FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Job Operation Software Administrator's Guide for Job Operation Manager Hook

J2UL-2459-02ENZ0(05)
September 2023

Preface

Purpose of This Manual

This manual describes feature of the job operation manager hook, the function of the Job Operation Manager in Technical Computing Suite.

Intended Readers

This manual is intended for the administrator who operates and manages the system with the Job Operation Software.

The manual assumes that readers have the following knowledge:

- Basic Linux knowledge
- General knowledge of the Job Operation Software from "Job Operation Software Overview"
- Knowledge of job operation from "Job Operation Software Administrator's Guide for Job Management"

Organization of This Manual

This manual is organized as follows.

Chapter 1 What are the Hooks of the Job Operation Management Function?

This chapter describes the mechanisms of hooks in the job operation management function.

Chapter 2 Creating and Incorporating Hooks

This chapter describes how to incorporate hooks created by the administrator into operation.

Appendix A Functional Comparison of Hooks

This appendix describes functional differences between hooks.

Appendix B Environment Variables

This appendix describes environment variables that are set by the job operation management function when executing hooks.

Appendix C References

This appendix is the reference for the job management exit function.

Notation Used in This Manual

Representation of units

The following table lists the prefixes used to represent units in this manual. Basically, disk size is represented as a power of 10, and memory size is represented as a power of 2. Be careful about specifying them when displaying or entering commands.

Prefix	Value	Prefix	Value
K (kilo)	10^3	Ki (kibi)	2^{10}
M (mega)	10^6	Mi (mebi)	2^{20}
G (giga)	10^9	Gi (gibi)	2^{30}
T (tera)	10^{12}	Ti (tebi)	2^{40}
P (peta)	10^{15}	Pi (pebi)	2^{50}

Notation of model names

In this manual, the computer that based on Fujitsu A64FX CPU is abbreviated as "FX server", and FUJITSU server PRIMERGY as "PRIMERGY server" (or simply "PRIMERGY").
Also, specifications of some of the functions described in the manual are different depending on the target model. In the description of such a function, the target model is represented by its abbreviation as follows:
[FX]: The description applies to FX servers.
[PG]: The description applies to PRIMERGY servers.

Administrators

The Job Operation Software has different types of administrator: system administrator, cluster administrator, and job operation administrator. However, the descriptions in this manual refer to functions available to administrators who have job operation administrator privileges or higher. For this reason, the text does not distinguish among the administrators but simply refers to them as "administrator."

Path names of the commands

In the examples of the operations, the path names of the commands in the directory /bin, /usr/bin, /sbin or /usr/sbin might not be represented by absolute path.

Symbols in This Manual

This manual uses the following symbols.

Note

The Note symbol indicates an item requiring special care. Be sure to read these items.

See

The See symbol indicates the written reference source of detailed information.

Information

The Information symbol indicates a reference note related to Job Operation Software.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Red Hat and Red Hat Enterprise Linux are registered trademarks of Red Hat, Inc. in the U.S. and other countries.
- All other trademarks are the property of their respective owners.

Date of Publication and Version

Version	Manual code
September 2023, Version 2.5	J2UL-2459-02ENZ0(05)
March 2023, Version 2.4	J2UL-2459-02ENZ0(04)
December 2020, Version 2.3	J2UL-2459-02ENZ0(03)
September 2020, Version 2.2	J2UL-2459-02ENZ0(02)
June 2020, Version 2.1	J2UL-2459-02ENZ0(01)
March 2020, Second version	J2UL-2459-02ENZ0(00)
January 2020, First version	J2UL-2459-01ENZ0(00)

Copyright

Update history

Changes	Location	Version
Fixed errata.	-	2.5
Added the commands that cannot be executed from prolog script and epilog script are listed in the Notes.	2.1.1	2.4
Fixed errata.	-	2.3
Added a countermeasure to suppress disturbance for execution performance of job.	2.3.1	2.2
Fixed incorrect descriptions of the directory where the prologue and epilogue scripts are placed and the directory on which they are run.	1.1.2 2.1.1	2.1
Added that node statistical information cannot be acquired or set if the job information acquisition function pjmx_getinfo_stats_item_*type*() or pjmx_getinfo_stats_item_array_*type*(), or the job information setting function pjmx_setinfo_stats_item_*type*() is called from the job manager exit function pjmx_quejob() or pjmx_startjob().	2.2.1 C.3.1 C.4.1	
Fixed incorrect descriptions about return values of the functions.	C.4.1 C.4.2 C.4.3	
Added the environment variable PJM_NET_ROUTE.	Appendix B	2
Added the job information acquisition functions pjmx_getinfo_execnode_list() and pjmx_getinfo_net_route().	C.3.1	
Added the job information setting function pjmx_setinfo_net_route().	C.4.1	
Changed the look according to product upgrades.	-	

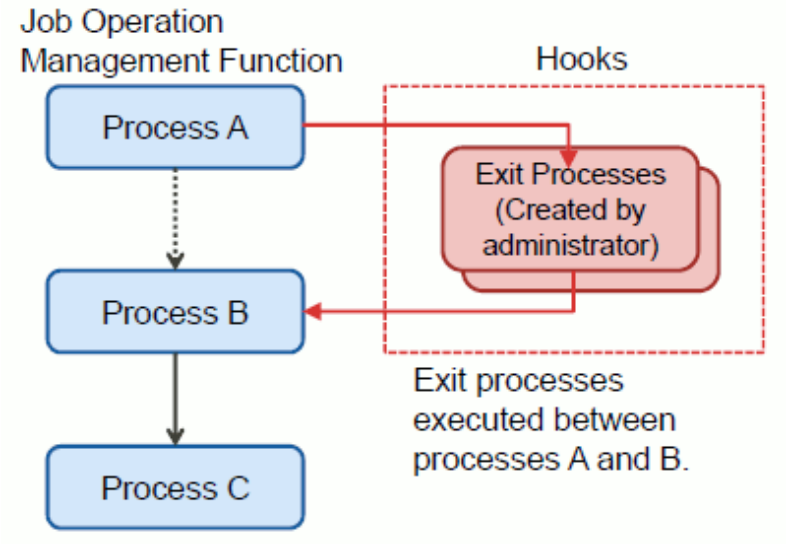
Contents

Chapter 1 What are the Hooks of the Job Operation Management Function?

Administrators may want to control jobs during job operations, based on their own criteria. For example, they may check the budget before executing a job. If the budget is not sufficient, they will reject acceptance of the job. To incorporate such unique criteria into job control, use the "hooks" of the job operation management function.

A hook is a mechanism that executes an administrator-prepared process (exit process) at a specific time during processing by the job operation management function. Job operations can be controlled according to the result returned by the exit process.

Figure 1.1 Conceptual diagram of hooks in the job operation management function



The job operation management function provides the following four types of hooks:

- Job manager exit function

  This is a hook for the job manager function. Use this hook for control related to accepting or executing a job.

- Job scheduler exit function

  This is a hook for the job scheduler function. Use this hook for control related to scheduling a job.

- Prologue and epilogue function

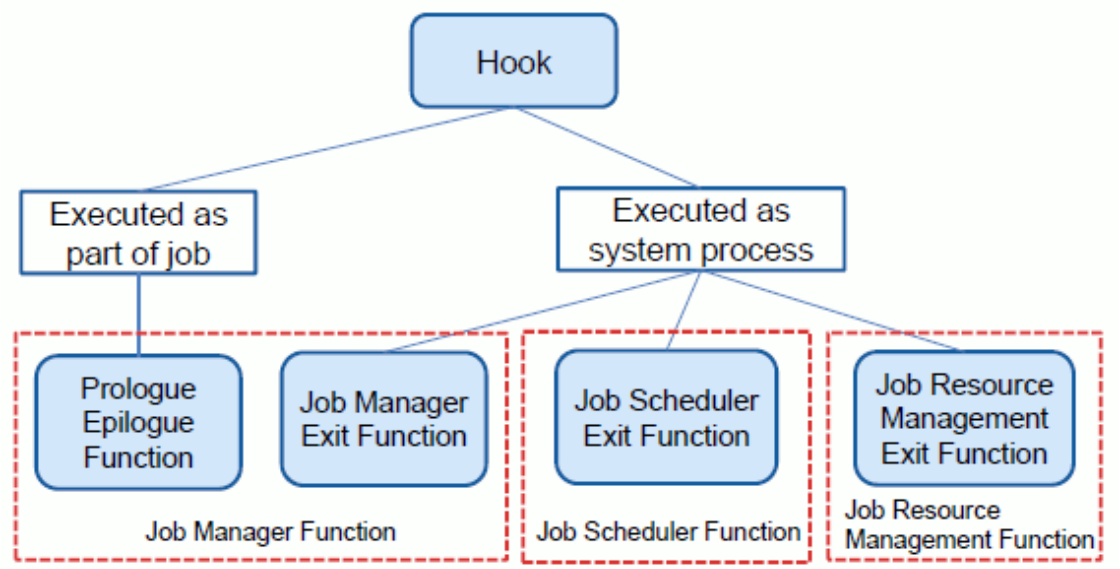
  This is a hook for the job manager function. Use this hook to execute a unique process before and after executing a job.

- Job resource manager exit function

  This is a hook for the job resource manager. Use this hook to execute a unique process with an awareness of the timing for allocating or releasing resources before and after the execution of each job.

The prologue and epilogue function is executed as part of a job. The other hooks are executed as system processes.

Figure 1.2 Types of hooks



The following sections describe the respective hooks.

See

For details on the features of each hook and the differences in timing, etc. when calling an exit process, see "Appendix A Functional Comparison of Hooks."

1.1 Prologue and epilogue function

Use prologue and epilogue function to apply settings and processes conforming to the job selection policy to all job scripts. For example, the function is used in the following ways.

- Set system-specific environment variables for use in job scripts.
- Create a work directory at a specific location where a job script can use it, and delete it when the job ends.

1.1.1 Prologue script and epilogue script

The exit processes of the prologue and epilogue function are shell scripts that are called before or after a job script. The shell script executed before it begins to execute the job script is called "Prologue script". The shell script executed after the job script execution ends is called "Epilogue script". The administrator creates prologue and epilogue scripts and incorporates them into job operations to use them. Different prologue and epilogue scripts can be incorporated for each resource unit.

Prologue and epilogue scripts are executed for each job that is a normal job and interactive job, and for each sub job in a step job or bulk job. The job state changes to RUNNING-P, when the prologue script is processed. The job state changes to RUNNING-E, when the epilogue script is processed.
The epilogue script is executed even if the job is terminated due to excessive use of resources or a job process error.

1.1.2 Execution environment

Prologue and epilogue scripts are executed below environment.

Execution node

Prologue and epilogue scripts are executed on one of the compute nodes allocated to the job. This compute node is the one executing the job script.

Note

If a job is forcibly terminated due to an error on the compute node executing the job script, the epilogue script is not executed.

Execution privileges

Setting execution privileges executed of prologue script and epilogue script by root privileges or job executing user privileges.

Execution directory

Prologue and epilogue scripts are executed on the current directory when the job script is submitted. Moving the directory within a prologue script does not affect the directory on which the job script runs.

Outputs

The standard output and standard error output of a prologue script and epilogue script are output to job execution results.

Environment variables

The job operation management function has set in advance some environment variables that can be referenced in prologue and epilogue scripts. For details on these environment variables, see "Appendix B Environment Variables."

None of the set environment variables in a prologue script is inherited by the subsequently executed job and epilogue scripts. None of the set environment variables in a job script is inherited by the subsequently executed epilogue script.

Elapsed time

Administrator can choose whether the execution time of a prologue script and an epilogue script is included in the execution elapsed time of a job. When including in execution elapsed time, the processing time of prologue script, the execution time of a user job script, and the execution time of epilogue are united, and it becomes the execution elapsed time of a job. Be careful about this time in operations like charging based on the length of the execution elapsed time.

Exit code of prologue script and epilogue script

Execution of a job is controllable by the exit code of prologue script and epilogue script. Problem occurs within a prologue script and an epilogue script, job can be controlled. The exit code of prologue script and epilogue script that can be set up, and the motion of a corresponding job are as follows.

Table 1.1 Exit code of prologue script and epilogue script and depend on job operation

Exit code of prologue script or epilogue script	Job operations
0	Job execution not controlled
1	Changing job state to ERROR
2	Re-executing the jobs
3	Changing job state to HOLD
4	Forcibly terminated the job
Other	Job execution not controlled

Note

- Even if the epilogue script returns exit code 2 or 3 when the job is not automatically re-executable, the job is deleted.
- If the job is instructed not to end normally with the end code of a prologue script or epilogue script, the job end code is 26. However, if the job is forcibly terminated due to a compute node error, the job end code is 20.
  Although the epilogue script is executed after the job script ends, the exit code of the epilogue script does not overwrite that of the job script.
- The exit codes of prologue and epilogue scripts control job operations only when the job ends normally. If the job ends abnormally, control by the job end code has priority over control by the exit codes of the prologue and epilogue scripts.
  For example, suppose that a job is forcibly terminated by the pjdel command. Then, the operation of the pjdel command has priority,

and the job is deleted. The same applies even for a given instruction to set the exit code of the prologue or epilogue script to 3 (Changing job state to HOLD). The operation of the prologue script and epilogue script takes effect after the job ends normally.

Referencing job execution results

An epilogue script can reference the end status of the job (job end code) and the exit code of the job script in the environment variables PJM_JOBEXIT and PJM_SHELLEXIT, respectively. This allows the administrator to create an epilogue process appropriate to the job execution results.

Deleting or holding a job

Prologue and epilogue scripts are executed even after the job is deleted by the pjdel command. However, if a job is deleted by the pjdel command with the --enforce option specified, the scripts operate as follows.

- The running prologue or epilogue script is aborted. If the running job is aborted, the epilogue process is not executed.

The same behavior occurs when the job is held with the pjhold command.

Interactive job

Prologue script and epilogue script is usually executed in an interactive job as well as a batch job. The standard output and standard error output destinations of prologue script and epilogue script become as follows.

- When the job begins, the execution result of prologue script is output to the terminal.
- The execution result of epilogue script is not output to the terminal.

See

You can set the prologue and epilogue function for individual resource units. For details on how to make the setting, see "2.1 Prologue and Epilogue Function" in "Chapter 2 Creating and Incorporating Hooks."

1.2 Job Manager Exit Function

The job manager exit function is used by administrators who want to determine whether to allow job submission and parameter changes based on the criteria that they themselves have decided. They can also use the function to execute a process based on the job information at job end. For example, the function is used in the following ways:

- Rejecting the submission of a job if the budget is insufficient for the submitted job and the parameter to be changed
- Subtracting from the budget after the end of a job
- Setting a value for job statistical information defined by the administrator

1.2.1 Exit Functions

The exit processes of the job manager exit function are C-language functions (exit functions). The administrator creates an exit function library and incorporates it into job operations to use it. A different exit function library can be incorporated for each resource unit or resource group.

The job manager exit function provides the following exit functions.

Table 1.2 Exit functions of the job manager function

Function name	Description	Application example
Initialization function pjmx_inithook()	Called when the exit function library of the job manager exit function is loaded. The exit function library is loaded when the job manager starts and when the library is incorporated by the pmpjmadm command. You can use this function to, for example, secure an area for use in the exit function library.	-

Function name	Description	Application example
Job registration function pjmx_quejob()	Executed when the job manager function is processing job acceptance. You can use it to judge whether to allow job acceptance.	If the budget is not sufficient to meet the amount of resources requested by the job, the function denies acceptance.
Job attribute change function pjmx_alterjob()	Executed when the resource unit, resource group, elapsed time limit, or other parameter of a job is changed. You can use it to judge whether to allow job parameter changes.	If the budget is not sufficient for the job parameter change, the function causes the change to fail.
Job pre-execution function pjmx_startjob()	Executed immediately before a QUEUED job transitions to the RUNNING-A state. You can use it to give an instruction on whether to allow job execution.	If the budget is not sufficient to meet the amount of resources requested by the job or the amount of allocated resources, the function suspends execution or deletes this job.
Job end function pjmx_endjob()	Executed when a job ends. This function is executed even if the job is forcibly terminated by the job deletion (pjdel) or hold (pjhold) command. You can use it to run unique processing for a job operation triggered by the job termination.	The function corrects budget management information based on the job end status and resource usage. Example: Budget returned or budget subtracted
Job deletion function pjmx_deljob()	Executed when the job manager function deletes information on an accepted job. You can use it to run unique processing for a job operation triggered by the job deletion.	The function corrects budget management information based on the job state. Example: Budget returned or budget subtracted
Finalize function pjmx_finihook()	Called when the exit function library of the job manager exit function is unloaded. The exit function library is unloaded when the job manager stops and when the incorporated exit function library is released by the pmpjmadm command. You can use this function to, for example, release the area used by the exit function library. Execution of this function is guaranteed only when the job manager stops normally.	-

An administrator who wants to add unique processes at the respective timing described above is requested to create the source programs of the exit functions and incorporate a library consisting of these functions in the job manager function. The job manager function executes these functions at the specific timing for the submission or execution of each job. Based on the return value of the exit function, an instruction on whether to continue processing of the target job is given.

You can register multiple exit function libraries for each resource unit or resource group, and you can prioritize them in their execution order.

For normal jobs and interactive jobs, the exit functions of the job manager exit function are executed for each job. The exit functions are executed as follows for step jobs and bulk jobs.

Table 1.3 Execution of exit functions on step jobs and bulk jobs

Job model	Target of executed exit function	
Step job	The function is executed on a sub job of the step job.	
Bulk job	The call target varies depending on the exit function.	
	pjmx_quejob()	The function is executed on each bulk job. It is not executed on any of the sub jobs.
	pjmx_alterjob()	The function is executed on a sub job of the bulk job.
	pjmx_startjob()	The function is executed on a sub job of the bulk job.
	pjmx_endjob()	The function is executed on a sub job of the bulk job.
	pjmx_deljob()	The function is executed on a sub job of the bulk job.

In the exit functions, the job information acquisition function can be used to reference detailed information on the jobs that act as execution triggers, and the job information setting function can be used to set job information.

See

- For notes on creating an exit function and details on how to create and incorporate a library in the job manager function, see "2.2 Job manager exit function and Job scheduler exit function" in "Chapter 2 Creating and Incorporating Hooks."
- For details on the call timing of each exit function, see "Table A.1 Execution timing of the hook" in "Appendix A Functional Comparison of Hooks."
- For details on the exit function APIs, job information acquisition function APIs, and job information setting function APIs, see "Appendix B Environment Variables."

1.2.2 Execution environment

The exit functions are executed as part of the job manager function daemon in the following environment.

Execution node

The exit functions are executed on the active compute cluster management node.

Execution privileges

Root privileges are used to execute the exit functions.

Execution directory

The exit functions are executed with /var/log/FJSVtcs/pjm as the current directory.

Outputs

If you want to output log or other information within an exit function, create a unique file and output the data to the file. Do not output it to the standard output and standard error output within the exit function.

Value returned by an exit function

The job manager function controls job acceptance and execution based on the return value of the exit function. For details, see "C.1 Job Manager Exit Functions" in "Appendix C References."

1.3 Job Scheduler Exit Function

The job scheduler exit function is used by administrators who want to determine whether to allow resource allocation to an accepted job and whether to reschedule the job. For example, the function is used to defer (reschedule) execution when the budget is insufficient to meet the amount of resources requested by a job.

1.3.1 Exit Functions

The exit processes of the job scheduler exit function are C-language functions (exit functions). The administrator creates an exit function library and incorporates it into job operations. A different exit function library can be incorporated for each resource unit or resource group.

The job scheduler exit function provides the following exit functions.

