# APITest XML Test Description format
# **DRAFT**

William McLendon
wcmclen@sandia.gov



# Overview

This document describes the grammar used by APItest for describing a test instance. Tests are scripted in XML with an XML Schema grammar provided. At this time, this document is not intended as a user-guide, but we attempt to insert useful information about how the XML test specifications are interpreted and executed by APItest. The test specification files should be named with the extension ".api".

We describe the individual elements that are used in a test specification file. In each section we provide a bit of schema description for that element and describe its usage and allowed attribute data where appropriate. Some extra attention is given to the dependency system. Example tests are provided and a full listing of the test specification schema grammar is also provided.

## Element: testCollection

testCollection is the root-level element that contains all the test specification. It can contain three elements: *name*, *info*, and *testList*. *name* is an optional element that would contain documentation about the particular test.

### XML Example:

```
<testCollection>
  <name>test1</name>
  <info> … </info>
  <testList> … </testList>
</testCollection>
```

### Schema Definition:

```
<xs:element name="testCollection">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="name"    type="xs:string"   minOccurs='1' maxOccurs='1'/>
         <xs:element name="info"    type="t_info"      minOccurs='0' maxOccurs='1'/>
         <xs:element name="testList" type="t_testList" minOccurs='1' maxOccurs='1'/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

## Element: *name*

The element *name* is fairly self explanatory. The CDATA of this element stores the name assigned to this test script. At the time of this writing, the *name* element is required, but is not used other than for reporting. In the future, we plan to use *name* for organizing multiple test files into suites.

### Attributes

none

### Description

*CDATA* – name of test file.

# Element: *info*

This element is a sub-element of *testCollection*. The purpose of this group is to provide information about the test described in this script. The information stored is *author* (person who wrote the test), *contact* (person responsible for test ie. Contact person for when the test fails), and a *description* of what this test is trying to accomplish. Table 1 describes the attributes for *author* and *contact*.

table 1: *author* and *contact* attribute definitions

| name | use | type | default | allowable values |
|---|---|---|---|---|
| name | required | string | '' | |
| email | optional | String | '' | |
| phone | optional | String | '' | |

1) **name** – name of the author or contact (depending on which element this is in)
2) **email** – email address of this person
3) **phone** – phone number where this person might be reached

## XML Example

```
<info>
   <author … />                                (optional)
   <contact …/>                                (optional)
   <description>…</description>                (optional)
</info>
```

## Schema Definition

```
<!-- Defined type for information -->
<xs:complexType name="t_info">
  <xs:sequence>
    <xs:element name="author" minOccurs='0' maxOccurs='1'>
      <xs:complexType>
        <xs:attribute name="name"  type="xs:string" use='required' />
        <xs:attribute name="email"  type="xs:string" use='optional' />
        <xs:attribute name="phone" type="xs:string" use='optional' />
      </xs:complexType>
    </xs:element>
    <xs:element name="contact" minOccurs='0' maxOccurs='1'>
      <xs:complexType>
        <xs:attribute name="name"  type="xs:string" use='required' />
        <xs:attribute name="email"  type="xs:string" use='optional' />
        <xs:attribute name="phone" type="xs:string" use='optional' />
      </xs:complexType>
    </xs:element>
    <xs:element name="description" type="xs:string" minOccurs='0' maxOccurs='1'/>
  </xs:sequence>
</xs:complexType>
```

## Element: *testList*

*testList* contains list of tests to perform. Valid test types are: *ssslibTest*, *shellTest*, *tcpipTest*, *httpTest*, and *portalsTest*. These elements may appear in any sequence.

Order of execution can only be guaranteed through explicit assignment of dependencies. One can not assume top-down execution order.

### testList Attributes

Table 2: testList attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| numReps | optional | integer | 1 | positive values |
| preDelay | optional | float | 0.0 | positive values |
| postDelay | optional | float | 0.0 | positive values |
| iterDelay | optional | float | 0.0 | positive values |

4) **numReps** – the number of times the *list of tests* should be repeated
5) **preDelay** – the delay (in seconds) to pause before running any tests after APItest is started.
6) **postDelay** – the delay (in seconds) to pause after *all* tests are finished prior to exiting.
7) **iterDelay** – the delay (in seconds) to pause between iterations. Note, this is relevant only when *numReps* > 1.

### XML Example

