CHAPTER 6

Creating User-Defined Tests

Access: WAE Coordinated Maintenance > Settings > User-Defined Tests tab

In addition to the built-in tests, you can create customized tests for inclusion in evaluations. The communication between your API and the WAE Coordinated Maintenance application is achieved through the Settings > User-Defined Tests page. When an evaluation is run that contains a user-defined test, the application sends the API a set of parameters on which to conduct the test. In return, the API returns the results to the application for inclusion in the evaluation.

For an example start to finish, including an example API, see the Example section. The full script used in this example is in `$WAE_HOME/mate/current/lib/web/apache-tomcat-6.0.37/webapps/ROOT/services/maintenance/samples`. By default, `$WAE_HOME` is `/opt/cariden/software`.

Workflow	See
1. Create the API that runs the test.	Input and Output sections
2. Configure the Settings > User-Defined Tests page so that the WAE Coordinated Maintenance application and the API can communicate.	Configure User-Defined Tests section
3. Either configure the default evaluation to include the custom test, or add the custom test when adding an event.	Configuring Evaluations chapter

Input

The input is passed to the custom test when the evaluation is run. The API must be able to take the following input in multipart/form-data format that will be passed using the HTTP POST method.

Input Parameter Passed to the API	Format	Description
startTime	yyyy-MM-dd HH:mm:ss	Date and time the event is to start in UTC.
stopTime	yyyy-MM-dd HH:mm:ss	Date and time the event is to stop in UTC.
operand	String for comparison; this can be a double string or an integer string. **Examples:** 90.00, 10, acme	The criteria value used in the evaluation.

Input Parameter Passed to the API	Format	Description
operator	> < = >= <=	Supported operators used when the operand value is compared to the result. When a comparison is true, the test fails. When a comparison is false, the test passes. **Example:** If the operand is 75, the result is 80, and operator is >=, then the test fails.
nodes	JSON string that identifies an array of nodes by nodeName	List of nodes used in the evaluation.
circuits	JSON string that identifies an array of circuits by nodeAName, intfA, nodeBName, intfB intf means “interface.”	List of circuits used in the evaluation.
ports	JSON string; array of ports by nodeAName, portA, nodeBName, portB	List of ports (LAG members) used in the evaluation.
planFile	File attachment byte stream	Representation of the network during the maintenance event.
origFile	File attachment byte stream	Representation of the network before evaluations are run.

Example Input

In this example, one node, two circuits, and two ports are passed to the custom test. The start time is July 7, 2016 at 10:00 AM (local time), and the duration is three hours. These are entered when adding the event.

The operand is 80, and the operator is > (greater than). These are specified in the UI when adding the custom test to the UI.

```
{
"planFile": (as a file attachment)
"origFile": (as a file attachment)
"nodes":[{"name":"AM_ATL_ER2"}],
"circuits":[{"nodeAName":"192.168.249.0/24:BgpPsn","nodeBName":"ASN65300:192.168.30.1",
"intfA":"TenGigE0/0/0/2","intfB":"to_EU_AMS_AMSIX"},{"nodeAName":"192.168.243.0/24:BgpP
sn","nodeBName":"ASN65370:192.168.37.1","intfA":"xe-1/0/1","intfB":"to_AM_NYC2_EQUINIX"
}],
"ports":[{"nodeAName":"AM_DC_NORTH_BB1","nodeBName":"AM_DC_NORTH_BB2","portA":"TenGigE1
/0/1","portB":"TenGigE1/0/4"},{"nodeAName":"AM_DC_NORTH_BB1","nodeBName":"AM_DC_NORTH_B
B2","portA":"TenGigE1/0/2","portB":"TenGigE0/0/4"}],
"startTime":"2016-07-07 10:00:00",
"stopTime":"2016-07-07 13:00:00",
"operand":"80.00","operator":">"
}
```

Output

The custom test's output is passed to the application's evaluation process once the test completes. The API must be able to produce text/plain and application/JSON. The API for the custom test must return these values to the application.

This information appears in the UI in the Summary Report section. The "details" are shown when you click the View Report button next to a failed test.

Output Parameter Passed to the Application	Format	Description
result	String, which must be either "pass" or "fail"	Identifies whether the test passed or failed
score	Real number	Test score
details	CSV	If the test passes (result = pass), this can be left empty. If the test fails (result = fail), this identifies the test report. It must include column headings and one row for each object that failed the test.

