

# FICO® Xpress Optimization

Last update 16 May, 2023

0.0.4

REFERENCE MANUAL

Package jobqueue

**FICO**®

©2019–2023 Fair Isaac Corporation. All rights reserved. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

FICO is a registered trademark of Fair Isaac Corporation in the United States and may be a registered trademark of Fair Isaac Corporation in other countries. Other product and company names herein may be trademarks of their respective owners.

jobqueue package

Deliverable Version: A

Last Revised: 16 May, 2023

Version 0.0.4

# Contents

---

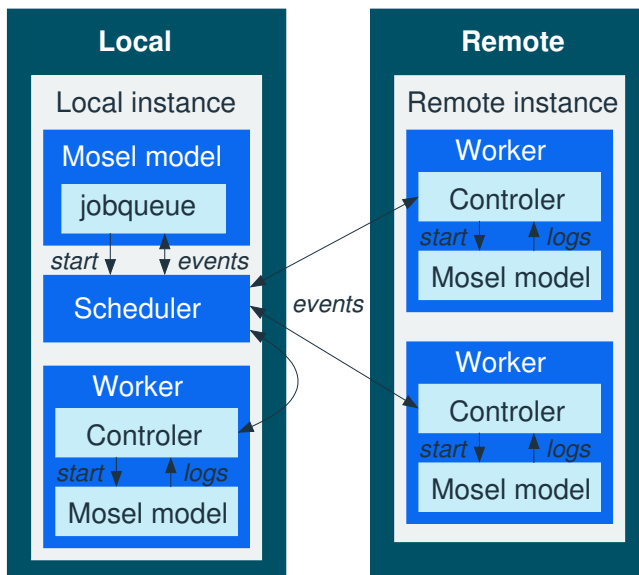
<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Control parameters</b>	<b>3</b>
<b>3</b>	<b>Constants</b>	<b>4</b>
<b>4</b>	<b>Types</b>	<b>5</b>
<b>5</b>	<b>Subroutines</b>	<b>6</b>
	jobadd . . . . .	7
	jobinit . . . . .	8
	jobsetresult . . . . .	9
	queueaddjob . . . . .	10
	queueaddnode . . . . .	11
	queuedel . . . . .	12
	queuedeltasks . . . . .	13
	queueflush . . . . .	14
	queuegetinfo . . . . .	15
	queuenew . . . . .	16
	queuepending . . . . .	17
	queuereset . . . . .	18
	queuwait . . . . .	19
	queuwaitnext . . . . .	20
	taskcancel . . . . .	21
	taskerrfile . . . . .	22
	taskfree . . . . .	23
	taskhostfile . . . . .	24
	taskoutfile . . . . .	25
	taskresfile . . . . .	26
	taskrestart . . . . .	27
	taskstatfile . . . . .	28
	<b>Index</b>	<b>29</b>

## 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 above, 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.

## CHAPTER 2

# 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*.

## CHAPTER 3

# 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

## CHAPTER 4

# 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 attempts 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

---

<code>jobadd</code>	Add a file to a job	p. 7
<code>jobinit</code>	Initialize a job	p. 8
<code>jobsetresult</code>	Define the result file for a job	p. 9
<code>queueaddjob</code>	Add a job to a queue	p. 10
<code>queueaddnode</code>	Add an execution node to a queue	p. 11
<code>queuedel</code>	Delete a queue	p. 12
<code>queuedeltasks</code>	Delete all tasks associated to a queue	p. 13
<code>queueflush</code>	Flushes a queue	p. 14
<code>queuegetinfo</code>	Retrieve queue information	p. 15
<code>queuenew</code>	Create a new queue	p. 16
<code>queuepending</code>	Retrieve the number of pending tasks of a queue	p. 17
<code>queuereset</code>	Reset a queue	p. 18
<code>queuwait</code>	Suspend execution until a queue has run all its tasks	p. 19
<code>queuwaitnext</code>	Suspend execution until a task finishes in a queue	p. 20
<code>taskcancel</code>	Cancel a task	p. 21
<code>taskerrfile</code>	File name of the error stream of a task	p. 22
<code>taskfree</code>	Delete a task	p. 23
<code>taskhostfile</code>	Host information file of a task	p. 24
<code>taskoutfile</code>	File name of the output stream of a task	p. 25
<code>taskresfile</code>	Result file of a task	p. 26
<code>taskrestart</code>	Restart a task	p. 27
<code>taskstatfile</code>	Status file of a task	p. 28

---

## 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, "simplifiedata.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 (that is, no 'connect'). If "" (empty string), a new instance is created via 'rcmd' on the current machine/node.
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 [queuwait](#)

### 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 [queuwait](#)

**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  
jq\_i     record where the queue status is returned

### Example

The following example retrieves and displays information about a queue.

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

queuegetinfo(queue, jq_i)      ! Queue info: workers, pending+running tasks
writeln("Queue info:", jq_i)
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 [queuwait](#)

### Further information

[queueflush](#) is called and all nodes are disconnected.

## queuwait

---

### Purpose

Suspend execution until a queue has run all its tasks

### Synopsis

```
function queuwait(qid:integer, maxtime:real):integer
function queuwait(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 'queuwait' and explicit termination.

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

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

! Alternative forms:
rti:=queuwaitnext(queue)      ! Wait for next task termination
rti:=queuwait(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)
```

## queuwaitnext

---

### Purpose

Suspend execution until a task finishes in a queue

### Synopsis

```
function queuwaitnext(qid:integer, maxtime:real):integer  
function queuwaitnext(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 [queuwait](#)

## 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

---

## J

jobadd, 7  
jobinit, 8  
jobsetresult, 9  
jq\_job, 5  
jq\_qinfo, 5  
jq\_verbose, 3

## Q

queueaddjob, 10  
queueaddnode, 11  
queuedel, 12  
queuedeltasks, 13  
queueflush, 14  
queuegetinfo, 15  
queuenew, 16  
queuepending, 17  
queuereset, 18  
queuwait, 19  
queuwaitnext, 20

## R

RT\_CANCELLED, 4  
RT\_COMPERR, 4  
RT\_FCPYERR, 4  
RT\_LOADERR, 4  
RT\_PENDING, 4  
RT\_SYSERR, 4

## T

taskcancel, 21  
taskerrfile, 22  
taskfree, 23  
taskhostfile, 24  
taskoutfile, 25  
taskresfile, 26  
taskrestart, 27  
taskstatfile, 28