

mitesterforsip_Developing_Test_Scripts

Release Version 2.0

Table of Contents

[1] Introduction	3
[1.1] Document contains	3
[2] XML File components and syntax	3
[2.1] Syntax and Semantics of XML Script.....	3
[2.2] Structure of XML Script	3
[2.3] General Definition and Syntax of tags	4
[3] Writing Server Script for a Call Flow.....	4
[3.1] Server Script for Normal Scenario.....	4
[3.1.1] Adding Headers to a Message	5
[3.1.2] Request line / Status Line Manipulation	7
[3.1.3] Values copied from previous request/response	8
[3.1.4] Custom Headers	9
[3.1.5] Adding Media to the Message	9
[3.2] Server Script for Abnormal Scenario.....	9
[3.2.1] Deleting headers in a message	10
[3.2.2] Duplication	10
[3.2.3] Header manipulation with CRLF.....	12
[3.2.4] Call Forking	13
[4] Validation in mitester	14
[4.1] Rules of Validation	15
[5] Sample Server Scripts	16
[5.1] Sample Server Script 1.....	16
[5.2] Sample Server Script 2.....	17
[5.3] Sample Server Script 3.....	18
[5.4] Sample Server Script 4.....	19
[5.5] Sample Server Script 5.....	20

[1] Introduction

mitester for SIP is developed to process, build and control the call flows which simulate the SIP messages to the system under test.

XML has emerged as the standard for exchanging data across disparate systems, and Java technology provides a platform for building portable applications. Hence JAXB Parsers are used to parse the scripts in XML format, which feeds the scenario into mitester for SIP.

[1.1] Document contains

The document contains few sign boards for users to work with mitester for SIP.



Warning. Care should be taken on working with this process



Note. This point should be noted for future reference

[2] XML File components and syntax

Options are provided for the user to create one or more Server 9++9Scripts to accommodate Test Scenarios to be executed. All Scripts should confine to XML format and syntax.

[2.1] Syntax and Semantics of XML Script

mitester for SIP characterize few syntax and semantics to write the scripts.

[2.2] Structure of XML Script

A Server Script will hold the following tags in a sequential order. The hierarchical order of tags in an XML Script is as follows

<TEST_FLOW>

This is the root element of any Server Script. This tag should be the first and last tag of a Script. No additional parameters are supported for this tag. This remains the root tag and any number of test scenarios should embed inside this tag. Hence there exists only one TEST_FLOW tag in a script file.

<TEST>

This tag is very important to define the full details of a test scenario. The various mandatory attributes in this tag are TYPE, TEST-ID, and DESCRIPTION. The optional attributes are TEST-SUITE and TEST-SHEET.

<ACTION>

This tag is the main identifier to deal with the transaction of the messages.

The ACTION tag has two main parameters SEND and RECV. These parameters are used to define and generate the process of Request or Response in the particular scenario.

The SEND and RECV parameters will carry the values of 'req' and 'res' for Request and Response respectively.

'value' is another parameter which defines the 'SIP method name' and 'Response Code_SIP method name' if it is Request or Response respectively.



More number of ACTION tags will constitute the transactional flow of a Test Scenario.

<WAIT>

This tag denotes the time to be delayed in the server during the execution of each transaction in a Test Scenario.

Syntax :

```
<WAIT unit='sec/msec' value='time to be delayed' />
```

eg :

```
<WAIT unit='msec' value='5000' />
```

```
<WAIT unit='sec' value='5' />
```

```
<WAIT unit='sec' value='1.5' />
```

[2.3] General Definition and Syntax of tags

The most important syntax to note from the above tags declaration is sending and receiving the request and responses embedded in ACTION tag.



RECV='req'	To receive request
RECV='res'	To receive response
SEND='req'	To send request
SEND='res'	To send response

This parameter and value will decide the flow of transaction to be carried out while executing a Call flow.

[3] Writing Server Script for a Call Flow

The scripting syntax is categorized into two for easy understanding of syntax. They are

- Server script for normal scenario
- Server script for abnormal scenario

The server scripts are constructed at the server simulator end. For any incoming 'offer' (request or response) the server simulator will construct the 'answer' with the appropriate values. Provisions are there to create and incorporate the user defined headers with values for any particular message. A usual Server script will have the default tags like TEST_FLOW, TEST with its attributes, ACTION and WAIT as described above.