```
<testList>
   <ssslibTest … > … </ssslibTest>
   <shellTest … > … </shellTest>
   <tcpipTest … > … </tcpipTest>
   <httpTest … > … </httpTest>
</testList>
```

### Schema Definition

```
<xs:complexType name="t_testList">
  <xs:complexContent>
    <xs:extension base='t_testListElement'>
      <xs:sequence minOccurs='0' maxOccurs='unbounded'>
        <xs:choice>
          <xs:element name="ssslibTest" type="t_ssslibTest" minOccurs='0'/>
          <xs:element name="shellTest"  type="t_shellTest"  minOccurs='0'/>
          <xs:element name="tcpipTest"  type="t_tcpipTest"  minOccurs='0'/>
          <xs:element name="httpTest"   type="t_httpTest"   minOccurs='0'/>
        </xs:choice>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

All tests share a set of attributes that contain metadata used by the APItest framework for handling test execution as well as result handling. These attributes are described in Table 3. Attributes that are specific to particular test types will be described with the individual test type specifications.

## Standard Test Metadata Attributes

Table 3: standard test metadata attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| name | required | string | '' | |
| numreps | optional | integer | 1 | |
| minPctMatch | optional | float | 0.0 | |
| maxPctMatch | optional | float | 100.0 | |
| preDelay | optional | float | 0.0 | |
| postDelay | optional | float | 0.0 | |
| iterDelay | optional | float | 0.0 | |
| onMismatch | optional | string | CONTINUE | CONTINUE,BREAK,HALT |

8) **name** – a text string that names each test. This name must be unique for each test. Dependencies are performed based on test names.
9) **numreps** – in integer, defaulting to 1. This specifies how many times *this test* should be interated.
10) **minPctMatch** – a floating point value between 0.0 and 100.0, defaulting to 0.0. This sets a lower-bound on the percent of test runs that must pass (ie. the actual output matches the expected output) for the overall run to be considered to have PASSED.
11) **maxPctMatch** – a floating point value between 0.0 and 100.0, defaulting to 100.0. This sets the upper-bound on the percent of test runs that must pass (ie. the actual output matches the expected output) for the overall test run result to be considered a PASS.
12) **preDelay** – a floating point value enumerating the number of seconds that apitest should wait prior to running this test sequence. Default value is 0.0.
13) **postDelay** – a floating point value enumerating the number of seconds that apitest should wait after executing this test sequence before going on to the next test. The default value is 0.0
14) **iterDelay** – a floating point value enumerating the number of seconds that apitest should wait between tests in the current sequence. The default value is 0.0.
15) **onMismatch** – a text string limited to either “CONTINUE”, “HALT”, or “BREAK” which tells APItest what to do if the actual and expected test results do not match. The default value is “CONTINUE”
    a. CONTINUE – Continue running subsequent tests with no interruption.
    b. BREAK – quit looping the immediate test but continue with the next test in testList.
    c. HALT – stop ALL tests and exit APItest wit appropriate error messages.

## Element: *ssslibTest*

*ssslibTest* contains a description of a test using the ssslib library and protocols. Use of this test type requires the ssslib suite to be installed on the computer running APItest.

### ssslibTest Attributes

Table 4: *ssslibTest* specific attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| destination | required | string | '' | |

16) **destination** – a text string that specifies the *destination* for the buffer. *destination* is the name of a service registered in the service directory. Transmission is handled via the standard ssslib communications library.

### XML Example

```
<ssslibTest>
   <input … > … </input>
   <output … > … </output>
   <dependencies> … </dependencies>
</ssslibTest>
```

### The *output* Element for ssslibTest

17) **type** - Specifics what output stream this expected buffer is associated with. Allowable values are:
    a. **recv** – received buffer from socket.
18) **format** – Specifies the format of the CDATA buffer. This value controls how APItest will validate the actual return with this expected buffer.
    a. **TXTSTR** – CDATA is a text string. Use exact matching.
    b. **REGEXP** – CDATA is a regular expression.

