ecflow watchdogs

A simple python client script may be used to watch over a suite:

simple example: ecflow_client.py

```
#!/usr/bin/env python
import ecf as ec
import sys
import time
class Observer(object):
    """ simple ecflow watchdog class """

    def __init__(self, defs, node, port, path):
        super(Observer, self).__init__()
        self.path = path
        self.node = node
        self.port = port

    def process_node(self, node):
        status = "%s" % node.get_dstate()
        if status not in ("queued", "complete"):
            print node.get_abs_node_path(), status
        for kid in node.nodes:
            self.process_node(kid)

    def clear(self):
        for n in range(0, 64, 1): print("\r\n")

    def run(self):
        client = ec.Client(self.node, self.port)
        while 1:
            client.sync_local() # get changes,
            node = client.get_defs().find_abs_node(self.path)
            assert node is not None
            self.clear()
            self.process_node(node)
            time.sleep(90)

if __name__ == "__main__":
    import os.path
    if len(sys.argv) < 3:
        node = "localhost"
        port = "31415"
        path = "/"
    else:
        node = sys.argv[1]
        port = sys.argv[2]
        path = sys.argv[3]
    if 1:  client = Observer(None, node, port, path)
    else:  client = ObserverHtml(None, node, port, path)
    client.run()
```

```
 python ecflow_client.py localhost 31415 /compo/main/12
```

This simple example may be extended to generate a html page.

html client

```
CSS = "/home/ma/map/http/sms.css"
HEADER = '''<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="refresh" content="600">
  <title>Status tree of {node_full_name} in {hostname}-{hostport}</title>
  <link rel="stylesheet" type="text/css" href="file:{css}">
```

```
</head>
<body>
  <table border=2 cellpadding=2> <tr>'''
FOOTER = '''</table>\n</body>\n</html>'''
class ObserverHtml(Observer):
    """ simple ecflow watchdog class + html output """
    def __init__(self, defs, node, port, path):
        super(ObserverHtml, self).__init__(defs, node, port, path)
        self.userdepth = 3
        self.status_mask = ("queued", "complete", "unknown")
        self.maxdepth = 100
        self.openkids = True
    def run(self):
        client = ec.Client(self.node, self.port)
        path   = self.path
        while 1:
            client.sync_local() # get changes, synced with local definition
            node = client.get_defs().find_abs_node(path)
            if 0: assert node is not None
            if node is None: node = client.get_defs()
            self.html_write(node)
            time.sleep(90)
    def header(self):
        print >>self.fp, HEADER.format(node_full_name=self.path,
                                       hostname= self.node,
                                       css= CSS,
                                       hostport=self.port)
    def tail(self, node=None):
        print >>self.fp, FOOTER
    def countnext(self, anything, depth):
      np = anything;
      n = 0;
      if np is not None and depth > self.maxdepth:
          self.maxdepth = depth;
      while np is not None:
          n += 1
          np = np.next()
      return n;
    def count(self, np=None, depth=1):
       n = 0;
       if np is None: return
       if ("%s" % np.get_state() not in self.status_mask
           or depth > self.userdepth):
           return
       if depth > self.maxdepth: self.maxdepth = depth;

    def html_write(self, np, fpp=None):
      if fpp is None:
          fpp = open("example.html", "w")
      self.count(np,0);
      self.fp = fpp;
      self.header();
      self.lineno = 0;
      self.table(np,0);
      self.tail(np);
      fpp.close()
    def write_link(self, node, horizontal, base, target):
      print >>self.fp, '<base target="{target}">'.format(target=target)
      for np in node.nodes:
          print >>self.fp,'      <td class="{status}">'.format(status="%s" % np.get_state())
          print >>self.fp,'<a href="{base}/{name}.html">{name}</a>'.format(
              base=base, name=np.name())
          if horizontal and np.next():
              print >>self.fp,"    </tr>"
              print >>self.fp,"    <tr>"
          print >>self.fp, "    </tr>"
    def img(self, name):
        print >>self.fp, "<img src=\"../gifs/%s.gif\" alt=\"[%s]\">" % (name,name)
    def decorations(self, node):
        status =  node.get_state()
        if status == "halted": self.img("halted")
```

```
        if status == "shutdown": self.img("shutdown")
        # rerun
        # messages
        # late
        # clock
    def html_links(self, np, fpp=None, title=0, horizontal=0):
      base = os.getenv("htmlbase");
      target = os.getenv("htmltarget");
      if base is not None: base = "od";
      if target is not None: target = "level1";
      self.fp = fpp;
      self.header();
      if title:      print >>self.fp,"      <th>{name}</th>".format(name=np.name())
      if horizontal: print >>self.fp,"    </tr>"
      if np.nodes:   print >>self.fp, "    <tr>"
      self.write_link(np, horizontal, base, target);
      self.tail();
    def table(self, node, depth):
        n = 0
        if node is None: return 0
        # if depth > self.userdepth: return 0 # depth
        try: print node.get_abs_node_path()
        except: pass
        if 1:
              try:    status = "%s" % node.get_state() # node
              except:
                  status = node.value()            # attribute
                  print >>self.fp, "<td>%s:%s" % (node.name(),status),"</td>"
                  return 1
              if status in self.status_mask: return 0
              try:
                  print >>self.fp, "<td class='%s'>" % status, "%s" % node.name(),"</td>"
              except:
                  for suite in node.suites: self.table(suite, depth) # Defs
                  return 0
              # decorations?
              if self.openkids:
                  # n += self.table(node.get_repeat(), depth+1)
                  for item in node.meters:
                      n += self.table(item, depth+1)
                  for item in node.events:
                      n += self.table(item, depth+1)
                  for item in node.labels:
                      n += self.table(item, depth+1)

                  i = 0
                  for kid in node.nodes:
                      if 1: print >>self.fp, "<tr>"
                      n += self.table(kid, depth+1)
                      i += 1
        if n == 0: print >>self.fp, "</tr>"; n=1
        return n
```