Scripting syntax defined for Server Scripts are applicable only for stimulating Server side transactions.

[3.1] Server Script for Normal Scenario

A normal scenario will hold the default tags like TEST_FLOW, TEST with its attributes, ACTION and WAIT. Whereas ACTION tag, which will carry its child tags to define the message, as per user's wish.

The default message construction at the server side will have the mandatory headers with appropriate values. The values are populated with respect to incoming message.



A message is populated with default values of headers and parameters when no HEADERS tags are added as nodes embedded inside the ACTION tag.

[3.1.1] Adding Headers to a Message

<ADD_MSG>

This tag is used to define the manipulation of a message with additional headers/parameters/values. If the user wishes to add any header, he/she can achieve this by scripting using ADD_MSG tag as the parent node with child nodes as headers.

<HEADERS>

This tag is used to define the headers and its parameters.

Syntax :

```
<HEADERS>
  <header name='header-name' value='header-value' >
    <param name='parameter-name' value='parameter-value' />
  </header>
</HEADERS>
```

eg 1 :

```
<HEADERS>
  <header name='max-forwards' value='170' />
</HEADERS>
```

'Max-Forwards: 170' is generated as a result of eg 1.

eg 2 :

```
<HEADERS>
  <header name='content-disposition' value='session' >
    <param name='handling' value='optional' />
  </header>
</HEADERS>
```

'Content-Disposition: session; handling=optional' is generated as a result of eg 2.

Each and every parameter value in a header can be defined as user specific through the above syntax.

<CONTENT>

This tag is used to add any content to the generated request or response message. The following detailed <SDP_BODY> MUST be embedded inside the CONTENT tag inside the ADD_MSG tag.

<SDP_BODY>

This tag is used to define the contents of the sdp body. A content-type header with sdp parameters are embedded in this tag. The parameters defined in that SDP BODY will be parsed as it is and it will not make any change in order while constructing the content.

Syntax :

```
<SDP_BODY>
  <header name='header-name' value='header-value' />
  <sdp name='sdp-name' value='sdp-value' />
</SDP_BODY>
```

eg :

```
<SDP_BODY>
  <header name='content-type' value='application/sdp' />
  <sdp name='v' value='0' />
  <sdp name='o' value='sip:a@127.0.0.1 234435 234435 IN IP4 127.0.0.1' />
  <sdp name='s' value='peer Session SDP' />
  <sdp name='c' value='IN IP4 127.0.0.1' />
  <sdp name='t' value='0 0' />
  <sdp name='m' value='audio 8000 RTP/AVP 97' />
  <sdp name='a' value='ptime:100' />
  <sdp name='a' value='maxptime:100' />
</SDP_BODY>
```

As a result the above mentioned example,

- 'Content-Type: application/sdp' is added to the generated SIP message along with the headers.
- The following sdp body is attached to the end of the generated message.

```
v=0
o=sip:a@127.0.0.1 234435 234435 IN IP4 127.0.0.1
s=peer Session SDP
c=IN IP4 127.0.0.1
t=0 0
m=audio 8000 RTP/AVP 97
a=ptime:100
a=maxptime:100
```

Other than SDP body, XML_BODY and TEXT_BODY can also be included in the SIP Message, by Specifying the Name of the file in the Server Script and the files to be in the "Content_Files" folder which is in the Server Scripts folder.

<XML_BODY>

This tag is used to define the XML body. A content-type header with the name of the XML body are embedded in this tag. The content defined in the XML will be parsed as it is and it will not make any change in order while constructing the content.

Syntax:

```
<XML_BODY>
  <header name='header-name' value='header-value' />
  <file source='Name of XML file' />
</XML_BODY>
```



Example of XML body to be placed in the "Content_Files" folder

<TXT_BODY>

This tag is used to define the TEXT body. A content-type header with the name of the TEXT body are embedded in this tag. The content defined in the TEXT file will be parsed as it is and it will not make any change in order while constructing the content.