Table 1.4 Exit functions of the job scheduler function

Function name	Description	Application example
Initialization function pjsx_inithook()	Called when the exit function library of the job scheduler exit function is loaded. The exit function library is loaded when the job scheduler starts and when the library is incorporated by the pmpjmadm command. You can use this function to, for example,	-

Function name	Description	Application example
	secure an area for use in the exit function library.	
Job preselection function pjsx_prejobselect()	Executed before the job scheduler function sorts jobs according to job priority. You can use this function to limit the total number of jobs to schedule. This function is executed once per job scheduling process.	If the budget is insufficient to meet the amount or resources requested by the job, the function defers execution. A comparison cannot be made to determine budget excesses and deficiencies based on the job execution order because the order has not yet been determined when this function is called. The function checks the amount of resources requested by the job to confirm that it is not greater than the entire budget.
Job post-selection function pjsx_postjobselect()	Executed immediately before compute resources are allocated, after the job scheduler sorts jobs according to job priority. You can use this function to judge whether to allow job scheduling. If scheduling of a job is denied, the job is not scheduled at the relevant scheduling trigger. The job is scheduled again at the next scheduling trigger.	If the budget is insufficient to meet the amount or resources requested by the job, the function defers execution. A comparison can be made to determine budget excesses and deficiencies based on the job execution order because the order has already been determined when this function is called. For example, if preceding jobs have consumed the budget, the function defers execution of subsequent jobs because the budget is insufficient.
Resource post-selection function pjsx_postrscalloc()	Executed after the job scheduler function determines the compute resources to allocate to a job. You can use this function to judge whether to allow allocation of the relevant compute resources. If allocation to a job is denied, the resources are not allocated to the job at the relevant scheduling trigger. Instead, the resources selected for allocation to the job are available for allocation to another job. The job is scheduled again at the next scheduling trigger.	If the budget is insufficient to meet the amount of resources allocated to the job, the function defers execution. The amount of resources to allocate to the job has already been determined when this function is called. Some resources are allocated with the requested amount rounded up (e.g., rounding up the number of nodes to the nearest Tofu unit). So this function can make a comparison to ultimately determine budget excesses and deficiencies.
Finalize function pjsx_finihook()	Called when the exit function library of the job scheduler exit function is unloaded. The exit function library is unloaded when the job scheduler stops and when the incorporated exit function library is released by the pmpjmadm command. You can use this function to, for example, release the area used by the exit function library. Execution of this function is guaranteed only when the job scheduler function stops normally.	-

An administrator who wants to add unique processes at the respective timing described above is requested to create the source programs of the exit functions and incorporate a library consisting of these functions in the job scheduler function. The job scheduler function executes these functions at specific times during the scheduling of each job. Based on the return value of the exit function, an instruction on whether to continue processing of the target job is given.

You can register multiple exit function libraries for each resource unit or resource group, and you can prioritize them in their execution order.

For normal jobs and interactive jobs, the exit functions of the job scheduler exit function are executed for each job. The exit functions are executed as follows for step jobs and bulk jobs.

Table 1.5 Execution of exit functions on step jobs and bulk jobs

Job type	Target of executed exit function
Step job	The function is executed on a sub job of the step job.
Bulk job	The call target varies depending on the exit function.

In the exit functions, the job information acquisition function can be used to reference detailed information on the jobs that act as execution triggers, and the job information setting function can be used to set job information.

See

- For notes on creating an exit function and details on how to create and incorporate a library in the job manager function, see "2.2 Job manager exit function and Job scheduler exit function" in "Chapter 2 Creating and Incorporating Hooks."
- For details on the call timing of each exit function, see "Table A.1 Execution timing of the hook" in "Appendix A Functional Comparison of Hooks."
- For details on the exit function APIs, job information acquisition function APIs, and job information setting function APIs, see "Appendix C References."

1.3.2 Execution environment

The exit functions are executed as part of the job scheduler function daemon in the following environment.

Execution node

The exit functions are executed on the active compute cluster management node.

Execution privileges

Root privileges are used to execute the exit functions.

Execution directory

The exit functions are executed with /var/log/FJSVtcs/pjm as the current directory.

Outputs

If you want to output log or other information within an exit function, create a unique file and output the data to the file. Do not output it to the standard output and standard error output within the exit function.

Value returned by an exit function

The job scheduler function controls job acceptance and execution based on the return value of the exit function. For details, see "C.2 Job Scheduler Exit Functions" in "Appendix C References."

1.4 Job Resource Management Exit Function

Use the job resource manager exit function when you want to process a job in a way that shows an awareness of the timing for processing before allocating resources and the timing for processing after releasing resources. For example, the function is used in the following ways:

- Performing pre-processing or post-processing when using the job execution environment
- Setting a value for job statistical information defined by the administrator

1.4.1 Exit script

The exit processes for the job resource manager exit function are shell, perl, and other scripts (exit scripts). They are called before job resource allocation, after job resource release, and at other times. The exit scripts include prealloc, predel, and postfree scripts. The administrator creates these exit scripts and incorporates them into job operations to use them. Different exit scripts can be incorporated for each resource unit and resource group.

The exit scripts can be created with any script, such as shell or perl (one that begins with a line in the #!<interpreter> format).

- prealloc
  A script coded to write the processing to be performed before the allocation of job resources.

  This script is executed when:

  - Before allocation of job resources.

    During processing the prealloc script, the job state is RUNNING-A.

  Note

  The processing time of a prealloc script (or the sum of the processing times of multiple prealloc scripts, if set) is included in the period of the RUNNING-A state, during which resources are allocated. Therefore, the longer processing time of a prealloc script may cause the wait for resource allocation in an interactive job to time out thereby cancelling the job. In order to avoid this, note the following.

  - Instruct users to specify an appropriate value in the --sparam "wait-time" option of the pjsub command when submitting an interactive job.
  - For interactive jobs in cases where the prealloc script references the environment variable "PJM_ENVIRONMENT," do not allow the script to exceed the maximum wait time. Examples of preventive actions include suppressing prealloc script processing and shortening the prealloc script processing time.

- predel
  A script coded to write the processing to be performed when deleting or holding a job.
  This script is executed when the following operations and events occur.

  - the pjdel command is executed;
  - the pjhold command is executed;
  - a job resource (elapsed time or memory) limit is exceeded;
  - the compute node fails (including the state where its service has stopped); and
  - the job is deleted by the system for a reason other than the above.

  Note

  - During execution of a prologue script, when the pjdel or pjhold command without the --enforce option specified is executed, an error occurs. In this case, requests to delete or hold the job are not accepted, so the predel script is not executed.
  - When the job is deleted by excess of elapse time in the job state of RUNNING, epilogue script is exected after the predel script. But, when elapse time for epilogue script is included in execution elapse time (ContainElapse item in the pmpjm.conf file is yes), epilogue script is not executed.
  - If the job is forcibly terminated by execution of the pjsig command to send SIGKILL, for example, the predel script is not executed.

- postfree
  A script coded to write the processing to be performed after the release of job resources.
  If the predel script is executed, postfree script will run after the release of job resources subsequent to the execution of the predel script.
  This script is executed when:

  - After release of job resources.
    The postfree script is executed immediately before the job state changes from RUNNING-E to RUNOUT (Immediately before changing from RUNNING to RUNOUT when the prologue epilogue function is not set).

For normal jobs and interactive jobs, these exit scripts are executed for each job. For bulk jobs and step jobs, they are executed for each sub job.

Register one set of three exit scripts: prealloc, predel, and postfree. (However, it is not necessary to prepare all three scripts.) If necessary, you can register multiple sets of these scripts. As a result, multiple scripts of the same type (e.g., prealloc) may be executed for a job.

Note

- There are cases where multiple exit scripts of the same type are executed. In addition to a situation where multiple sets of exit scripts are registered, other cases include times when the pjdel command is executed while simultaneously job resources are used excessively.
- If multiple exit scripts of the same type are registered, they are executed in the specified order of priority. However, if an exit script ends in an error, subsequent exit scripts are not executed.

You can set the following job statistical information In the exit scripts of the job resource manager:

- REASON item in job statistical information (using the pmsetjobinfo command)
- Value of an administrator-defined item in job statistical information (using the pmsetstats command)

See

For details on how to set the job statistical information by using the commands, see "2.3.1 Creating exit scripts" in "Chapter 2 Creating and Incorporating Hooks."

1.4.2 Execution environment

Exit scripts are executed below environment.

Execution node

The scripts are executed on every compute node allocated to the job.

Note

If a job is forcibly terminated due to a failure on a compute node, no exit script is executed on the compute node.

Execution privileges

The scripts are executed with root privileges. The permission of the exit script is automatically set, when the exit script is registered. Therefore, the job operation manager need not set it.

Execution directory

The scripts are executed with the root home directory as the current directory.

Outputs

The results are written to the standard output and standard error output within the exit script but are not output. If you want to output the results, create an original file and output the results to the file.

Environment variables

The job operation management function has set in advance some environment variables that can be referenced in exit scripts. For details on these environment variables, see "Appendix B Environment Variables."

Elapsed time

The exit script execution time is not included in the elapsed time for execution of the job.

Exit code of exit script

Execution of a job is controllable by the exit code of exit script. Problem occurs within an exit script, job can be controlled.

The exit code of exit script that can be set up, and the motion of a corresponding job are as follows.

Table 1.6 Exit code of exit script and depend on job operation

Exit code of exit script	Job operations
0	Job execution not controlled

Exit code of exit script	Job operations
1	Changing job state to ERROR
2	Re-executing the jobs
3	Changing job state to HOLD
4	Forcibly terminated the job
Other	Job execution not controlled

Note

- If the compute node running the job goes down, the exit script exit code cannot control the job. he job transitions to the QUEUED state. If auto rerun of the job is enabled, the job transitions to the QUEUED state and then reruns. If automatic rerun of the job is disabled, the job ends.
- Even if the exit script executed after the job state changes to RUNNING returns exit code 2 or 3, when the job is not automatically re-executable, the job is deleted.
- If the job is instructed not to end normally with the exit code of the exit script and if the set exit code of the exit script is a value other than 0, the job end code is 27 (error in job resource management exit processing). The end code is not to 20 (node failure) or 11 (job execution timeout due to elapsed time limit violation). This happens even if, for example, the job ends because a node failed or the elapsed time exceeded the limit.
  Although the postfree script is executed after the job script ends, the exit code of the postfree script does not overwrite the exit code of the job script.
- If multiple exit scripts of the same type are registered, the result from the exit script executed last on each compute node is the result for that compute node. The exit code with the highest priority among all the results of the compute nodes is the result for the exit script for the job.

  ```
  Exit code priority: 1 > 4 > 3 > 2 > 0
  ```

  For example, if the exit codes of exit scripts on three nodes are 0, 1, and 4, exit code 1 has the highest priority and is adopted as the exit code of the exit script for the job.
- The priority of the exit script operations is higher than for ordinary job errors. For example, when the pjdel command is executed, if the specified end code of the exit script is 3, then the job is not deleted but enters the HOLD state.

Referencing job execution results

A postfree script can reference the end status of the job (job end code) and the exit code of the job script in the environment variables PJM_JOBEXIT and PJM_SHELLEXIT, respectively. This allows the administrator to create an exit process appropriate to the job execution results.

Operation of a command that interrupts a job

When the exit script prealloc or postfree is executed, control of the jobs on the node will not start until the processing within the script ends. Therefore, if a command (pjdel, pjhold, pjsig, etc.) that interrupts the job is executed during execution of prealloc or postfree, the command will start job interruption processing after the exit script processing ends.

See

You can configure the resource management exit function for each resource unit and resource group. For details on how to configure it, see "2.3 Job resource management exit function" in "Chapter 2 Creating and Incorporating Hooks."

1.4.3 Embedded Exit Script for Job Operation Software

Even if the administrator does not register an exit script, information indicating that the exit script worked (Item prealloctrigger, etc.) may be output in the job statistical information.

This is because, in addition to the exit scripts that the administrator registers, there are exit scripts (the embedded exit script) that are already embedded into the job operation software.

The embedded exit script runs before the exit script that the administrator registers. If the embedded exit script fails, the exit script that the administrator registers is not executed.

Chapter 2 Creating and Incorporating Hooks

This chapter describes creating and incorporating hooks of the job operation management function.

Note

Hooks are functions with settings that do not change frequently during operation. Before the start of operation, confirm that hooks operate as expected.

2.1 Prologue and Epilogue Function

2.1.1 Creating a script

Create prologue and epilogue scripts with shell scripts. In settings, you can select a shell for executing the scripts. Create scripts that return the exit codes shown in "1.1.2 Execution environment" in "Chapter 1 What are the Hooks of the Job Operation Management Function?" according to the purpose of each script. There are no rules on the file names of the scripts. Place the created scripts in any directory on the compute cluster management node.

Note

- You do not need to always create both a prologue script and epilogue script.
- You do not need to give execution permission to prologue and epilogue scripts.
- Currently, prologue and epilogue scripts support only shell scripts.
- Prologue and epilogue processes are executed as part of a job submitted by an end user. Therefore, create the processes by taking into account the following information.
  - The administrator can give the privilege to execute prologue and epilogue scripts to the job submission user or the root user in settings (see "2.1.2 Modifying a configuration file"). Note the following about using root user privileges to execute the scripts.
    - Set the permission for these scripts in such a way that prevents users other than the root user from editing the scripts.
    - The processing details and quality of these scripts, such as processing load and time, disk usage, and post-processing, may affect job operations and system security. Duly check the operation of the scripts before operating them.
  - Prologue and epilogue processes are added to job statistical information. However, they are included in elapsed job execution time only when the ContainElapse item is "yes" (see "2.1.2 Modifying a configuration file").
  - The resource limit value for a job also applies to prologue and epilogue processes.
- The mpiexec command, pjrsh command, and pjexe command [PG] cannot be executed from the prologue script and epilogue script. Use the job resource management exit function when executing pre-processing and post-processing of a job script on a node other than the node where the job script is executed.

2.1.2 Modifying a configuration file

To use the prologue and epilogue function, settings in the pmpjm.conf file (system management node: /etc/opt/FJSVtcs/Rscunit.d/*resource unit name*/pmpjm.conf) need to be configured for each resource unit.

Write the settings for the prologue and epilogue function in the ResourceUnit section.

```
[System management node]
# cat /etc/opt/FJSVtcs/Rscunit.d/ResourceUnit/pmpjm.conf
ResourceUnit {
    ResourceUnitName = ResourceUnit
   ...
    PrologueEpilogue {                  <- Prologue and epilogue function settings
```

```
        ShellName = /bin/sh             <- Execution shell
        ExecUser = ROOT                 <- Execution privilege
        PrologueName = /work/prologue   <- Path to prologue script
        EpilogueName = /work/epilogue   <- Path to epilogue script
        PrologueTime = 300              <- Estimate of prologue script execution time
        EpilogueTime = 180              <- Estimate of epilogue script execution time
        ContainElapse = no              <- Whether to include elapsed time of
                                           prologue and epilogue function into job
    }
    ...
}
```

Table 2.1 Prologue and epilogue function setting items (PrologueEpilogue subsection)

Item name	Definition contents	Specifiable value	Default value
ShellName	Execution shell setting for prologue and epilogue scripts	Character string that can be used as a file name	/bin/sh
ExecUser	Execution user setting for prologue and epilogue scripts	ROOT : Execution by a root user JOBUSER : Execution by a job execution user	ROOT
PrologueName	Prologue script path on the compute cluster management node setting	Absolute path	No setting
EpilogueName	Epilogue script path on the compute cluster management node setting	Absolute path	No setting
PrologueTime	Estimated execution time (seconds) setting for prologue scripts	0 - 2147483648	0
EpilogueTime	Estimated execution time (seconds) setting for epilogue scripts	0 - 2147483648	0
ContainElapse	Elapsed time for prologue and epilogue scripts setting	yes : including in execution elapsed time no: excluding in execution elapsed time	no

Note

- The PrologueTime and EpilogueTime items are not limit values. They are used to calculate the scheduled end time of a job in job scheduling.
- Suppose that ContainElapse is set to yes and a range from a minimum value to a maximum value is specified for the elapsed execution time limit of a job. In this case, the job may be terminated before the elapsed execution time reaches the minimum value.

2.1.3 Incorporating a script

Use the pmpjmadm command to get the system to reflect the contents of the set pmpjm.conf file. Execute the command on the system management node.

```
[System management node]
# pmpjmadm -c cluster --set --rscunit resourceunit
```

Note

If you have not modified the pmpjm.conf file but have modified the prolog or epilog script, run the pmpjmadm command. The execution of the pmpjmadm command also incorporates the deployed prolog and epilog scripts into the job operation.

2.1.4 Releasing an incorporated script

To release incorporated prologue and epilogue scripts, delete the relevant PrologueEpilogue section from the pmpjm.conf file, and use the pmpjmadm command to reflect the settings.

2.2 Job manager exit function and Job scheduler exit function

This section describes how to create exit function libraries for the job manager exit function and job scheduler exit function, and also how to incorporate the libraries into operation.

2.2.1 Creating the source files of exit functions

The source files of exit functions of the job manager exit function and job scheduler exit function are installed on the compute cluster management node. Copy them to an appropriate work directory, and modify them there.

Table 2.2 Source files for a job manager exit function library

Source file	Description
/usr/src/FJSVtcs/pjm/hook/pjmx/Makefile	Sample Makefile for job manager exit function library
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_inithook.c	Source file of initialization function pjmx_inithook() of job manager exit function
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_quejob.c	Source file of job registration function pjmx_quejob()
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_startjob.c	Source file of job pre-execution function pjmx_startjob()
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_alterjob.c	Source file of job attribute change function pjmx_alterjob()
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_endjob.c	Source file of job end function pjmx_endjob()
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_deljob.c	Source file of job deletion function pjmx_deljob()
/usr/src/FJSVtcs/pjm/hook/pjmx/pjmx_finihook.c	Source file of exit process function pjmx_finihook() of job manager exit function

Table 2.3 Source files for a job scheduler exit function library

Source file	Description
/usr/src/FJSVtcs/pjm/hook/pjsx/Makefile	Sample Makefile for job scheduler exit function library
/usr/src/FJSVtcs/pjm/hook/pjsx/pjsx_inithook.c	Source file of initialization function pjsx_inithook() of job scheduler exit function
/usr/src/FJSVtcs/pjm/hook/pjsx/pjsx_prejobselect.c	Source file of job preselection exit function pjsx_prejobselect()
/usr/src/FJSVtcs/pjm/hook/pjsx/pjsx_postjobselect.c	Source file of job post-selection exit function pjsx_postjobselect()
/usr/src/FJSVtcs/pjm/hook/pjsx/pjsx_postrscalloc.c	Source file of resource post-selection exit function pjsx_postrscalloc()
/usr/src/FJSVtcs/pjm/hook/pjsx/pjsx_finihook.c	Source file of exit process function pjsx_finihook() of job scheduler exit function

The exit functions in the installed source files only end normally without doing anything. Create processes for these functions according to your purpose.
For example, the exit function pjmx_quejob() performs the following processing to manage the budget at job submission.

1. The function acquires information on the target job and the amount of resources requested by the job by using the job information acquisition function together with the passed argument (QueJobInfo_t) information.
2. The function calls the budget management function prepared by the administrator, and checks whether the amount of resources requested by the job is within the budget.
3. If the amount exceeds the budget, the function returns an error to reject acceptance or execution of the job.

- You can prepare multiple exit function libraries and incorporate them into operation.
- You do not have to implement all the exit functions for an exit function library.

