Callbacks

This page refers to **job-specific** callbacks. For callbacks that are applied to all jobs see the documentation for Universal Callbacks

A **callback** is Qube's mechanism to allow custom execution of queuing logic depending upon the events which occur during the lifetime of a Qube job or upon pre-defined system events.

The use of callbacks can range from sending email when a job has completed, to designing complex dependency trees or direct integration with an asset tracking system. There is also a special case that allows for delayed execution of the job, see: How to submit a job that will wait until later to run

When specifying a callback, Qube requires a little information in order to execute it properly. Each callback must include:

1. The **trigger** expression to specify when the callback should be executed.
2. The scripting **language** to use for the callback.
3. The action or **code** the callback will execute each time the trigger condition is met.

Callbacks In Detail:

Callbacks are defined as a list of one or more key/value pairs which is appended to the job's `callback` list.

- Callback execution environment
- Callback languages
- Triggers
- Properties and Uses

Python API Examples - refer to the above pages for more information

JobB waits for the entire jobA to complete

Using the "qube" callback language

```
frameRange = '1-10'

jobA = {
 'name': 'my ribgen job',
 'label': 'ribgenLabel',
 'prototype': 'cmdrange',
 'package': {
  'cmdline': 'my ribgen command...'
 },
 'agenda': qb.genframes(frameRange),
}

jobB = {
 'name': 'my render job',
 'label': 'render',
 'status': 'blocked',
 'prototype': 'cmdrange',
 'package': {
  'cmdline': 'my render command...'
 },
 'agenda': qb.genframes(frameRange),
}

callbacks = [
 {
  'triggers': 'complete-job-ribgenLabel',
  'language': 'qube',
  'code': 'unblock-self'
 }
]
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB] )
```

Using the "python" callback language is identical to the first example, except the callback's "language" and "code" is different

```
jobA = {
 .
 .
 .
}

jobB = {
 .
 .
 .
}

cbCode = 'jobId = qb.jobid()\n'
cbCode += 'qb.unblock(jobId)\n'

callbacks = [
 {
  'triggers': 'complete-job-ribgenLabel',
  'language': 'python',
  'code': cbCode
}
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB] )
```

Each frame in JobB waits for the its corresponding frame in jobA to complete

Using the "qube" callback language

```
frameRange = '1-10'

jobA = {
 'name': 'my ribgen job',
 'label': 'ribgenLabel',
 'prototype': 'cmdrange',
 'package': {
  'cmdline': 'my ribgen command...'
 },
 'agenda': qb.genframes(frameRange),
}

jobB = {
 'name': 'my render job',
 'label': 'render',
 'status': 'blocked',
 'prototype': 'cmdrange',
 'package': {
  'cmdline': 'my render command...'
 },
 'agenda': qb.genframes(frameRange),
}

callbacks = []
for work in jobB['agenda']:
 work['status'] = 'blocked'
 callbacks.append(
  {
   'triggers': 'complete-work-ribgenLabel-%s' % work['name'],
   'language': 'qube',
   'code': 'unblock-work-self',
  }
 )
]
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB] )
```

Using the "python" callback language is identical to the first example, except the callback's "language" and "code" is different

```
jobA = {
 .
 .
 .
}

jobB = {
 .
 .
 .
}

callbacks = []
for work in jobB['agenda']:
 work['status'] = 'blocked'
 frameNumber = work['name']

    # the agenda item's callback should unblock both itself and the job
    cbCode = 'jobId = qb.jobid()\n'
    cbCode += 'qb.workunblock("%%s:%s" %% jobId)\n' % frameNumber
    cbCode += 'qb.unblock(jobId)\n'

 callbacks.append(
  {
   'triggers': 'complete-work-ribgenLabel-%s' % frameNumber,
   'language': 'python',
   'code': cbCode,
  }
 )
]
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB] )
```

JobC waits for jobA and jobB

This can be done using any of the callback languages, the only difference is the **triggers** value

Using the "python" callback language is identical to the first example, except the callback's "language" and "code" is different

```
jobA = {
 'label': 'ribGen',
 .
 .
}

jobB = {
 'label': 'render',
 .
 .
}

jobC = {
 'label': 'composite'
 .
 .
}

callbacks = []
for work in jobC['agenda']:
 work['status'] = 'blocked'
 frameNumber = work['name']

    # the agenda item's callback should unblock both itself and the job
    cbCode = 'jobId = qb.jobid()\n'
    cbCode += 'qb.workunblock("%%s:%s" %% jobId)\n' % frameNumber
    cbCode += 'qb.unblock(jobId)\n'

 triggerStr = 'complete-work-ribgen-%s' % frameNumber'
 triggerStr += ' && '
 triggerStr += 'complete-work-render-%s' % frameNumber'

 callbacks.append(
  {
   'triggers': triggerStr,
   'language': 'python',
   'code': cbCode,
  }
 )
]
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB, jobC] )
```

JobD waits for either (jobA AND jobB) OR jobC

This is identical to the above example, except the 'triggerStr'" is different

```
jobA = {
 'label': 'ribGen',
 .
 .
}

jobB = {
 'label': 'render',
 .
 .
}

jobC = {
 'label': 'composite',
 .
 .
}

jobD = {
 'label': 'sendToDailies',
 .
 .
}
callbacks = []
for work in jobD['agenda']:
 work['status'] = 'blocked',
 frameNumber = work['name']

    # the agenda item's callback should unblock both itself and the job
    cbCode = 'jobId = qb.jobid()\n'
    cbCode += 'qb.workunblock("%%s:%s" %% jobId)\n' % frameNumber
    cbCode += 'qb.unblock(jobId)\n'

 triggerSt = '('
 triggerStr += 'complete-work-ribgen-%s' % frameNumber'
 triggerStr += ' && '
 triggerStr += 'complete-work-render-%s' % frameNumber'
 triggerStr += ')'

 triggerStr += ' || '
 triggerStr = 'complete-work-composite-%s' % frameNumber'

 callbacks.append(
  {
   'triggers': triggerStr,
   'language': 'python',
   'code': cbCode,
  }
 )
]
jobB['callbacks'] = callbacks

qb.submit( [jobA, jobB, jobC, jobD] )
```