Restcomm Diameter Stack

Table of Contents

The Diameter Stack is the core component of the presented Diameter solution. It performs all necessary tasks to allow user interaction with the Diameter network. It manages the state of diameter peers and allows to route messages between them. For more details, refer to RFC 3588.

The Diameter Stack currently supports the following application sessions:

- Base
- Credit Control Application (CCA)
- Sh
- Ro
- Rf
- Cx/Dx
- Gx
- Gq'
- Rx

Diameter Stack Design

Diameter Stack Extensibility

Diameter Stack has been designed to be extensible. In order to achieve that, two set of APIs are defined by the stack - one that defines basic contracts between the user application and the stack, and one that defines contracts allowing the instance to inject custom objects into the stack to perform certain tasks (for example, `SessionFactory`). `ISessionFactory` declares additional methods that allow the developer to declare custom behaviour (for example, custom application sessions). Please refer to Session Factory for more detailed information.

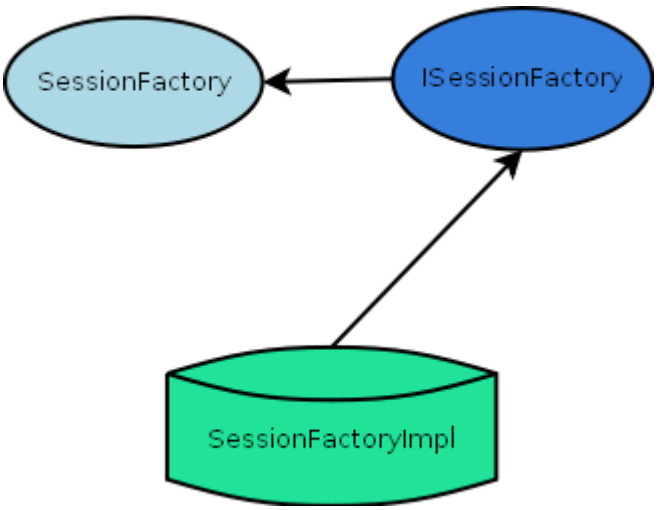

Figure 1. Diameter Stack Extensibility Visualization

The general pattern for interface declaration is:

- The interface `ComponentInterface` declares the minimal set of methods for a component to perform its task.
- The interface `IComponentInterface` provides additional behavior methods. Please refer to the java doc for a list of interfaces and descriptions of method contracts.

Diameter Stack Model

Diameter Stack performs the following tasks:

- Manages connections to remote peers.
- Manages session objects.
- Routes messages on behalf of sessions.
- Receives and delivers messages to assigned listeners (usually a session object).

Sessions use the stack and the services it provides to communicate with remote peers. The application is the only place that holds references to sessions. It can be seen as follows:

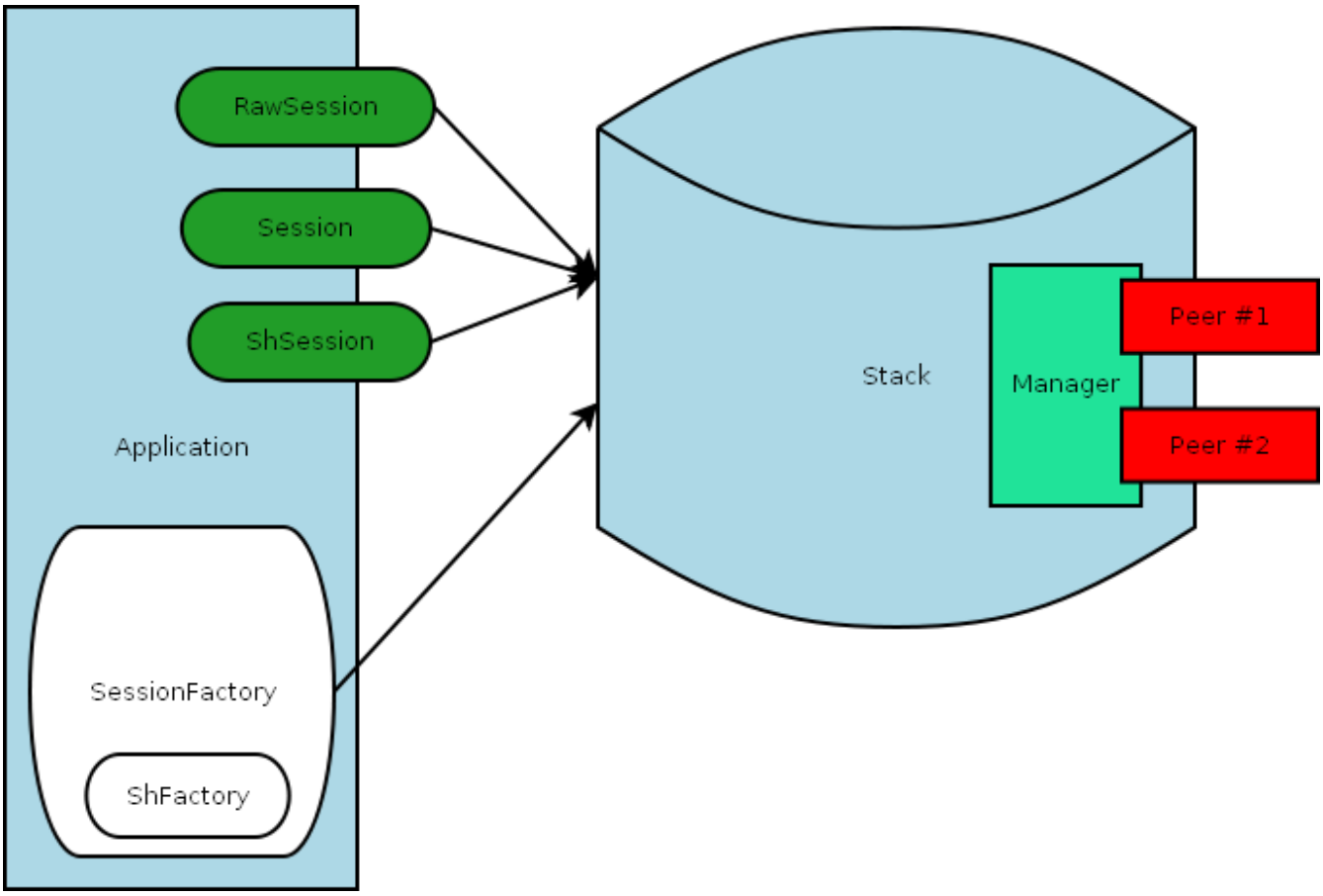

Figure 2. Diameter Application and Stack Model

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 and context.

The figure below shows the relationship between Application Session Factories and User Applications:

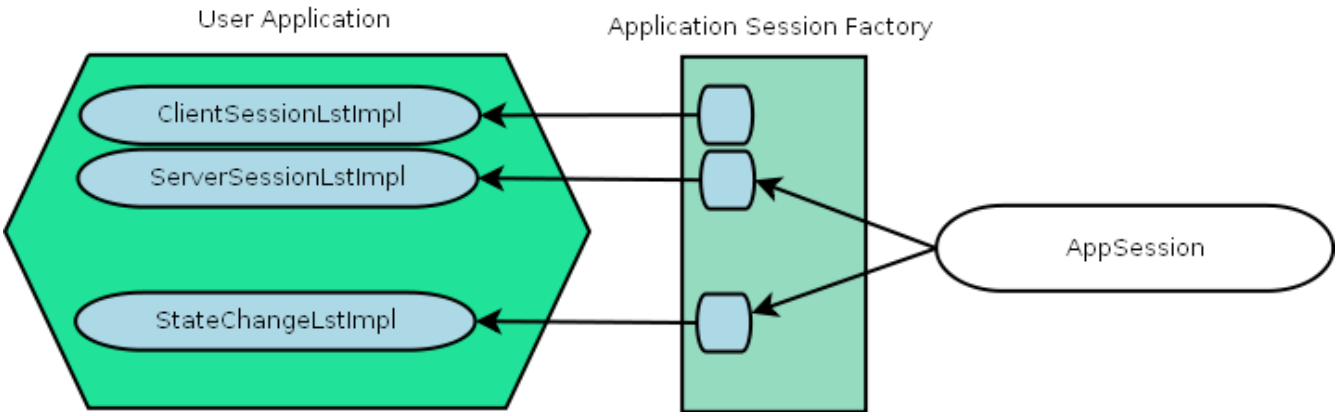

Figure 3. Application Session Factory and User Application

Session context is a callback interface defined by some sessions.

