Introducing Zombies

A *zombie* is a running job, where the embedded *ecflow_client* child command(init,complete,abort) fails authentication when communicating with the *ecflow_server*

How are zombies created ?

There are a wide variety of reasons why a *zombie* is created.
The most common causes are due to user action:

- The *node* tree is deleted, replaced or reloaded whilst jobs are running
- A *task* is rerun, whilst in a *submitted* or *active* state
- A job is forced to a new state, i.e. *complete*

Rarer causes might be:

- *ecf script* errors, where we have multiple calls to init and complete *child command* s
- The *child command* s in the *ecf script* is placed in the background. In this case, the order in which the *child command* contact the server may be indeterminate.
- Load leveller submitting a job twice
- Server crash and recovered *check point* file is out of date
- Machine crash

How can zombie's be handled ?

The default behaviour for init, complete, abort and wait child commands, is to **block** the job, and for event, label, meter to continue(fob). (With **fob**, the job no longer blocks, but the server will **not** change the node tree)

When blocking the *child command* continues attempting to contact the *ecflow_server*.

There are two environment variables that control how **ecflow_client** handles wait times when trying to connect to the server.

- ECF_TIMEOUT This defines the maximum time client will wait for **any** child command. Hence this includes zombies. Typically applicable when the server is down. It is specified in seconds. The default value is 24 hours. See *ecflow_client*.
- ECF_ZOMBIE_TIMEOUT This is applied to zombies only. It is specified in seconds. The default value is 12 hours. This would apply for **each zombie** init, abort, and complete in the script.

When any of the above timeout is exceeded, **ecflow_client** exits with a failure. Depending on your script, this can be caught by a trap, which will typically call abort child command, this again can wait for 12/24 hours before exiting the process.
Hence it is worth considering if this is appropriate behaviour for your system.

The jobs can also be configured, so that if the server denies the communication, then the *child command* can be set to fail immediately.
(This can be done setting the environment variable ECF_DENIED in your scripts. See *ecflow_client*)
This can be useful to detect network issues early.

ecflow_ui provides a tab that lists all the zombies and the actions that can be taken.

The zombie's tab is shown, in the info panel when the server node(i.e. topmost) is selected.

The actions include:

- Terminate:

  The *child command* is asked to **fail**.
  Depending on your scripts, if trapping is enabled, this may cause the abort *child command* to be called.
  Which again will be flagged as a *zombie*.
- Fob:

  Allow the job to continue. The *child command* completes and hence no longer blocks the job.

  Great care should be taken when this action is chosen.
  If we have two jobs running, they may cause data corruption.
  Even when we have a single job, issues can arise.
  i.e.. if the associated command was an event *child command*, then the *event* would not be set. If this *event* was used in a *trigger* expression,it would never evaluate.
- Delete:

  Remove the *zombie* from the server. The job will continue blocking, hence

when the *child command* next contacts the *ecflow_server*, the *zombie* will re-appear.
If the job is killed manually, then this option can be used.

- Rescue:

  Adopt the zombie and update the node tree.
  The unique password(ECF_PASS) on the zombie is copied over to the *task*, so that the next *child command* will continue as normal.
  This should only be used when the user is sure there are no additional jobs.
- Kill:

  Applies the kill command (ECF_KILL_CMD) using the process id stored on the *zombie*.
  If the script has correct signal trapping, this should end up calling abort.
  Note: path zombies will need to be killed manually.

Of the four actions above, only Rescue will allow *child command* to change the state of the node tree.

What to do

1. Create a *zombie* by starting a *task*, and setting it to *complete* immediately via ecflow_ui
2. Inspect the log file, it will show you how the zombie has arisen.
3. Inspect the zombie tab in *ecflow_ui* (select the host node, then select the zombie's tab)
4. Experiment with the different actions on the zombie
5. Since the default ECF_ZOMBIE_TIMEOUT is 12hr, change this to 1 minute, by editing your **head.h.**

```
export ECF_ZOMBIE_TIMEOUT=60 # specified in seconds
```

Previous Up Next