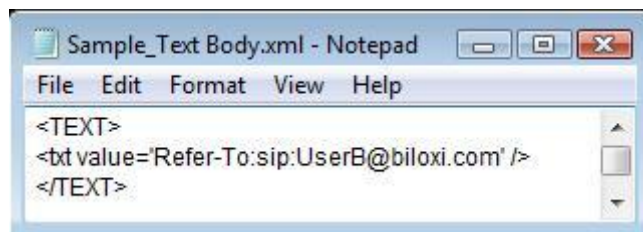
Syntax:

<TXT_BODY>

<header name='header-name' value='header-value'/>

<file source='Name of TEXT body file'/>

</TXT_BODY>



Example of TEXT body to be placed in the "Content_Files" folder



The Content-Length of the sdp body attached is automatically valued and added to the generated message.
File extension of XML and TEXT body should be .xml

[3.1.2] Request line / Status Line Manipulation

A request line / status line will be constructed by default based on the 'value' parameter in ACTION tag.

Syntax for Request line :

<ADD_MSG>

<LINE>

<fline name='req-line' />

```

    <req-line name='name' value='value' />
  </LINE>
</ADD_MSG>

```

Syntax for Status line :

```

<ADD_MSG>
  <LINE>
    <fline name='status-line' />
    <status-line name='name' value='value' />
  </LINE>
</ADD_MSG>

```

ADD_MSG tag is used to define, adding something to a message. LINE tag is used to change the content of Request line or Status line. A 'fline' tag (firstline) with 'name' parameter is used to define whether it is a Request line or Status line.

The values in 'name' parameters are reserved ones. Only values of 'req-line' or 'status-line' should be assigned to name parameter. Depending on the value of 'name' parameter in firstline, the parameters constituting the first line varies.

eg 1 :

```

<LINE>
  <fline name='req-line' />
  <req-line name='method' value='INVITE' />
  <req-line name='req-uri' value='sip:sandy@alexa.com' />
  <req-line name='sip-version' value='SIP/2.0' />
</LINE>

```

'INVITE sip:sandy@alexa.com SIP/2.0' is the request line generated as a result of eg 1.

eg 2 :

```

<LINE>
  <fline name='status-line' />
  <status-line name='sip-version' value='SIP/2.0' />
  <status-line name='status-code' value='200' />
  <status-line name='reason-phrase' value='OK' />
</LINE>

```

'SIP/2.0 200 OK' is the status line generated as a result of eg 2.

All the parameters or only a single parameter in Request line / Status line can be modified according to the user's request.

[3.1.3] Values copied from previous request/response

A successful scenario will definitely have the dependent values in most of the mandatory and other headers/parameters which will be carried throughout the flow of a scenario. In such cases the headers/parameters which requires to carry the same value from the previous message can be achieved using '***' (triple star) in the value parameter of headers.



Copying of values through scripting with *** is applicable only when the value to be copied is available in the previous message.

[3.1.4] Custom Headers

Any header portrayed by the user with his own value is called a 'Custom header'. The Custom header need not be confined to RFC standards. Injecting fault in the default headers can also be achieved by defining the particular header as Custom headers.

Syntax :

```
<ADD_MSG>
  <HEADERS>
    <header name='header-name' value='header-value' />
  </HEADERS>
</ADD_MSG>
```

[3.1.5] Adding Media to the Message

<MEDIA> tag is used to add Media (Audio/Video) to the SIP Messages. Media files can be RTP and RTCP. All RTCP packet types RTCP SR, RTCP RR, RTCP SDES and RTCP APP are supported.

Syntax:

```
<ACTION SEND='media' value='RTP/RTCP' >
  <MEDIA>
    <file source='Path and Name of the Media file' />
  </MEDIA>
</ACTION>
```

Eg:

```
<ACTION SEND='media' value='RTP' >
  <MEDIA>
    <file source='media/RTP.txt' />
  </MEDIA>
</ACTION>
```

mitester can receive the media from the SUT. The syntax to receive media is;

```
<ACTION RECV='media' value='Name of the media to be received' >
  </ACTION>
```

Eg:

```
<ACTION RECV='media' value='RTCP_SR' >
  </ACTION>
```



The media files to be added to the SIP messages should be placed in the folder and the related path and name of the file should be included in the Server Script. The extension of the media files can be .txt and .pcap.



To successfully simulate the media call flows, the 127.0.0.1 IP address specified in the test script must be set to current machine IP address in which SIP application and mitester run.

[3.2] Server Script for Abnormal Scenario

Abnormal Test Scripts are webbed around the following features taken into consideration.

[3.2.1] Deleting headers in a message

Deleting headers is the process of removing a particular header/parameter and its value. Using this tag the value of a parameter can even be removed as required by the user.

A complete header with parameters can be removed using <DEL_MSG> tag.