- If the compute cluster management node is redundantly configured, modify the source files on both the active and standby compute cluster management nodes.
  Do not edit files other than the source files shown in "Table 2.2 Source files for a job manager exit function library" and "Table 2.3 Source files for a job scheduler exit function library."
- The source files are configured to permit access only by root users. When modifying a source file, be careful not to grant access privileges to a non-root user.
- Processing within an exit function is executed as part of the job manager function daemon. Therefore, the performance and quality of the exit function have a significant effect on job operations. Be careful with the following when modifying a source file. Adequately check the modifications before using the file.
  - Do not execute processes that have a significant processing cost. The following processes are examples:
    - Exclusive control of a process, thread, or file
    - Process of searching in a large file
  - Use thread-safe functions in the exit functions.
  - Do not use the system calls fork() and exec() in the exit functions.
  - After a file opens in an exit function, close the opened file within the same exit function.
  - After a memory area is acquired in an exit function, release the acquired area within the same exit function.
  - Do not change operations on signals.
  - Do not output to the standard output. Otherwise, the job manager function log may not be output normally. If you want to output a log or other information in an exit function, create your own file and output the information to the file.
  - When preparing multiple exit function libraries, consider whether or not to make processes in the libraries exclusive to one another before creating them. For example, if the libraries have a process that outputs data to a file, considerations must be made to ensure that their output destinations do not overlap.
  - Be careful that the function does not include any process that would cause the abnormal end of the daemon, hang-up, or a security hole.

[Job statistical information settings]

With the job manager exit function, you can set values for administrator-defined items in job statistical information in the papjmstats.conf file by using the job information setting function pjmx_setinfo_stats_item_*type*().

To set a value for an administrator-defined item in job statistical information from the job manager exit function, use the job information setting function pjmx_setinfo_stats_item_*type*(). If you want to reference a set value, use the job information acquisition function pjmx_getinfo_stats_item_*type*(). The "*type*" part in the function names varies depending on the type of value to set. For details on the functions, see "Appendix C References."

- Values for administrator-defined items in job statistical information must be set with job manager exit functions or exit scripts of the job resource manager exit function described below ("2.3.1 Creating exit scripts").
  For details on administrator-defined items in job statistical information, see "Settings for job statistical information in a cluster (papjmstats.conf)" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management."
- Node statistical information cannot be acquired or set if the job information acquisition function pjmx_getinfo_stats_item_*type*() or pjmx_getinfo_stats_item_array_*type*(), or the job information setting function pjmx_setinfo_stats_item_*type*() is called from the job manager exit function pjmx_quejob() or pjmx_startjob().

2.2.2 Modifying a configuration file

[Incorporating a new exit function library]

To use the job manager exit function and the job scheduler exit function, the pmpjm.con file (system management node: /etc/opt/FJSVtcs/Rscunit.d/*resource unit name*/pmpjm.conf) must be configured for each resource unit. There is no difference in the setting method between the job manager exit function and the job scheduler exit function.

Write settings for a resource unit and a resource group in the ExitFunc subsection in the ResourceUnit and ResourceGroup sections, respectively, in the pmpjm.conf file. In the ExitFunc subsection, register the exit function libraries created as respective exit functions.

```
[System management node]
# cat /etc/opt/FJSVtcs/Rscunit.d/resourceunit/pmpjm.conf
ResourceUnit {
    ResourceUnitName = resourceunit
   ...
    ExitFunc {                      <- Settings for resource unit
        ExitFuncLib = libpjmx.so    <- Specifies exit function library
        ExitFuncPri = 127           <- Execution priority of exit function
        ExitFuncType = pjm          <- Exit function type
    }
    ...
    ResourceGroup {
        ResourceGroupName = resourcegroup
        ...
        ExitFunc {                  <- Settings for resource group
            ExitFuncLib = libpjsx.so
            ExitFuncPri = 127
            ExitFuncType = pjs
        }
        ...
    }
}
```

Table 2.4 Definition items (ExitFunc subsection) of the job manager exit function and job scheduler exit function

Item name	Definition contents	Specifiable value	Default value
ExitFuncLib	File name of exit function library * Not the name of the installation path to the exit function library	Character string of up to 255 bytes	Not omissible
ExitFuncPri	Priority of exit function library If multiple exit function libraries are registered, they are executed in descending order of priority.	Value from 0 to 255 The larger the number is, the higher the priority is.	127
ExitFuncType	Type of exit function library	pjm: Job manager exit function pjs: Job scheduler exit function	Not omissible

Note

The job preselection exit function pjsx_prejobselect() is called only if it is included in the exit function library registered in the ResourceUnit section. The job preselection exit function pjsx_prejobselect() in the exit function library registered in the ResourceGroup section is not called.

To prepare multiple exit function libraries, write an ExitFunc section for each of them, and set their priorities.

```
ResourceUnit {
    ResourceUnitName = unit1
   ...
    ExitFunc {
        ExitFuncLib = libpjmx.so
        ExitFuncPri = 127
```

```
        ExitFuncType = pjm
    }
    ...
    ResourceGroup {
        ResourceGroupName = groupA
        ...
        ExitFunc {
            ExitFuncLib = libpjmxA.so
            ExitFuncPri = 64
            ExitFuncType = pjm
        }
        ExitFunc {
            ExitFuncLib = libpjmxB.so
            ExitFuncPri = 63
            ExitFuncType = pjm
        }
        ...
    }
}
```

Information

- If multiple ExitFunc subsections are written, the execution order of the exit function libraries is determined for each exit function library type (pjm or pjs) according to the priority specified in the ExitFuncPri item.
  In the above example, the job manager exit functions for a job executed in the resource group groupA are executed in order from libpjmx.so for the resource unit to libpjmxA.so to libpjmxB.so for the resource group.
- Initialization functions are executed in descending order of priority at the exit function library load time. Exit processing functions are executed in ascending order of priority at the exit function library unload time.
- If multiple exit function libraries are incorporated, they are executed in order according to priority regardless of whether they are in a resource unit or resource group. However, if the same priority is set, the order is determined by the following rules.
  - The set exit function library for the resource unit has priority over the set library for the resource group.
  - The exit function library that is written earlier in the configuration file has priority.

[Replacing an incorporated library]

If you want to replace an incorporated exit function library, release the library before installing the modified exit function library ("2.2.3 Installing an exit function library"). Replacing or deleting the exit function library without releasing it may cause the job management function service to stop.

To replace an exit function library, perform the following operations.

1. Comment out the relevant ExitFunc section in the pmpjm.conf file by adding the character # to the beginning of the line.
2. Reflect the settings with the pmpjmadm command (see "2.2.4 Incorporating a library"). This releases the incorporated exit function library.
   You can check whether or not the exit function library has been incorporated by using a command such as lsof. The check shown in the following step uses the lsof command.

```
[Active compute cluster management node]

When the exit function library is incorporated, the display is like the following:
# lsof /var/opt/FJSVtcs/shared_disk/pjm/hook/libpjmx.so
COMMAND   PID USER  FD   TYPE DEVICE SIZE/OFF    NODE NAME
pjmd    12090 root mem    REG  253,0    17961 2062572 /var/opt/FJSVtcs/shared_disk/pjm/hook/
libpjmx.so

When no exit function library is incorporated, nothing is displayed.
```

3. After confirming that the incorporated exit function library has been released, restore the commented-out part of the pmpjm.conf file. Then, proceed to "2.2.3 Installing an exit function library."

Information

Aside from than the above, another method is to create and modify an exit function library with a name different from the name of the incorporated library, change the ExitFuncLib value to the name of the modified library, and incorporate the new library by using the pmpjmadm command. In this method, the old exit function library is released at the same time that the new one is incorporated.

2.2.3 Installing an exit function library

After modifying a source file, create an exit function library by performing the following procedure. If the compute cluster management node is redundantly configured, install the exit function library on the active compute cluster management node.

Note

- If an exit function library is installed when another exit function library is already incorporated, the existing one is overwritten, which may have an impact, such as unstable exit process operation. Therefore, when changing an incorporated exit function library, first release it as described in "[Replacing an incorporated library]" in "2.2.2 Modifying a configuration file."
- The procedure described below installs a job manager exit function library. To install a job scheduler exit function library, replace "pjmx" with "pjsx" when reading the text.

1. Modifying a Makefile

   If the library name has changed or the file name of the corresponding sample source has changed, modify PJMXLIBOBJS and PJMXLIBNAME as required in the sample Makefile.
   In PJMXLIBOBJS, specify the object files to include in the exit function library. In PJMXLIBNAME, specify the name of the exit function library. The library name "lib$(PJMXLIBNAME).so." is given to the exit function library.
   Place the modified Makefile in the same directory as the source file.

```
PJMXLIBOBJS = \
        pjmx_quejob.o \
        pjmx_startjob.o \
        pjmx_endjob.o \
        pjmx_deljob.o \
        pjmx_alterjob.o \
        pjmx_inithook.o \
        pjmx_finihook.o
PJMXLIBNAME = pjmx
```

   ### Note

   - As shown in the above example, you do not need to specify all the object files (*.o) in PJMXLIBOBJS. Instead, specify only the object files that contain created exit functions.
   - ROOTDIR and PJMX_LIBPATH in the Makefile show the installation path to the exit function library. Do not change their values.
   - Multiple exit function libraries can be created according to the purpose. Create a library by copying and modifying the set of sample source files for each library. Do not give a duplicate name to the library.
   - If not using the sample Makefile, specify the following options at the compile time.

```
-shared -fPIC -pthread
```

2. Installing an exit function library

   Create an exit function library as shown below on the active compute cluster management node.
   Perform this work with root privileges.

```
[Active compute cluster management node]
# make install
```

   When successful, the exit function library is installed as shown below.

```
[When exit function library libpjmx.so is installed]
Path: /var/opt/FJSVtcs/shared_disk/pjm/hook/libpjmx.so
Owner: root
Group: root
Permission: 0444
```

2.2.4 Incorporating a library

To incorporate an installed exit function library into job operations, use the pmpjmadm command to get the system to reflect the contents of the pmpjm.conf file.

```
[System Management node]
# pmpjmadm -c cluster --set --rscunit resourceunit
```

Note

- The contents of the pmpjm.conf file are reflected immediately after the pmpjmadm command with the --set option is executed, so submitted jobs too will reflect the settings. Therefore, pay attention to the processing details of the exit function library and the timing for incorporating it.
  For example, the function pjmx_quejob() is not executed for jobs that have already been submitted when the settings are reflected. If the exit function pjmx_deljob() performs processing based on the execution results of the function pjmx_quejob(), required information may be missing.
- Executing the pmpjmadm command incorporates or releases only the exit function library that was changed in the configuration file. Accordingly, only the target exit function library is loaded or unloaded.

2.2.5 Releasing an incorporated library

To delete an exit function library that is no longer used, delete the relevant ExitFunc section as described in "[Replacing an incorporated library]" in "2.2.2 Modifying a configuration file," and use the pmpjmadm command to reflect the settings. This releases the incorporated exit function library.

After that, using a command such as lsof, confirm that the incorporated exit function library has been released. Then, delete the library.

2.3 Job resource management exit function

2.3.1 Creating exit scripts

Create exit scripts for the job resource manager exit function in any script language, such as shell or perl. Create them such that they return the exit codes shown in "1.4.2 Execution environment" in "Chapter 1 What are the Hooks of the Job Operation Management Function?" according to their respective purposes.

The file name of an exit script must be prealloc, predel, or postfree. Place created scripts in any directory on the system management node. The job resource manager exit function executes files that have these names and are located in the directory where the scripts were placed. If you want to execute different exit scripts for each resource unit or resource group, prepare different directories and place the scripts there.

Place created exit scripts as a set of prealloc, predel, and postfree in any directory on the system management node. However, you do not need to always create all three scripts.
If you want to prepare multiple sets of exit scripts, place each set in a different directory. This allows you to execute a different set of exit

scripts for each resource unit or resource group and to execute multiple sets of exit scripts in a resource unit or resource group. For a specific example, see "2.3.2.1 Job resource manager function settings."

Note

- Exit scripts are executed with root user privileges. Note the following:
  - For exit scripts, set the owner to a user with job operation administrator privileges or higher and the access permission to 0660 to prevent users without the privileges from changing the scripts.
  - The processing details and quality of exit scripts, such as processing load and time, disk usage, and post-processing, may affect job operations and system security. Duly check the operation of the scripts before operating them.
- One of the setting items for the job resource manager is RespWaitTime, which is the maximum wait time for a response in inter-node communication. (For details, see "Settings for job resource management in a resource unit (pmrsc.conf file)" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management.")
  Note the following about RespWaitTime and the processing time of exit scripts when creating exit scripts.
  - If the processing time of any exit script exceeds the set value of RespWaitTime, the compute node executing the exit script is isolated from operation.

[Job statistical information settings]

You can set some job statistical information on the target jobs with exit scripts of the job resource manager. Execute the following command from an exit script as necessary.

- Value of an administrator-defined item in job statistical information (using the pmsetstats command)

  With an exit script, the pmsetstats command can set a value for an administrator-defined item in job statistical information in the papjmstats.conf file.

```
pmsetstats --record {JI|JN} --item item_name --type value_type --value value
```

Note

Values for administrator-defined items in job statistical information must be set with exit scripts of the job resource manager exit function or job manager exit functions described above ("2.2.1 Creating the source files of exit functions"). For details on administrator-defined items in job statistical information, see "Settings for job statistical information in a cluster (papjmstats.conf)" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management."

Table 2.5 Job statistical information settings with the pmsetstats command

Option	Description
--record {JI|JN}	Record type of the job statistical information to set JI: Job statistical information (excluding the node statistical information below) JN: Node statistical information (information per node in job statistical information)
--item *item_name*	Item name for a set value Name specified in ItemName in the papjmstats.conf file
--type *value_type*	Value type You can specify char, int8, uint8, int16, uint16, int32, uint32, int64, uint64, double, float, timespec, time, size, or string. For details on each type, see "Table 2.6 Types that can be specified in the --type option of the pmsetstats command." The set value for this item must match the type specified in DataType in the papjmstats.conf file.
--value *value*	Value

Table 2.6 Types that can be specified in the --type option of the pmsetstats command

Type	Description
char	Character Specifying 2 or more characters (e.g., --value "ABC") causes an error.
int8	Signed integer (8 bits long) Specifying a value exceeding the range of the type causes an error.
uint8	Unsigned integer (8 bits long) Specifying a value exceeding the range of the type causes an error. Specifying a negative value also causes an error.
int16	Signed integer (16 bits long) Specifying a value exceeding the range of the type causes an error.
uint16	Unsigned integer (16 bits long) Specifying a value exceeding the range of the type causes an error. Specifying a negative value also causes an error.
int32	Signed integer (32 bits long) Specifying a value exceeding the range of the type causes an error.
uint32	Unsigned integer (32 bits long) Specifying a value exceeding the range of the type causes an error. Specifying a negative value also causes an error.
int64	Signed integer (64 bits long) Specifying a value exceeding the range of the type causes an error.
uint64	Unsigned integer (64 bits long) Specifying a value exceeding the range of the type causes an error. Specifying a negative value also causes an error.
double	Double-precision floating point
float	Single-precision floating point If the specified value is of the double type, it is rounded to the nearest value of the float type.
timespec	Value represented by the timespec structure Specify the values to set in the members tv_sec (seconds) and tv_nsec (nanoseconds) of the timespec structure, separating them by a period (.). Example: Setting of 10.5 seconds --type timespec --value 10.500000000
time	Value represented by the time_t type
size	Value represented by the size_t type
string	Character string

- REASON item in job statistical information (using the pmsetjobinfo command)

  Use this item to notify end users of a reason, such as why a job ended in an error in an exit script, by using the REASON item of statistical information.

```
pmsetjobinfo --type reason --value "msg"
```

  The specified character string *msg* is output to the REASON item in job statistical information when the exit code of an exit script indicates an error (1 to 4).

Note

- The pmsetstats command ignores the following settings. Since no messages are output at this time, confirm that the settings for job statistical information work as expected before the start of operation with them.
  - An item defined by the job operation management function is specified in the --item option.

- The contents of specified arguments are not consistent with the papjmstats.conf file.

  For example, an item that is not defined in the papjmstats.conf file may be specified in the --item option, or the item specified in the --item option and the type specified in the --type option may not match the contents of a definition.

- You can set job statistical information with the pmsetjobinfo and pmsetstats commands only from exit scripts of the job resource manager.

[Suppression of disturbance to execution performance of job]

When an OS memory recovery process occurs during the execution of a job, it becomes a disturbance to the execution performance of the job. To prevent this, the administrator can use the exit script prealloc to free the OS memory cache just before the job runs. This increases the amount of free memory in the compute node, and it is expected to reduce the occurrence of memory recovery processes that cause disturbances during job execution.

The following example shows how to free the memory cache.

```
echo 3 > /proc/sys/vm/drop_caches
```

See

Check the OS specifications for details on how to free the memory cache.

When running a UDI-specified job in Docker or KVM mode, the process of extracting the image file and freeing the memory cache with the exit script prealloc may run concurrently. Because the process of extracting an image file involves obtaining a memory cache, it conflicts with the freeing memory cache in the exit script prealloc, which can result in a memory cache release process that takes more than 10 minutes. If the exit script timeout value (ExitFuncTimer in the pmrsc.conf file) is short for this processing time, the job ends with an error. The administrator should consider the above before deciding whether to incorporate the process of freeing memory cache in the exit script prealloc. Also, change the exit script timeout value if necessary.

2.3.2 Modifying a configuration file

2.3.2.1 Job resource manager function settings

To use the job resource manager exit function, settings in the pmrsc.conf file (system management node: /etc/opt/FJSVtcs/Rscunit.d/*resource unit name*/pmrsc.conf) need to be configured for each resource unit.

Write the settings for a resource unit and resource group in the ExitFunc subsection in the ResourceUnit and ResourceGroup sections, respectively, in the pmrsc.conf file. In the ExitFunc subsection, register an exit script. Each ExitFunc subsection corresponds to 1 exit script, and up to 100 ExitFunc subsections can be written in a resource unit. If the number of ExitFunc subsections exceeds 100, an error occurs when the settings are reflected by the pmrscadm command.

The following examples show preparation of exit scripts for each of the resource unit runit1 and the resource groups rgroupA and rgroupB.

- Exit scripts
  Place each set of exit scripts in a different directory on the system management node.

  - Exit scripts for the resource unit runit1

    ```
    /work/hook/runit1/prealloc
                     predel
                     postfree
    ```

  - Exit scripts for the resource group rgroupA

    ```
    /work/hook/rgroupA/prealloc
                      predel
                      postfree
    ```

- Exit scripts for the resource group rgroupB

```
/work/hook/rgroupB/prealloc
                   predel
                   postfree
```

- Example of settings in the pmrsc.conf file

```
[System management node]
# cat /etc/opt/FJSVtcs/Rscunit.d/runit1/pmrsc.conf
Cluster {
    ClusterName = clstname
    ResourceUnit {
        ResourceUnitName = runit1
        ...
        ExitFunc {                          <- Exit script settings for resource unit
            ExitFuncTimer = 10              <- Timeout value
            ExitFuncScriptDir = /work/hook/runit1  <- Directory where exit script is located
            ExitFuncPri = 100               <- Priority of exit script
        }
        ResourceGroup {
            ResourceGroupName = rgroupA
            ExitFunc {                      <- Exit script settings for resource group rgroupA
                ExitFuncTimer = 30
                ExitFuncScriptDir = /work/hook/rgroupA
                ExitFuncPri = 120
            }
        }
        ResourceGroup {
            ResourceGroupName = rgroupB
            ExitFunc {                      <- Exit script settings for resource group rgroupB
                ExitFuncTimer = 30
                ExitFuncScriptDir = /work/hook/rgroupB
                ExitFuncPri = 130
            }
        }
    }
}
```

Table 2.7 Definition items of the job resource manager exit function

Subsection name	Item name	Definition contents	Specifiable value	Default value
ExitFunc	ExitFuncTimer	Timeout value for exit function execution time (seconds)	1 - 1800	10
	ExitFuncScriptDir	Directory where the exit script is located If none of the exit scripts prealloc, postfree, and predel are located immediately under the directory, an error occurs when the settings are reflected.	Path name that has up to 2,048 characters and includes directory name and exit script name	Not omissible
	ExitFuncPri	Priority of exit script If multiple exit scripts are registered, they are executed in descending order of priority.	Value from 0 to 255 The larger the value is, the higher the priority is.	127

Note

- If multiple ExitFunc subsections are written, the execution order of the exit scripts is determined for each exit script type (prealloc, postfree, or predel) according to the priority specified in the ExitFuncPri item.
- If multiple ExitFunc sections with the same priority are written, they are executed in the order that they are written in the configuration file. If the ExitFunc sections with the same priority are set in a resource unit and a resource group, the set exit function library in the resource unit is called earlier.
- One of the setting items for the job resource manager is RespWaitTime, which is the maximum wait time for a response in inter-node communication. (For details, see "Settings for job resource management in a resource unit (pmrsc.conf file)" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management.")
  Note the following about the RespWaitTime item when configuring exit scripts.
  - We recommend adjusting the values of the ExitFuncTimer and RespWaitTime items so that they meet the following formula. This is intended to prevent inter-node communication of the job resource manager from timing out earlier than an exit script running on a compute node when the job resource manager is waiting for the exit script to end.