### The *input* Element for ssslibTest

19) **type** – Specifies the buffer that we wish to transmit. (ie. input to the component under test)
    a. **send** – specifies this input element is a send buffer.

### Schema Definition

```
<xs:complexType name="t_ssslibTest">
  <xs:complexContent>
    <xs:extension base='t_ssslibTestElement'>
      <xs:sequence>
        <xs:element name='input' type='t_input' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='output' type='t_output' minOccurs='0'
maxOccurs='unbounded'/>
        <xs:element name='dependencies'          type='t_dependencies' minOccurs='0'
maxOccurs='1'/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

</xs:complexType>

## Element: *shellTest*

*shellTest* specifies a command to be run via a command shell.

### shellTest Attributes

Table 5: *shellTest* specific attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| command | required | string | '' | |

20) **command** – string containing the command line to be executed.

### XML Example

```
<shellTest>
   <output … > … </output>
   <dependencies> … </dependencies>
</shellTest>
```

### The *output* Element for shellTest

21) **type** – Specifics what output stream this expected buffer is associated with. Allowable values are:
    a. **stderr** – associates the expected buffer with standard error of the command.
    b. **stdout** – associates the expected buffer with standard output of the command.
    c. **status** – associates the expected buffer with the exit status of the command.
22) **format** – Specifies the format of the CDATA buffer. This value controls how APItest will validate the actual return with this expected buffer.
    a. **TXTSTR** – CDATA is a text string. Use exact matching.
    b. **REGEXP** – CDATA is a regular expression.

### Schema Definition

```
<xs:complexType name="t_shellTest">
  <xs:complexContent>
    <xs:extension base='t_shellTestElement'>
      <xs:sequence>
        <xs:element name='output' type='t_output' minOccurs='0'
maxOccurs='unbounded'/>
        <xs:element name='dependencies' type='t_dependencies' minOccurs='0'
maxOccurs='1' />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

# Element: *tcpipTest*

## tcpipTest Attributes

Table 5: *tcpipTest* specific attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| desturl | required | string | '' | |
| destport | required | integer | - | any valid port number |
| clearRecvBuf | optional | t_bool | | YES, NO |
| comm. | optional | commOpts | KEEPALIVE | KEEPALIVE, CLOSE, CLOSEITER |

23) **desturl** – destination URL for the buffer we want to transmit.
24) **destport** – destination port for the buffer we want to transmit.
25) **clearRecvBuf** – YES/NO – do we clear the receive buffer between receives
26) **comm** – Tells how to handle the socket
    a. **KEEPALIVE** – keep the socket open between iterations
    b. **CLOSE** – close the socket after this test is finished, but keep it open over multiple iterations.
    c. **CLOSEITER** – close the socket after EVERY iteration

## XML Example

```
<tcpipTest>
  <input … > … </input>
  <output … > … </output>
  <dependencies> … </dependencies>
</tcpipTest>
```

## The *output* Element for tcpipTest

27) **type** - Specifics what output stream this expected buffer is associated with. Allowable values are:
    a. **recv** – received buffer from socket.
28) **format** – Specifies the format of the CDATA buffer. This value controls how APItest will validate the actual return with this expected buffer.
    a. **TXTSTR** – CDATA is a text string. Use exact matching.
    b. **REGEXP** – CDATA is a regular expression.

## The *input* Element for tcpipTest

29) **type** – Specifies the buffer that we wish to transmit. (ie. input to the component under test)
    a. **send** – specifies this input element is a send buffer.