Syntax :

```
<DEL_MSG>
  <HEADERS>
    <header name='header-name' />
  </HEADERS>
</DEL_MSG>
```

eg :

```
<DEL_MSG>
  <HEADERS>
    <header name='cseq' />
  </HEADERS>
</DEL_MSG>
```

As a result of the above example, 'Cseq' header is deleted while generating the message.

[3.2.2] Duplication

Duplicating the entire message

Duplication is the process of repetitive occurrence of headers/parameters and its value in a SIP request / response message.

Syntax :

```
<ACTION SEND='req/res' value='method-name/response-code_method-name' count='number'>
</ACTION>
```

The 'count' parameter in the syntax claims the process of sending the number of request or response messages. Depending on the count value, the messages are generated and sent during execution.

eg :

```
<ACTION SEND='req' value='INVITE' count='2' >
</ACTION>
```

As a result of the above example, 'INVITE' request is sent twice during execution.

Duplicating the headers in a message

Headers in a request / response message can be duplicated.

Syntax :

```
<DUP_MSG>
  <HEADERS>
    <header name='header-name' count='number' />
  </HEADERS>
</DUP_MSG>
```

eg :

```
<DUP_MSG>
  <HEADERS>
    <header name='cseq' count='2' />
  </HEADERS>
</DUP_MSG>
```

As a result of the above example, 'cseq' header is duplicated while generating the message. Depending on the count value, the headers in the messages are generated and sent during execution.

Duplicating the header parameters

Header Parameters in a request / response message can be duplicated.

Syntax :

```
<DUP_MSG>
  <HEADERS>
    <header name='header-name' >
      <param name='parameter-name' count='number' />
    </header>
  </HEADERS>
</DUP_MSG>
```

eg :

```
<DUP_MSG>
  <HEADERS>
    <header name='Via' >
      <param name='branch' count='5' />
    </header>
  </HEADERS>
</DUP_MSG>
```

As a result of the above example, 'branch' parameter in Via header is duplicated. Depending on the count value, the specific header parameter will be duplicated during execution.

Duplicating the entire test case

Load testing can be performed by executing the test cases, 'n' number of times. The number of times that the test case has to be executed is specified in the "COUNT" (upper-case) parameter included inside the "TEST" tag.

Syntax :

```
<TEST TYPE = 'test-type' TEST-ID = 'test-id' DESCRIPTION = 'test-description'
COUNT='number' >
  ...
</TEST>
```

eg :

```
<TEST TYPE = 'Normal' TEST-ID = 'SS-1' DESCRIPTION = 'Registration' COUNT='5' >
...
</TEST>
```

As a result, the test case “SS-1” will be executed '5' times. Depending on the value mentioned in the “COUNT” parameter, the specific test case will be executed.

[3.2.3] Header manipulation with CRLF

CRLF plays a major role in defining a message. Without CRLF, the complete SIP message will be incomprehensible. RFCs insist on finite number of CRLF at places required. Each header should be separated at least through a single pair of CRLF.



Appropriate numbers of CRLFs can be added while constructing the message by mitester for SIP.

Adding CRLF

Any number of CRLF can be added to the end of the headers.

Syntax :

```
<ADD_CRLF>
<HEADERS>
  <header name='header-name' count='number' />
</HEADERS>
</ADD_CRLF>
```

The ‘count’ parameter in the syntax claims the process of adding the number of CRLF. Depending on the count value, the CRLF pairs are added.

eg :

```
<ADD_CRLF>
<HEADERS>
  <header name='from' count='2' />
</HEADERS>
</ADD_CRLF>
```

As a result of the above example, two pairs of CRLF are added at the end of 'From' header.



The user can add CRCLF, LFLF pairs at the end of headers. The syntax tag varies according to the usage of the CRLF.

Removing CRLF

CRLF can be removed from the end of the headers.

Syntax :

```
<DEL_CRLF>
<HEADERS>
  <header name='header-name' />
</HEADERS>
</DEL_CRLF>
```

eg :

```
<DEL_CRLF>
  <HEADERS>
    <header name='from' />
  </HEADERS>
</DEL_CRLF>
```

As a result of the above example, CRLF is removed at the end of 'From' header.



The user can remove CR / LF at the end of headers. The syntax tag varies according to the removal.



The Request line and Status line can also be subjected to the above fault Injections.

