>. Optional element, which specifies the mapping between the vendor name, vendor ID, and vendor code. The element supports the following required attributes:

name
: Specifies the vendor name. For example "Hewlett Packard".

vendor-id
: Specifies the unique ID associated with this vendor. For example, "HP".

code
: Specifies the alpha-numeric code allocated to the vendor by IANA. For example "11". The value must be unique for each <vendor> declaration.

Example 2.3. <vendor> XML Attributes

```
...
<vendor vendor-id="None" code="0" name="None" />
<vendor vendor-id="HP" code="11" name="Hewlett Packard" />
<vendor vendor-id="Merit" code="61" name="Merit Networks" />
<vendor vendor-id="Sun" code="42" name="Sun Microsystems, Inc." />
<vendor vendor-id="USR" code="429" name="US Robotics Corp." />
<vendor vendor-id="3GPP2" code="5535" name="3GPP2" />
<vendor vendor-id="TGPP" code="10415" name="3GPP" />
<vendor vendor-id="TGPPCX" code="16777216" name="3GPP CX/DX" />
<vendor vendor-id="TGPPSH" code="16777217" name="3GPP SH" />
<vendor vendor-id="Ericsson" code="193" name="Ericsson" />
<vendor vendor-id="ETSI" code="13019" name="ETSI" />
```

```
<vendor vendor-id="Vodafone" code="12645" name="Vodafone" />
```

<typedefn>. Defines the simple Attribute Value Pair (AVP) types. The element supports the following required attributes:

type-name
: Specifies a type name in accordance with the acceptable base types defined in RFC 3588. For example; "Enumerated", "OctetString", "Integer32".

type-parent
: Specifies the parent type name used to define the base characteristics of the type. The values are restricted to defined <typedefn> elements. For example; "OctetString", "UTF8String", "IPAddress".

Example 2.4. <typedefn> XML Attributes

```
<!-- Primitive types, see http://tools.ietf.org/html/rfc3588#section-4.2 -->
<typedefn type-name="OctetString" />
<typedefn type-name="Float64" />
<typedefn type-name="Float32" />
<typedefn type-name="Integer64" />
<typedefn type-name="Integer32" />
<typedefn type-name="Unsigned64" />
<typedefn type-name="Unsigned32" />

<!-- derived avp types, see http://tools.ietf.org/html/rfc3588#section-4.3 -->
<typedefn type-name="Address" type-parent="OctetString" />
<typedefn type-name="Time" type-parent="OctetString" />
<typedefn type-name="UTF8String" type-parent="OctetString" />
<typedefn type-name="DiameterIdentity" type-parent="OctetString" />
```

<application>. Defines the specific applications used within the dictionary. Two child elements are supported by <application>: <avp> and <command>. The <application> element supports the following attributes:

id
: Specifies the unique ID allocated to the application. The attribute is used in all messages and forms part of the message header.

name
: Optional attribute, which specifies the logical name of the application.

uri
: Optional attribute, which specifies a link to additional application information.

Example 2.5. <application> XML Attributes

```
<application id="16777216" name="3GPP Cx/Dx" uri="http://www.ietf.org/rfc/rfc3588.txt?
number=3588">
```

<avp>. Element containing information necessary to configure the Attribute Value Pairs. *Table 2.2, “<avp> Attributes”* contains the complete list of supported attributes, and their available values (if applicable).

The <avp> element supports a number of child elements, which are used to set finer parameters for the individual AVP. The supported elements are <type>, <enum>, and <grouped>.

Note

Different sets of elements are supported by <avp> depending on its position in the dictionary.xml file.

Example 2.6. <avp> Child Elements and Attributes

```
<avp name="Server-Assignment-Type" code="614" mandatory="must" vendor-bit="must"
 vendor-id="TGPP" may-encrypt="no">
    <type type-name="Unsigned32" />
    <enum name="NO_ASSIGMENT" code="0" />
    <enum name="REGISTRATION" code="1" />
    <enum name="RE_REGISTRATION" code="2" />
    <enum name="UNREGISTERED_USER" code="3" />
    <grouped>
       <gavp name="SIP-Item-Number" multiplicity="0-1"/>
       <gavp name="SIP-Authentication-Scheme" multiplicity="0-1"/>
       <gavp name="SIP-Authenticate" multiplicity="0-1"/>
       <gavp name="Line-Identifier" multiplicity="0+"/>
    </grouped>
</avp>
```

<type>. Child element of <avp>, which is used to match the AVP with the AVP type as defined in the <typedefn> element. The element supports the following mandatory attribute:

type-name
: Specifies the type name, which is used to match to the type-name value specified in the <typedefn> element.

