# FICO® Xpress Optimization

0.0.4

REFERENCE MANUAL

Package jobqueue

FICO®

jobqueue package

Deliverable Version: A

Last Revised: 5 May, 2023

Version 0.0.4

# Contents

# CHAPTER 1

# Introduction

The package *jobqueue* provides a set of Mosel subroutines for managing the remote execution of submodels via *mmjobs* and *mmhttp* functionality. Individual model files, optionally with data files, are run on one worker from a pool (queue) of remote machines that are configured via this package. *jobqueue* also implements the handling of output and errors from the submodels and the generation and retrieval of individual result output files.
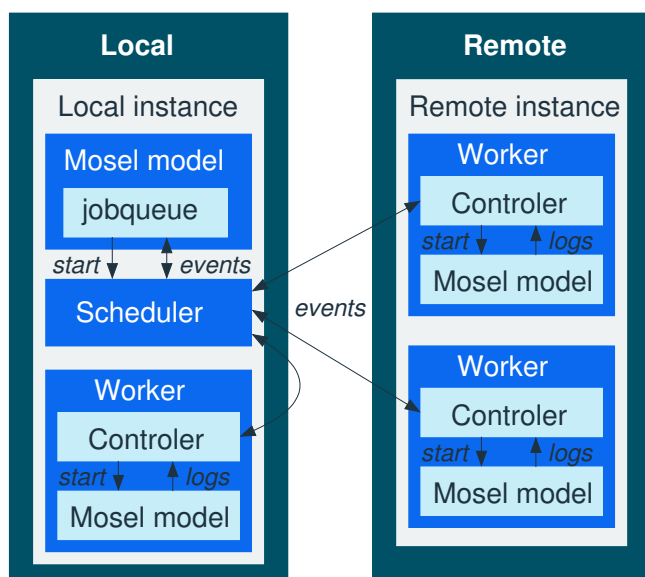


**Figure 1.1**: Architecture of jobqueue runs

- Step 1: Configuration of queues (node lists)

    - definition of worker instances (remote via rcmd or xprmsrv, on local node, or on same Mosel instance as the main model), initial status check, connections are opened when needed

- Step 2: Configuration of *computation jobs* (models+other required files)

    - Mosel file to be run (in source or compiled form), optional addition of other required (data) files, optional specification of a result file, optional specification of retry attempts in case of server failure

- Step 3: Management of *task queues*

    - jobs are turned into computation tasks by adding them to a queue, submodel execution starts automatically

- – optional specification of runtime parameters

- ■ Step 4: Supervision of nodes and model execution on workers via *controler programs*

  - – waiting for job termination: can perform conditional wait (time limit)
  - – explicit deletion of tasks or queues if not busy/being processed

- ■ Step 5: *Reporting* functionality

  - – display of queue information: workers, pending and running tasks
  - – submodel status (status file, host information file), output produced by submodel run (output log, error log), and optional result file with model-specific contents (only if specified in job definition)

The package *jobqueue* requires (at least) Mosel 5. That is, the main model on the local instance that uses this package needs to be run with Mosel 5 or newer, but remote (worker) instances can use older releases (starting with Mosel 4.*).

The main model to be provided by the developer implements the steps outlined avove, that is, it defines the computing tasks, assigns them to a job queue, waits for their termination and (optionally) reports on their status and results. All other components shown in Figure *Architecture of jobqueue runs* (*jobqueue* scheduler, controler programs on workers) are created and managed automatically by the jobqueue library. See the file `mainjq.mos` that is provided as test with the *jobqueue* distribution for a complete example.

# Control parameters

**jq_verbose** : `integer`

> Package output level

| | |
|---|---|
| **Default value** | 0 |
| **Notes** | 1. Positive values up to 10 or 0 to disable output from this package. |
| | 2. The parameter is accessed via Mosel's getparam/setparam routines, for example, use `setparam("jq_verbose", 2)` to enable logging and some debugging output by *jobqueue*. |