[3.2.4] Call Forking

mitester for SIP supports Call Forking. Generates request or response messages with user-specified dialog, irrespective of the current dialog. For the purpose of Call Forking, a new parameter “dialog” is added to the server test script.

Syntax :

```
<ACTION SEND='req/res' value='method-name/response-code_method-name' dialog='already-existing-dialog' />
```

eg :

```
<ACTION SEND='req' value='NOTIFY' dialog='INVITE' />
```

As a result, mitester generates and sends NOTIFY request with already existing INVITE dialog irrespective of the current dialog.

[4] Validation in mitester

Validation is done in two ways using the Validation.xml file.

- i) Checking Presence of headers
- ii) Checking the Syntax of headers

To check for Validation, Validation.xml file should be placed in the mitester package folder.

I) Checking Presence of headers

Validation.xml file is used to predefine and check the presence of the defined header in the incoming SIP message.

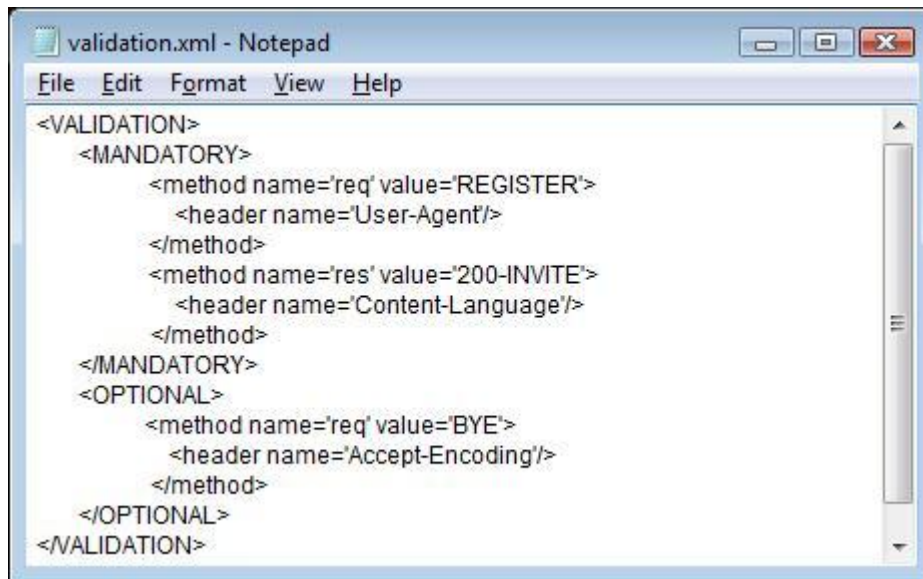
mitester will check the presence of header in the incoming SIP messages, only if CHECK_PRESENCE_OF_HEADER is enabled in the mitester.properties file.

ii) Checking the syntax of headers

mitester will check the header syntax of the headers specified in the Validation.xml file, if VALIDATION is enabled in the mitester.properties file. If VALIDATION is enabled, "**validation.properties**" file should be placed inside the 'lib' folder. This file consists of the regular expression of all the headers. Users can also define their own value to the headers in the validation.properties file. mitester validates the headers' syntax against the validation.properties file. Syntax of headers in Validation.xml file

```
<VALIDATION>
  <MANDATORY>
    <method name='req' value='SIP Request Message'>
      <header name='Header'/>
    </method>
    <method name='res' value='SIP Response Message'>
      <header name='Header'/>
    </method>
  </MANDATORY>
  <OPTIONAL>
    <method name='req' value=' SIP Request Message '>
      <header name='Header'/>
    </method>
  </OPTIONAL>
</VALIDATION>
```

Eg:



```
<VALIDATION>
  <MANDATORY>
    <method name='req' value='REGISTER'>
      <header name='User-Agent'/>
    </method>
    <method name='res' value='200-INVITE'>
      <header name='Content-Language'/>
    </method>
  </MANDATORY>
  <OPTIONAL>
    <method name='req' value='BYE'>
      <header name='Accept-Encoding'/>
    </method>
  </OPTIONAL>
</VALIDATION>
```

[4.1] Rules of Validation

- I) CHECK_PRESENCE_OF_HEADER will have the priority if both CHECK_PRESENCE_OF_HEADER and VALIDATION are enabled.
- II) MANDATORY and OPTIONAL tags do not take effect if CHECK_PRESENCE_OF_HEADER only is enabled. mitester will treat the headers present inside the OPTIONAL tags as MANDATORY one, validates and puts FAIL if not present in the incoming SIP message.
- III) mitester will treat the headers (MANDATOR or OPTIONAL) as specified in the Validation.xml file, If VALIDATION is enabled. For example, if headers specified inside the OPTIONAL tags are not found in the incoming SIP message, then mitester will not put the result FAIL. It will ignore the header and will not validate, since it is OPTIONAL.