Example Output

Pass—The test passed with a score of 47.58. No report is added.

```
{"result":"pass","score":47.58,"details":""}
```

Fail—The test failed with a score of 118.77. The report includes one row of headings and one row of data for the circuit that caused the test to fail.

```
{"result":"fail","score":118.77,"details":"[[\"Node\",\"Circuits\",\"Utilization\",\"Traffic Level\",\"Start
Time\",\"Stop Time\"],[\"AM_BOS_ER2\",\"TenGigE0/0/1\",\"118.77\",\"Default\",\"2015-04-07
10:00:00\",\"2015-04-07 13:00:00\"]]
```

Configure User-Defined Tests

Once the API that executes the custom test is created, you must configure the Settings > User-Defined Tests page to set up communication between the API and the application.

General

- Test Name—Name of the test. These names appear as a selectable options to add to groups of tests on the Settings > Evaluations page.
- Description—(Optional) Summarized explanation of the test, which is particularly useful for communication purposes when there are multiple users.

Resource

This section defines how the application communicates with the API.

- URI—Full name of the resource used by the application to reach the API. The URI format is as follows. For scheme, only http and https are supported.

  Required format: scheme:[//[host[:port]][/]path

  Example: https://172.12.123.0:8888/opt/acme/tests

  Format with optional query and fragment: scheme:[//[host[:port]][/]path[?query][#fragment]

  Example: https://172.12.123.0:8888/opt/acme/tests?event=23#results
- Authentication—(Optional) If selected, the URI requires authentication to access it.
  - Type—Only Basic authentication is supported.
  - Username and password—Username and password that secures the access to the URI.

Test Results

- Score Label—Label that appears next to the score in the evaluation report to identify what that score represents.
- Pass Message—(Optional) Message that appears next to the word “Pass” in the evaluation report.
- Fail Message—(Optional) Message that appears next to the word “Fail” in the evaluation report. This message is particularly useful when other users are generating evaluations with a custom test that they did not develop.

Report

- Name—Name of the report included in the evaluation results.
- Display only if failed—If selected, display the report only if the test fails.
- Require original plan—The application always passes the API a plan file that represents the network during the maintenance event. By selecting this option, you are telling the application to also pass the API the original plan file that represents the network prior to the maintenance event. This original plan can be useful when comparing plan files before and during the event.

Example

This example user-defined test is an over simplified way of demonstrating the most basic components of creating a custom test. The test itself looks for nodes that have the letter “o” in their names. If more than two nodes are found that meet this failure condition, the test fails.

Example API

This section of an API focuses on the handler section of code that takes the application input, loads it into the API framework, and returns results back to the application. This example is using a Tornado python web application package, which must be separately installed. Comments as to what this handler is doing are inline.

```
# Handlers are invoked based on which resource is being accessed in this application.
# There is one handler per resource.

# This handler is for the simple "noOrouters" resource. This resource implements a simple
# test that examines the list of specified nodes and returns a list of any nodes that have
# the letter "o" in their names. It also compares the number of these matching nodes
# against the specified test fault condition (criterion) to return "pass" or "fail,"
# as well as the number of matches.
class NoORoutersHandler(RequestHandler):
    def initialize(self,db) :
        self.db = db
    def post(self,foo='foo') :

        # The test report is returned as an array of arrays. The first contained array
        # is a list of column names for what will be rendered as a table in the UI.
        # There is only 1 column called "Nodes with O," so the array has only 1 element.
        # Note that the details passed must include a column name per column of data.
        matching = [["Nodes with O"]]

        # Get the list of nodes that were specified in the event. Consider only nodes,
        # and ignore circuits and LAG members.
        nodes = json.loads(self.get_argument('nodes'))

        # Get the threshold value.
        operand = self.get_argument('operand')

        # The value of the threshold is a count of matching nodes, so convert it
        # to an integer.
    try:
        thresh = int(operand)
    except:
        thresh = int(float(operand))

    # Get the comparison operator that was selected.
    oper = self.get_argument('operator')

    # Use the "operator" python module to match this to a usable comparison operator.
    import operator
    ops = {
        '>' : operator.gt,
        '<' : operator.lt,
        '=' : operator.eq
    }
    compare = ops[oper]

    for n in nodes :
        if 'o' in n['name'].lower() :  # Check for "o" in the name.
            if n['name'] not in matching : # Check for duplicates.
                matching.append([n['name']]) # Append it to the list of matching nodes.

    # If any matches were found, the "matching" array will contain 1 entry for the column
    # title and 1 entry for each match. Therefore, the number of matching entries is the
    # length of the "matching" array minus the title row. Note that a score
    # and a result are passed back to the application.
    score = len(matching) - 1
```

```
    self.db['response']['score'] = score

    # Check how the score compares with the threshold.
    if compare(score,thresh) :
        self.db['response']['result'] = 'fail'
    else :
        self.db['response']['result'] = 'pass'

    # Add the report. Note that details must be passed back to the application. All
    # matching nodes are placed in details such that there is one row per matching node.
    self.db['response']['details'] = json.dumps(matching)

    # Tornado automatically handles setting the content-type of the response.
    self.write(self.db['response'])

def make_app():
    return Application([
        url(r"/noOrouters/(.*)", NoORoutersHandler,dict(db=db),name='noOrouters'),
        ])
```