It clearly needs to be refined to provide the expected output.

Next step may be to introduce javaScript and jQuery, using JSON format to dump the tree content into a file.

JavaScript + jQuery

```
JQUERY = HOME + "jquery-1.10.2.min.js"
SIMPLE_CSS = """
<style type="text/css">
body {background: #ffffff; color: #000000 }
A:link {color: blue}
A:visited {color: purple}
A:active {color: blue}

A:link, A:visited, A:active {text-decoration: underline; }
li.unknown   { background: #bfbfbf }
```

```
li.complete  { background: yellow }
li.queued    { background: #add9e7 }
li.submitted { background: #04e1d1 }
li.active    { background: #00ff00 }
li.suspended { background: #ffa600 }
li.aborted   { background: #ff0000 }
li.shutdown  { background: #ffc1cc }
li.halted    { background: #ef83ef }
li.set       { background: #bbbbff }
li.clear     { background: #bbbbbb }

/* ul li { list-style: disc; }
ul ul li { list-style: circle; }
ul ul ul li { list-style: square; } */
</style>
"""
# NODES: def, suite, family, task
# ATTRIBUTES: autocancel, clock, complete, cron, date, day, defstatus, edit
# event, inlimit, label, late, limit, meter, repeat, time, today, trigger

class ObserverJS(Observer):
    """ simple ecflow watchdog class + html output """

    def __init__(self, defs, node, port, path, fname="ecflow.html"):
        super(ObserverJS, self).__init__(defs, node, port, path)
        self.fname = fname
        self.status_mask = ("queued", "complete", "unknown")

    def process_node(self, node, fp):
        if node is None: return
        elif isinstance(node, ec.Alias): return
        elif isinstance(node, ec.Defs):
            status = "%s" % node.get_server_state()
            status = status.lower()
            name   = "%s@%s" % (self.node, self.port)
        else:
            status = "%s" % node.get_state()
            name   = node.name()
            if status in self.status_mask: return

        print >>fp, "<ul>"
        print >>fp, "<li class='%s fold'>%s" % (status, name)
        if isinstance(node, ec.Defs):
            for kid in node.suites:
                self.process_node(kid, fp)
        else:
            for kid in node.nodes:
                self.process_node(kid, fp)
        print >>fp, "</li></ul>"

    def gen_html(self, fp, node):
        print >>fp, HEADER.format(node_full_name= self.path,
                                  hostname= self.node,
                                  css= "", # CSS,
                                  hostport= self.port)

        print >>fp,"""
<h2>Status tree of {node_full_name} in {hostname}-{hostport}
</h2>""".format(node_full_name= self.path,
                hostname= self.node,
                hostport= self.port)

        print >>fp, """<!--
wget http://code.jquery.com/jquery-1.10.2.min.js
python ecflow_client.py localhost 31415 /verify example.html
firefox example.html
wget https://github.com/mbostock/d3/archive/master.zip # http://d3js.org/
# http://www.randelshofer.ch/treeviz/
-->
<script src="{jquery}"></script>""".format(jquery= JQUERY)
        print >>fp, SIMPLE_CSS
```

```
        if isinstance(node, ec.Defs):
            status = "%s" % node.get_server_state()
            d = [ { "name": "%s-%s" % (self.node, self.port),
                  "status": status.lower(),