[5] Sample Server Scripts

The following are the snippets of server scripts which pave easy way for the tester. These few commonly used server scripts are for your immediate reference.

[5.1] Sample Server Script 1

```
<TEST_FLOW>
  <TEST TYPE = 'Normal' TEST-ID = 'SS-1'

    DESCRIPTION = ' SUT sends First REGISTER request
                  mitester sends 401 REGISTER response,
                  SUT sends Second REGISTER request,
                  mitester sends 200 REGISTER response,
                  SUT sends De-REGISTER request,
                  mitester sends 200 REGISTER response' >

    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='401_REGISTER' >
      <ADD_MSG>
        <HEADERS>
          <header name='www-authenticate' value='Digest'>
            <param name='realm' value='atlanta.com' />
            <param name='domain' value='sip:boxesbybob.com' />
            <param name='qop' value='auth' />
            <param name='nonce' value='f84f1cec41e6cbe5aea9c8e88d359' />
            <param name='opaque' value='NULL' />
            <param name='stale' value='FALSE' />
            <param name='algorithm' value='MD5' />
          </header>
        </HEADERS>
      </ADD_MSG>
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>

  </TEST>
</TEST_FLOW>
```


[5.2] Sample Server Script 2

```

<TEST_FLOW>
  <TEST TYPE = 'Normal' TEST-ID = 'SS-2'

    DESCRIPTION = ' SUT sends REGISTER request,
                  mitester sends 200 REGISTER response,
                  mitester sends INVITE request,
                  SUT sends 200 INVITE response,
                  mitester sends ACK request,
                  SUT sends BYE request,
                  mitester sends 200 BYE response,
                  SUT sends De-REGISTER request,
                  mitester sends 200 REGISTER response' >

    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>
    <ACTION SEND='req' value='INVITE' >
      <ADD_MSG>
        <CONTENT>
          <SDP_BODY>
            <header name='content-type' value='application/sdp' />
            <sdp name='v' value='0' />
            <sdp name='o' value='sip:UserA@127.0.0.1 234435 234435 IN IP4 127.0.0.1' />
            <sdp name='s' value='peer Session SDP' />
            <sdp name='c' value='IN IP4 127.0.0.1' />
            <sdp name='t' value='0 0' />
            <sdp name='m' value='audio 5000 RTP/AVP 0 8 97 3 5 4' />
            <sdp name='a' value='rtptime:4 G723/8000' />
            <sdp name='a' value='fmtp:4 annexa=no;bitrate=6.3' />
            <sdp name='m' value='video 5002 RTP/AVP 99 34 26 31' />
            <sdp name='a' value='rtptime:99 H264/90000' />
            <sdp name='a' value='maxptime:100' />
            <sdp name='a' value='fmtp:99 packetization-mode=1' />
            <sdp name='a' value='recvonly' />
          </SDP_BODY>
        </CONTENT>
      </ADD_MSG>
    </ACTION>
    <ACTION RECV='res' value='200_INVITE'>
    </ACTION>
    <ACTION SEND='req' value='ACK' >
    </ACTION>
    <ACTION RECV='req' value='BYE' >
    </ACTION>
    <ACTION SEND='res' value='200_BYE'>
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>

  </TEST>
</TEST_FLOW>