Note

<type> is ignored if the <avp> element contains the <grouped> element.

<enum>. Child element of <avp>, which specifies the enumeration value for the specified AVP. <enum> is used only when the type-name attribute of <type> is specified. The element supports the following mandatory attributes:

name
: Specifies the name of a constant value that applies to the AVP.

code
: Specifies the integer value associated with the name of the constant. The value is passed as a value of the AVP, and maps to the name attribute.

Note

<enum> is ignored if the <avp> element contains the <grouped> element.

<grouped>. Child element of <avp>, which specifies the AVP is a grouped type. A grouped AVP is one that has no <typedefn> element present. The element does not support any attributes, however the <gavp> element is allowed as a child element.

<gavp>. Child element of <grouped>, which specifies a reference to a grouped AVP. The element supports one mandatory attribute:

name
: Specifies the name of the grouped AVP member. The value must match the defined AVP name.

Table 2.2. <avp> Attributes

Attribute Name (optional in brackets)	Explicit Values (default in brackets)	Description
name		Specifies the name of the AVP, which is used to match the AVP definition to any grouped AVP references. For further information about grouped AVPs, refer to the element description in this section.
code		Specifies the integer code of the AVP.

Attribute Name (optional in brackets)	Explicit Values (default in brackets)	Description
(vendor-id)	(none)	Used to match the vendor ID reference to the value defined in the <vendor> element.
(multiplicity)		Specifies the number of acceptable AVPs in a message using an explicit value.
	0	An AVP *must not* be present in the message.
	(0+)	Zero or more instances of the AVP *must be* present in the message.
	0-1	Zero, or one instance of the AVP *may be* present in the message. An error occurs if the message contains more than one instance of the AVP.
	1	One instance of the AVP *must be* present in the message.
	1+	At least one instance of the AVP *must be* present in the message.
may-encrypt	Yes | (No)	Specifies whether the AVP can be encrypted.
protected	may | must | mustnot	Determines actaul state of AVP that is expected, if it MUST be encrypted , may or MUST NOT.
vendor-bit	must | mustnot	Specifies whether the Vendor ID should be set.
mandatory	may | must | mustnot	Determines if support for this AVP is madnatory in order to consume/process message.
vendor		Specifies the defined vendor code, which is used by the <command> child element.

Example 2.7. <avp> XML Attributes

```
<!-- MUST -->
<avp name="Session-Id" code="263" vendor="0" multiplicity="1" index="0" />
<avp name="Auth-Session-State" code="277" vendor="0" multiplicity="1" index="-1" />

<!-- MAY -->
<avp name="Destination-Host" code="293" vendor="0" multiplicity="0-1" index="-1" />
<avp name="Supported-Features" code="628" vendor="10415" multiplicity="0+" index="-1" />

<!-- FORBBIDEN -->
<avp name="Auth-Application-Id" code="258" vendor="0" multiplicity="0" index="-1" />
<avp name="Error-Reporting-Host" code="294" vendor="0" multiplicity="0" index="-1" />
```

<command>. Specifies the command for the application. The element supports the <avp> element, which specifies the structure of the command. The element supports the following attributes:

name
: Optional parameter, which specifies the message name for descriptive purposes.

code
: Mandatory parameter, which specifies the integer code of the message.

request
: Mandatory parameter, which specifies whether the declared command is a request or answer. The available values are "true" (request) or "false" (answer).

Note

If the <avp> element is specified in <command> it does not support any child elements. The <avp> element only refers to defined AVPs when used in this context.

Example 2.8. <command> Elements and Attributes

```
<command name="User-Authorization" code="300" vendor-id="TGPP" request="true">
        <avp name="Server-Assignment-Type" code="614" mandatory="must" vendor-bit="must" vendor-id="TGPP" may-encrypt="no"/>
</command>
```

2.5.2. Validator Source Overview

Validator API defines methods to access its data base of AVP and check if AVP and message has proper structure.