## Schema Definition

```
<xs:complexType name="t_tcpipTest">
 <xs:complexContent>
  <xs:extension base='t_tcpipTestElement'>
   <xs:sequence>
    <xs:element name='input' type='t_input' minOccurs='0' maxOccurs='unbounded'/>
    <xs:element name='output' type='t_output' minOccurs='0'
maxOccurs='unbounded'/>
    <xs:element name='dependencies' type='t_dependencies' minOccurs='0'
maxOccurs='1'/>
```

```
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

## Element: *httpTest*

Work in progress

## Element: *portalsTest*

Work in progress

## Element: *output*

The *output* element describes an expected result buffer that is to be matched against the actual output from the component under test. The schema definition does not preclude multiple *output* buffers with the same tag from being listed, but the handling of this will be test dependent.

Table 5: *output* attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| format | optional | string | REGEXP | REGEXP, TXTSTR |
| type | required | string | - | dependent upon test type |

30) **format** – Specifies the format of the CDATA buffer.
    a. **REGEXP** – regular expression matching.
    b. **TXTSTR** – text string (match via diff)
31) **type** – Describes the type of buffer we have. This functions as a tag and is test-dependent. For more detail, see the shellTest, ssslibTest, etc. sections.

### XML Example

```
<output format='REGEXP' type='stdout'>.*stdout.*</output>
```

This example shows an expected output buffer for a shellTest. The *format* attribute tells us that the CDATA is a regular expression, in this case any buffer that contains the word 'stdout' at least once will match. The *type* attribute stores the value 'stdout', which is used to tell APItest that this expected result is associated with the stdout stream.

### Schema Definition

```
<xs:complexType name='t_output'>
  <xs:simpleContent>
    <xs:extension base='xs:string'>
      <xs:attribute name='format' type='bufType' use='optional'/>
      <xs:attribute name='type' type='xs:string' use='required'/>
    </xs:extension>
  </xs:simpleContent>
```

</xs:complexType>

## Element: *input*

Table 5: *input* attributes

| name | use | type | default | allowable values |
|---|---|---|---|---|
| type | required | string | - | dependent upon test type. |

### XML Example

<**input** type='send'>Hello World!</**input**>

This shows an *input* element for a tcpipTest that would send the text string “Hello World!” to the destination component.

### Schema Definition

```
<xs:complexType name='t_input'>
  <xs:simpleContent>
    <xs:extension base='xs:string'>
      <xs:attribute name='type' type='xs:string' use='required'/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

## Element: *dependencies*

The test dependency system used by APItest allows for some fairly complex inter-test dependencies to be created. A test can depend on another test purely in a temporal sense (ie. Test B is preceded by Test A and will execute regardless of how test A did). We can also complicate this by adding a condition to the prerequisite such as: Test B executes iff test A executed and >= 80% of the time test A's expected and actual results matched.

We further enhanced this by allowing not only multiple dependencies to be specified for a given test, but allow boolean expressions in the dependencies. An example of this would be: "Test A depends on tests B,C,D and will execute only if B and C both matched expected results or if D executed. The structure of these dependencies can be represented as a DAG. We will provide a couple of examples illustrating how this works.

The *dependencies* element has only one child element that can be any one of: *dependency, AND, OR, NOR, NAND,* or *NOT.*

### Schema Definition

```
<xs:complexType name='t_dependencies'>
  <xs:choice maxOccurs='1'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:group ref='g_Logical' minOccurs='0' maxOccurs='unbounded'/>
  </xs:choice>
</xs:complexType>

<xs:group name='g_Logical'>
  <xs:choice>
    <xs:element name='OR'            type='t_or' />
    <xs:element name='AND'   type='t_and' />
    <xs:element name='NOR'   type='t_nor' />
    <xs:element name='NAND' type='t_nand' />
    <xs:element name='NOT'   type='t_not' />     <!-- note: UNARY OP! -->
  </xs:choice>
</xs:group>
```

### XML Example

32) Specify a dependency on test B. In this case, it's a pure ordering… this test will execute regardless of whether or not B actually executed, but only after APItest *attempts* to execute B (It is possible that B failed its dependencies). In this case, the defaults for minPctMatch and maxPctMatch are 0% and 100%, respectively.

```
<dependencies>
  <dependency name="B"/>
</dependencies>
```

33) Perhaps we want to enforce that B must have executed and that it's expected results matched its actual results. To do this, simply override the default value of minPctMatch to 100%:

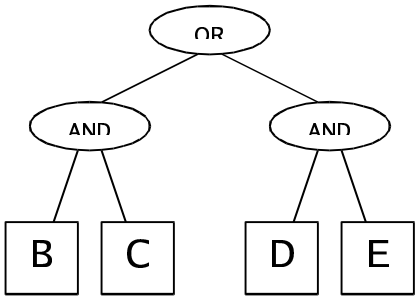
```
<dependencies>
  <dependency name="B" minPctMatch="100" />
```

```
</dependencies>
```