```

[5.3] Sample Server Script 3

```

<TEST_FLOW>
  <TEST TYPE = 'Normal' TEST-ID = 'SS-3'

    DESCRIPTION = ' SUT sends REGISTER request,

                    mitester sends 200 REGISTER response,
                    SUT sends INVITE request,
                    mitester sends 200 INVITE response,
                    SUT sends ACK request,
                    mitester sends BYE request,
                    SUT sends 200 BYE response,
                    SUT sends De-REGISTER request,
                    mitester sends 200 REGISTER response' >

    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>
    <ACTION RECV='req' value='INVITE' >
    </ACTION>
    <ACTION SEND='res' value='200_INVITE'>
    <ADD_MSG>
      <CONTENT>
        <SDP_BODY>
          <header name='content-type' value='application/sdp' />
          <sdp name='v' value='0' />
          <sdp name='o' value='sip:UserA@127.0.0.1 234435 234435 IN IP4 127.0.0.1' />
          <sdp name='s' value='peer Session SDP' />
          <sdp name='c' value='IN IP4 127.0.0.1' />
          <sdp name='t' value='0 0' />
          <sdp name='m' value='audio 5000 RTP/AVP 0 8 97 3 5 4' />
          <sdp name='a' value='rtpmap:4 G723/8000' />
          <sdp name='a' value='fmtp:4 annexa=no;bitrate=6.3' />
          <sdp name='m' value='video 5002 RTP/AVP 99 34 26 31' />
          <sdp name='a' value='rtpmap:99 H264/90000' />
          <sdp name='a' value='maxptime:100' />
          <sdp name='a' value='fmtp:99 packetization-mode=1' />
          <sdp name='a' value='recvonly' />
        </SDP_BODY>
      </CONTENT>
    </ADD_MSG>
    </ACTION>
    <ACTION RECV='req' value='ACK' >
    </ACTION>
    <ACTION SEND='req' value='BYE' >
    </ACTION>
    <ACTION RECV='res' value='200_BYE'>
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>

  </TEST>
</TEST_FLOW>

```

[5.4] Sample Server Script 4

```

<TEST_FLOW>
  <TEST TYPE = 'Normal' TEST-ID = 'SS-4'

    DESCRIPTION = ' SUT sends REGISTER request,
                  mitester sends 200 REGISTER response,
                  mitester sends INVITE request,
                  SUT sends 486 INVITE response,
                  mitester sends ACK request,
                  SUT sends De-REGISTER request,
                  mitester sends 200 REGISTER response' >

    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>
    <ACTION SEND='req' value='INVITE' >
      <ADD_MSG>
        <CONTENT>
          <SDP_BODY>
            <header name='content-type' value='application/sdp' />
            <sdp name='v' value='0' />
            <sdp name='o' value='sip:UserA@127.0.0.1 234435 234435 IN IP4 127.0.0.1' />
            <sdp name='s' value='peer Session SDP' />
            <sdp name='c' value='IN IP4 127.0.0.1' />
            <sdp name='t' value='0 0' />
            <sdp name='m' value='audio 5000 RTP/AVP 0 8 97 3 5 4' />
            <sdp name='a' value='rtpmap:4 G723/8000' />
            <sdp name='a' value='fmtp:4 annexa=no;bitrate=6.3' />
            <sdp name='m' value='video 5002 RTP/AVP 99 34 26 31' />
            <sdp name='a' value='rtpmap:99 H264/90000' />
            <sdp name='a' value='maxptime:100' />
            <sdp name='a' value='fmtp:99 packetization-mode=1' />
            <sdp name='a' value='recvonly' />
          </SDP_BODY>
        </CONTENT>
      </ADD_MSG>
    </ACTION>
    <ACTION RECV='res' value='486_INVITE'>
    </ACTION>
    <ACTION SEND='req' value='ACK' >
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>

  </TEST>
</TEST_FLOW>

```

[5.5] Sample Server Script 5

```
<TEST_FLOW>
  <TEST TYPE = 'Normal' TEST-ID = 'SS-5'

    DESCRIPTION = ' SUT sends REGISTER request,
                  mitester sends 200 REGISTER response,
                  SUT sends INVITE request,
                  mitester sends 180 INVITE response,
                  SUT sends CANCEL request,
                  mitester sends 200 CANCEL response,
                  mitester sends 487 INVITE response,
                  SUT sends ACK request,
                  SUT sends De-REGISTER request,
                  mitester sends 200 REGISTER response' >

    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>
    <ACTION RECV='req' value='INVITE' >
    </ACTION>
    <ACTION SEND='res' value='180_INVITE'>
    </ACTION>
    <ACTION RECV='req' value='CANCEL' >
    </ACTION>
    <ACTION SEND='res' value='200_CANCEL'>
    </ACTION>
    <ACTION SEND='res' value='487_INVITE'>
    </ACTION>
    <ACTION RECV='req' value='ACK' >
    </ACTION>
    <ACTION RECV='req' value='REGISTER'>
    </ACTION>
    <ACTION SEND='res' value='200_REGISTER' >
    </ACTION>

  </TEST>
</TEST_FLOW>
```