# Constants

**RT_CANCELLED = -11**

> Model status: cancelled before execution

**RT_COMPERR = -12**

> Model status: compilation failed

**RT_FCPYERR = -14**

> Model status: files could not be uploaded to worker

**RT_LOADERR = -13**

> Model status: bim file could not be loaded

**RT_PENDING = -10**

> Model status: waiting for execution

**RT_SYSERR = -15**

> Model status: controler could not be run on worker

# Types

**jq_job** : `record`

    A job definition

    **srcfiles** : `list of text`

        list of files to upload on the worker

    **dstfiles** : `list of text`

        names of the files on the worker

    **resultfile** : `text`

        file containing the result after execution

    **maxretry** : `integer`

        maximum number of attemps at running the job in case of server disconnection

    **Note**        Entities of this type should be populated using `jobinit`, `jobadd` or `jobsetresult`.

**jq_qinfo** : `record`

    Queue information as returned by `queuegetinfo`

    **nbwk** : `integer`

        total number of workers

    **nbdis** : `integer`

        number of disabled workers

    **running** : `list of integer`

        list of running tasks

    **pending** : `list of integer`

        list of tasks waiting for execution

# CHAPTER 5
# Subroutines

# jobadd

**Purpose**

Add a file to a job

**Synopsis**

```
procedure jobadd(j:jq_job, src:text, dst:text)
procedure jobadd(j:jq_job, src:text)
```

**Arguments**

| | |
|---|---|
| `j` | a job |
| `src` | a data file |
| `dst` | filename to be used on the host for src |

**Example**

See `jobinit`

**Further information**

The destination filename must be an actual filename, not including any I/O driver. If the 'dst' argument is not specified its value is deduced from the 'src' argument, in which case the filename specified for 'src' must be a complete path (excluding the use of I/O drivers, including 'tmp:'). If the source filename in 'src' is specified as an extended filename using an I/O driver, then the 'dst' filename must be stated explicitly, using the 3-argument version of this subroutine.

# jobinit

## Purpose

Initialize a job

## Synopsis

```
procedure jobinit(j:jq_job, src:text, dst:text, maxretry:integer)
procedure jobinit(j:jq_job, src:text)
procedure jobinit(j:jq_job, src:text, maxretry:integer)
```

## Arguments

| | |
|---|---|
| `j` | job to initialize |
| `src` | file containing the model to run (.bim or .mos) |
| `dst` | filename to be used on the host for src |
| `maxretry` | maximum number of attempts at running the model in case of server disconnection (default: 0) |

## Example

The following example defines three jobs with different configurations.

```
public declarations
  job,job2,job3: jq_job
  tasks: list of integer
  mpar: text
end-declarations

jobinit(job,"simple.bim")           ! Model is already compiled

jobinit(job2,"simplewdata.mos")     ! Model gets compiled by 'jobqueue'
jobadd(job2,"simpledata.txt")       ! Add a data file for this model
jobsetresult(job2,"results.txt")    ! Specify the result file (single file)

jobinit(job3,"simple.mos",2)        ! 2 attempts to run model if server failure
```

## Further information

The destination filename must be an actual filename, not including any I/O driver. If the 'dst' argument is not specified its value is deduced from the 'src' argument, in which case the filename specified for 'src' must be a complete path (excluding the use of I/O drivers, including 'tmp:'). If the source filename in 'src' is specified as an extended filename using an I/O driver, then the 'dst' filename must be stated explicitly, using the 4-argument version of this subroutine.

# jobsetresult

**Purpose**

Define the result file for a job

**Synopsis**

```
procedure jobsetresult(j:jq_job, rfile:text)
```

**Arguments**

j       a job

rfile   file containing the result of the execution

**Example**

See `jobinit`

# queueaddjob

## Purpose

Add a job to a queue

## Synopsis

```
function queueaddjob(qid:integer, j:jq_job, rtp:text):integer
function queueaddjob(qid:integer, j:jq_job):integer
```

## Arguments

qid      a queue ID

j      job to add

rtp      parameter string to be used for execution

## Return value

A unique task ID

## Example

The following example queues several instances of a job, specifying different runtime parameter settings per task. It then adds an instance of a second job.

```
public declarations
  job,job2: jq_job
  tasks: list of integer
  queue: integer
  mpar: text
end-declarations

!... create 'queue' and initialize the jobs ...

forall(i in 1..NB) do                ! Queue some jobs with runtime parameters
  setmodpar(mpar, "WAIT_IN_SUBMODEL", i)
  setmodpar(mpar, "IMPORTANT_PARAM", text("Hello World ") + (i^2))
  tasks+=[queueaddjob(queue, job, mpar)]
end-do
tasks+=[queueaddjob(queue, job2)]   ! Queue another job
```