Example Input

Following is the input passed to the user-defined test as a result of the UI input shown in Figure 6-1 and Figure 6-2.

```
{
"planFile": (as a file attachment)
"nodes":[{"name":"AM_ATL_ER1"},{"name":"AP_BEI_ER4"},{"name":"AP_TOK_BB3"},{"name":"AM_
BOS_ER2"},{"name":"EU_LON_ER4"}]
"startTime":"2016-04-07 17:00:00",
"stopTime":"2016-04-07 18:00:00",
"operand":"2","operator":">"
}
```

Figure 6-1 Example UI Input When Adding Event

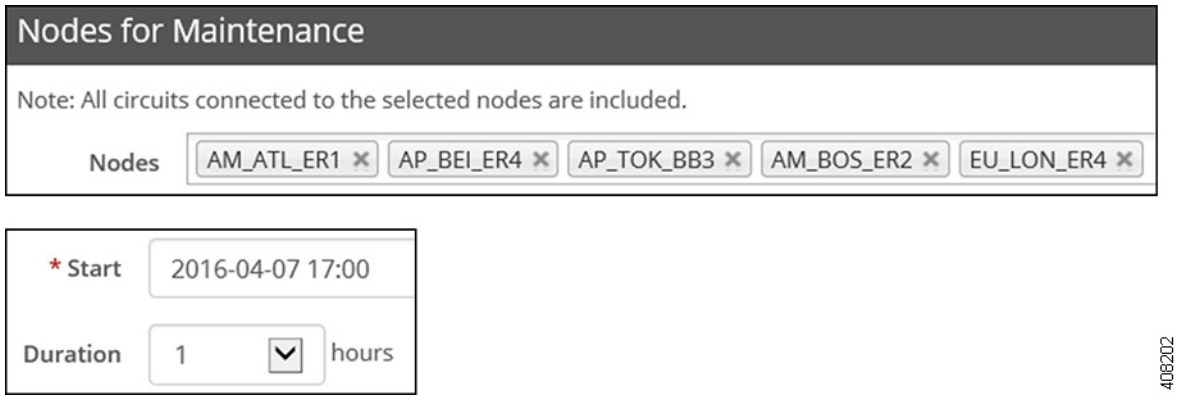
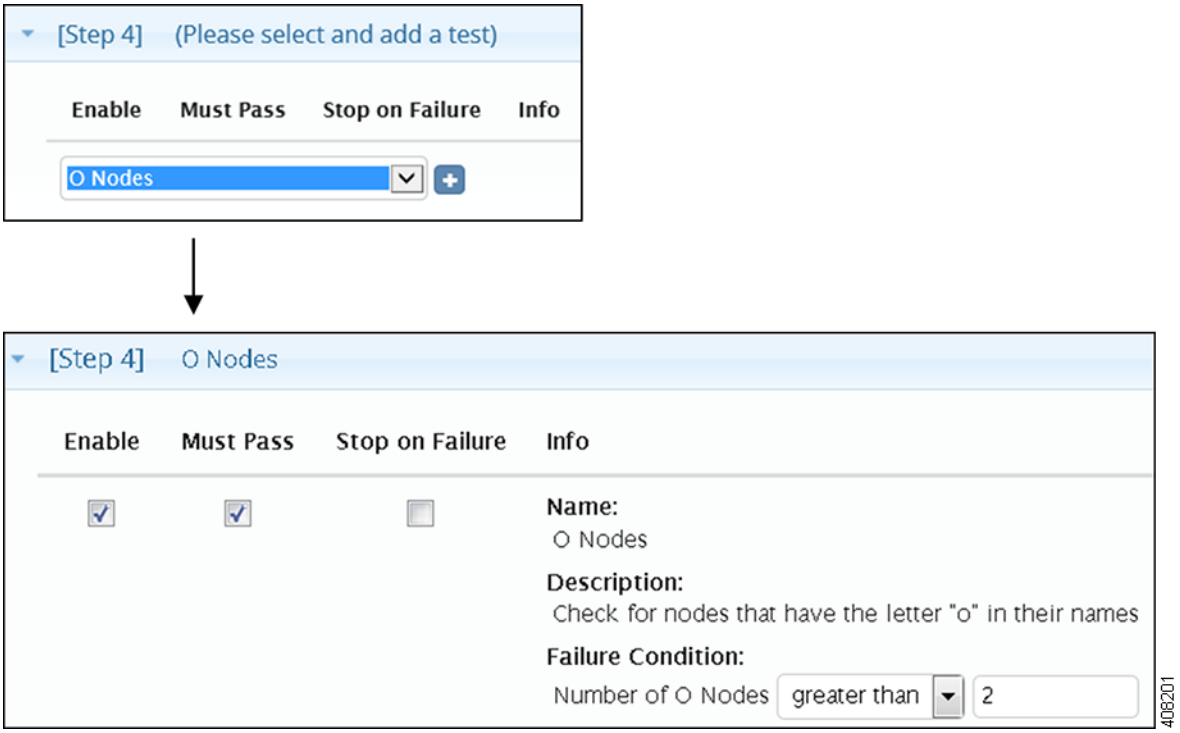