Currently Validator is message oriented, that is it declared methods which center on message consitency checks. Class containing all validation logic is `org.jdiameter.common.impl.validation.DiameterMessageValidator`. It exposes the following methods:

public boolean isOn();
: Simple method to determine if `Validator` is enabled.

public ValidatorLevel getSendLevel();
: Returns validation level for outgoing messages. It can have one of following values: `OFF`, `MESSAGE`, `ALL`

public ValidatorLevel getReceiveLevel()
: Returns validation level for incoming messages. It can have one of following values: `OFF`, `MESSAGE`, `ALL`

public void validate(Message msg, boolean incoming) throws JAvpNotAllowedException
: Performs validation on message. Based on *`incoming`* flag, the correct validation level is applied. If validation fails, an exception with details is thrown.

public void validate(Message msg, ValidatorLevel validatorLevel) throws JAvpNotAllowedException
: Performs validation on message with the specified level. It is a programatical way to allow different level of validation different than configured. If validation fails, a `JAvpNotAllowedException` with details is thrown.

Current implementation provides more methods, however those are out of scope for this documentation.

Simple example of validator use case look as follows:

Example 2.9. Validator Message Check Example

The example below is pseudo-code.

```


...
boolean isRequest = true;
boolean isIncoming = false;

DiameterMessageValidator messageValidator = DiameterMessageValidator.getInstance();
```

```
Message message = createMessage(UserDataRequest.MESSAGE_CODE, isRequest,
   applicationId);

//add AVPs
...
//perform check
try{
   messageValidator.validate(message, isIncoming);
}
catch(JAvpNotAllowedException e) {
        System.err.println("Failed to validate ..., avp code: " + e.getAvpCode() + " avp
 vendor:" + e.getVendorId() + ", message:" + e.getMessage());
}
```

JBoss Communications Diameter Multiplexer

The JBoss Communications Diameter Multiplexer (MUX) is designed as a stack wrapper. It serves two purposes:

Expose stack
: It exposes the Diameter Stack and allows it to be shared between multiple listeners. The stack follows the MUX life cycle, ie, it is created and destroyed with the MUX.

Expose management operations
: It exposes management operations for JMX clients, one of them being the JBoss Operations Network Console. For specifc information please refer to the JBoss Communications Diameter Management Console User Guide.

3.1. JBoss Communications Diameter Multiplexer (MUX) Design

JBoss Communications Diameter MUX is a simple service provided on behalf of stack. Entities interested in receiving messages, for a certain Diameter application, may register for that in the MUX. Upon registration entity passes set of Application-Ids, which are of interest to it. Based on message content and registered listeners, MUX either drops message or passes it to proper listener. MUX checks Application-Ids present in message to match target listener.