    ```
    RespWaitTime > A x B + C

    A = Number of jobs that can be simultaneously executed on 1 node
    B = Sum of following values of ExitFunc sections
        ExitFuncTimer x (total number of prealloc and postfree scripts)
    C = Greatest ExitFuncTimer value in all ExitFunc sections where predel script is set
    ```

  - If an exit script does not end even after reaching the set value of ExitFuncTimer, the SIGXCPU signal is sent. Therefore, capture and process for the SIGXCPU signal in the exit script as necessary.

  If the above value exceeds the value of the RespWaitTime item, inter-node communication of the resource manager times out, and the compute node executing the exit script may be isolated from operation.
- If the compute cluster management node or compute cluster sub management node is restarted while the exit script prealloc or postfree is running, it may take as much time as the above value of A x B + C at the maximum until each node is operational.
- If the following events occur while an exit script is running, exit scripts with a lower priority are not executed.
  - The execution time exceeds the timeout value.
  - The exit code of the exit script is a value from 1 to 4.

2.3.2.2 Job manager function settings

Factors that determine the execution start time during scheduling of a job include the elapsed execution time of the job and the buffer time for the job execution interval. The execution time of exit scripts is one of the factors that affect the execution start time of a job. However, it is not included in the elapsed execution time of a job but needs to be considered as a buffer time.
The administrator is requested to set the sum of the maximum processing times of the exit scripts that may be executed for a single job as the buffer time DecidedGap in the pmpjm.conf configuration file of the job operation management function.

```
[System management node]
# cat /etc/opt/FJSVtcs/Rscunit.d/resourceunit/pmpjm.conf
ResourceUnit {
    ResourceUnitName = resourceunit
    LogLevel = 1
    Backfill = yes
    DecidedGap = 00:01:00   <- Sets buffer time for job execution interval in "hours:minutes:seconds"
    Grace = 00:02:00
    ...
}
```

See

For details on the buffer time, see "Job scheduling parameters" in "Chapter 2 Details of the Job Operation Management Function" in "Job Operation Software Administrator's Guide for Job Management."

2.3.3 Incorporating a script

Use the pmrscadm and pmpjmadm commands to reflect the contents of the set pmrsc.conf and pmpjm.conf files, respectively. Execute this command on the system management node.

```
[System management node]
# pmrscadm --set --rscunit resourceunit             <- Reflects contents of pmrsc.conf file
...
# pmpjmadm -c cluster -set --rscunit resourceunit   <- Reflects contents of pmpjm.conf file
...
```

Note

After confirming that the target resource unit has no running jobs, incorporate the job resource manager exit function. The pmrsc.conf file is reflected immediately after the pmrscadm command with the --set option is executed, so running jobs too will reflect the settings. For example, in processing by the exit script postfree based on the prealloc execution results, postfree may not be processed as expected for the running jobs.

2.3.4 Releasing an incorporated script

To release an incorporated exit script, delete the relevant ExitFunc section from the pmrsc.conf file, and use the pmrscadm command to reflect the settings.

Appendix A Functional Comparison of Hooks

This appendix describes functional differences between hooks.

Purpose

Prologue and epilogue function	Used to execute job pre-processing and post-processing.
Job manager exit function	Used to incorporate unique criteria (e.g., budget management) relating to job acceptance judgment and execution control.
Job scheduler exit function	Used to incorporate unique criteria (e.g., budget management) relating to job scheduling.
Job resource management exit function	Used to execute job pre-processing and post-processing. If you are aware of timing of resource allocation or release, this function is especially suitable.

Operation unit

Prologue and epilogue function	Resource unit
Job manager exit function	Resource unit or resource group
Job scheduler exit function	Resource unit or resource group
Job resource management exit function	Resource unit or resource group

Execution target

Prologue and epilogue function	For normal jobs and interactive jobs, the function is executed for each job. For bulk jobs and step jobs, it is executed for each sub job.
Job manager exit function	For normal jobs and interactive jobs, the function is executed for each job. For bulk jobs and step jobs, the exit function pjmx_quejob() is executed for each job, and the other exit functions are executed for each sub job.
Job scheduler exit function	For normal jobs and interactive jobs, the function is executed for each job. For bulk jobs and step jobs, it is executed for each sub job.
Job resource management exit function	For normal jobs and interactive jobs, the function is executed for each job. For bulk jobs and step jobs, it is executed for each sub job.

Implementation method

Prologue and epilogue function	Scripts (Shell scripts)
Job manager exit function	Programs (C language)
Job scheduler exit function	Programs (C language)
Job resource management exit function	Scripts

Job control

Prologue and epilogue function	Jobs can be controlled based on the script exit status: - Continuing a job - Placing a job in the ERROR state - Holding a job - Forcibly terminating a job - Re-executing a job
Job manager exit function	Jobs can be controlled based on the exit function return value:

	- Permitting or denying job acceptance - Permitting or denying job execution
Job scheduler exit function	Jobs can be controlled based on the exit function return value: - Permitting or denying job execution
Job resource management exit function	Jobs can be controlled based on the script exit status: - Continuing a job - Placing a job in the ERROR state - Holding a job - Forcibly terminating a job - Re-executing a job

Node on which hook is executed

Prologue and epilogue function	Node on which the job script is executed
Job manager exit function	Compute cluster management node (active)
Job scheduler exit function	Compute cluster management node (active)
Job resource management exit function	All compute nodes allocated to the job

Current directory at hook execution

Prologue and epilogue function	Current directory when the job is submitted
Job manager exit function	/var/log/FJSVtcs/pjm
Job scheduler exit function	/var/log/FJSVtcs/pjm
Job resource management exit function	root user's home directory

Operation for step job

Prologue and epilogue function	Executed on a sub job of the step job.
Job manager exit function	Executed on a sub job of the step job.
Job scheduler exit function	Executed on a sub job of the step job.
Job resource management exit function	Executed on a sub job of the step job.

Operation for bulk job

Prologue and epilogue function	Executed on a sub job of the bulk job.
Job manager exit function	Executed on the bulk job and its sub jobs.
Job scheduler exit function	Executed on a sub job of the bulk job.
Job resource management exit function	Executed on a sub job of the bulk job.

Positioning of process

Prologue and epilogue function	Executed as part of a job. You can set whether or not to add the elapsed execution time of the prologue and epilogue function to the job.
Job manager exit function	Executed as a system process.
Job scheduler exit function	Executed as a system process.
Job resource management exit function	Executed as a system process.

The following table shows the call timing of each hook.

Table A.1 Execution timing of the hook

Hook	Name of the hook	Execution timing
Prologue and epilogue function	Prologue script	The prologue script is executed in the job state of RUNNING-P.
	Epilogue script	The epilogue script is executed in the job state of RUNNING-E.
Job manager exit function	pjmx_inithook() function	This function is called when the exit function library of the job manager exit function is loaded. The exit function library is loaded when the job manager starts and when the library is incorporated by the pmpjmadm command.
	pjmx_quejob() function	This function is executed at a job state transition from ACCEPT to QUEUED.
	pjmx_alterjob() function	This function is called before the job manager function changes a parameter as instructed by the pjalter or pmalter command.
	pjmx_startjob() function	This function is executed at a job state transition from QUEUED to RUNNING-A.
	pjmx_endjob() function	This function is executed when the job state is RUNOUT.
	pjmx_deljob() function	This function is executed when the job state is EXIT.
	pjmx_finihook() function	This function is called when the exit function library of the job manager exit function is unloaded. The exit function library is unloaded when the job manager stops and when the incorporated library is released by the pmpjmadm command.
Job scheduler exit function	pjsx_inithook() function	This function is called when the exit function library of the job scheduler exit function is loaded. The exit function library is loaded when the job scheduler starts and when the library is incorporated by the pmpjmadm command.
	pjsx_prejobselect() function	This function is executed before the job scheduler function sorts jobs according to job priority.
	pjsx_postjobselect() function	This function is executed before the job scheduler function allocates compute resources to a job.
	pjsx_postrscalloc() function	This function is executed after the job scheduler function determines the compute resources to allocate to a job.
	pjsx_finihook() function	This function is called when the exit function library of the job scheduler exit function is unloaded. The exit function library is unloaded when the job scheduler stops and when the incorporated library is released by the pmpjmadm command.
Job resource management exit function	prealloc script	The prealloc script is executed in the job state of RUNNING-A.
	postfree script	The postfree script is executed immediately before the job state changes from RUNNING-E to RUNOUT (Immediately before changing from RUNNING to RUNOUT when the prologue and epilogue function is not set).
	predel script	The predel script is executed when the job is deleted by execution of the pjdel or pjhold command, excess of job resources (elapse time or memory), or the compute node (or its service) failure.

Figure A.1 Hook execution timing relationship diagram (1)

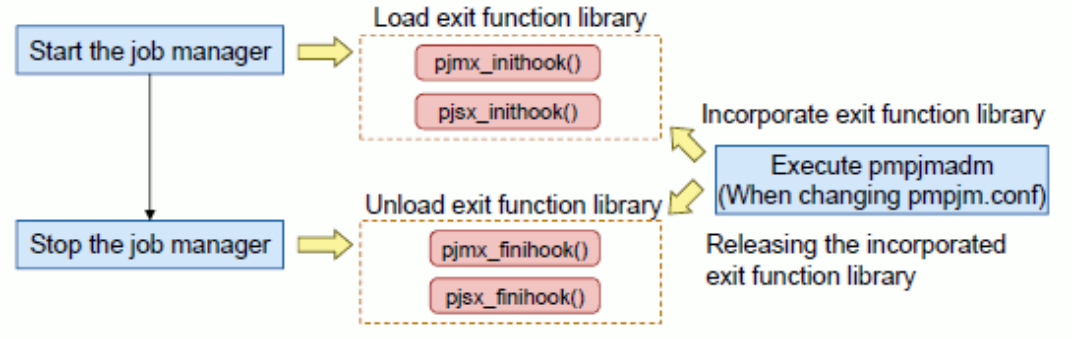


Figure A.2 Hook execution timing relationship diagram (2)

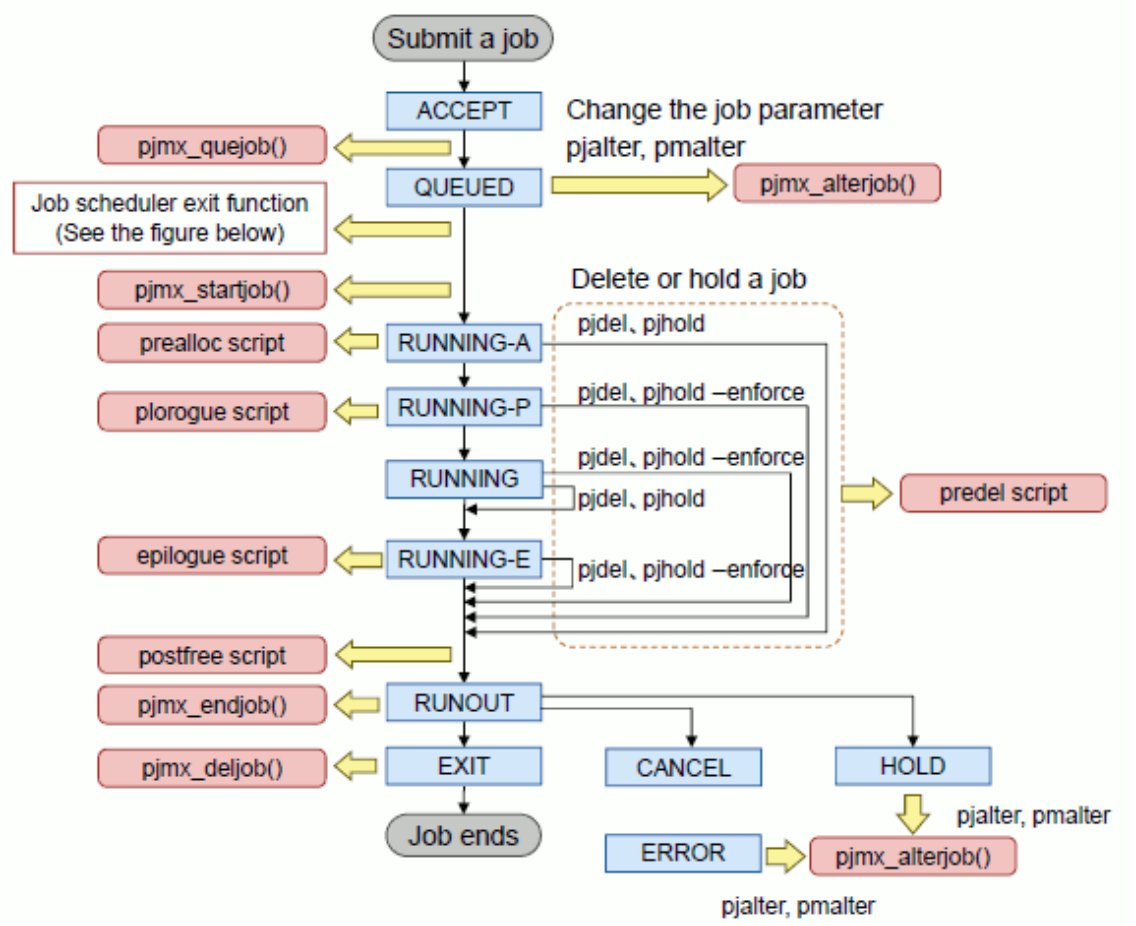
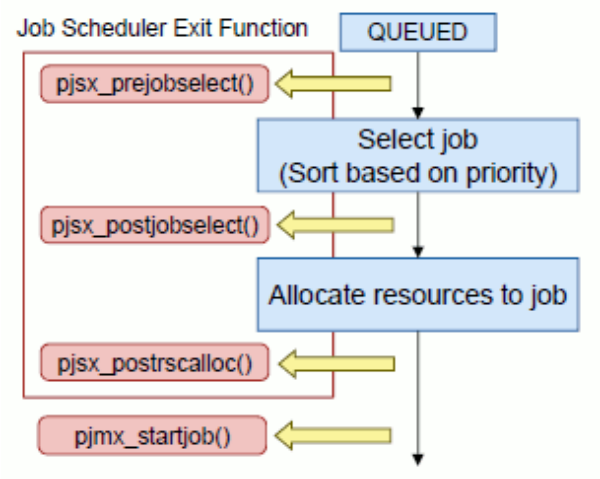


Figure A.3 Hook execution timing relationship diagram (3)



Appendix B Environment Variables

This appendix provides a list of environment variables that are set by the job operation management function. They are among the environment variables that can be referenced by the prologue and epilogue scripts of exit processes and the exit scripts of the job resource manager exit function.

Note

Depending on the exit process, some environment variables may not be set. In the following table, "Yes" indicates that the environment variable is set, and "-" indicates that it is not set.