Session Replication

Diameter Stack supports replication of session data and state. Clustered stack instances can perform operations on session regardless of physical location. Imagine the logically clustered stack as follows:

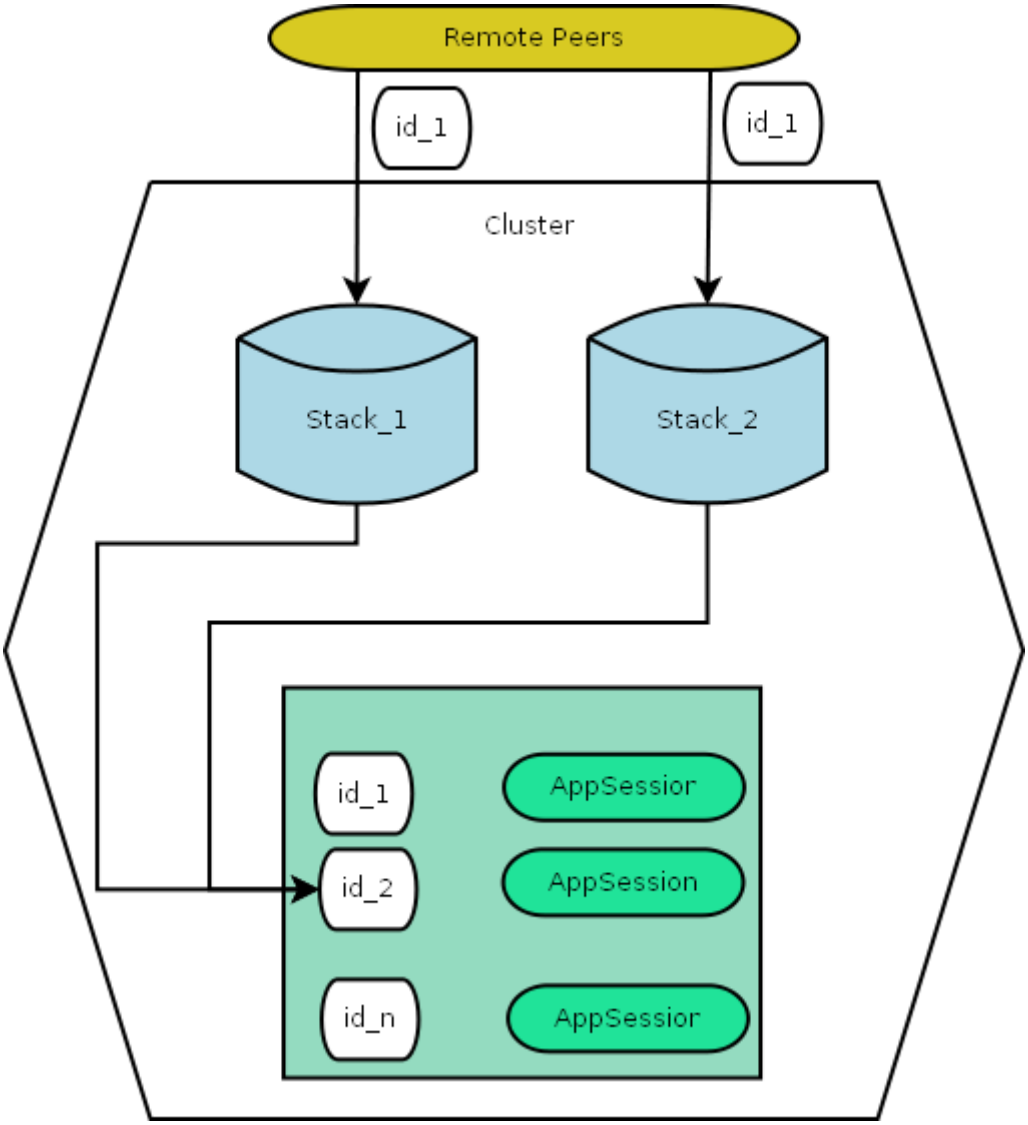

Figure 4. Diameter Cluster

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

- RawSessions
- Sessions

Diameter Cluster replicates the full state of sessions. However, it does not replicate resources that are entirely local to the stack instance, like session listeners. Local resource references are recreated once the session is being prepared to be used in the stack instance. Listeners (state and events) are fetched from the respective session factory instance. See Application Session Factories for more details.