34) More complex dependencies are specified by using the appropriate conjunction. Perhaps we wish to create a dependency on tests B,C,D, and E that is satisfied either if both B and C matched 100% or both D and E both matched 100% (ie. ($B_{100\%}$ AND $C_{100\%}$) OR ($C_{100\%}$ AND $D_{100\%}$). The XML would be the following:

```
<dependencies>
 <OR>
  <AND>
   <dependency name="B" minPctMatch="100"/>
   <dependency name="C" minPctMatch="100"/>
  </AND>
  <AND>
   <dependency name="D" minPctMatch="100"/>
   <dependency name="E" minPctMatch="100"/>
  </AND>
 </OR>
</dependencies>
```

a. The expression tree for this example is the following:



## Element: *dependency*

Element *dependency* specifies an individual dependency.

### Schema Definition

```
 <xs:complexType name='t_dependency'>
  <xs:attribute name='name'  type='t_name' use='required'/>
  <xs:attribute name='type'  type='t_dep' use='optional'/>                    <!-- default:
PREREQ -->
  <xs:attribute name='minPctPass' type='xs:decimal' use='optional'/>    <!-- default: 0.0
-->
  <xs:attribute name='maxPctPass' type='xs:decimal' use='optional'/>    <!-- default:
100.0  -->
</xs:complexType>
```

## Element: *AND*

Defines a logical AND operation amongst ALL sub-elements.

### Schema Definition

```
<xs:complexType name='t_and'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'     type='t_and'/>
    <xs:element name='OR'     type='t_or'/>
    <xs:element name='NAND'     type='t_nand'/>
    <xs:element name='NOR'     type='t_nor'/>
    <xs:element name='NOT'     type='t_not'/>
  </xs:choice>
</xs:complexType>
```

## Element: *OR*

Defines a logical OR operation amongst ALL sub-elements.

### Schema Definition

```
<xs:complexType name='t_or'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>
```

## Element: *NOT*

Defines a negation operation. It should be noted that this is a *unary* operation.

### Schema Definition

```
<xs:complexType name='t_not'>
  <xs:choice maxOccurs='1'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'       type='t_and'/>
    <xs:element name='OR'        type='t_or'/>
    <xs:element name='NAND'      type='t_nand'/>
    <xs:element name='NOR'       type='t_nor'/>
    <xs:element name='NOT'       type='t_not'/>
  </xs:choice>
</xs:complexType>
```

## Element: *NAND*

Defines a logical NAND operation amongst ALL sub-elements.

### Schema Definition

```
<xs:complexType name='t_nand'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>
```

## Element: *NOR*

Defines a logical NOR operation amongst ALL sub-elements.

### Schema Definition

```
<xs:complexType name='t_nor'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>
```

# APItest Example Test Specifications

## Shell Tests

35) Executes two tests, (A and B) with some testList delays added as well as output checking in test B.

**File:** shellTest1.api

```
<?xml version="1.0" encoding="utf-8" ?>
<testCollection
    xmlns:xs = "http://www.w3.org/2001/XMLSchema-instance"
    xs:noNamespaceSchemaLocation='apitest.xsd'>
  <name>t15</name>
  <testList numReps='1' preDelay='1' postDelay='1' iterDelay='.2'>
     <shellTest name='A' command='python apitest.py --test'/>
     <shellTest name='B' command='python apitest.py --test'>
        <output format='REGEXP' type='stdout'>.*stdout.*</output>
        <output format='REGEXP' type='stderr'>.*stderr.*</output>
        <output format='REGEXP' type='status'>[123][0]*</output>
     </shellTest>
  </testList>
</testCollection>
```

36) We can add a bit more complex test to the mix by adding the following two tests to the *testList* section above in *shellTest1.api* .