Figure 6-2 Example UI Input When Adding User-Defined Test

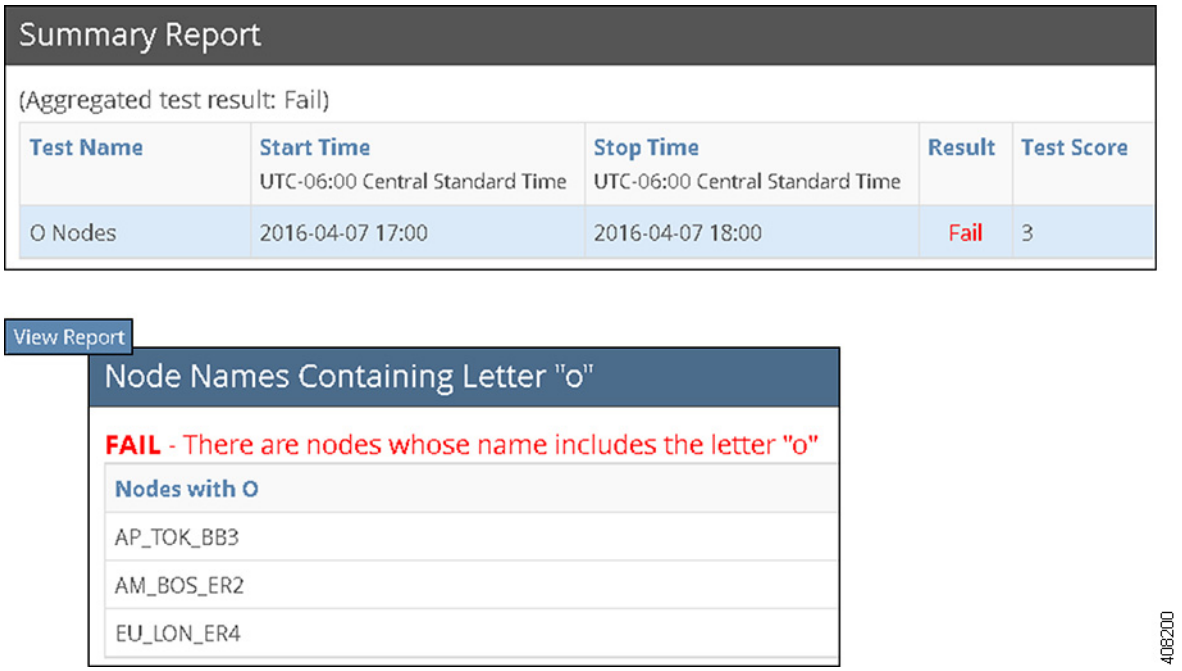


Example Output

In the above API, the score, result, and details are these lines. Figure 6-3 shows how these results appear in the UI.

```
{"result":"fail","score":"3","details":"[[\"Nodes with
O\"],[\"AP_TOK_BB3\"],[\"AM_BOS_ER2\"],[\"EU_LON_ER4\"]]"}
```

Figure 6-3 Example UI Output from User-Defined Test



Related Topics

- *WAE Design Integration and Development Guide* (for information on plan files)