Diameter Stack Validator

Table of Contents

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 1. <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. <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 3. <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 4. <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 1. <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 5. <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 6. <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 7. 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());
}
```