   a. *complex1* implements a dependency on the expression: [($A_{>90\%}$ AND $B_{>90\%}$) OR $C_{>90\%}$]

```
<shellTest name='C' command='python apitest.py --test'/>

<shellTest name='complex1' command='python apitest.py --test'>
   <dependencies>
     <OR>
       <AND>
         <dependency name='A' type='PREREQ' minPctMatch='90'/>
         <dependency name='B' type='PREREQ' minPctMatch='90'/>
       </AND>
         <dependency name='C' type='PREREQ' minPctMatch='90'/>
     </OR>
   </dependencies>
  </shellTest>
```

# APItest Test File Schema

The following is the APItest as of September 18, 2003.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
            xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:annotation>
  <xs:documentation>
    apitest input file schema
    SciDAC SSS project
    William McLendon   wcmclen@sandia.gov
  </xs:documentation>
</xs:annotation>

<xs:element name="testCollection">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name"     type="xs:string"  minOccurs='1' maxOccurs='1'/>
      <xs:element name="info"     type="t_info"     minOccurs='0' maxOccurs='1'/>
      <xs:element name="testList" type="t_testList" minOccurs='1' maxOccurs='1'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="t_testList">
 <xs:complexContent>
   <xs:extension base='t_testListElement'>
    <xs:sequence minOccurs='0' maxOccurs='unbounded'>
      <xs:choice>
        <xs:element name="ssslibTest" type="t_ssslibTest" minOccurs='0'/>
        <xs:element name="shellTest"  type="t_shellTest"  minOccurs='0'/>
        <xs:element name="tcpipTest"  type="t_tcpipTest"  minOccurs='0'/>
        <xs:element name="httpTest"   type="t_httpTest"   minOccurs='0'/>
      </xs:choice>
    </xs:sequence>
   </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name='t_testListElement'>
 <xs:attribute name='numReps'   use='optional' type='xs:integer'/>
 <xs:attribute name='preDelay'  use='optional' type='xs:float'/>
 <xs:attribute name='postDelay' use='optional' type='xs:float'/>
 <xs:attribute name='iterDelay' use='optional' type='xs:float'/>
</xs:complexType>

<!-- Defined Test for TCPIP messages -->
<xs:complexType name="t_tcpipTest">
 <xs:complexContent>
   <xs:extension base='t_tcpipTestElement'>
    <xs:sequence>
```

```
      <xs:element name='input' type='t_input' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element name='output' type='t_output' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element name='dependencies' type='t_dependencies' minOccurs='0' maxOccurs='1'/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='t_tcpipTestElement'>
  <xs:attribute name='name'       use='required' type='xs:string'/>
  <xs:attribute name='minPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='maxPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='numReps'    use='optional' type='xs:integer'/>
  <xs:attribute name='preDelay'   use='optional' type='xs:float'/>
  <xs:attribute name='postDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='iterDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='onFail'     use='optional' type='t_failAction'/>
  <xs:attribute name='desturl'    use='required' type='xs:string'/>
  <xs:attribute name='destport'   use='required' type='xs:integer'/>
  <xs:attribute name='clearRecvBuf' use='optional' type='t_bool'/>
  <xs:attribute name='comm'       use='optional' type='commOpts'/>
</xs:complexType>

<!-- Defined Test for HTTP components -->
<xs:complexType name="t_httpTest">
  <xs:complexContent>
   <xs:extension base='t_httpTestElement'>
    <xs:sequence>
      <xs:element name='input' type='t_input' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element name='output' type='t_output' minOccurs='0' maxOccurs='unbounded'/>
      <xs:element name='dependencies' type='t_dependencies' minOccurs='0'
maxOccurs='unbounded'/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='t_httpTestElement'>
  <xs:attribute name='name'       use='required' type='xs:string'/>
  <xs:attribute name='minPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='maxPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='numReps'    use='optional' type='xs:integer'/>
  <xs:attribute name='preDelay'   use='optional' type='xs:float'/>
  <xs:attribute name='postDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='iterDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='onFail'     use='optional' type='t_failAction'/>
  <xs:attribute name='desturl'    use='required' type='xs:string'/>
  <xs:attribute name='destpath'   use='optional' type='xs:string'/>
  <xs:attribute name='action'     use='required' type='t_httpReqClass'/>  <!-- GET/POST -->
</xs:complexType>

