Add Trigger

Previous Up Next
In the previous exercise, we saw that the two tasks were running simultaneously.
We would like now to make sure that **t2** only runs once **t1** is complete.
For this, we have to define a *trigger*
Triggers are used to declare *dependencies* between two tasks.
For instance, the second task might need data created by the first task.
When ecFlow tries to start a task, it evaluates the *trigger* expression.
If the condition is correct, the task is started, otherwise, the task stays *queued*.
Triggers can be between tasks, or between families, or a mixture.
Remember the two rules:

- A family is *complete* when all its tasks are *complete*
- A task will be started if its triggers and the triggers of **all** its parent families evaluate to true

A *node* can only have one trigger expression, but very complex expressions can be built (and keep in mind that the triggers of the parent nodes are also implicit triggers).
Sometimes triggers are also used to prevent too many jobs from running at the same time. In this case, the use of a *limit* may be a better solution (we will cover limits later on, in the *Limits* section).
Nodes can be addressed in trigger expressions using full names: **/test/f1/t1** refers to the *task* **t1**, and **/test/f1** refers to the *family* **f1**.
In some contexts, ecFlow will accept relative names, such as **../t1**.

```
trigger /test/f1/t1 == complete
```

Triggers can be very complex, and ecFlow supports all kinds of conditions
(not, and, or, ...), in addition, they can also reference Node attributes like
event, *meter*, *variable*, *repeat*, limits and generated variables.

Text

```
# Definition of the suite test.
suite test
   edit ECF_INCLUDE "$HOME/course"   # replace '$HOME' with the path to your home directory
   edit ECF_HOME    "$HOME/course"
   family f1
     edit SLEEP 20
     task t1
     task t2
         trigger t1 eq complete
   endfamily
endsuite
```

Python

The trigger expression can be checked, this is **especially** important when dealing with very large suites and **complex** triggers.

$HOME/course/test.py

```
import os
from ecflow import Defs,Suite,Family,Task,Edit,Trigger

def create_family_f1():
    return Family("f1",
                Edit(SLEEP=20),
                Task("t1"),
                Task("t2",Trigger("t1 == complete")))

print("Creating suite definition")
home = os.path.join(os.getenv("HOME"), "course")
defs = Defs(
        Suite("test",
            Edit(ECF_INCLUDE=home,ECF_HOME=home),
            create_family_f1()))
print(defs)

print("check trigger expressions")
check = defs.check()
assert len(check) == 0, check

print("Checking job creation: .ecf -> .job0")
print(defs.check_job_creation())

print("Saving definition to file 'test.def'")
defs.save_as_defs("test.def")
```

What to do

1. Edit the *suite definition* file to add the *trigger*.
2. Replace the *suite*
   python: python3 test.py ; python3 client.py
   text: ecflow_client --suspend=/test ; ecflow_client --replace=/test test.def
3. Observe the tasks in *ecflow_ui* .
4. See the triggers by selecting **t1** or **t2**.
5. See the trigger relation by clicking on the trigger tab.
6. Search any reference to **t1** by using the search menu.
7. Introduce an error in the trigger expression and ensure that this error is trapped. i.e. change the trigger to.

   Check trigger expressions

   ```
   Trigger("t == complete")  # there is no node with name t, this should be reported as an error
   ```

Previous Up Next