Table B.1 Environment variables set by the job operation management function

Environment variables	Description	Prologue Epilogue	Exit Script
PJM_ADAPTIVE_ELAPSED_TIME_MAX [FX]	Maximum value (seconds) for the elapsed time limit of a job, as specified in the -L elapse option of the pjsub command If specified with a range, like with "-L elapse=30:00-1:00:00", the maximum value (1:00:00) is set as a number of seconds (3600). If the -L elapse option is not specified, the default value defined by the job ACL function is set. If specified like "-L elapse=30:00", this environment variable itself is not set.	Yes	Yes
PJM_ADAPTIVE_ELAPSED_TIME_MIN [FX]	Minimum value (seconds) for the executable time of a job, as specified in the -L elapse option of the pjsub command If the executable time is specified with a minimum value and a maximum value, like with "-L elapse=30:00-1:00:00", the minimum value (30:00) is set as a number of seconds (1800). If the -L elapse option is not specified, the default value defined by the job ACL function is set. If specified like "-L elapse=30:00", this environment variable itself is not set.	Yes	Yes
PJM_APPNAME	Character string specified in the --appname option of the pjsub command If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_ASSIGN_LOGICAL_CPU [PG]	Range types for the logical CPUs that can be used for job processes This environment variable is valid for any job process other than those of "C/C++/Fortran programs created in Development Studio." job: Job processes can use only the logical CPUs for the job in the allocated CPU core. all: Job processes can use all the logical CPUs in the allocated CPU core. For details on the logical CPUs for a job, see "Setting the Range of CPU Resources Available to a Job" in Appendix "Executing programs of MPI processing system other than Development Studio" in "Job Operation Software End-user's Guide."	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
PJM_ASSIGN_ONLINE_NODE [FX]	If the --mpi assign-online-node option of the pjsub command is specified, 1 is set. If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_AT	Job execution start time *YYYY*/*MM*/*DD hh*:*mm*:*ss* specified in the --at option of the pjsub command (*YYYY*: year, *MM*: month, *DD*: day, *hh*: hour, *mm*: minute, *ss*: second) If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_AUXGID	Auxiliary group IDs of the users who submitted jobs (set the group IDs separated by a comma)	-	Yes
PJM_BULKNUM	Bulk number (set only for a bulk job)	Yes	Yes
PJM_COMMENT	Character string specified in the --comment option of the pjsub command If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_CORE_MEM_LIMIT [PG]	Amount of memory (bytes) per CPU core specified in the -L core-mem option of the pjsub command If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_CPUINF	nformation on the CPU cores allocated to the job (specify the CPU core IDs separated by a comma)	-	Yes
PJM_CUSTOM_RESOURCES	Custom resource name and requested quantity (or requested type) specified in the -L option of the pjsub command (can be referenced from prealloc, predel, and postfree) Example: customrscname=1 (customrscname indicates a defined custom resource name) If the option is not specified, the default value defined by the job ACL function is set. If no default value is defined by the job ACL function, 0 is set for the requested quantity In this case, the requested type is not set. If multiple custom resources are requested, the resource information is delimited by a comma (","). If a custom resource per node is requested, "/n" is added to the custom resource name. For details on custom resources, see "Job scheduling function using custom resources" in "Chapter 2 Details of the Job Operation Management Function" of "Job Operation Software Administrator's Guide for Job Management."	Yes	Yes
PJM_DELINF	Trigger for predel script execution (can be referenced only in the predel script) 1: pjdel command 2: pjhold command 3: Execution triggered by job deletion for a reason (*) other than above (1 and 2) (*) The job is forcibly deleted by the --enforce option of the padeadline command. 4: Compute node error 5: CPU time limit violation 6: Elapsed time limit violation 7: Excess memory usage 8: Execution triggered by OOM Killer operation	-	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
PJM_DPREFIX	Prefix indicating a command line from a job script to the job operation software. By default, "#PJM" is set. However, if a value is explicitly specified in the -C or --dir-prefix option of the pjsub command, the value is set.	Yes	Yes
PJM_ELAPSED_TIME_MODE [FX]	Specification method for the executable time specified for a job in the -L elapse option of the pjsub command Either of the following is set: - "fixed" This is the setting when the executable time is specified like "-L elapse=30:00". - "adaptive" This is the setting when the executable time is specified with a minimum value and a maximum value, like with "-L elapse=30:00-" or "-L elapse=30:00-1:00:00". If the -L elapse option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_ELAPSE_LIMIT	Maximum executable time (seconds) for the job specified in the-L elapse option of the pjsub command If the option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_ENVIRONMENT	"BATCH" for a batch job, and "INTERACT" for an interactive job are set.	Yes	Yes
PJM_EXEC_POLICY [PG]	Execution mode policy specified in the -P exec-policy option of the pjsub command simplex: SIMPLEX (Use nodes exclusively) share: SHARE (Share nodes) If this option is not specified, the default value defined by the job ACL function is set. For a node allocation job, this environment variable is not set.	Yes	Yes
PJM_FSNAME	Character string specified in the --fs option of the pjsub command If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_GID	Group ID of the user who executed the job	-	Yes
PJM_JOBDIR	Path of the current directory of the beginning to execute job script	Yes	Yes
PJM_JOBEXIT	Job end code (which can only be referenced by postfree. Note that this environment variable cannot be referenced if the script execution was triggered by job swapping or an error that occurred before job start.) 1: Normal end of job 3: Job execution timeout due to elapsed time limit violation 4: Waiting for changing the status of the bulk job 5: Generating job fails because of failed creating of the process 6: Forcibly terminated request by PJM 7: Hardware error (ICC error) 8: Termination due to OOM Killer 9: Failure of collecting the enhanced CPU statistical information 11: Own memory acquisition failed.	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
	21: Epilogue script execution failed. 255: Other This environment variable is set only in epilogue scripts and the exit script postfree. However, the environment variable is not set when execution of the exit script postfree is triggered by an error that occurred before job start.		
PJM_JOBGRPID	Group ID of the job execution user	Yes	-
PJM_JOBID	Job ID	Yes	Yes
PJM_JOBNAME	Job name	Yes	Yes
PJM_JOBNUM	Number of jobs running on the relevant compute node However, it does not include jobs that reference this environment variable. This environment variable is set only in the exit scripts prealloc and postfree.	-	Yes
PJM_JOBUSRID	User ID of the job execution user	Yes	-
PJM_MAILOPTION	E-mail notification operation specified in the -m option of the pjsub command An OR operation on the following values is set in hexadecimal. 0x1: Send notification at job execution start (-m b) 0x2: Send notification at job end (-m e) 0x4: Send notification at job re-execution (-m r) 0x8: Include job statistical information in notification (-m s) 0x10: Include job and node statistical information in notification (-m S) Example: PJM_MAILOPTION=0x3 If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_MAILSENDTO	User specified as the e-mail destination in the --mail-list option of the pjsub command If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_MPI_PROC	Maximum number of processes generated at the start of the MPI program specified in the --mpi proc option of the pjsub command	Yes	Yes
PJM_MPI_SHAPE_X PJM_MPI_SHAPE_Y PJM_MPI_SHAPE_Z [FX]	Number of nodes in the X-, Y-, and Z-axis directions for the process shape specified in the --mpi shape option of the pjsub command If this option is not specified, the number of nodes for the node shape specified in the -L node option of the pjsub command is used. The environment variables that do not correspond to the dimensions of the shape are not set. For example, the PJM_MPI_SHAPE_Y and PJM_MPI_SHAPE_Z environment variables are not set for a one-dimensional shape.	Yes	Yes
PJM_NET_ROUTE [FX]	Operation when a link goes down for the Tofu interconnect used as the communication path of a job dynamic: Change the communication path, and job execution continues.	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
	static: Do not change the communication path. The job ends abnormally.		
PJM_NODE	Value specified in the -L node option of the pjsub command (For FX servers, the number of nodes is calculated from the specified node shape.) If the option is not specified, the default value defined in the "joblimit node" or "joblimit interact-node" item by the job ACL function is set. This environment variable is set only when physical nodes are allocated, and it is not set when virtual nodes are allocated.	Yes	Yes
PJM_NODE_ALLOCATION_MODE [FX]	Node allocation method specified in the -L node option of the pjsub command torus: Torus mode mesh: Mesh mode noncont: Non-contiguous mode If this option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_NODE_MEM_LIMIT [FX]	Upper limit of memory usage (bytes) per node specified in the -L node-mem option of the pjsub command If this option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_NODE_TYPE	Type of compute node that executes an exit script 0: Node where no job script is running 1: Node where a job script is running An exit script operates on all compute nodes allocated to a job. However, if you want to distinguish the compute nodes where a job script is operating from the other compute nodes, use this environment variable.	-	Yes
PJM_NODE_X PJM_NODE_Y PJM_NODE_Z [FX]	Number of nodes in the X-, Y-, and Z-axis directions for the shape specified in the -L node option of the pjsub command The environment variables that do not correspond to the dimensions of the shape are not set. For example, the PJM_NODE_Y and PJM_NODE_Z environment variables are not set for a one-dimensional node shape.	Yes	Yes
PJM_NORESTART	If the --norestart option of the pjsub command is specified, 1 is set. If the option is not specified, this environment variable is not set.	Yes	Yes
PJM_O_HOME	Environment variable HOME of the user who executed the pjsub command	Yes	Yes
PJM_O_HOST	Name of the host that executed the pjsub command	Yes	Yes
PJM_O_LANG	Environment variable LANG of the user who executed the pjsub command	Yes	Yes
PJM_O_LOGNAME	Environment variable LOGNAME of the user who executed the pjsub command	Yes	Yes
PJM_O_MAIL	Environment variable MAIL of the user who executed the pjsub command	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
PJM_O_NODEINF [PG]	Path of the allocated node list file For a job to which virtual nodes are allocated, the IP addresses of the physical nodes where the virtual nodes are placed are written one per line.	Yes	Yes
PJM_O_PATH	Environment variable PATH of the user who executed the pjsub command	Yes	Yes
PJM_O_SHELL	Environment variable SHELL of the user who executed the pjsub command	Yes	Yes
PJM_O_TZ	Environment variable TZ of the user who executed the pjsub command	Yes	Yes
PJM_O_WORKDIR	Current directory at the pjsub command execution time	Yes	Yes
PJM_PROC_BY_NODE	Maximum number of processes generated per physical node or virtual node by an MPI program However, if single physical node (node=1) or virtual node (vnode=1) is allocated and the --mpi option of the pjsub command is not specified, this environment variable is not set.	Yes	Yes
PJM_RANK_MAP_BYCHIP	Rank allocation rule specified in the --mpi rank-map-bychip option of the pjsub command - FX servers: The XY, YX, XYZ, XZY, YXZ, YZX, ZXY, or ZYX character string is set. - PRIMERGY servers: The number of virtual nodes to be allocated per physical node is set. If the option is not specified, this environment variable is not set. If no argument of this option is specified, DEF is displayed as the environment variable.	Yes	Yes
PJM_RANK_MAP_BYNODE	Rank allocation rule specified in the --mpi rank-map-bynode option of the pjsub command - FX servers: The XY, YX, XYZ, XZY, YXZ, YZX, ZXY, or ZYX character string is set. - PRIMERGY servers: If this option is specified, 1 is set. If the option is not specified, this environment variable is not set. If no argument of this option is specified, DEF is displayed as the environment variable.	Yes	Yes
PJM_RETRYNUM	Number of job retries	Yes	Yes
PJM_RSCGRP	Resource group name specified in the -L rscgrp or -L rg option of the pjsub command If this option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_RSCUNIT	Resource unit name specified in the -L rscunit or -L ru option of the pjsub command If this option is not specified, the default value defined by the job ACL function is set.	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
PJM_SHELL	Path of the shell which executes a job	Yes	Yes
PJM_SHELLEXIT	End code of user job script This environment variable is set only in epilogue scripts and the exit script postfree. However, the environment variable is not set when execution of the exit script postfree is triggered by an error that occurred before job start.	Yes	Yes
PJM_STDERR_PATH	Path of the standard error output file for jobs as specified in the -e option of the pjsub command If this option is not specified, the default file name "*job name.e.job ID*" is set. However, for bulk jobs and step jobs, "*job ID*" contains a sub job ID. If the -j option of the pjsub command is specified, the same value as the PJM_STDOUT_PATH environment variable value is used.	Yes	Yes
PJM_STDOUT_PATH	Path of the standard output file for jobs as specified in the -o option of the pjsub command If this option is not specified, the default file name "*job name.o.job ID*" is set. However, for bulk jobs and step jobs, "*job ID*" contains a sub job ID.	Yes	Yes
PJM_STEPNUM	Step number (set only for a step job)	Yes	Yes
PJM_SUBJOBID	Sub job ID For a normal job, the job ID is set.	Yes	Yes
PJM_UID	User ID of the user who executed the job	-	Yes
PJM_VNID	Virtual node IDs (specify the virtual node IDs, separated by a comma, to be generated on the relevant compute node)	-	Yes
PJM_VNODE	Value specified in the -L vnode option of the pjsub command If the option is not specified, the default value defined in the "joblimit vnode" or "joblimit interact-vnode" item by the job ACL function is set. This environment variable is set only when virtual nodes are allocated, and it is not set when physical nodes are allocated.	Yes	Yes
PJM_VNODE_CORE	Value specified in the -L vnode-core or -L vnode=(core=) option of the pjsub command If the option is not specified, the default value defined in the "joblimit vnode-core" or "joblimit interact-vnode-core" item by the job ACL function is set. This environment variable is set only when virtual nodes are allocated, and it is not set when physical nodes are allocated.	Yes	Yes
PJM_VNODE_MEM_LIMIT [PG]	Amount of memory (bytes) per virtual node specified in the -L vnode-mem or -L mem option of the pjsub command If this option is not specified, the default value defined by the job ACL function is set.	Yes	Yes
PJM_VN_POLICY [PG]	Virtual node placement policy specified in the -P vn-policy option of the pjsub command abs-pack: Absolutely PACK pack: PACK abs-unpack: Absolutely UNPACK unpack: UNPACK	Yes	Yes

Environment variables	Description	Prologue Epilogue	Exit Script
	If this option is not specified, the default value defined by the job ACL function is set.		
Others	Environment variable specified by the -x or -X option of pjsub command. (Environment variables starting with "PJM _" only)	Yes	Yes

Appendix C References

This appendix describes the exit functions of two of the hooks--the job manager exit function and the job scheduler exit function--in detail.

C.1 Job Manager Exit Functions

This section describes job manager exit functions.

C.1.1 Job Information Structure UsrJobInfo_t

The job information structure UsrJobInfo_t is a structure for storing information on the target job at the exit function call time. It is defined in the exit function header file FJSVtcs/pjm/pjmx.h.

```
typedef struct UsrJobInfo {
    uint32_t        jobid;
    uint16_t        job_model;
    uint16_t        job_type;
    PjmsBulkno_t    bulkinfo;
    PjmsStepno_t    stepinfo;
    uid_t           uid;
    gid_t           gid;
    char            rscunit[64];
    char            rscgrp[64];
    void            *extended_p;
} UsrJobInfo_t;
```

Table C.1 Members of the job information structure UsrJobInfo_t

Member	Type	Meaning
jobid	uint32_t	Job ID
job_model	uint16_t	Job model The bits shown in the following macro are set according to the job model. PJM_JOBMODEL_NORMAL : Normal job PJM_JOBMODEL_BULK : Bulk job PJM_JOBMODEL_STEP : Step job
job_type	uint16_t	Job type PJM_JOBTYPE_BATCH : Batch job PJM_JOBTYPE_INTERACT : Interactive job
bulkinfo	PjmsBulkno_t	Bulk job information See "Table C.2 Members of the structure PjmsBulkno_t" in a subsequent section. If the target job is neither a bulk job nor one of its sub jobs, these contents have no meaning.
stepinfo	PjmsBulkno_t	Step job information See "Table C.3 Members of the structure PjmsStepno_t" in a subsequent section. If the target job is not a sub job of a step job, these contents have no meaning.
uid	uid_t	User ID of the job
gid	gid_t	Group ID of the job
rscunit[64]	char	Resource unit name in which job is executed
rscgrp[64]	char	Resource group name for which job was executed
extended_p	void *	Pointer for extended information This member is provided as an argument when the job information acquisition function or the job information setting function, described in a subsequent section, is called.

The structure PjmsBulkno_t is as follows.

```
typedef struct PjmsBulkno {
    uint32_t        start;
    uint32_t        end;
    uint32_t        step;
    uint32_t        num_bulklist;
    uint32_t        list_type;
    int32_t         reserved;
    off_t           bulklist_off;
    void            *extended_p;
} PjmsBulkno_t;
```

Table C.2 Members of the structure PjmsBulkno_t

Member	Type	Meaning
start	uint32_t	Start value for bulk numbers For sub jobs of a bulk job, this value is the bulk number of a sub job.
end	uint32_t	End value for bulk numbers For sub jobs of a bulk job, this value is the bulk number of a sub job.
step	uint32_t	Not supported. Currently, the value is always 1.
num_bulklist	uint32_t	Not supported. Currently, the value is always 0.
list_type	uint32_t	Not supported. Currently, the value is always 0.
reserved	int32_t	Reserved area
bulklist_off	off_t	Not supported. Currently, the value is always 0.
extended_p	void *	Pointer for extended information It is used for future extension.

The structure PjmsStepno_t is as follows.

```
typedef struct PjmsStepno {
    uint16_t        stepno;
    int16_t         pad;
    void            *extended_p;
} PjmsStepno_t;
```

Table C.3 Members of the structure PjmsStepno_t

Member	Type	Meaning
stepno	uint16_t	Step number of sub job of step job
pad	int16_t	Padding
extended_p	void *	Pointer for extended information It is used for future extension.

C.1.2 Initialization function pjmx_inithook()

The initialization function pjmx_inithook() is called when the exit function library of the job manager exit function is loaded and when the job manager starts.

```
#include <FJSVtcs/pjm/pjmx.h>

void pjmx_inithook(const char *rscuname_p, const char *rscgname_p);
```

The resource unit and resource group name are set in *rscuname_p* and *rscgname_p*, respectively, and passed to the function. If the job manager exit function is defined in a resource unit section, NULL is set in *rscgname_p*.

C.1.3 Job registration function pjmx_quejob()

The job registration function pjmx_quejob() is a user exit function that is called after the end of the job acceptance process. The job manager function judges whether to accept or deny the job based on the return value of pjmx_quejob(). Depending on the value of the pjsub command option -w {nowait | jobchk}, the pjsub command that submitted the job ends after this exit function returns or the command returns before the exit function returns.

```
#include <FJSVtcs/pjm/pjmx.h>

int  pjmx_quejob(UsrJobInfo_t *usrjobinfo, QueJobInfo_t *quejobinfo)
```

Information on the target job is stored in the job information structure UsrJobInfo_t and structure QueJobInfo_t and passed to the function.

```
typedef struct QueJobInfo {
    uint64_t    elapse_limit;
    uint64_t    cputime_limit;
    uint64_t    sum_mem_limit;
    uint64_t    sum_node_quota_limit;
    int         num_node;
    int         num_node_d;
    int         sum_num_cpu;
    int         respawn_flag;
    uint32_t    num_vn;
    uint32_t    num_cpu;
    uint64_t    vnode_mem;
    char        comment[256];
    char        fs_name[64];
    char        app_name[64];
    void        *extended_p;
} QueJobInfo_t;
```

Table C.4 Members of the structure QueJobInfo_t

Member	Type	Meaning
elapse_limit	uint64_t	Upper limit on elapsed execution time (seconds) PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is specified for the elapsed execution time limit, the maximum value is set.
cputime_limit	uint64_t	Not used. The value is undefined.
sum_mem_limit	uint64_t	- For node allocated jobs Upper limit on memory usage for each node (Mi bytes) x number of nodes requested by job - For virtual node allocated jobs Upper limit on memory usage for each virtual node (Mi bytes) x number of virtual nodes requested by job PJM_RLIM_INFINITY is the setting that indicates no limit.
sum_node_quota_limit	uint64_t	Not used. The value is undefined.
num_node	int	Number of nodes requested by job For virtual node allocated jobs on FX server, the number of virtual nodes requested by the job is specified. For virtual node allocated jobs on PRIMERGY server, 0 is specified.
num_node_d	int	This is always 1.
sum_num_cpu	int	Number of CPU cores requested by job If the number of nodes is specified when the job is submitted, it is the total number of mounted CPU cores on the nodes.

Member	Type	Meaning
respawn_flag	int	Re-execution enable/disable flag 0: Enable re-execution. 1: Disable re-execution.
num_vn	uint32_t	Number of virtual nodes requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
num_cpu	uint32_t	Number of CPU cores per virtual node requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
vnode_mem	uint64_t	Upper limit on memory usage for each virtual node (Mi bytes) PJM_RLIM_INFINITY is the setting that indicates no limit. This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
comment[256]	char	Character string specified in --comment option of pjsub command
fs_name[64]	char	Character string specified in --fs option of pjsub command
app_name[64]	char	Character string specified in --appname option of pjsub command
extended_p	void *	Pointer for extended information It is used for future extension.

You can control job acceptance by setting the function pjmx_quejob() return value as follows.

Table C.5 Return values and operations of the job registration function pjmx_quejob()

Return value	Operation
PJMX_SUCCESS	Permits job acceptance. The job transitions to the QUEUED state.
PJMX_FAILED	Denies job acceptance.
Other	The same as for PJMX_SUCCESS

C.1.4 Job attribute change function pjmx_alterjob()

The job attribute change function pjmx_alterjob() is called when a job parameter, such as the resource unit or resource group of a job, is changed. The job manager function determines whether or not to change a job attribute based on the return value of pjmx_alterjob ().

```
#include <FJSVtcs/pjm/pjmx.h>

int pjmx_alterjob(UsrJobInfo_t *usrjobinfo, AlterJobInfo_t *alterjobinfo);
```

Information on the target job is stored in the job information structure UsrJobInfo_t and structure AlterJobInfo_t, and passed to the function.

```
typedef struct AlterJobInfo {
    uint64_t    elapse_limit;
    uint64_t    aprio_p;
    uint64_t    uprio_p;
    uint64_t    cputime_limit;
    uint64_t    sum_mem_limit;
    uint64_t    sum_node_quota_limit;
    int         num_node;
    int         num_node_d;
    int         sum_num_cpu;
    int         respawn_flag;
    uint32_t    num_vn;
    uint32_t    num_cpu;
    uint64_t    vnode_mem;
    char        comment[256];
    char        fs_name[64];
```

```
    char        app_name[64];
    char        rscunit[64];
    char        rscgrp[64];
    uint16_t    requester;
    uint8_t     pad[6];
    void        *extended_p;
} AlterJobInfo_t;
```

Table C.6 Members of the structure AlterJobInfo_t

Member	Type	Meaning
elapse_limit	uint64_t	Upper limit on elapsed execution time after the change (seconds) PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is specified for the elapsed execution time limit, the maximum value is set.
aprio_p	uint64_t	Job priority within the resource unit after the change
uprio_p	uint64_t	Job priority within the same user after the change
cputime_limit	uint64_t	Not used. The value is undefined.
sum_mem_limit	uint64_t	- For node allocated jobs Upper limit on memory usage for each node (Mi bytes) x number of nodes requested by job For virtual node allocated jobs Upper limit on memory usage for each virtual node (Mi bytes) x number of virtual nodes requested by job PJM_RLIM_INFINITY is the setting that indicates no limit.
sum_node_quota_limit	uint64_t	Not used. The value is undefined.
num_node	int	Number of nodes requested by job For virtual node allocated jobs on FX server, the number of virtual nodes requested by the job is specified. For virtual node allocated jobs on PRIMERGY server, 0 is specified.
num_node_d	int	This is always 1.
sum_num_cpu	int	Number of CPU cores requested by job If the number of nodes is specified when the job is submitted, it is the total number of mounted CPU cores on the nodes.
respawn_flag	int	Re-execution enable/disable flag 0: Enable re-execution. 1: Disable re-execution.
num_vn	uint32_t	Number of virtual nodes requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
num_cpu	uint32_t	Number of CPU cores per virtual node requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
vnode_mem	uint32_t	Upper limit on memory usage for each virtual node (Mi bytes) PJM_RLIM_INFINITY is the setting that indicates no limit. This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
comment[256]	char	Character string specified in --comment option of pjsub command
fs_name[64]	char	Character string specified in --fs option of pjsub command
app_name[64]	char	Character string specified in --appname option of pjsub command
rscunit[64]	char	Resource unit name after the change

Member	Type	Meaning
rscgrp[64]	char	Resource group name after the change
requester	uint16_t	The command invoked the function PJM_REQUESTER_PJ (0x7069) is set for the pjalter command and PJM_REQUESTER_PM (0x706d) for the pmalter command
pad[6]	uint8_t	Padding
extended_p	void *	Pointer for extended information It is used for future extension.