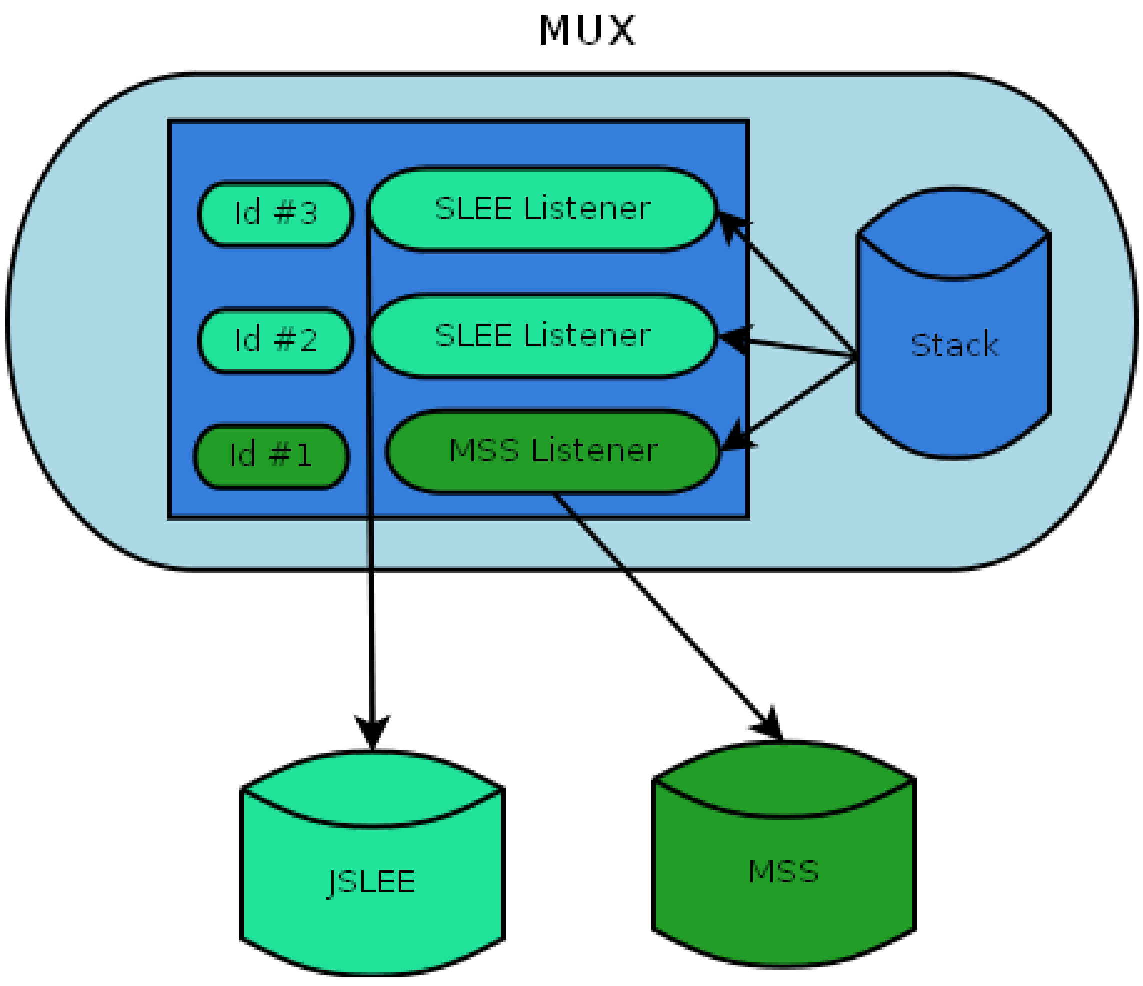

Figure 3.1. JBoss Communications Diameter Multiplexer (MUX) Design Overview

3.2. JBoss Communications Diameter Multiplexer (MUX) Setup

3.2.1. Pre-Install Requirements and Prerequisites

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

3.2.1.1. Hardware Requirements

MUX does not have any hardware requirements.

3.2.1.2. Software Prerequisites

MUX must be deployed either in JBoss Enterprise Application Platform v4.x or v5.x. However it is possible to adapt configuration files and run in any JMX container.

3.2.2. Source Code

This section provides instructions on how to obtain and build the JBoss Communications Diameter MUX from source code.

3.2.2.1. 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 ${THIS.MUX_RELEASE_SOURCE_CODE_URL}, then add the specific release version, lets consider 1.3.1.FINAL.

```
[usr]$ svn co ${THIS.MUX_RELEASE_SOURCE_CODE_URL}/1.3.1.FINAL
 mux-1.3.1.FINAL
```

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 deployable unit binary.

```
[usr]$ cd mux-1.3.1.FINAL
[usr]$ mvn install
```

   Once the process finishes you should have the SAR built. If `JBOSS_HOME` environment variable is set, after execution SAR will be deployed in container.

Note

By default JBoss Communications Diameter MUX; deploys in JBoss Enterprise Application Platform v4.x SAR. To change it run **maven** with profile switch: -*`Pjboss5`*

3.2.2.2. Development Trunk Source Building

Similar process as for *Section 3.2.2.1, “Release Source Code Building”*, the only change is the SVN source code URL, which is ${THIS.MUX_TRUNK_SOURCE_CODE_URL}.

3.3. JBoss Communications Diameter Multiplexer (MUX) Configuration

MUX requires has three configuration files:

`jboss-service.xml`
: This file is specific to SAR, please refer to **JBoss Enterprise Application Platform** manual for explanation. By default this file binds JBoss Communications Diameter MUX under following JMX object name: `diameter.mobicents:service=DiameterStackMultiplexer`.