# queueaddnode

**Purpose**

Add an execution node to a queue

**Synopsis**

```
function queueaddnode(qid:integer, cstr:text, mt:integer):integer
function queueaddnode(qid:integer, cstr:text):integer
```

**Arguments**

qid      a queue ID

cstr     connection string (if "*" the current instance is used)

mt       maximum number of models to run on this node

**Return value**

ID for the new node

**Example**

See `queuenew`

# queuedel

**Purpose**

Delete a queue

**Synopsis**

```
function queuedel(qid:integer):boolean
```

**Return value**

true if successful or false if the queue is busy

**Example**

See `queuewait`

**Further information**

A queue cannot be deleted when it is running tasks

# queuedeltasks

**Purpose**

Delete all tasks associated to a queue

**Synopsis**

```
procedure queuedeltasks(qid:integer)
```

**Argument**

qid      a queue ID

**Example**

See `queuewait`

**Further information**

All tasks waiting for execution are removed from the queue and deleted, all tasks that have finished execution are also deleted.

# queueflush

## Purpose

Flushes a queue

## Synopsis

```
procedure queueflush(qid:integer)
```

## Argument

qid      a queue ID

## Further information

All tasks waiting for execution are removed from the queue and deleted.

# queuegetinfo

## Purpose

Retrieve queue information

## Synopsis

```
procedure queuegetinfo(qid:integer, jqi:jq_qinfo)
```

## Arguments

qid      a queue ID

jqi      record where the queue status is returned

## Example

The following example retrieves and displays information about a queue.

```
public declarations
  queue: integer
  jqi: jq_qinfo
end-declarations

queuegetinfo(queue,jqi)         ! Queue info: workers, pending+running tasks
writeln("Queue info:", jqi)
writeln(queuepending(queue))    ! Display pending tasks
```

# queuenew

## Purpose

Create a new queue

## Synopsis

```
function queuenew:integer
```

## Return value

ID for the new queue

## Example

The following example shows different configuration options for queues.

```
public declarations
  queue: integer
  nd1,nd2,nd3,nd4: integer
end-declarations

queue:=queuenew                    ! Create a new queue

nd1:=queueaddnode(queue,"*",2)   ! Use same instance as current (no 'connect')
                                 ! with 2 workers (=2 concurrent executions)
nd2:=queueaddnode(queue,"",3)    ! New instance (via rcmd) on current machine
nd3:=queueaddnode(queue,"somemach")   ! New inst. (xprmsrv) on remote machine
nd4:=queueaddnode(queue,"localhost")  ! New inst. via xprmsrv on current
```

# queuepending

**Purpose**

Retrieve the number of pending tasks of a queue

**Synopsis**

```
function queuepending(qid:integer):integer
```

**Argument**

qid     a queue ID

**Return value**

Number of tasks executing and waiting for execution

**Example**

See `queuegetinfo`

# queuereset

**Purpose**

Reset a queue

**Synopsis**

```
function queuereset(qid:integer):boolean
```

**Argument**

qid      a queue ID

**Return value**

true if successful or false if queue is still executing tasks

**Example**

See `queuewait`

**Further information**

`queueflush` is called and all nodes are disconnected.

# queuewait

## Purpose

Suspend execution until a queue has run all its tasks

## Synopsis

```
function queuewait(qid:integer, maxtime:real):integer
function queuewait(qid:integer):integer
```

## Arguments

qid   a queue ID

maxtime  maximum amount of time to wait (in seconds)

## Return value

Number of tasks executing and waiting for execution

## Example

The following example shows how to use different forms of 'queuewait' and explicit termination.

```
public declarations
  queue: integer
  rti: integer
  rtb: boolean
end-declarations

rti:=queuewait(queue)          ! Wait for all tasks to terminate
writeln("End of wait: ",rti)   ! Number of tasks executing or waiting

! Alternative forms:
rti:=queuewaitnext(queue)      ! Wait for next task termination
rti:=queuewait(queue,10)       ! Wait for 10 seconds

! Optional: explicit termination/deletion
queuedeltasks(queue)                ! Delete all pending+terminated tasks
rtb:=queuereset(queue)              ! Del. pending + disconnect nodes (if not busy)
rtb:=queuedel(queue)                ! Delete a queue (only if not busy)
```

