Using python scripting

As you have already seen, ecFlow has a *ecFlow Python Api*: (for both python2 and python3)

```
import ecflow
```

This allows the *suite definition* to be built with python.
It also allows communication with the *ecflow_server*.
This is a very powerful feature, that helps to define very complex suites in a relatively compact way.
Consider the following *suite*:

```
suite test
 family f1
     task a
     task b
     task c
     task d
     task e
 endfamily
 family f2
     task a
     task b
     task c
     task d
     task e
 endfamily
 family f3
     task a
     task b
     task c
     task d
     task e
 endfamily
 family f4
     task a
     task b
     task c
     task d
     task e
 endfamily
 family f5
     task a
     task b
     task c
     task d
     task e
 endfamily
 family f6
     task a
     task b
     task c
     task d
     task e
 endfamily
endsuite
```

This can be written in python as:

```
def create_suite(name) :
    suite = Suite(name)
    for i in range(1, 7) :
        fam = suite.add_family("f" + str
(i))
        for t in ( "a", "b", "c", "d", "e"
) :
            fam.add_task(t)
    return suite
```

```
def create_suite(name) :
    return Suite(name,
            [ Family("f{0}".format(i),
                [ Task(t) for t in ( "a", "b", "c", "d",
"e") ])
              for i in range(1,7) ])
```

Python variables can be used to generate *trigger dependencies*.
Imagine that we want to chain the families f1 to f6, so that f2 runs after f1, f3 after f2 and so on.
The following will do the trick:

```
def create_sequential_suite(name) :
    suite = Suite(name)
    for i in range(1, 7) :
        fam = suite.add_family("f" + str(i))
        if i != 1:
            fam += Trigger("f" + str(i-1) + " == complete")  # or fam.add_family( "f%d == complete" % (i-1) )
        for t in ( "a", "b", "c", "d", "e" ) :
            fam.add_task(t)
    return suite
```

For more detailed example please see the user manual

Adding Node attributes

There are several styles for adding node attributes(Repeat,Time,Today,Date,Day,Cron,Clock,DefStatus,Meter,Event,Variable,Label,Trigger, Complete, Limit,Inlimit,Zombie,Late)

```
# Functional style
node.add_variable(home,'COURSE')          # c++ style
node.add_limit('limitX',10)               # c++ style

# Using <node>.add(<attributes>)
node.add(Edit(home=COURSE),               # Notice that add() allows you adjust the indentation
         Limit('limitX',10))              # node.add(<attributes>)

# in place. When creating a Node, attributes are additional arguments (preferred)
# This also allows indentation.
#   Task(name,<attributes>)
#   Family(name,Node | <attributes>)
#   Suite(name,Node  | <attributes>)
node = Family('t1',
           Edit(home='COURSE'),
           Limit('limitX',10),
           Task('t1',
              Event('e')))

# Using <node> += <attribute>     adding a single attribute
node += Edit(home='COURSE')

# Using <node> += [ <attributes> ]  - use list to add multiple attributes
node += [ Edit(home='COURSE'), Limit('limitY',10), Event(1) ]

# Using node + <attributes>  - A node container(suite | family) must appear on the left hand side. Use brackets
to control scope.
node + Edit(home=COURSE) + Limit('limitZ',10)

# In this example, variable 'name' is added to suite 's/' and not task 't3'
suite = Suite("s") + Family("f") + Family("f2") + Task("t3") + Edit(name="value")

suite s
  edit name 'value'
  family f
  endfamily
  family f2
  endfamily
  task t3
endsuite


# here we use parenthesis to control where the variable gets added
suite = Suite("s") + Family("f") + Family("f2") + (Task("t3") + Edit(name="value"))

 suite s
  family f
  endfamily
  family f2
  endfamily
  task t3
    edit name 'value'
endsuite
```

Previous Up Next