# Trigger Event Syntax

Breaking down the trigger event specification syntax, it's composed of 4 fields:

```
name-type-context-extra
```

Examples are:

- `complete-job-self` (when this job completes)
- `failed-work-self-*` (when any frame in this job fails)
- `complete-work-render-5` (when frame 5 in the job labelled "render" completes) - see Referring to a job by label

## Trigger syntax fields

1. Event names
2. Event Types
3. Event Contexts
4. Context-specific extra fields (only certain context need this)

## Referring to other jobs in a callback via job labels vs job id's

Qube introduces the concept of job **labeling**. A job label is a separate field in a Qube job which is used to help other jobs **refer to that job by name rather than its job ID**.

When designing a job dependency graph, developers were previously forced to submit the jobs in order of precedence, collecting job ids and using them to initialize the child jobs. This technique is messy and a takes a significant amount of development to implement. It also limits the dependencies to a directed graph, and will not lend itself to a feedback loop job easily.

Another alternative is to use the job's name to identify the job's dependency relationship. This method doesn't work well because a user is then committed to a strict naming convention when submitting to the farm.

To solve this, Qube uses its **process group** job attribute in combination with the job's label. The only prerequisite is that the jobs be submitted with the same process group ID. All jobs submitted with the same `qb.submit()` call are automatically joined into a new pgrp; each has the same process group ID, which is the job ID of the first job submitted (also known as the ***pgrp leader***).

> ⚠ Job Labels **must be unique within the process group**.
>
> The Qube Supervisor automatically enforces the uniqueness requirement and will not allow duplicate job labels to be submitted, instead the submission will be rejected

During submission, a developer *may* link jobs to the same process group by collecting the lead job's process group ID and then using that to submit the successive jobs, setting each job's **`pgrp`** value. A simpler method is to submit all the jobs under the same API submit call, which automatically attaches all jobs to the same process group.

This process group/label system solves 2 problems:

1. The developer isn't forced to collect the job ids.
2. The developer isn't required to use a naming convention for their jobs.

The system also offers several major benefits:

1. Resubmission, cloning and storage of a process group are simpler.
2. Feedback loop job relationships are made possible.

## Event Names

The *name* is the component of the event which details when the event should take place. This is either pre-defined, or user-defined.

The possible pre-defined event names for jobs are:

| Event Name | Event Trigger |
|---|---|
| complete | Job is set to complete |
| done | Job is set to complete or killed or failed |
| submit | Job has been submitted |

| | |
|---|---|
| killed | Job has been killed |
| blocked | Job has been blocked |
| failed | Job has failed |
| running | Job has started running |
| waiting | Job has been set to waiting |
| assigned | Job has been assigned to a host |
| removed | Job has been removed |
| modified | Job has been modified |
| dummy | Event is time-based - see here |

*Table 3: Event Names*

## Event Types

The *type* of the event allows the system to identify the kind of event referred to. The available pre-defined names are relative to the specification of the type.
The "types" of events

| | |
|---|---|
| job | Specifies the entire job |
| subjob | Specifies a single subjob |
| work | Specifies a single work agenda item |
| host | Specifies the event belongs to a host |
| time | Specifies a time-based event - see here |
| repeat | Specifies an interval event |
| global | Specifies a time-based event that exists independent of a job |

*Table 4: Event Types*

## Event Contexts

The *context* or the "label" of the event is a specification to narrow the scope of the event. When someone specifies 'job' they don't normally mean all jobs, so a context is required to determine which job they are describing. A context can be specified in 3 different forms:

1. pre-defined label
2. process group label (see: Job Labels)
3. job ID

*Pre-defined labels*

| label | description |
|---|---|
| self | This job |
| parent | The job referred to by *pid* |

## Context-specific extra fields

The *extra* in the event specification refers to the type of event. Each type of event may require additional information.

In the case of the job, it requires nothing more. However in the case of the subjob it requires the subjob's ID number.

**Example:**

When the job's instance #5 is done, execute callback: done-subjob-self-5

### Extra information required by each *type*:

| | |
|---|---|
| job | None |
| subjob | *subjobid* |
| work | *workid/name* |
| host | *hostname* |
| time | *timeofday* |
| repeat | *timeofday-interval* |
| global | None |
| globaltime | *timeofday* |
| globalrepeat | *timeofday-interval* |

⚠ The time of day is given in Qube time format. Intervals are in seconds.