Information

Values at job submission are set in members other than rscunit, rscgrp, elapse_limit, aprio_p, and uprio_p.

You can control job attribute changes by setting the return value of the function pjmx_alterjob() as follows.

Table C.7 Return values and operations of the job attribute change function pjmx_alterjob()

Return value	Operation
PJMX_SUCCESS	Permits job attribute changes.
PJMX_FAILED	Denies job attribute changes.
Other	The same as for PJMX_SUCCESS

Note

If the return value of this function is PJMX_FAILED, the pjalter or pmalter command that triggered the call trigger has ended in an error. If you want to change the error message shown at this time for the command, set a message with the job information setting function pjmx_setinfo_message().

C.1.5 Job pre-execution function pjmx_startjob()

The job pre-execution function pjmx_startjob() is called immediately before job execution. The job manager function judges whether to execute the job based on the return value of pjmx_startjob().

```
#include <FJSVtcs/pjm/pjmx.h>

int pjmx_startjob(UsrJobInfo_t *usrjobinfo, StartJobInfo_t *startjobinfo);
```

Information on the target job is stored in the job information structure UsrJobInfo_t and structure StartJobInfo_t and passed to the function.

```
typedef struct StartJobInfo {
    uint64_t    elapse_limit;
    uint64_t    cputime_limit;
    uint64_t    sum_mem_limit;
    uint64_t    sum_node_quota_limit;
    int         num_node;
    int         num_node_d;
    int         num_node_alloc;
    int         sum_num_cpu;
    int         sum_num_cpu_alloc;
    int         pad;
    uint32_t    num_vn;
    uint32_t    num_cpu;
    uint64_t    vnode_mem;
    void        *extended_p;
} StartJobInfo_t;
```

Table C.8 Members of the structure StartJobInfo_t

Member	Type	Meaning
elapse_limit	uint64_t	Upper limit on elapsed execution time (seconds) PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is specified for the elapsed execution time limit, the maximum value is set.
cputime_limit	uint64_t	Not used. The value is undefined.
sum_mem_limit	uint64_t	- For node allocated jobs Upper limit on memory usage for each node (Mi bytes) x number of nodes requested by job For virtual node allocated jobs Upper limit on memory usage for each virtual node (Mi bytes) x number of virtual nodes requested by job PJM_RLIM_INFINITY is the setting that indicates no limit.
sum_node_quota_limit	uint64_t	Not used. The value is undefined.
num_node	int	Number of nodes requested by job For virtual node allocated jobs on FX server, the number of virtual nodes requested by the job is specified. For virtual node allocated jobs on PRIMERGY server, 0 is specified.
num_node_d	int	This is always 1.
num_node_alloc	int	Number of nodes allocated to job For virtual node allocated jobs, the number of nodes actually allocated is specified.
sum_num_cpu	int	Number of CPU cores requested by job If the number of nodes is specified when the job is submitted, it is the total number of mounted CPU cores on the nodes.
sum_num_cpu_alloc	int	Number of CPU cores allocated to job
pad	int	Padding
num_vn	uint32_t	Number of virtual nodes requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
num_cpu	uint32_t	Number of CPU cores per virtual node requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
vnode_mem	uint64_t	Upper limit on memory usage for each virtual node (Mi bytes) PJM_RLIM_INFINITY is the setting that indicates no limit. This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
extended_p	void *	Pointer for extended information It is used for future extension.

You can control job acceptance by setting the pjmx_startjob() return value as follows.

Table C.9 Return values and operations of the job pre-execution function pjmx_startjob()

Return value	Operation
PJMX_SUCCESS	Permits job execution. The job transitions to the RUNNING-A state.
PJMX_FAILED_QUEUED	Denies job execution. The job transitions to the QUEUED state. This job is not scheduled until one of the following events occurs: - Running jobs that belong to the same group as this job end. - This job transitions to the HOLD state.

Return value	Operation
	- The job manager function service restarts.
PJMX_FAILED_CANCEL	Denies job execution. The job transitions to the CANCEL state and is deleted.
PJMX_FAILED_RESTART	Denies job execution. The job transitions to the QUEUED state and is scheduled.
PJMX_FAILED_ERROR	Denies job execution. The job transitions to the ERROR state.
Other	The same as for PJMX_SUCCESS

C.1.6 Job end function pjmx_endjob()

The job end function pjmx_endjob() is an exit function that is called when a job ends. It is called even if the job is forcibly terminated when deleted or held. This function returns no value. The job manager function does not use the results of this function to control the job state.

```
#include <FJSVtcs/pjm/pjmx.h>

void  pjmx_endjob(UsrJobInfo_t *usrjobinfo, EndJobInfo_t *endjobinfo);
```

Information on the target job is stored in the job information structure UsrJobInfo_t and structure EndJobInfo_t and passed to the function.

```
typedef struct EndJobInfo {
    uint64_t    elapse_limit;
    uint64_t    cputime_limit;
    uint64_t    sum_mem_limit;
    uint64_t    sum_node_quota_limit;
    int         num_node;
    int         num_node_d;
    int         num_node_alloc;
    int         sum_num_cpu;
    int         sum_num_cpu_alloc;
    int         pad;
    uint32_t    num_vn;
    uint32_t    num_cpu;
    uint64_t    normal_mem;
    int         pjm_code;
    uid_t       lastcancel_uid;
    uint64_t    detail_code;
    uint64_t    exec_elapse;
    uint64_t    utime;
    uint64_t    stime;
    uint        used_nodenum;
    uint32_t    used_cpunum;
    uint64_t    used_memory;
    void        *extended_p;
} EndJobInfo_t;
```

Table C.10 Members of the structure EndJobInfo_t

Member	Type	Meaning
elapse_limit	uint64_t	Upper limit on elapsed execution time (seconds) PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is specified for the elapsed execution time limit, the maximum value is set.
cputime_limit	uint64_t	Not used. The value is undefined.
sum_mem_limit	uint64_t	For node allocated jobs Upper limit on memory usage for each node (Mi bytes) x number of nodes requested by job For virtual node allocated jobs Upper limit on memory usage for each virtual node (Mi bytes) x number of virtual nodes requested by job PJM_RLIM_INFINITY is the setting that indicates no limit.

Member	Type	Meaning
sum_node_quota_limit	uint64_t	Not used. The value is undefined.
num_node	int	Number of nodes requested by job For virtual node allocated jobs on FX server, the number of virtual nodes requested by the job is specified. For virtual node allocated jobs on PRIMERGY server, 0 is specified.
num_node_d	int	This is always 1.
num_node_alloc	int	Number of nodes allocated to job For virtual node allocated jobs, the number of nodes actually allocated is specified.
sum_num_cpu	int	Number of CPU cores requested by job If the number of nodes is specified when the job is submitted, it is the total number of mounted CPU cores on the nodes.
sum_num_cpu_alloc	int	Number of CPU cores allocated to job
pad	int	Padding
num_vn	uint32_t	Number of virtual nodes requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
num_cpu	uint32_t	Number of CPU cores per virtual node requested by job This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
normal_mem	uint64_t	Upper limit on memory usage for each virtual node (Mi bytes) PJM_RLIM_INFINITY is the setting that indicates no limit. This value is 0 for jobs on the FX server and for node-allocated jobs on the PRIMERGY server.
pjm_code	int	PJM code The meaning of the value of this item is the same as for the item PJM CODE in the job statistical information output by the pjstat or pjsub command. See the man page for pjstatsinfo(7).
lastcancel_uid	uid_t	User ID of last user who canceled or held job
detail_code	uint64_t	Detail code (for troubleshooting)
exec_elapse	uint64_t	Elapsed time since execution start (seconds)
utime	uint64_t	User CPU time used by job (milliseconds)
stime	uint64_t	System CPU time used by job (milliseconds)
used_nodenum	uint	Number of nodes used by job
used_cpunum	uint32_t	Number of CPU cores used by job
used_memory	uint64_t	Amount of memory used by job (bytes)
extended_p	void *	Pointer for extended information It is used for future extension.

C.1.7 Job deletion function pjmx_deljob()

The job deletion function pjmx_deljob() is an exit function that is called when job information is deleted. This exit function is called not only when a job is deleted by the pjdel command but also for any job accepted by the job manager function. This function returns no value. The job manager function does not use the results of this function to control the job state.

```
#include <FJSVtcs/pjm/pjmx.h>

void pjmx_deljob(UsrJobInfo_t *usrjobinfo, DelJobInfo_t *deljobinfo);
```

Information on the target job is stored in the job information structure UsrJobInfo_t and structure DelJobInfo_t and passed to the function.

```
typedef struct DelJobInfo {
    int         pjm_code;
    uid_t       lastcancel_uid;
```

```
    uint64_t    detail_code;
} DelJobInfo_t;
```

Table C.11 Members of the structure DelJobInfo_t

Member	Type	Meaning
pjm_code	int	PJM code
lastcancel_uid	uid_t	User ID of last user who canceled or held job
detail_code	uint64_t	Detail code (for troubleshooting)

C.1.8 Finalization function pjmx_finihook()

The finalization function pjmx_finihook() is called when the job manager exit function in a library format is unloaded and when the job manager stops.

```
#include <FJSVtcs/pjm/pjmx.h>

void pjmx_finihook(const char *rscuname_p, const char *rscgname_p);
```

The resource unit and resource group name are set in *rscuname_p* and *rscgname_p*, respectively, and passed to the function. If the job manager exit function is defined in a resource unit section, NULL is set in *rscgname_p*.

C.2 Job Scheduler Exit Functions

This section describes the job scheduler exit function APIs.

C.2.1 Initialization function pjsx_inithook()

The initialization function pjsx_inithook() is called when the job scheduler exit function is loaded and when the job scheduler starts.

```
#include <FJSVtcs/pjm/pjsx.h>

void pjsx_inithook(const char *rscuname_p, const char *rscgname_p)
```

The resource unit and resource group name are set in *rscuname_p* and *rscgname_p*, respectively, and passed to the function. If the job scheduler exit function is defined in a resource unit section, NULL is set in *rscgname_p*.

C.2.2 Job preselection exit function pjsx_prejobselect()

The job preselection exit function pjsx_prejobselect() is called immediately before jobs are sorted according to priority in a job scheduling process. This function is called once every time a job scheduling process runs.

```
#include <FJSVtcs/pjm/pjsx.h>

int pjsx_prejobselect(JobSelectInfo_t *jobselectinfo)
```

Information on the target job is stored in the structure JobSelectInfo_t and passed to the function.

```
typedef struct JobSelectInfo {
    uint32_t job_num;
    void     **extended_pp;
} JobSelectInfo_t;
```

Table C.12 Members of the structure JobSelectInfo_t

Member	Type	Meaning
job_num	uint32_t	Number of jobs
extended_pp	void **	Pointer for extended information on job_num jobs is set as an array.

Member	Type	Meaning
		Specify the pointer for extended information on the target job as an argument when calling the job information acquisition function or job information setting function.

Table C.13 Return values and operations of the job preselection exit function pjsx_prejobselect()

Return value	Operation
PJSX_SUCCESS	Succeeded.
Other	The same as for PJSX_SUCCESS

Note

The job preselection exit function pjsx_prejobselect() is called only when it is included in the exit function library registered in the ResourceUnit section. The job preselection exit function pjsx_prejobselect() of the exit function library registered in the ResourceGroup section is not called.

C.2.3 Job post-selection exit function pjsx_postjobselect()

After jobs are sorted according to priority in a job scheduling process, the job post-selection exit function pjsx_postjobselect() is called immediately before compute resources are allocated. The job manager function determines whether or not to execute a job based on the return value of pjmx_postjobselect().

```
#include <FJSVtcs/pjm/pjsx.h>

int pjsx_postjobselect(void *extended_p)
```

Acquire information on the target job by specifying the argument *extended_p* for the job information acquisition function.

You can control job execution by setting the return value at the end of pjsx_postjobselect() as follows.

Table C.14 Return values and operations of the job post-selection exit function pjsx_postjobselect ()

Return value	Operation
PJSX_SUCCESS	Permits job execution.
PJSX_FAILED_QUEUED	Denies job execution. The job transitions to the QUEUED state. The job manager function automatically makes the job subject to scheduling again.
Other	The same as for PJSX_SUCCESS

C.2.4 Resource post-selection exit function pjsx_postrscalloc()

The resource post-selection exit function pjsx_postrscalloc() is a user exit function that is called after compute resources are allocated to a job in a job scheduling process.

```
#include <FJSVtcs/pjm/pjsx.h>

int pjsx_postrscalloc(void *extended_p)
```

Acquire information on the target job by specifying the argument *extended_p* for the job information acquisition function.

You can control job execution by setting the return value of pjsx_postrscalloc() as follows.

Table C.15 Return values and operations of the resource post-selection exit function pjsx_postrscalloc()

Return value	Operation
PJSX_SUCCESS	Permits job execution.
PJSX_FAILED_QUEUED	Denies job execution. The job transitions to the QUEUED state. The job manager function automatically makes the job subject to scheduling again.

Return value	Operation
Other	The same as for PJSX_SUCCESS

C.2.5 Finalization function pjsx_finihook()

The finalization function pjsx_finihook() is called when the job scheduler exit function is unloaded and when the job scheduler stops.

```
#include <FJSVtcs/pjm/pjsx.h>

void pjsx_finihook(const char *rscuname_p, const char *rscgname_p)
```

The resource unit and resource group name are stored in *rscuname_p* and *rscgname_p*, respectively, and passed to the function. If the job scheduler exit function is defined in a resource unit section, NULL is set in *rscgname_p*.

C.3 Job Information Acquisition Functions

This section describes the job information acquisition function APIs that can be used in exit functions.

There are three types of job information acquisition functions, classified as follows based on what functions they can call:

- Functions that can be called only from job manager exit functions (the names of the functions begin with "pjmx")
- Functions that can be called only from job scheduler exit functions (the names of the functions begin with "pjsx")
- Functions that can be called from both (the names of the functions begin with "pjmsx")

The job information acquisition function returns the following values.

Table C.16 Return values of the job information acquisition function

Return value	Meaning
0	The function ended normally.
1	The function cannot be called from this exit function.
2	The area for information storage is insufficient. Example: len is too small. len is the length of the job name storage area.
3	A specified parameter is invalid. Example: The function tried to acquire step-job information even though the job is not a step job.
4	Job information could not be acquired. Example: The specified job does not exist.
8	This information acquisition function tried to acquire a type of item that does not match the type of value acquired (when the functions pjmx_getinfo_stats_item_*type*() and pjmx_getinfo_stats_item_array_*type*() are called).
9	Information could not be acquired for a specified item. (when the functions pjmx_getinfo_stats_item_*type*() and pjmx_getinfo_stats_item_array_*type*() are called)
10	No information has been set.

C.3.1 pjmx functions

This section describes the job information acquisition functions (functions whose names begin with "pjmx") that can be called only from job manager exit functions.

When using a job information acquisition function, provide the member *extended_p* as an argument of the job information structure UsrJobInfo_t. The member is passed as an argument of a job manager exit function.

Acquired information	Number of retries
Function name	int pjmx_getinfo_numretry(uint16_t **numretry_p*, const void **extended_p*)
Description	Stores the number of job retries in **numretry_p*.

Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job name
Function name	int pjmx_getinfo_jobname(char **jobname_p*, int *len*, const void **extended_p*)
Description	Stores the job name in **jobname_p*. The maximum length of the job name is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Host name on which the pjsub command executed
Function name	int pjmx_getinfo_host(char **host_p*, int *len*, const void **extended_p*)
Description	Stores the host name of the node that executed the pjsub command, in **host_p*. The maximum length of the host name is 16 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Last state of job
Function name	int pjmx_getinfo_last_state(int16_t **last_state_p*, const void **extended_p*)
Description	Stores the last state of the job in **last_state_p*. The meaning of each value is as follows: 1: Accepted job submission (ACCEPT) 2: Waiting for job execution (QUEUED) 5: Acquiring resources required for job execution (RUNNING-A) 6: Executing job (RUNNING) 7: Waiting for completion of job termination processing (RUNOUT) 12: Holding status by user operation (HOLD) 13: In fixed state due to an error (ERROR)
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	State of job
Function name	int pjmx_getinfo_state(int16_t **state_p*, const void **extended_p*)
Description	Stores the state of the job in **state_p*. The meaning of each value is as follows: 1: Accepted job submission (ACCEPT) 2: Waiting for job execution (QUEUED) 5: Acquiring resources required for job execution (RUNNING-A) 6: Executing job (RUNNING) 7: Waiting for completion of job termination processing (RUNOUT) 9: Completed job termination processing (EXIT) 11: Ended due to cancellation of job execution (CANCEL) 12: Holding status by user operation (HOLD) 13: In fixed state due to an error (ERROR)
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job processing error message
Function name	int pjmx_getinfo_reason(char **reason_p*, int *len*, const void **extended_p*)
Description	Stores the job processing error message in **reason_p*. The maximum length of the error message is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.