<!-- Defined Test for SSSLIB components  -->
<xs:complexType name="t_ssslibTest">
  <xs:complexContent>
   <xs:extension base='t_ssslibTestElement'>
```

```
      <xs:sequence>
        <xs:element name='input' type='t_input' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='output' type='t_output' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='dependencies' type='t_dependencies' minOccurs='0' maxOccurs='1'/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='t_ssslibTestElement'>
  <xs:attribute name='name'       use='required' type='xs:string'/>
  <xs:attribute name='minPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='maxPctPass' use='optional' type='xs:float'/>
  <xs:attribute name='numReps'    use='optional' type='xs:integer'/>
  <xs:attribute name='preDelay'   use='optional' type='xs:float'/>
  <xs:attribute name='postDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='iterDelay'  use='optional' type='xs:float'/>
  <xs:attribute name='onFail'     use='optional' type='t_failAction'/>
  <xs:attribute name='destination' use='required' type='xs:string'/>
</xs:complexType>

<!-- Defined Test for execution via the shell -->
<xs:complexType name="t_shellTest">
  <xs:complexContent>
    <xs:extension base='t_shellTestElement'>
      <xs:sequence>
        <xs:element name='output' type='t_output' minOccurs='0' maxOccurs='unbounded'/>
        <xs:element name='dependencies' type='t_dependencies' minOccurs='0' maxOccurs='1' />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="t_shellTestElement">
  <xs:attribute name='name'       use='required' type='xs:string'/>
  <xs:attribute name='command'    use='required' type='xs:string'/>
  <xs:attribute name='minPctPass' use='optional' type='xs:decimal'/>
  <xs:attribute name='maxPctPass' use='optional' type='xs:decimal'/>
  <xs:attribute name='numReps'    use='optional' type='xs:integer'/>
  <xs:attribute name='preDelay'   use='optional' type='xs:decimal'/>
  <xs:attribute name='postDelay'  use='optional' type='xs:decimal'/>
  <xs:attribute name='iterDelay'  use='optional' type='xs:decimal'/>
  <xs:attribute name='onFail'     use='optional' type='t_failAction'/>
</xs:complexType>

<xs:complexType name='t_output'>
  <xs:simpleContent>
    <xs:extension base='xs:string'>
      <xs:attribute name='format' type='bufType' use='optional'/>
      <xs:attribute name='type' type='xs:string' use='required'/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name='t_input'>
```

```
  <xs:simpleContent>
    <xs:extension base='xs:string'>
      <xs:attribute name='type' type='xs:string' use='required'/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!-- Defined type for information -->
<xs:complexType name="t_info">
   <xs:sequence>
      <xs:element name="author" minOccurs='0' maxOccurs='1'>
         <xs:complexType>
            <xs:attribute name="name"  type="xs:string" use='required' />
            <xs:attribute name="email" type="xs:string" use='optional' />
                <xs:attribute name="phone" type="xs:string" use='optional' />
         </xs:complexType>
      </xs:element>
      <xs:element name="contact" minOccurs='0' maxOccurs='1'>
         <xs:complexType>
            <xs:attribute name="name"  type="xs:string" use='required' />
            <xs:attribute name="email" type="xs:string" use='optional' />
            <xs:attribute name="phone" type="xs:string" use='optional' />
         </xs:complexType>
      </xs:element>
      <xs:element name="description" type="xs:string" minOccurs='0' maxOccurs='1'/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name='t_not'>
  <xs:choice maxOccurs='1'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'        type='t_and'/>
    <xs:element name='OR'         type='t_or'/>
    <xs:element name='NAND'       type='t_nand'/>
    <xs:element name='NOR'        type='t_nor'/>
    <xs:element name='NOT'        type='t_not'/>
  </xs:choice>
</xs:complexType>

<xs:complexType name='t_and'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'        type='t_and'/>
    <xs:element name='OR'         type='t_or'/>
    <xs:element name='NAND'       type='t_nand'/>
    <xs:element name='NOR'        type='t_nor'/>
    <xs:element name='NOT'        type='t_not'/>
  </xs:choice>
</xs:complexType>

