Handling zombies

Manual

When *zombie* s arise they can be handled manually by ecflow_ui. (See *Zombie*) **or** via the command-line interface:

zombie commands

```
ecflow_client –-zombie_get                # This will list all the task/jobs the server thinks are zombies.
ecflow_client --zombie_kill=<task-path>   # Ask the server kill the zombie process. Use ECF_KILL_CMD
ecflow_client –-zombie_fail=<task-path>   # Ask the zombie to fail. This may result in another zombie because
abort child command in the job, will be called.
ecflow_client –-zombie_fob=<task-path>    # Used to unblock the child, allows the job to proceed. However this
will only work for zombies where the password does not match.
ecflow_client –-zombie_adopt=<task-path>  # Copies the password stored on the zombie onto the task. Allows the
job to proceed, and update the state in the server
                                          # ( i.e. due to init,complete,abort).
                                          # It is up to the user, to ensure that the zombie has been dealt
with  before doing this.
ecflow_client –-zombie_remove=<task-path> # Remove the zombie representation in the server. Typically this is
done, when we are sure we have handled the zombie.
                                          # The zombie will re-appear next time it communicates with server, if
this is not the case.
ecflow_client –-zombie_block =<task-path> # Ask the jobs to block at the child command in the job. Prevents the
job from proceeding.
                                          # (This is the default behaviour for the init, complete and abort
child commands)
```

Sometimes we may want the job to proceed but "ecflow_client –zombie_adopt=<task-path>" does not work. i.e. we have the case where zombies password matches, but the process id (ECF_RID) are different.

ecflow_client –zombie_adopt=<task-path>, will not allow this, due to the potential for data corruption.

In this case, the normal behaviour would kill both processes, and re-queue the task.

In the **extreme**, we can bypass the authentication. (i.e. allowing the request to be handled by the server).

This should **ONLY** be done when you are sure the zombie has been killed, and you don't want to re-queue the job.

```
> ecflow_client --alter=add variable ECF_PASS FREE  /path/to/task
```

This is also available from the GUI. Select the task. RMB->Special-> Free password.

After the job has completed, be sure to delete this variable. Otherwise, if zombies arise again, there is a considerable risk of data corruption.

Automated

It is also possible to ask *ecflow_server* to make the same response in an automated fashion. However, **very** careful consideration should be made before doing this. Otherwise, it could mask a serious underlying problem.

The automated response can be defined statically using python and text interface or dynamically (add/remove) via alter.:

- python interface(See ecflow.ZombieAttr)
- text interface (See *Definition file Grammar*)

```
zombie             ::=  "zombie" >> `zombie_type` >> ":" >> !(`client_side_action` | `server_side_action`)
>> ":" >> *`child` >> ":" >> !`zombie_life_time`
zombie_type        ::=  "user" | "ecf" | "path" | "ecf_pid" | "ecf_passwd" | "ecf_pid_passwd"
child              ::=  "init" | "event" | "meter" | "label" | "wait" | "abort" | "complete" | "queue"
client_side_action ::=  "fob" | "fail" | "block"
server_side_action ::=  "adopt" | "delete | "kill"
zombie_life_time   ::=  unsigned integer( default:  user(300), ecf(3600), path(900)  ), the server poll
timer runs every 60 seconds, hence this is the effective minimum value
Where:
```

ecf_pid - PID miss-match, password matches. Job scheduled twice. Check submitter

ecf_pid_passwd - Both PID and password miss-match. Re-queue & submit of the active job?

ecf_passwd - Password miss-match, PID matches, system has re-cycled PID or hacked job file?

ecf - Two init commands or task complete or aborted but receives another child cmd

ecf_user - Created by user action

ecf_path - Task not found. Nodes replaced whilst jobs were running

- --alter command(dynamic)
  ecflow_client --alter add zombie <zombie-attribute> <path>
  ecflow_client --later delete zombie < ecf | path | user> <path>
  However note, the effect will only be seen, when the child command, makes the next attempt to communicate with the server.