	The error message is the same as the item REASON in the job statistical information output by the pjstat or pjsub command. For details, see "Appendix A Job Information" in the "Job Operation Software End-User's Guide."
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Process CPU time limit (seconds)
Function name	int pjmx_getinfo_prc_cputm_lmt(uint64_t **prc_cputm_lmt_p*, const void **extended_p*)
Description	Stores the CPU time limit for each process in **prc_cputm_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Core file size limit value for each process (bytes)
Function name	int pjmx_getinfo_prc_corefile_lmt(uint64_t **prc_corefile_lmt_p*, const void **extended_p*)
Description	Stores the maximum core file size for each process in **prc_corefile_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Maximum number of processes generated with a real user ID for the processes
Function name	int pjmx_getinfo_prc_cre_proc_lmt(uint64_t **prc_cre_proc_lmt_p*, const void **extended_p*)
Description	Stores the maximum number of processes generated with a real user ID for the processes, in **prc_cre_proc_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Maximum size of process data segment (bytes)
Function name	int pjmx_getinfo_prc_data_lmt(uint64_t **prc_data_lmt_p*, const void **extended_p*)
Description	Stores the maximum size of the process data segment in **prc_data_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	POSIX message queue size limit value for each process (bytes)
Function name	int pjmx_getinfo_prc_psx_msg_que_lmt(uint64_t **prc_psx_msg_que_lmt_p*, const void **extended_p*)
Description	Stores the limit value of the POSIX message queue size for each process, in **prc_psx_msg_que_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Maximum number of file descriptors opened by each process
Function name	int pjmx_getinfo_prc_openfiles_lmt(uint64_t **prc_openfiles_lmt_p*, const void **extended_p*)
Description	Stores the maximum number of file descriptors opened by each process, in **prc_openfiles_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Limit value of number of signals for each process
Function name	int pjmx_getinfo_prc_pndng_sgnl_lmt(uint64_t **prc_pndng_sgnl_lmt_p*, const void **extended_p*)
Description	Stores the limit value of the number of signals for each process, in **prc_pndng_sgnl_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Maximum size of the process stack segment (bytes)
Function name	int pjmx_getinfo_prc_stack_lmt(uint64_t **prc_stack_lmt_p*, const void **extended_p*)

Description	Stores the maximum size of the process stack segment, in **prc_stack_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	Maximum size of process virtual memory (bytes)
Function name	int pjmx_getinfo_prc_vmem_lmt(uint64_t **prc_vmem_lmt_p*, const void **extended_p*)
Description	Stores the maximum size of process virtual memory, in **prc_vmem_lmt_p*.
Possible caller	pjmx_quejob()

Acquired information	File mask
Function name	int pjmx_getinfo_umask(int **umask_p*, const void **extended_p*)
Description	Stores the file mask value (umask value) in **umask_p*.
Possible caller	pjmx_quejob()

Acquired information	E-mail transmission flag
Function name	int pjmx_getinfo_mailflag(int **mailflag_p*, const void **extended_p*)
Description	Stores the parameters specified in the -m option of the pjsub command, in **mailflag_p*. Each bit of the value corresponds to the respective parameter. An OR operation on the following values is performed: 1: b (e-mail transmission at job start) 2: e (e-mail transmission at job end) 4: r (e-mail transmission at job re-execution) 8: s (statistical information notification - without node/virtual node information) 16: S (statistical information output - with node/virtual node information)
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Mail address
Function name	int pjmx_getinfo_address(char **address_p*, int *len*, const void **extended_p*)
Description	Stores the e-mail address in **address_p*. The maximum length of the mail address is 256 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Job execution shell
Function name	int pjmx_getinfo_shell(char **shell_p*, int *len*, const void **extended_p*)
Description	Stores the path name of the job script execution shell, in **shell_p*. The maximum length of the path name is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Comment of job
Function name	int pjmx_getinfo_comment(char **comment_p*, int *len*, const void **extended_p*)
Description	Stores the character string specified in the -c option of the pjsub command, in **comment_p*. The maximum length of the character string is 256 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Step job dependency expression
Function name	int pjmx_getinfo_stp_endcode(char **stp_endcode_p*, int *len*, const void **extended_p*)
Description	Stores the step job dependency expression specified in the "--sparam sd=" option of the pjsub command, in **stp_endcode_p*. The maximum length of the step job dependency expression is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Path name of standard output file
Function name	int pjmx_getinfo_std_file(char **std_file_p*, int *len*, const void **extended_p*)
Description	Stores the path name of the job standard output file, in **std_file_p*. The maximum length of the path name is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Path name of standard error output file
Function name	int pjmx_getinfo_std_err_file(char **std_err_file_p*, int *len*, const void **extended_p*)
Description	Stores the path name of the job standard error output file, in **std_err_file_p*. The maximum length of the path name is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Path name of statistical information file
Function name	int pjmx_getinfo_info_file(char **info_file_p*, int *len*, const void **extended_p*)
Description	Stores the path name of the job statistical information storage file, in **info_file_p*. The maximum length of the path name is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Path name of the directory on which the job was submitted
Function name	int pjmx_getinfo_pjsub_dir(char **pjsub_dir_p*, int *len*, const void **extended_p*)
Description	Stores the path name of the current directory at the job submission time, in **pjsub_dir_p*. The maximum length of the path name is 4096 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Argument in the --fs option of the pjsub command
Function name	int pjmx_getinfo_fs_name(char **fs_name_p*, int *len*, const void **extended_p*)
Description	Stores the character string specified in the --fs option of the pjsub command, in **fs_name_p*. The maximum length of the character string is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Argument in the --appname option of the pjsub command
Function name	int pjmx_getinfo_app_name(char **app_name_p*, int *len*, const void **extended_p*)

Description	Stores the application name specified in the pjsub command, in **app_name_p*. The maximum length of the application name is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	Rank allocation rule
Function name	int pjmx_getinfo_rankmap_type(int **rankmap_type_p*, const void **extended_p*)
Description	Stores the rank allocation rule in **rankmap_type_p*. The meaning of each value is as follows: 0: No rule specified 1: rank-map-bynode 2: rank-map-bychip
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Number of rank map arrangement
Function name	int pjmx_getinfo_bychip_n(int **bychip_n_p*, const void **extended_p*)
Description	Stores the number of rank map arrangements specified in the --mpi rank-map-bychip=*n* option of the pjsub command, in **bychip_n_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Number of virtual nodes arrangement
Function name	int pjmx_getinfo_unpack_n(int **unpack_n_p*, const void **extended_p*)
Description	Stores the number of virtual nodes allocated as specified in the -P vn-policy=unpack=*n* or -P vn-policy=abs-unpack=*n* option of the pjsub command, in **unpack_n_p*. The value does not have any meaning for the jobs on FX servers or the node allocated jobs on PRIMERGY servers.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Compute node type
Function name	int pjmx_getinfo_node_type(int **node_type_p*, const void **extended_p*)
Description	Stores the model of the compute node in **node_type_p*. The meaning of each value is as follows: 1: PRIMERGY server 2: FX server
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Virtual node placement policy and execution mode policy
Function name	int pjmx_getinfo_policy_ex(uint64_t **policy_p*, const void **extended_p*)
Description	Stores the virtual node placement policy and execution mode policy, in **policy_p*. Based on the value acquired by this function, the macro RET_VN_POLICY() or RET_E_POLICY() is used to reference each policy. - Macro RET_VN_POLICY(*pol*,*ret*) When the information **policy* acquired by the function pjmx_getinfo_policy_ex() is assigned to the argument *pol*, the virtual node placement policy of the target job is stored in *ret*. The type of *ret* is uint64_t. The meaning of each value is as follows: 0: PACK 1: UNPACK

	2: ABS_PACK 3: ABS_UNPACK - Macro RET_E_POLICY(*pol*,*ret*) When the information *policy acquired by the function pjmx_getinfo_policy_ex() is assigned to the argument pol, the execution mode policy of the target job is stored in ret. The type of *ret* is uint64_t. The meaning of each value is as follows: 0: SIMPLEX 1: SHARE
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Number of requested static MPI processes
Function name	int pjmx_getinfo_req_mpi_static_proc(int32_t **req_mpi_static_proc_p*, const void **extended_p*)
Description	Stores the number of requested static MPI processes in **req_mpi_static_proc_p*. (It is the number of processes generated when an MPI program starts.)
Possible caller	pjmx_quejob()

Acquired information	Number of requested MPI processes
Function name	int pjmx_getinfo_req_mpi_proc(int32_t **req_mpi_proc_p*, const void **extended_p*)
Description	Stores the upper limit on the number of requested MPI processes in **req_mpi_proc_p*. (It is the sum of the number of static processes generated when an MPI program starts and the number of dynamic processes generated during MPI program execution.)
Possible caller	pjmx_quejob()

Acquired information	Number of allocated static MPI processes
Function name	int pjmx_getinfo_alloc_mpi_static_proc(int32_t **alloc_mpi_static_proc_p*, const void **extended_p*)
Description	Stores the number of allocated static MPI processes in **alloc_mpi_static_proc_p*. (It is the number of processes generated when an MPI program starts.)
Possible caller	pjmx_quejob()

Acquired information	Number of allocated MPI processes
Function name	int pjmx_getinfo_alloc_mpi_proc(int32_t **alloc_mpi_proc_p*, const void **extended_p*)
Description	Stores the upper limit on the number of allocated processes, in **alloc_mpi_proc_p*. (It is the sum of the number of static processes generated when an MPI program starts and the number of dynamic processes generated during MPI program execution.)
Possible caller	pjmx_quejob()

Acquired information	NUMA allocation policy
Function name	int pjmx_getinfo_numa_policy(uint64_t **numa_policy_p*, const void **extended_p*)
Description	Stores the NUMA allocation policy set for the job ACL function in **numa_policy_p*. The meaning of each value is as follows: 0: PACK 1: UNPACK
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Start bulk number

Function name	int pjmx_getinfo_start_bulkno(uint32_t **start_bulkno_p*, const void **extended_p*)
Description	Stores the start bulk number in **start_bulkno_p*. If the target job is not a bulk job, 0 is stored.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	End bulk number
Function name	int pjmx_getinfo_end_bulkno(uint32_t **end_bulkno_p*, const void **extended_p*)
Description	Stores the end bulk number in **start_bulkno_p*. If the target job is not a bulk job, 0 is stored.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Scheduling start time
Function name	int pjmx_getinfo_last_sched_date(struct timespec **last_sched_date_p*, const void **extended_p*)
Description	Stores the job scheduling start date and time in **last_sched_date_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Information on the options specified for the pjsub command
Function name	int pjmx_getinfo_pjsub_option_flg(uint64_t **pjsub_option_flg_p*, const void **extended_p*)
Description	Stores information on the specified options in the pjsub command, in **pjsub_option_flg_p*. Each bit in the value **pjsub_option_flg_p* corresponds to an option that can be specified for the pjsub command. If an option is specified, the corresponding bit is set. For the correspondence between the options and bits, see "Job submission option flag" in the man page pjstatsinfo(7).
Possible caller	pjmx_quejob()

Acquired information	Information on the arguments specified in the -L option of the pjsub command
Function name	int pjmx_getinfo_pjsub_L_arg_flg(uint64_t **pjsub_L_arg_flg_p*, const void **extended_p*)
Description	Stores information on the arguments specified in the -L or --rsc-list option of the pjsub command, in **pjsub_L_arg_flg_p*. Each bit in the value **pjsub_L_arg_flg_p* corresponds to an argument that can be specified for the -L or --rsc-list option. If an argument is specified, the corresponding bit is set. For the correspondence between the arguments and bits, see "-L option argument flag of the pjsub command" in the man page pjstatsinfo(7).
Possible caller	pjmx_quejob()

Acquired information	Information on the arguments specified in the --mpi option of the pjsub command
Function name	int pjmx_getinfo_pjsub_mpi_arg_flg(uint64_t **pjsub_mpi_arg_flg_p*, const void **extended_p*)
Description	Stores information on the arguments specified in the --mpi option of the pjsub command, in **pjsub_mpi_arg_flg_p*. Each bit in the value **pjsub_mpi_arg_flg_p* corresponds to an argument that can be specified for the --mpi option. If an argument is specified, the corresponding bit is set. For the correspondence between the arguments and bits, see "--mpi option argument flag of the pjsub command" in the man page pjstatsinfo(7).
Possible caller	pjmx_quejob()

Acquired information	Information on the arguments specified in the --step option of the pjsub command
Function name	int pjmx_getinfo_pjsub_step_arg_flg(uint64_t **pjsub_step_arg_flg_p*, const void **extended_p*)

Description	Stores information on the arguments specified in the --step option of the pjsub command, in **pjsub_step_arg_flg_p*. Each bit in the value **pjsub_step_arg_flg_p* corresponds to an argument that can be specified for the --step option. If an argument is specified, the corresponding bit is set. For the correspondence between the arguments and bits, see "--step option argument flag of the pjsub command" in the man page pjstatsinfo(7).
Possible caller	pjmx_quejob()

Acquired information	Information on the arguments specified in the -P option of the pjsub command
Function name	int pjmx_getinfo_pjsub_P_arg_flg(uint64_t **pjsub_P_arg_flg_p*, const void **extended_p*)
Description	Stores information on the arguments specified in the -P option of the pjsub command, in **pjsub_P_arg_flg_p* Each bit in the value **pjsub_P_arg_flg_p* corresponds to an argument that can be specified for the -P option. If an argument is specified, the corresponding bit is set. For the correspondence between the arguments and bits, see "-P option argument flag of the pjsub command" in the man page pjstatsinfo(7).
Possible caller	pjmx_quejob()

Acquired information	Environment variable
Function name	int pjmx_getinfo_envdata(void ***data*, const void **extended_p*)
Description	Sets the pointer to the environment variable storage area in *data*. Each environment variable is stored in the "*name=value*" format, with multiple environment variables separated by a NULL character (\0). Two consecutive NULL characters indicate the end of the area. This area is secured within the function and released by the calling side. This function can acquire the environment variables specified in the -x or -X option of the pjsub command and environment variables (names beginning with "PJM_O_") automatically added by the job operation management function.
Possible caller	pjmx_quejob(), pjmx_startjob()

Acquired information	List of allocated nodes
Function name	int pjmx_getinfo_allocnode_list(uint32_t ***allocnod_list_pp*, void **extended_p*)
Description	Stores the address of the list (array) of node IDs of the nodes allocated to a job, in **allocnod_list_pp*. The size of the node ID list is the number of nodes or virtual nodes allocated to the job. For a virtual node allocated job, the node ID of the node actually allocated for the respective virtual nodes is stored. This means that the same node ID may be output multiple times. The node ID list storage area is secured within the function and released by the calling side.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Allocation mode of job
Function name	int pjmx_getinfo_allocationmode(uint32_t **allocationmode_p*, const void **extended_p*)
Description	Stores the allocation mode of job. The following values are set in **allocationmode_p*. For a job on FX servers: 0 : torus mode. 1 : mesh mode. 2 : a node-sharing job (virtual node allocated job). 3 : non-contiguous mode. For a job on PRIMERGY servers: 0 : a virtual node allocated job. 1 : a node allocated job.

Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	List of execution nodes for a job
Function name	int pjmx_getinfo_execnode_list(uint32_t **execnode_num_p*, uint32_t ***execnodelist_pp*, const void **extended_p*);
Description	Stores the addresses of a node ID list (array) of nodes that will or actually executed a job in **execnodelist_pp*. The size of the node ID list (number of nodes) is stored in *execnode_num_p. The node ID list storage area is secured within the function and released by the calling side.
Possible caller	pjmx_startjob(), pjmx_endjob()

Acquired information	Power consumption of a job
Function name	int pjmx_getinfo_total_electric_energy(double **electric_energy_p*, const void **extended_p*)
Description	Stores the power consumption (Wh) of the job in **electric_energy_p*. It fails in the acquisition of power information if the job acquiring power information has already been executed by the same node. In this case, the return value of the function is four. If other jobs were executed while acquiring power information on the job by the same node, power information that other jobs executed by the same node used is included.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Maximum power consumption of jobs
Function name	int pjmx_getinfo_max_electric_power(double **max_electric_power_p*, const void **extended_p*)
Description	Stores the maximum power consumption (W) of jobs in **max_electric_power_p*. It fails in the acquisition of power information if the job acquiring power information has already been executed by the same node. In this case, the return value of the function is four. If other jobs were executed while acquiring power information on the job by the same node, power information that other jobs executed by the same node used is included.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Minimum power consumption of jobs
Function name	int pjmx_getinfo_min_electric_power(double **min_electric_power_p*, const void **extended_p*)
Description	Stores the minimum power consumption (W) of jobs in **min_electric_power_p*. It fails in the acquisition of power information if the job acquiring power information has already been executed by the same node. In this case, the return value of the function is four. If other jobs were executed while acquiring power information on the job by the same node, power information that other jobs executed by the same node used is included.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Average power consumption of jobs
Function name	int pjmx_getinfo_ave_electric_power(double **ave_electric_power_p*, const void **extended_p*)
Description	Stores the average power consumption (W) of jobs in **ave_electric_power_p*. It fails in the acquisition of power information if the job acquiring power information has already been executed by the same node. In this case, the return value of the function is four. If other jobs were executed while acquiring power information on the job by the same node, power information that other jobs executed by the same node used is included.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Supplementary information (Item SUPPLEMENTARY INFORMATION in job statistical information)

Function name	int pjmx_getinfo_supplementary_info(char **info_p*, int *len*, const void **extend_p*)
Description	Stores the supplementary information in **info_p*. The maximum length of the character string is 256 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify the size in *len*.
Possible caller	pjmx_quejob()

Acquired information	strict and strict-io specification of job
Function name	int pjmx_getinfo_strictmode(int16_t **strictmode_p*, const void **extended_p*)
Description	Acquires information on whether strict or strict-io is specified for the job. The following values are set in **strictmode_p*. 0: Neither strict nor strict-io is specified. 1: strict is specified. 2: strict-io is specified.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	User communication receive data size
Function name	int pjmx_getinfo_tofu_user_comm_rsv_byte(int64_t **tofu_user_comm_rsv_byte*, const void **extended_p*)
Description	Stores the receive data size (bytes) used for user-level communication via Tofu in **tofu_user_comm_rsv_byte*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	User communication send data size
Function name	int pjmx_getinfo_tofu_user_comm_send_byte(int64_t **tofu_user_comm_send_byte*, const void **extended_p*)
Description	Stores the send data size (bytes) used for user-level communication via Tofu in **tofu_user_comm_send_byte*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	File I/O communication receive data size
Function name	int pjmx_getinfo_tofu_sys_comm_rsv_byte(int64_t **tofu_sys_comm_rsv_byte*, const void **extended_p*)
Description	Stores the receive data size (bytes) used for file I/O via Tofu in **tofu_sys_comm_rsv_byte*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	File I/O communication send data size
Function name	int pjmx_getinfo_tofu_sys_comm_send_byte(int64_t **tofu_sys_comm_send_byte*, const void **extended_p*)
Description	Stores the send data size (bytes) used for file I/O via Tofu in **tofu_sys_comm_send_byte*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Number of times the Fujitsu profiler is used
Function name	int pjmx_getinfo_fj_profiler(int64_t **fj_profiler*, const void **extended_p*)
Description	Stores the number of times that the job has used the profiler (started the profiler command) in **fj_profiler*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Whether the sector cache is used
Function name	int pjmx_getinfo_sector_cache(int64_t **sector_cache*, const void **extended_p*)

Description	Stores the number of times that the job has used (initialized) the sector cache in **sector_cache*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Whether the inter-core hardware barrier is used
Function name	int pjmx_getinfo_intra_node_barrier(int64_t **barrier*, const void **extended_p*)
Description	Stores the number of times that the job has used (initialized) the inter-core hardware barrier in **barrier*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Tofu logical coordinates of the nodes used by the job
Function name	int pjmx_getinfo_tofu_logical_coordinate(PjmxLogicalCoordinate_t ***tofu_logical_coordinate*, const void **extended_p*)

Description	Stores an array of the Tofu logical coordinates of the nodes used by the job in **tofu_logical_coordinate*.
	The number of structures is the number of nodes used by the job, that is, the member num_node_alloc of the structure StartJobInfo_t or EndJobInfo_t passed to the exit function of the caller. If the caller function is pjmx_alterjob() or pjmx_deljob(), use pjmx_getinfo_stats_item_uint32() to acquire the number of nodes (nnumu).
	If the caller function is pjmx_quejob(), there are none of nodes are assigned and the function fails to get the tofu logical ccordinates of the node used by the job, returninig code 4.
	The storage area is secured within the function and released by the calling side.
	