`jdiameter-config.xml`
: This file configures stack exposed by MUX, please refer to *Section 2.3, “JBoss Communications Diameter Stack Configuration”* for details. It is located in `mobicents-diameter-mux-1.3.1.FINAL.sar/config`

`dictionary.xml`
: This file configures dictionary. Its structure and content is identical to file described in *Section 2.5.1, “Validator Configuration”*.

3.4. JBoss Communications Diameter Multiplexer (MUX) Source Overview

JBoss Communications Diameter MUX capabilities are defined by a MBean interface: `org.mobicents.diameter.stack.DiameterStackMultiplexerMBean`. This interface defines two types of methods:

Management
: Used by JBoss Operations Network console, those methods are out of scope for user.

Stack accessors
: Methods which allow retrieval and usage of wrapped stack.

The following methods are of interest to JBoss Communications Diameter MUX user:

`public Stack getStack();`
: Returns the stack instance wrapped by multiplexer. It's present as a convenience method, and stack should only be changed directly by expert users.

`public void registerListener(DiameterListener listener, ApplicationId[] appIds) throws IllegalStateException;`
: Method for registering a Diameter listener, to be triggered when a message for certain application id is received.

`public void unregisterListener(DiameterListener listener);`
: Method for unregistering a Diameter listener, for all its applications.

`public DiameterStackMultiplexerMBean getMultiplexerMBean();`
: Returns the actual instance of MUX.

Listener interface is defined as follows:

```
package org.mobicents.diameter.stack;

import java.io.Serializable;

import org.jdiameter.api.Answer;
import org.jdiameter.api.EventListener;
import org.jdiameter.api.NetworkReqListener;
import org.jdiameter.api.Request;

public interface DiameterListener extends NetworkReqListener, Serializable,
   EventListener<Request, Answer>
{

}
```

MUX can be used as follows:

```
public class DiameterActor implements DiameterListener
{
   private ObjectName diameterMultiplexerObjectName = null;
   private DiameterStackMultiplexerMBean diameterMux = null;

   private synchronized void initStack() throws Exception {
        this.diameterMultiplexerObjectName =
          new ObjectName("diameter.mobicents:service=DiameterStackMultiplexer");

        Object[] params = new Object[]{};
        String[] signature = new String[]{};

        String operation = "getMultiplexerMBean";
        this.diameterMux=mbeanServer.invoke(this.diameterMultiplexerObjectName, operation,
          params, signature);

        long acctAppIds = new long[]{19312L};
        long acctVendorIds = new long[]{193L};
        long authAppIds = new long[]{4L};
        long authVendorIds = new long[]{0L};
        List<ApplicationId> appIds = new ArrayList<ApplicationId>();
        for(int index = 0;index<acctAppIds.length;index++) {
          appIds.add(ApplicationId.createByAccAppId(acctVendorIds[index], acctAppIds[index]));
        }

        for(int index = 0;index<authAppIds.length;index++) {
          appIds.add(ApplicationId.createByAuthAppId(authVendorIds[index], authAppIds[index]));
        }

        this.diameterMux.registerListener(this, appIds.toArray(new ApplicationId[appIds.size()]));
        this.stack = this.diameterMux.getStack();
        this.messageTimeout = stack.getMetaData().getConfiguration().getLongValue(
          MessageTimeOut.ordinal(), (Long) MessageTimeOut.defValue());
    }
}
```

3.5. JBoss Communications Diameter Multiplexer (MUX) Dictionary

Dictionary is part of Diameter MUX package. Its purpose is to provide unified access to information regarding AVP structure, content and definition. Dictionary is configured via a XML file, `dictionary.xml`.

The Dictionary logic is contained in `org.mobicents.diameter.dictionary.AvpDictionary` class. It exposes the following methods:

`public AvpRepresentation getAvp(int code)`
: Return an `AvpRepresentation` object representing the AVP with the given code (assuming vendor id as 0 (zero)). If there is no AVP defined, `null` is returned.

`public AvpRepresentation getAvp(int code, long vendorId)`
: Returns an `AvpRepresentation` object representing the AVP with the given code and vendor id. If there is no AVP defined, `null` is returned.