Restcomm Diameter Stack Setup

Pre-Install Requirements and Prerequisites

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

Hardware Requirements

Restcomm Diameter Stack does not have any hardware requirements.

Software Prerequisites

Restcomm Diameter Stack has the following software dependencies:

- Pico Container
- slf4j

Clustered setup also requires following:

- JDiameter HA
- JBoss Cache

Source Code

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

Release Source Code Building

1. Downloading the source code

   

   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 https://github.com/Restcomm/jdiameter , then add the specific release version, lets consider &THIS.VERSION;.

   ```
   [usr]$ git clone git@github.com:RestComm/jss7.git
   ```

2. Building the source code

   

   Maven 3.2.5 (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 -
   [usr]$ mvn install
   ```

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

Development Trunk Source Building

Follow the process in Release Source Code Building, replacing the SVN source code URL with https://github.com/Restcomm/jdiameter .

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:3868"/>
    <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 that 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. The URI has the following format: "aaa://FQDN:port".

<IPAddresses>

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

<*FirmwareRevision*>

Specifies the version of the firmware.

<*OverloadMonitor*>

Optional parent element containing child elements that specify settings relating to the Overload Monitor.

<*Entry*>

Supports <ApplicationID> child elements that specify the ID of the tracked application(s). It also supports the 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 that specify information about the application. The child elements create a unique application identifier. The child elements are:

<VendorId> Specifies the 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".

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

<*Applications*>

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

```
<Parameters>

    <AcceptUndefinedPeer value="true"/>
    <DuplicateProtection value="true"/>
  <DuplicateTimer value="240000"/>
  <DuplicateSize value="5000"/>
    <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"/>

    <!-- Peer FSM Thread Count Configuration -->
    <PeerFSMThreadCount value="3" />

    <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 that 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 the value of the 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 (in extreme cases, it can live up to 2 * DuplicateTimer - 1 milliseconds). The default, minimum value is `240000` (4 minutes in milliseconds).

<*DuplicateSize*>

Specifies the number of requests stored for duplicate protection. The default value is `5000`.

<*UseUriAsFqdn*>

Determines whether the URI should be used as FQDN. If it is set to `true`, the stack expects the destination/origin host to be in the format of "aaa://isdn.domain.com:3868" rather than the normal "isdn.domain.com". The default value is `false`.

<*QueueSize*>

Determines how many tasks the peer state machine can have before rejecting the next task. This queue contains FSM events and messaging.

<*MessageTimeOut*>

Determines the timeout for messages other than protocol FSM messages. The delay is in milliseconds.

<*StopTimeOut*>

Determines how long the stack waits for all resources to stop. The delays are in milliseconds.

<*CeaTimeOut*>

Determines how long it takes for CER/CEA exchanges to timeout if there is no response. The delays are in milliseconds.

<*IacTimeOut*>

Determines how long the stack waits to retry the communication with a peer that has stopped answering DWR messages. The delay is in milliseconds.

<*DwaTimeOut*>

Determines how long it takes for a DWR/DWA exchange to timeout if there is no response. The delay is in milliseconds.

<*DpaTimeOut*>

Determines how long it takes for a DPR/DPA exchange to timeout if there is no response. The delay is in milliseconds.

<*RecTimeOut*>

Determines how long it takes for the reconnection procedure to timeout. The delay is in milliseconds.

<*PeerFSMThreadCount*>

Determines the number of threads for handling events in the Peer FSM.

<*Concurrent /*>

Controls the thread pool sizes for different aspects of the stack. It supports multiple `Entity` child elements. `Entity` elements configure thread groups. These elements support the following properties:

name Specifies the name of the entity.

size Specifies the thread pool size of the entity.

The default supported entities are:

ThreadGroup Determines the maximum thread count in other entities.

ProcessingMessageTimer Determines the thread count for message processing tasks.

DuplicationMessageTimer Specifies the thread pool for identifying duplicate messages.

RedirectMessageTimer Specifies the thread pool for redirecting messages that do not need any further processing.

PeerOverloadTimer Determines the thread pool for managing the overload monitor.

ConnectionTimer Determines the thread pool for managing tasks regarding peer connection FSM.

StatisticTimer Determines the thread pool for statistic gathering tasks.