                  "kids": [ self.get_dict(kids)
                            for kid in node.suites ] } ]
        elif (isinstance(node, ec.Suite)
              or isinstance(node, ec.Family)
              or isinstance(node, ec.Task)):
            d = [ self.get_dict(node) ]

        else: d = {}; print type(node); return
        import json
        print d
        print >>fp, """<script type="text/javascript">
var html = [];
var tree= """, json.dumps(d), """;

function createList(arr) {
html.push('<ul>');

$.each(arr, function(index, item) {
if (item==null) { return; }
html.push("<li class='" + item.status + " fold'>"+item.name);
if (item.kids) { createList(item.kids); }
html.push("</li>"); });

html.push('</ul>');
}

createList(tree);
$('body').append(html.join(''));
</script>
"""
        self.process_node(node, fp)
        print >>fp, FOOTER

    def get_dict(self, node):
        status = "%s" % node.get_state()
        if status in self.status_mask: return # reduce output
        return { "name": node.name(),
                 "status": status.lower(),
                 "kids": [ self.get_dict(kid)
                           for kid in node.nodes ]}
    def run(self):
        client = ec.Client(self.node, self.port)
        while 1:
            client.sync_local() # get changes,
            if self.path == "/": node = client.get_defs()
            else: node = client.get_defs().find_abs_node(self.path)
            if 0: assert node is not None

            fp = open(self.fname, "w")
            self.gen_html(fp, node)
            fp.close()
            time.sleep(90)
```

Or even to use one of these nice libraries for fancy rendering (d3 as Treemap or SunBurst):

TreeMap

```
SIZE = {
    "aborted": 50,
    "queued": 10,
    "complete": 10,
    "unknown": 10,
    "submitted": 30,
```

```
    "active": 50,
    "running": 10,
    "shutdown": 30,
    "halted": 50,
}

STATUS = {
    "aborted": 5,
    "queued": 3,
    "complete": 2,
    "unknown": 10,
    "submitted": 4,
    "active": 6,
    "running": 10,
    "shutdown": 7,
    "suspended": 1,
    "halted": 8,
}

class ObserverD3(Observer):
    """ simple ecflow watchdog class + html output """

    def __init__(self, defs, node, port, path, fname="ecflow.html"):
        super(ObserverD3, self).__init__(defs, node, port, path)
        self.fname = fname
        self.status_mask = ("queued", "complete", "unknown")
        self.d3body = """
<!DOCTYPE html>
<meta charset="utf-8">
<style>

body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  margin: auto;
  position: relative;
  width: 960px;
}

form {
  position: absolute;
  right: 10px;
  top: 10px;
}

.node {
  border: solid 1px white;
  font: 10px sans-serif;
  line-height: 12px;
  overflow: hidden;
  position: absolute;
  text-indent: 2px;
}

</style>
<form>
  <label><input type="radio" name="mode" value="size" checked>Size</label>
  <label><input type="radio" name="mode" value="count"> Count</label>
</form>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

var margin = {top: 40, right: 10, bottom: 10, left: 10},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var color = d3.scale.category20c();

var treemap = d3.layout.treemap()
    .size([width, height])
    .sticky(true)
    .value(function(d) { return d.size; });
```