`public AvpRepresentation getAvp(String avpName)`
: Returns an `AvpRepresentation` object representing the AVP with the given name. If there is no AVP defined, `null` is returned.

Dictionary uses a POJO class (`org.mobicents.diameter.dictionary.AvpRepresentation`) to provide access to stored information. It exposes the following methods:

`public int getCode()`
: Returns the code assigned to represented AVP.

`public long getVendorId()`
: Returns the vendor id assigned to represented AVP.

`public String getName()`
: Returns the name assigned to represented AVP. If no name is defined, `null` is returned.

`public boolean isGrouped()`
: Returns `true` if AVP is of grouped type.

`public String getType()`
: Returns a String with the name of the represented AVP type. Return value is equal to one of defined types, for instance OctetString or Unsiged32.

`public boolean isMayEncrypt()`
: Returns `true` if AVP can be encrypted.

`public boolean isProtected()`
: Returns `true` if AVP must be encrypted. It returns `true` only if `public String getRuleProtected()` returns `must`.

`public boolean isMandatory()`
: Returns `true` if AVP must be supported by agent to properly consume message. It returns `true` only if `public String getRuleMandatory()` returns `must`.

`public String getRuleMandatory()`
: Returns the mandatory rule value. It can return one of following values: `may`, `must` or `mustnot`.

`public String getRuleProtected()`
: Returns the protected rule value. It can have one of following values: `may`, `must` or `mustnot`.

`public String getRuleVendorBit()`
: Returns the vendor rule value. It can have one of following values: `must` or `mustnot`.

The JBoss Communications Diameter MUX Dictionary can be used as follows:

```
public static void addAvp(Message msg, int avpCode, long vendorId, AvpSet set, Object avp) {
```

```
AvpRepresentation avpRep = AvpDictionary.INSTANCE.getAvp(avpCode, vendorId);

if(avpRep != null) {
    DiameterAvpType avpType = DiameterAvpType.fromString(avpRep.getType());

    boolean isMandatoryAvp = avpRep.isMandatory();
    boolean isProtectedAvp = avpRep.isProtected();

    if(avp instanceof byte[]) {
     setAvpAsRaw(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp, (byte[]) avp);
    }
    else
    {
        switch (avpType.getType()) {
        case DiameterAvpType._ADDRESS:
        case DiameterAvpType._DIAMETER_IDENTITY:
        case DiameterAvpType._DIAMETER_URI:
        case DiameterAvpType._IP_FILTER_RULE:
        case DiameterAvpType._OCTET_STRING:
        case DiameterAvpType._QOS_FILTER_RULE:
            setAvpAsOctetString(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
                avp.toString());
            break;

        case DiameterAvpType._ENUMERATED:
        case DiameterAvpType._INTEGER_32:
            setAvpAsInteger32(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
                (Integer) avp);
            break;

        case DiameterAvpType._FLOAT_32:
            setAvpAsFloat32(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
                (Float) avp);
            break;

        case DiameterAvpType._FLOAT_64:
            setAvpAsFloat64(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
                (Float) avp);
            break;

        case DiameterAvpType._GROUPED:
            setAvpAsGrouped(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
                (DiameterAvp[]) avp);
            break;
        case DiameterAvpType._INTEGER_64:
          setAvpAsInteger64(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
            (Long) avp);
          break;

        case DiameterAvpType._TIME:
          setAvpAsTime(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
            (Date) avp);
          break;

        case DiameterAvpType._UNSIGNED_32:
          setAvpAsUnsigned32(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
            (Long) avp);
          break;

        case DiameterAvpType._UNSIGNED_64:
          setAvpAsUnsigned64(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
            (Long) avp);
          break;

        case DiameterAvpType._UTF8_STRING:
          setAvpAsUTF8String(msg, avpCode, vendorId, set, isMandatoryAvp, isProtectedAvp,
            (String) avp);
          break;
        }
      }
    }
  }
```

Appendix A. Revision History

Revision History

Revision 1.0	Thu Mar 11 2010	BartoszBaranowski, AlexandreMendonça

Creation of the JBoss Communications Diameter User Guide.

Revision 1.0	Mon July 12 2010	BartoszBaranowski, AlexandreMendonça

Creation of the JBoss Communications Diameter User Guide.

Index

F

feedback, viii