<xs:complexType name='t_or'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
```

```
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>

<xs:complexType name='t_nor'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>

<xs:complexType name='t_nand'>
  <xs:choice maxOccurs='unbounded'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:element name='AND'  type='t_and'/>
    <xs:element name='OR'   type='t_or'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:complexType>

<xs:group name='g_Logical'>
  <xs:choice>
    <xs:element name='OR'   type='t_or' />
    <xs:element name='AND'  type='t_and' />
    <xs:element name='NOR'  type='t_nor'/>
    <xs:element name='NAND' type='t_nand'/>
    <xs:element name='NOT'  type='t_not'/>
  </xs:choice>
</xs:group>

<xs:complexType name='t_dependency'>
  <xs:attribute name='name'  type='t_name' use='required'/>
  <xs:attribute name='type'  type='t_dep' use='optional'/>              <!-- default: PREREQ -->
  <xs:attribute name='minPctPass' type='xs:decimal' use='optional'/>    <!-- default: 0.0    -->
  <xs:attribute name='maxPctPass' type='xs:decimal' use='optional'/>    <!-- default: 100.0  -->
</xs:complexType>

<xs:complexType name='t_dependencies'>
  <xs:choice maxOccurs='1'>
    <xs:element name='dependency' type='t_dependency'/>
    <xs:group ref='g_Logical' minOccurs='0' maxOccurs='unbounded'/>
  </xs:choice>
</xs:complexType>

<xs:simpleType name='t_delay'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='pre'/>
```

```
    <xs:enumeration value='post'/>
    <xs:enumeration value='iter'/>
  </xs:restriction>
</xs:simpleType>

<!--
Fail Action:
          CONTINUE = Testing will continue regardless of how it finishes or prereqs finish.
          BREAK    = If a test is running in a looped mode, this breaks out of that loop and
                     goes on to the next test.
          HALT     = Stop everything, run no more tests.
-->
<xs:simpleType name="t_failAction">
   <xs:restriction base="xs:string">
      <xs:enumeration value="CONTINUE" />
      <xs:enumeration value="BREAK" />
      <xs:enumeration value="HALT" />
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name='t_dep'>
  <xs:restriction base='xs:string'>
    <xs:enumeration value='PREREQ'/>
  </xs:restriction>
</xs:simpleType>

<!--
CCode (ConditionCode)
          PASS = TEST PASSED
          FAIL = TEST FAILED
          UNKN = UNKNOWN
-->
<xs:simpleType name="t_condCode">
  <xs:restriction base="xs:string">
    <xs:enumeration value="UNKN" />
    <xs:enumeration value="PASS" />
    <xs:enumeration value="FAIL" />
  </xs:restriction>
</xs:simpleType>

<!--
Boolean String, restricts to YES or NO (or yes/no)
-->
<xs:simpleType name="t_bool">
   <xs:restriction base="xs:string">
      <xs:enumeration value="YES" />
      <xs:enumeration value="yes" />
      <xs:enumeration value="NO" />
      <xs:enumeration value="no" />
   </xs:restriction>
</xs:simpleType>

<!--
-->
<xs:simpleType name="bufType">
```

```
  <xs:restriction base="xs:string">
    <xs:enumeration value="REGEXP" />
    <xs:enumeration value="SCHEMA" />
    <xs:enumeration value="TXTSTR" />
  </xs:restriction>
</xs:simpleType>

<!-- -->
<xs:simpleType name="commOpts">
  <xs:restriction base="xs:string">
    <xs:enumeration value="KEEPALIVE" />
    <xs:enumeration value="CLOSE"     />
    <xs:enumeration value="CLOSEITER" />
  </xs:restriction>
</xs:simpleType>

<!-- Restricted string, for names, etc. max length 70, min length 1 character -->
<xs:simpleType name="t_name">
  <xs:restriction base="xs:string">
    <xs:maxLength value="70" />
    <xs:minLength value="1" />
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="t_httpReqClass">
  <xs:restriction base="xs:string">
    <xs:enumeration value="GET"/>
    <xs:enumeration value="POST"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

<!-- EOF -->
```