ApplicationSession Determines the thread pool for managing the invocation of application session FSMs, 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 that specify parameters for external peers. The available elements and attributes are listed for reference.

<Network> Elements and Attributes

<Peers>

Parent element containing the child element <Peer>, which specifies external peers and the way they connect. <Peer> specifies the name of external peers, whether they should be treated as a

server or client, and what rating the peer has externally.

<Peer> supports the following properties:

name Specifies the name of the peer in the form of a URI. The structure is "aaa://[fqdn|ip]:port" (for example, "aaa://192.168.1.1:3868").

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

rating Specifies the rating of this peer in order to achieve peer priorities/sorting.

<Realms>

Parent element containing the child element <Realm>, which specifies all realms that connect into the Diameter network. <Realm> contains attributes and elements that describe different realms configured for the Core. It supports <ApplicationID> child elements, which define the applications supported.

<Realm> supports the following parameters:

peers Comma separated list of peers. Each peer is represented by an 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. That is, peers that connect to peers with this realm name will be added to the realm peer list if not present already.

exp_time The time before a peer belonging to this realm is removed if no connection is available.

Below is 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" />
    <DuplicateSize value="5000" />
        <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" />

        <PeerFSMThreadCount value="3" />

        <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>
```

Cluster configuration

The following list defines the requirements for enabling stack cluster mode

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

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

- A proper `JBoss Cache` configuration file: *jdiameter-jbc.xml* (located in the *config* directory).

  The following content is sufficient for the JBoss Cache configuration file:

```
<?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>
```

Diameter Stack Source overview

Diameter stack is built with the following basic components:

Session Factory

The Session Factory governs the 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 the data present.

Stack

The Stack governs all necessary components, which are used to establish connection and communicate with remote peers.

For more detailed information, please refer to the Javadoc or the simple examples that can be found here: Git Testsuite HEAD.

Session Factory

`SessionFactory` provides the stack user with access to session objects. It manages registered application session factories in order to allow for the creation of specific application sessions. A Session Factory instance can be obtained from the stack using the `getSessionFactory()` method. The base `SessionFactory` interface is defined below:

```
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 the stack is extensible, it is safe to cast the `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 a `RawSession`. Raw sessions are meant as handles for code performing part of the routing decision on the stack's, such as rely agents for instance.

`Session getNewSession() throws InternalException;`

This method creates a session that acts as the endpoint for peer communication (for a given session ID). It declares the method that works with the `Request` and `Answer` objects. A session created with this method has an autogenerated ID. It should be considered as a client session.

`Session getNewSession(String sessionId) throws InternalException;`

As above. However, the created session has an ID equal to that passsed as an argument. This created session should be considered a server session.

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

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

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

As above. However, the session Id is equal to the argument passed. New sessions should be considered server sessions.

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

As above. However, it allows the stack to pass some additional arguments. Passed values are implementation specifc.

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

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

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

Removes the application session factory registered for the `sessionClass`.

Example 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. 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);
}
```

Sessions

`RawSessions`, `Sessions` and `ApplicationSessions` provide the means for dispatching and receiving messages. Specific implementation of `ApplicationSession` may provide non standard methods.

The `RawSession` and the `Session` life span is controlled entirely by the application. However, the `ApplicationSession` life time depends on the 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 the time stamp of this session creation.

`long getLastAccessedTime();`
: Returns the time stamp indicating the last sent or received 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 the user that the session should free any associated resource - it shall not be used anymore.

`Future<Message> send(Message message)`

Sends a message in async mode. The `Future` reference provides the means of accessing the answer once it is received

`void send(Message message, EventListener<Message, Message> listener, long timeOut, TimeUnit timeUnit)`

As above. Allows to specify the time out value for send operations.

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

Creates a Diameter message. It should be explicitly set either as a request or answer. Passed parameters are used to build messages.

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

As above. However, it also allows for the Hop-by-Hop and End-to-End Identifiers in the message header to be set. This method should be used to create answers.

`Message createMessage(Message message, boolean copyAvps);`

Clones a message and returns the created object. The copyAvps parameter defines whether basic AVPs (Session, Route and Proxy information) should be copied to the new object.

`void send(Message message, EventListener<Message, Message> listener)`

Sends a message. The answer will be delivered by the specified listener

`void send(Message message, EventListener<Message, Message> listener, long timeOut, TimeUnit timeUnit)`

As above. It also allows for the answer to be passed after 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;
}
```