The zombie attribute is inherited in the same manner as *Variable inheritance*.

Example: For tasks under suite “s1” add a zombie attribute, such that child label commands(i.e.. ecflow_client –label) never blocks the job: (not strictly needed as this is the default behaviour)

- python

```
s1 = ecflow.Suite('s1')
child_list = [ ChildCmdType.label ]
zombie_attr = ZombieAttr(ZombieType.ecf, child_list, ZombieUserActionType.fob, 300)
s1.add_zombie(zombie_attr)
```

- text

```
suite s1
   zombie ecf:fob:label:
```

- alter
  ecflow_client --alter=add zombie "ecf:fob:label:" /s1

Example: For tasks under suite “s1” add a zombie attribute, such that job that issues the child commands(event, meter, label) never blocks: (not strictly needed as this is the default behaviour)

- python

```
s1 = ecflow.Suite('s1')
child_list = [ ChildCmdType.label, ChildCmdType.event, ChildCmdType.meter ]
zombie_attr = ZombieAttr(ZombieType.ecf, child_list, ZombieUserActionType.fob, 300)
s1.add_zombie(zombie_attr)
```

- text

```
suite s1
   zombie ecf:fob:label,event,meter:
```

- alter
  ecflow_client --alter=add zombie "ecf:fob:label,event,meter:" /s1

Example: For all tasks under family “critical”, if any zombies arise then fail the job(i.e. the zombies process will exit with a failure):

- python

```
with ecflow.Suite('s1') as s1:
   with s1.add_family("critical") as crit :
      child_list = []  # empty child list means apply to all child commands
      for zombie_type in (ZombieType.ecf,ZombieType.path,ZombieType.user,ZombieType.ecf_pid,ZombieType.
ecf_passwd,ZombieType.ecf_pid_passwd):
         crit.add_zombie(ZombieAttr(zombie_type, child_list, ZombieUserActionType.fail, 300))
```

- text

```
   suite s1
     family critical
       zombie ecf:fail::
       zombie path:fail::
       zombie user:fail::
       zombie ecf_pid:fail::
       zombie ecf_passwd:fail::
       zombie ecf_pid_passwd:fail::
```

- alter
  ecflow_client --alter=add zombie "ecf:fail::" /s1
  ecflow_client --alter=add zombie "path:fail::" /s1
  ecflow_client --alter=add zombie "user:fail::" /s1

```
ecflow_client --alter=add zombie "ecf_pid:fail::"  /s1

ecflow_client --alter=add zombie "ecf_passwd:fail::"  /s1

ecflow_client --alter=add zombie "ecf_pid_passwd:fail::"  /s1
```

Here are some further example of using --alter:

- ecflow_client --alter=add zombie "ecf:fob::" /suiteX # fob (init,event, meter, label,abort, complete) child commands. This prevents zombies from blocking the script. Use with **great** care.
- ecflow_client --alter=add zombie "ecf:fail::" /suiteY # fail the script straight away for any child command, in the job file.

You can only add one zombie attribute of each time(ecf, path, user).

To delete a zombie attribute, please use one of:

- ecflow_client --alter=delete zombie ecf /suiteX
- ecflow_client --alter=delete zombie path /suiteX
- ecflow_client --alter=delete zombie user /suiteX

Here are some more examples:

- Add a zombie attribute, that kills the zombie process automatically when a init/complete child is received by the server. This will use whatever is defined for ECF_KILL_CMD

  ```
  ecflow_client --alter=add zombie "ecf:kill:init,complete:" /suiteZ
  ```

- Add a zombie automatically kills zombies process, created out of user action.

  ```
  ecflow_client --alter=add zombie "user:kill::" /suiteZ
  ```

- Add a zombie attribute that adopts all child complete zombies.

  ```
  ecflow_client --alter=add zombie "ecf:adopt:complete:" /suiteZ
  ```

Previous Up Next