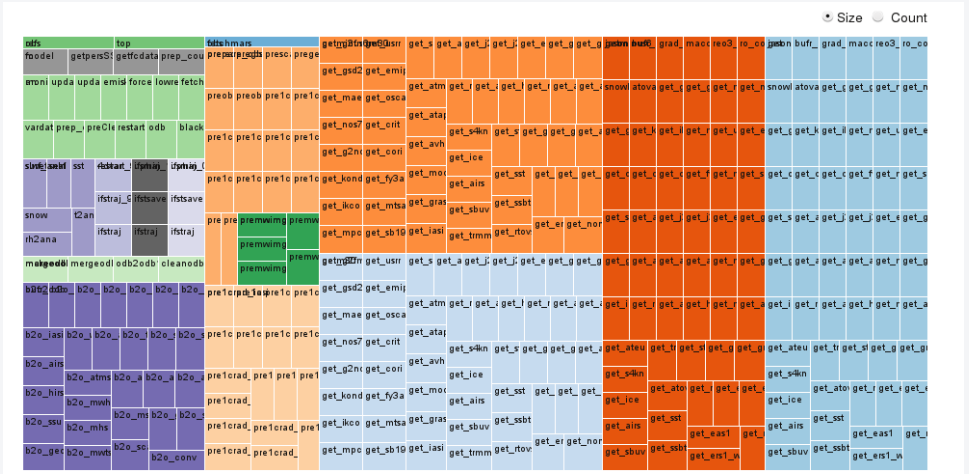
```
var div = d3.select("body").append("div")
    .style("position", "relative")
    .style("width", (width + margin.left + margin.right) + "px")
    .style("height", (height + margin.top + margin.bottom) + "px")
    .style("left", margin.left + "px")
    .style("top", margin.top + "px");

d3.json("ecflow.json", function(error, root) {
  var node = div.datum(root).selectAll(".node")
      .data(treemap.nodes)
    .enter().append("div")
      .attr("class", "node")
      .call(position)
      .style("background", function(d) { return d.children ? color(d.name) : null; })
      .text(function(d) { return d.childrenkids ? null : d.name; });

  d3.selectAll("input").on("change", function change() {
    var value = this.value === "count"
        ? function() { return 1; }
        : function(d) { return d.size; };

    node
        .data(treemap.value(value).nodes)
      .transition()
        .duration(1500)
        .call(position);
  });
});

function position() {
  this.style("left", function(d) { return d.x + "px"; })
      .style("top", function(d) { return d.y + "px"; })
      .style("width", function(d) { return Math.max(0, d.dx - 1) + "px"; })
      .style("height", function(d) { return Math.max(0, d.dy - 1) + "px"; });
}

</script>
"""

    def process_node(self, node, fp):
        if node is None: return
        elif isinstance(node, ec.Alias): return
        elif isinstance(node, ec.Defs):
            status = "%s" % node.get_server_state()
            status = status.lower()
            name   = "%s@%s" % (self.node, self.port)
        else:
            status = "%s" % node.get_state()
            name   = node.name()
            if status in self.status_mask: return

        print >>fp, "<ul>"
        print >>fp, "<li class='%s fold'>%s" % (status, name)
        if isinstance(node, ec.Defs):
            for kid in node.suites:
                self.process_node(kid, fp)
        else:
            for kid in node.nodes:
                self.process_node(kid, fp)
        print >>fp, "</li></ul>"
        # def, suite, family, task
        # autocancel, clock, complete, cron, date, day, defstatus, edit
        # event, inlimit, label, late, limit, meter, repeat, time, today, trigger

    def get_dict(self, node):
        if isinstance(node, ec.Alias): return
        status = "%s" % node.get_state()
        try: a = SIZE[status],
        except: print SIZE.keys(), status; raise
        return { "name": node.name(),
```

```
                "status": STATUS[status],
                "state": status,
                "size":  SIZE[status],
                "children": [ self.get_dict(kid) for kid in node.nodes ]}

    def run(self):
        client = ec.Client(self.node, self.port)
        fp = open(self.fname, "w")
        print >> fp, self.d3body
        fp.close()
        while 1:
            client.sync_local() # get changes,
            if self.path == "/": node = client.get_defs()
            else: node = client.get_defs().find_abs_node(self.path)
            if 0: assert node is not None

            if isinstance(node, ec.Defs):
                status = "%s" % node.get_server_state()
                d = {"name": "%s-%s" % (self.node, self.port),
                     "status": STATUS[status.lower()],
                     "size":  SIZE[status.lower()],
                     "state": status,
                     "children": [self.get_dict(kid) for kid in node.suites]}

            elif isinstance(node, ec.Alias): pass

            elif (isinstance(node, ec.Suite)
                  or isinstance(node, ec.Family)
                  or isinstance(node, ec.Task)):
                d = self.get_dict(node)

            else: d = {}; print type(node) # FIXME
            import json
            print d
            fp = open("ecflow.json", "w")
            print >>fp, json.dumps(d)
            fp.close()
            time.sleep(90)
```