# queuewaitnext

**Purpose**

Suspend execution until a task finishes in a queue

**Synopsis**

```
function queuewaitnext(qid:integer, maxtime:real):integer
function queuewaitnext(qid:integer):integer
```

**Arguments**

qid        a queue ID

maxtime    maximum amount of time to wait (in seconds)

**Return value**

Number of tasks executing and waiting for execution

**Example**

See `queuewait`

# taskcancel

**Purpose**

Cancel a task

**Synopsis**

```
procedure taskcancel(tid:integer)
```

**Argument**

tid     a task ID

**Further information**

A running task will be stopped.

# taskerrfile

**Purpose**

File name of the error stream of a task

**Synopsis**

```
function taskerrfile(tid:integer):string
```

**Argument**

tid       a task ID

**Return value**

a file name

**Example**

See taskstatfile

**Further information**

This file is populated only after execution of the task.

# taskfree

## Purpose
Delete a task

## Synopsis
```
function taskfree(tid:integer):boolean
```

## Argument
tid      a task ID

## Return value
true if successful or false if the task does not exist or is currently running

## Further information
A task waiting for execution will be removed from its queue.

# taskhostfile

**Purpose**
Host information file of a task

**Synopsis**
```
function taskhostfile(tid:integer):string
```

**Argument**
tid      a task ID

**Return value**
a file name

**Example**
See `taskstatfile`

**Further information**

1. This file contains information on the host running the task. It is available as soon as the task is running.

2. Model status values are documented under subroutine `getstatus` of *mmjobs* in the Mosel Language Reference Manual, with the additional values `RT_PENDING`, `RT_CANCELLED`,`RT_COMPERR`, `RT_LOADERR`, `RT_FCPYERR`, and `RT_SYSERR` defined by *jobqueue*.

# taskoutfile

## Purpose

File name of the output stream of a task

## Synopsis

```
function taskoutfile(tid:integer):string
```

## Argument

tid      a task ID

## Return value

a file name

## Example

See `taskstatfile`

## Further information

This file is populated only after execution of the task.

# taskresfile

**Purpose**

Result file of a task

**Synopsis**

```
function taskresfile(tid:integer):string
```

**Argument**

tid       a task ID

**Return value**

a file name

**Example**

The following example retrieves and displays information from the result file produced by a task.

```
public declarations
  tasks: list of integer
end-declarations
! ... run tasks and wait for termination ...

! Model-specific result data
declarations
  Sol: dynamic array(range) of real
  L: list of text
  val: real
end-declarations

with t=tasks.last do    ! A result file must have been specified for this task
  initializations from taskresfile(t)
    Sol L val
  end-initializations
  writeln("**Result values: Sol=", Sol, " L=", L, " val=", val)
end-do
```

**Further information**

This file is populated only after execution of the task.

# taskrestart

## Purpose
Restart a task

## Synopsis
```
procedure taskrestart(tid:integer)
```

## Argument
tid     a task ID

## Further information
This routine has no effect if the task is already scheduled or running.

# taskstatfile

**Purpose**

Status file of a task

**Synopsis**

```
function taskstatfile(tid:integer):string
```

**Argument**

tid      a task ID

**Return value**

a file name

**Example**

The following example shows how to retrieve and display status information and the output files produced by tasks.

```
public declarations
  status, code: integer
  tasks: list of integer
end-declarations

forall(t in tasks) do
  initializations from taskstatfile(t)  ! Model execution status file
    status code
  end-initializations
  writeln("Status of task ", t, ": ", status, "/", code)
  writeln("Host info task ", t, ":")
  fcopy(taskhostfile(t),0,"",0)          ! Host (node) information file
  if status=0 then
    writeln("Output of task ", t, ":")
    fcopy(taskoutfile(t),0,"",0)         ! Output log file
  else
    writeln("Errors of task ", t, ":")
    fcopy(taskerrfile(t),0,"",0)         ! Error log file
  end-if
end-do
```

**Further information**

This file can be used to retrieve the status of a task.

# Index