Application Session Factories

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

Table 1. Application Factories

Factory class	Application type & id	Application	Reference
org.jdiameter.common.impl.app.acc.AccSessionFactoryImpl	AccountingId[0:3]	Acc	FC3588
org.jdiameter.common.impl.app.auth.AuthSessionFactoryImpl	Specific	Auth	RFC3588
org.jdiameter.common.impl.app.cca.CCASessionFactoryImpl	AuthId[0:4]	CCA	RFC4006
org.jdiameter.common.impl.app.sh.ShSessionFactoryImpl	AuthId[10415:16777217]	Sh	TS.29328, TS.29329
org.jdiameter.common.impl.app.cxdx.CxDxSessionFactoryImpl	AuthId[13019:16777216]	Cx	TS.29228, TS.29229
org.jdiameter.common.impl.app.cxdx.CxDxSessionFactoryImpl	AuthId[10415:16777216]	Dx	TS.29228, TS.29229
org.jdiameter.common.impl.app.acc.AccSessionFactoryImpl	AccountingId[10415:3]	Rf	S.32240
org.jdiameter.common.impl.app.cca.CCASessionFactoryImpl	AuthId[10415:4]	Ro	TS.32240

There is no specific factory for Ro and Rf. Those applications reuse the respective session and session factories.

Application IDs contain two numbers - [VendorId:ApplicationId].

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

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.

The Validator is only capable of checking structural integrity, not the content of the message.

The Validator performs the following checks:

Index

Checks that the AVPs are in the correct place. For example, `Session-Id` must always be encoded before any other AVP.

Multiplicity

Checks that the message AVPs occur the proper number of times. For example, the Session-ID should only be present once.

The `Validator` is called by the stack implementation. It is invoked after the message is received, but before it is dispatched to a remote peer.

This means that if the peer does not exist in the local peer table, the validator is not called, as the stack fails before calling it.

Validator Configuration

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

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

The configuration file contains more data that `Validator` uses to build its data base. This is because the `Dictionary` uses the same file to configure itself. It reuses the AVP definitions, with some extra information like AVP type and flags.

The configuration file has the following structure:

```
<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 that 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 messages 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 checked (ALL).

receiveLevel Determines the validation level for messages 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 checked (ALL).

<*vendor*>

Optional element that 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 the 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 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 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 that specifies the logical name of the application.

uri Optional attribute that specifies a link to additional application information.

Example 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_avp_attributes] contains the complete list of supported attributes, and their available values (if applicable). The <avp> element supports a number of child elements that are used to set finer parameters for the individual AVP. The supported elements are <type>, <enum>, and <grouped>.

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

Example 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> that 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 of the element. This is used to match the type-name value in the <typedefn> element.

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

<enum>

Child element of <avp> that 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.

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

<grouped>

Child element of <avp> that 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.

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

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

Table 2. <avp> Attributes

Attribute Name (optional in brackets)	Explicit Values (default in brackets)	Description
name		Specifies the name of the AVP. This 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.
(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 actual 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 mandatory in order to consume/process message.

Attribute Name (optional in brackets)	Explicit Values (default in brackets)	Description
vendor		Specifies the defined vendor code, which is used by the <command> child element

Example 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 that specifies the message name for descriptive purposes.

code Mandatory parameter that specifies the integer code of the message.

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

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 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>
```

Validator Source Overview

The ValidatorAPI defines methods to access its database of AVPs and check if the AVP and message have the proper structure.

The Validator is currently message oriented. This means that it declares methods that center on message consistency checks. The 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 the `Validator` is enabled.

public ValidatorLevel getSendLevel();

Returns the validation level of outgoing messages. It can have one of the following values: `OFF`, `MESSAGE`, `ALL`.

public ValidatorLevel getReceiveLevel()

Returns the validation level of incoming messages. It can have one of the following values: `OFF`, `MESSAGE`, `ALL`.

public void validate(Message msg, boolean incoming) throws JAvpNotAllowedException

Performs validation on a message. Based on the `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 messages with a specified level. It is a programatical way to allow different levels of validation from those configured. If validation fails, a `JAvpNotAllowedException` with details is thrown.

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

A simple example of a Validator use case is shown below:

Example 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());
}
```