SunBurst

```
class ObserverD3SunBurst(ObserverD3):
    def __init__(self, defs, node, port, path, fname="ecflow.html"):
        super(ObserverD3SunBurst, self).__init__(defs, node, port, path, fname)
        self.d3body = """
<!DOCTYPE html>
<meta charset="utf-8">
<style>

path {
  stroke: #fff;
  fill-rule: evenodd;
}

</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

var width = 960,
    height = 700,
    radius = Math.min(width, height) / 2;

var x = d3.scale.linear()
    .range([0, 2 * Math.PI]);

var y = d3.scale.sqrt()
    .range([0, radius]);
```

```
var color = d3.scale.category20c();

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height)
  .append("g")
    .attr("transform", "translate(" + width / 2 + "," + (height / 2 + 10) + ")");

var partition = d3.layout.partition()
    .value(function(d) { return d.size; });

var arc = d3.svg.arc()
    .startAngle(function(d) { return Math.max(0, Math.min(2 * Math.PI, x(d.x))); })
    .endAngle(function(d) { return Math.max(0, Math.min(2 * Math.PI, x(d.x + d.dx))); })
    .innerRadius(function(d) { return Math.max(0, y(d.y)); })
    .outerRadius(function(d) { return Math.max(0, y(d.y + d.dy)); });

d3.json("ecflow.json", function(error, root) {
  var path = svg.selectAll("path")
      .data(partition.nodes(root))
    .enter().append("path")
      .attr("d", arc)
      .attr("name", function(d) { return d.name; })
      .style("fill", function(d) { return color((d.children ? d : d.parent).name); })
      .on("click", click);

  function click(d) {
    path.transition()
      .duration(750)
      .attrTween("d", arcTween(d));
  }
});

d3.select(self.frameElement).style("height", height + "px");

// Interpolate the scales!
function arcTween(d) {
  var xd = d3.interpolate(x.domain(), [d.x, d.x + d.dx]),
      yd = d3.interpolate(y.domain(), [d.y, 1]),
      yr = d3.interpolate(y.range(), [d.y ? 20 : 0, radius]);
  return function(d, i) {
    return i
        ? function(t) { return arc(d); }
        : function(t) { x.domain(xd(t)); y.domain(yd(t)).range(yr(t)); return arc(d); };
  };
}

</script>
"""
```


Size
Count
olfs
top
fdbhmars