	The structure PjmxLogicalCoordinate_t is as follows:
	
	`typedef struct PjmxLogicalCoordinate {`
	`  uint32_t  tofu_x;  // Tofu X coordinate`
	`  uint32_t  tofu_y;  // Tofu Y coordinate`
	`  uint32_t  tofu_z;  // Tofu Z coordinate`
	`  uint8_t pad[4];  // Padding`
	`} PjmxLogicalCoordinate_t;`

Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job execution environment start time
Function name	int pjmx_getinfo_jobexecute_begin_time(time_t **begin_time*, const void **extended_p*)
Description	Stores the time taken to start the job execution environment in **begin_time*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job execution environment end time
Function name	int pjmx_getinfo_jobexecute_shut_time(time_t **shut_time*, const void **extended_p*)
Description	Stores the time taken to exit the job execution environment in *shut_time.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Operation when a Tofu interconnect link goes down
Function name	int pjmx_getinfo_net_route(uint8_t **net_route_dynamic_p*, const void **extended_p*)
Description	Stores the operation for whether to change the communication path when a link goes down for a communication path of the Tofu interconnect used by a job, in **net_route_dynamic_p*. 0: Do not change the communication path. The job ends abnormally. 1: Dynamically change the communication path, and job execution continues.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job power consumption information

Function name	int pjmx_getinfo_power_consumption(PjmxPower_t **power_p*, const void **extended_p*)

Description	Stores the power consumption information used by the job in * *power_p*
	
	`struct PjmxPower {`
	`  double  max_tofu;  // Maximum Tofu power consumption (W)`
	`  double  min_tofu;  // Minimum Tofu power consumption (W)`
	`  double  ave_tofu;  // Average Tofu power consumption (W)`
	`  double  max_cpu_peripherals; // Maximum power consumption at`
	`  // the periphery inside the CPU (W)`
	`  double  min_cpu_peripherals; // Minimum power consumption at`
	`  // the periphery inside the CPU (W)`
	`  double  ave_cpu_peripherals; // Average power consumption at`
	`  // the periphery inside the (W)`
	`  double  max_opticalmodule;  // Maximum power consumption of`
	`  // the optical module (W)`
	`  double  min_opticalmodule;  // Minimum power consumption of`
	`  // the optical module (W)`
	`  double  ave_opticalmodule;  // Average power consumption of`
	`  // the optical module (W)`
	`  double  max_pcle;  // Maximum PCIe power consumption (W)`
	`  double  min_pcle;  // Minimum PCIe power consumption (W)`
	`  double  ave_pcle;  // Average PCIe power consumption (W)`
	`  double  max_node_est;  // Maximum node power consumption`
	`  // (Estimate) (W)`
	`  double  min_node_est;  // Minimum node power consumption`
	`  // (Estimate) (W)`
	`  double  ave_node_est;  // Average node power consumption`
	`  // (Estimate) (W)`
	`  double  max_node_measure;  // Maximum node power consumption`
	`  // (actual measurement) (W)`
	`  double  min_node_measure;  // Minimum node power consumption`
	`  // (actual measurement) (W)`
	`  double  ave_node_measure;  // Average node power consumption`
	`  // (actual measurement) (W)`
	`  uint32_t  num_cmg;  // Numger of CMGs (Core Memory Group)`
	`  uint8_t  pad[4];  // Padding`
	`  PjmPowerCmg_t *cmg_p;  // Data for each CMG`
	`} PjmxPower_t`
	
	The member cmg_p points to the area that has PjmxPowerCmg_t structures (data for each CMG) at num_cmg consecutive locations. This area is secured within the function and released by the calling side.
	
	`struct PjmxPowerCmg {`
	`  int32_t  cmgno;  // CMG number`
	`  uint8_t pad[4];  // padding`
	`  double  max_core;  // Maximum power consumption of`
	`  // the compute core group (W)`
	`  double  min_core;  // Minimum power consumption of`
	`  // the compute core group (W)`
	`  double  ave_core;  // Average power consumption of`
	`  // the compute core group (W)`
	`  double  max_l2cache; // Maximum power consumption of`
	`  // the L2 cache (W)`
	`  double  min_l2cache; // Minimum power consumption of`
	`  // the L2 cache (W)`
	`  double  ave_l2cache; // Average power consumption of`
	`  // the L2 cache (W)`
	`  double  max_mem;  // Maximum memory power consumption (W)`
	`  double  min_mem;  // Minimum memory power consumption (W)`
	`  double  ave_mem;  // Average memory power consumption (W)`
	`} PjmxPowerCmg_t`
	
	If power acquisition failed or nothing was acquired, 0 is set.

	Average power consumption, maximum power consumption, and minimum power consumption have the following meaning: - Average power consumption Value calculated by dividing the power consumption from job start to job end by the job execution time - Maximum power consumption Maximum value among the time average values of power consumption collected at a certain interval from job start to job end. The maximum value per job is the sum of the per-node maximum values. - Minimum power consumption Minimum value among the time average values of power consumption collected at a certain interval from job start to job end. The minimum value per job is the sum of the per-node minimum values.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Power knob operation information
Function name	int pjmx_getinfo_power_knob_flg(uint8_t **power_knob_flg_p*, const void **extended_p*)
Description	Stores information on power nob operations by applications in **power_knob_flg_p*. The meanings of the values are as follows: 0: Not operated 1: Operated For details on the power knob, see "Job Operation Software API user's Guide for Power API."
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job information notification structure
Function name	int pjmx_getinfo_pjmapi_info_job(Pjmapi_info_job_t ***infojob_pp*, const void **extended_p*)
Description	Stores the address of the storage area for the job information notification structure (Pjmapi_info_job_t) reported by the information acquisition external API, in ***info_job_pp*. The job information notification structure contains job-related information, such as the amount of resources requested by the job, limit value, scheduling result information, and statistical information. For details on the structure Pjmapi_info_job_t, see "Job Operation Software API user's Guide for Job Information Notification API." The storage area for the job information notification structure is secured within the function and released by the calling side.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

Acquired information	Job statistical information item value
Function name	int pjmx_getinfo_stats_item_*type*(const char **record_name*, uint32_t *id*, const char **item_name*, *type***value*, const void **extended_p*) The function is one of multiple functions that differ in the *type* part of the function name.
Description	Stores the value of the item *item_name* of the record specified by *record_name* and *id* in the area indicated by *value*. The type of stored value is *type*. There is a function for each type of value to acquire, and the *type* part of the function name indicates the type. For details on each function name, see the table below. *record_name*: Record name ("JI": Job statistical information, "JN": Node statistical information, "SUM": Summary information of a bulk or step job). Attempting to acquire node statistical information by this function calling from pjmx_quejob() or pjmx_startjob() results in an error (Return value 3). *id* : Record ID. For node statistical information, specify a node ID. You can acquire node IDs with the job information acquisition function pjmx_getinfo_allocnode_list(). For job statistical information, *id* is ignored. *item_name*: Item name *value*: Storage area for the value. The area must be large enough to store a value of the type *type*. *extended_p*: Job information that is passed to the exit function The type *type* of the item *item_name* to acquire is as follows:

	- For an item defined by the job operation management function For the item type, see the man page pjstatsinfo(7). - For an item defined by the job operation administrator DataType defined in the papjmstats.conf file
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

The following table lists the functions prepared for each type of item.

Type of item	Function
PJMX_DATATYPE_CHAR	int pjmx_getinfo_stats_item_char(const char **record_name*, uint32_t *id*, const char **item_name*, char **value*, const void **extended_p*)
PJMX_DATATYPE_INT8	int pjmx_getinfo_stats_item_int8(const char **record_name*, uint32_t *id*, const char **item_name*, int8 **value*, const void **extended_p*)
PJMX_DATATYPE_UINT8	int pjmx_getinfo_stats_item_uint8(const char **record_name*, uint32_t *id*, const char **item_name*, uint8 **value*, const void **extended_p*)
PJMX_DATATYPE_INT16	int pjmx_getinfo_stats_item_int16(const char **record_name*, uint32_t *id*, const char **item_name*, int16 **value*, const void **extended_p*)
PJMX_DATATYPE_UINT16	int pjmx_getinfo_stats_item_uint16(const char **record_name*, uint32_t *id*, const char **item_name*, uint16 **value*, const void **extended_p*)
PJMX_DATATYPE_INT32	int pjmx_getinfo_stats_item_int32(const char **record_name*, uint32_t *id*, const char **item_name*, int32 **value*, const void **extended_p*)
PJMX_DATATYPE_UINT32	int pjmx_getinfo_stats_item_uint32(const char *record_name, uint32_t *id*, const char **item_name*, uint32 **value*, const void **extended_p*)
PJMX_DATATYPE_INT64	int pjmx_getinfo_stats_item_int64(const char **record_name*, uint32_t *id*, const char **item_name*, int64 **value*, const void **extended_p*)
PJMX_DATATYPE_UINT64	int pjmx_getinfo_stats_item_uint64(const char **record_name*, uint32_t *id*, const char **item_name*, uint64 **value*, const void **extended_p*)
PJMX_DATATYPE_FLOAT	int pjmx_getinfo_stats_item_float(const char **record_name*, uint32_t *id*, const char **item_name*, float **value*, const void **extended_p*)
PJMX_DATATYPE_DOUBLE	int pjmx_getinfo_stats_item_double(const char **record_name*, uint32_t *id*, const char **item_name*, double **value*, const void **extended_p*)
PJMX_DATATYPE_TIMESPEC	int pjmx_getinfo_stats_item_timespec(const char **record_name*, uint32_t *id*, const char **item_name*, struct timespec **value*, const void **extended_p*)
PJMX_DATATYPE_TIME	int pjmx_getinfo_stats_item_time(const char **record_name*, uint32_t *id*, const char **item_name*, time_t **value*, const void **extended_p*)
PJMX_DATATYPE_SIZE	int pjmx_getinfo_stats_item_size(const char **record_name*, uint32_t *id*, const char **item_name*, size_t **value*, const void **extended_p*)
PJMX_DATATYPE_STRING	int pjmx_getinfo_stats_item_string(const char **record_name*, uint32_t *id*, const char **item_name*, char **value*, const void **extended_p*) (*) The argument *value* must specify an area large enough to store the character string acquired according to the item *item_name*.

Acquired information	Job statistical information item value (array)
Function name	int pjmx_getinfo_stats_item_array_*type*(const char **record_name*, uint32_t *id*, const char **item_name*, *type* ***value*, int32_t **num*, const void **extended_p*) The function is one of multiple functions that differ in the *type* part of the function name.
Description	Acquires the value of the item *item_name* of the record specified by *record_name* and *id* in an array of the type *type*. *value* points to the area storing the location of the acquired array, and *num* points to the area storing the number of elements in the array.

	There is a function for each type of value to acquire, and the *type* part of the function name indicates the type. For details on each function name, see the table below. *record_name*: Record name ("JI": Job statistical information, "JN": Node statistical information, "SUM": Summary information of a bulk or step job). Attempting to acquire node statistical information by this function calling from pjmx_quejob() or pjmx_startjob() results in an error (Return value 3). *id*: Record ID. For node statistical information, specify the node ID. For job statistical information, *id* is ignored. *item_name*: Item name *value*: Storage area for the location of the array. The array (area that **value* points to) is stored within the function and released by the calling side. *num*: Storage area for the number of elements in the array *extended_p*: Job information that is passed to the exit function The type *type* of the item *item_name* to acquire is as follows: - For an item defined by the job operation management function For the item type, see the man page for pjstatsinfo(7). - For an item defined by the job operation administrator DataType defined in the papjmstats.conf file
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

The following table lists the functions prepared for each type of item.

Type of item	Function
PJMX_DATATYPE_INT8_ARRAY	int pjmx_getinfo_stats_item_array_int8(const char **record_name*, uint32_t *id*, const char **item_name*, int8 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_UINT8_ARRAY	int pjmx_getinfo_stats_item_array_uint8(const char **record_name*, const uint32_t *id*, const char **item_name*, uint8 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_INT16_ARRAY	int pjmx_getinfo_stats_item_array_int16(const char **record_name*, uint32_t *id*, const char **item_name*, int16 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_UINT16_ARRAY	int pjmx_getinfo_stats_item_array_uint16(const char **record_name*, uint32_t *id*, const char **item_name*, uint16 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_INT32_ARRAY	int pjmx_getinfo_stats_item_array_int32(const char **record_name*, uint32_t *id*, const char **item_name*, int32 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_UINT32_ARRAY	int pjmx_getinfo_stats_item_array_uint32(const char **record_name*, uint32_t *id*, const char **item_name*, uint32 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_INT64_ARRAY	int pjmx_getinfo_stats_item_array_int64(const char **record_name*, uint32_t *id*, const char **item_name*, int64 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_UINT64_ARRAY	int pjmx_getinfo_stats_item_array_uint64(const char **record_name*, uint32_t *id*, const char **item_name*, uint64 ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_FLOAT_ARRAY	int pjmx_getinfo_stats_item_array_float(const char **record_name*, uint32_t *id*, const char **item_name*, float ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_DOUBLE_ARRAY	int pjmx_getinfo_stats_item_array_double(const char **record_name*, uint32_t *id*, const char **item_name*, double ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_TIMESPEC_ARRAY	int pjmx_getinfo_stats_item_array_timespec(const char **record_name*, uint32_t *id*, const char **item_name*, struct timespec ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_TIME_ARRAY	int pjmx_getinfo_stats_item_array_time(const char **record_name*, uint32_t *id*, const char **item_name*, time_t ***value*, int32_t **num*, const void **extended_p*)
PJMX_DATATYPE_SIZE_ARRAY	int pjmx_getinfo_stats_item_array_size(const char **record_name*, uint32_t *id*, const char **item_name*, size_t ***value*, int32_t **num*, const void **extended_p*)

C.3.2 pjsx functions

This section describes the job information acquisition functions (functions whose names begin with "pjsx") that can be called only from job scheduler exit functions.

When using a job information setting function, provide the pointer *extended_p* for extended information as an argument. The pointer is passed as an argument of a job scheduler exit function.

Acquired information	Job ID
Function name	int pjsx_getinfo_jobid(uint32_t **jobid_p*, const void **extended_p*)
Description	Stores the job ID in **jobid_p*.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Job model
Function name	int pjsx_getinfo_jobmodel(uint16_t **job_model_p*, const void **extended_p*)
Description	Stores the job model in **job_mode_p*. The bits shown in the following macro are set according to the job model. PJM_JOBMODEL_NORMAL : Normal job PJM_JOBMODEL_BULK : Bulk job PJM_JOBMODEL_STEP : Step job
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Job type
Function name	int pjsx_getinfo_jobtype(uint16_t **job_type_p*, const void **extended_p*)
Description	Stores the job type in **job_type_p*. PJM_JOBTYPE_BATCH : Batch job PJM_JOBTYPE_INTERACT : Interactive job
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Bulk job information
Function name	int pjsx_getinfo_bulkinfo(PjmsBulkno_t **bulkinfo_p*, const void **extended_p*)
Description	Stores bulk job information in **bulkinfo_p*. For details on the structure PjmsBulkno_t, see "Table C.2 Members of the structure PjmsBulkno_t." After confirming with the function pjsx_getinfo_jobmodel() that the job is a bulk job, ensure that this function is called. If the target job is neither a bulk job nor one of its sub jobs, these contents have no meaning.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Step job information
Function name	int pjsx_getinfo_stepinfo(PjmsStepno_t **stepinfo_p*, const void **extended_p*)
Description	Stores step job information in **stepinfo_p*. For details on the structure PjmsStepno_t, see "Table C.3 Members of the structure PjmsStepno_t." After confirming with the function pjsx_getinfo_jobmodel() that the job is a step job, ensure that this function is called. If the target job is not a sub job of a step job, these contents have no meaning.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Job user ID
Function name	int pjsx_getinfo_uid(uid_t **uid_p*, const void **extended_p*)
Description	Stores the job user ID in **uid_p*.

Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Job group ID
Function name	int pjsx_getinfo_gid(gid_t **gid_p*, const void **extended_p*)
Description	Stores the job group ID in **gid_p*.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Name of the resource unit executing the job
Function name	int pjsx_getinfo_rscunit(char **rscuname_p*, int *len*, const void **extended_p*)
Description	Stores the name of the resource unit executing the job in **rscuname_p*. The maximum length of the resource unit name is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify its size in *len*.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Name of the resource group executing the job
Function name	int pjsx_getinfo_rscgroup(char **rscgname_p*, int *len*, const void **extended_p*)
Description	Stores the name of the resource group executing the job in **rscgname_p*. The maximum length of the resource group name is 64 bytes (including the trailing NULL character). Secure the storage area on the calling side, and specify its size in *len*.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Upper limit on elapsed execution time
Function name	int pjsx_getinfo_elapse_limit(uint64_t **elapse_limit_p*, const void **extended_p*)
Description	Stores the upper limit (seconds) on elapsed execution time in **elapse_limit_p*. PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is specified for the elapsed execution time limit, the maximum value is set.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Limit on the total amount of memory for each physical/virtual node
Function name	int pjsx_getinfo_mem_limit(uint64_t **mem_limit_p*, const void **extended_p*)
Description	- For a physical node allocated job Upper limit value (mebibytes) of memory usage of each physical node x number of nodes requested by the job - For a virtual node allocated job Upper limit value (mebibytes) of memory usage of each virtual node x number of virtual nodes requested by the job Determine whether the job requested physical nodes or virtual nodes by using the function pjsx_getinfo_num_vn(). PJM_RLIM_INFINITY is the setting that indicates no limit.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Number of nodes requested by the job
Function name	int pjsx_getinfo_num_node(int **num_node_p*, const void **extended_p*)

Description	Stores the number of nodes requested by the job in **num_node_p*. If the job is a virtual node allocated job on an FX server, this is the number of virtual nodes requested by the job. If it is a virtual node allocated job on a PRIMERGY server, 0 is set.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Number of virtual nodes requested by the job
Function name	int pjsx_getinfo_num_vn(uint32_t **num_vn_p*, const void **extended_p*)
Description	Stores the number of virtual nodes requested by the job in **num_vn_p*. If the job is a node allocated job, 0 is set.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Number of CPU cores per virtual node requested by the job
Function name	int pjsx_getinfo_cpu(uint32_t **num_cpu_p*, const void **extended_p*)
Description	Stores the number of CPU cores per virtual node requested by the job in **num_cpu_p*. If the job is a node allocated job, 0 is set.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Upper limit on memory usage per virtual node
Function name	int pjsx_getinfo_vnode_mem(uint64_t **vnode_mem_p*, const void **extended_p*)
Description	Stores the upper limit value (mebibytes) of memory usage per virtual node in **vnode_mem_p*. If the job is a node allocated job, 0 is set. PJM_RLIM_INFINITY is the setting that indicates no limit.
Possible caller	pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Number of physical nodes allocated to the job
Function name	int pjsx_getinfo_node_alloc(int **num_node_alloc_p*, const void **extended_p*)
Description	Stores the number of physical nodes allocated to the job in **num_node_alloc_p*. Even if the job is a virtual node allocated job, the number of physical nodes that are actually allocated is set, instead of the number of virtual nodes.
Possible caller	pjsx_postrscalloc()

Acquired information	Number of CPU cores allocated to the job
Function name	int pjsx_getinfo_cpu_alloc(int **num_cpu_alloc_p*, const void **extended_p*)
Description	Stores the number of CPU cores allocated to the job in **num_cpu_alloc_p*.
Possible caller	pjsx_postrscalloc()

Acquired information	Flag for judging whether the job is excluded from scheduling
Function name	int pjsx_getinfo_sched_skip(int **sched_skip_p*, const void **extended_p*)
Description	Stores the flag to judge whether the job is excluded from scheduling in **sched_skip_p*. The meanings of the stored values are as follows: 1: The job is excluded from scheduling. Other than 1: The job is subject to scheduling.
Possible caller	pjsx_prejobselect()

C.3.3 pjmsx functions

This section describes the job information acquisition functions (functions whose names begin with "pjmsx") that can be called from both job manager exit functions and job scheduler exit functions.

When using a job information setting function, provide the member *extended_p* as an argument of the job information structure UsrJobInfo_t, or provide the pointer *extended_p* for extended information as an argument. The member is passed as an argument of a job manager exit function. The pointer is passed as an argument of a job scheduler exit function.

Acquired information	Job priority in the resource unit
Function name	int pjmsx_getinfo_aprio(int16_t **aprio_p*, const void **extended_p*)
Description	Stores the job priority within the resource unit in **aprio_p*. The priority is a value from 0 to 255. The larger the value is, the higher the priority is.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Job priority within the same user
Function name	int pjmsx_getinfo_uprio(int16_t **uprio_p*, const void **extended_p*)
Description	Stores the job priority within the same user in **uprio_p*. The priority is a value from 0 to 255. The larger the value is, the higher the priority is.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Size in the X-axis direction of the node shape requested by a job
Function name	int pjmsx_getinfo_node_require_x(uint **node_require_x_p*, const void **extended_p*)
Description	Stores the following size in **node_require_x_p*: size in the X-axis direction of the node shape requested by a job. For a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Size in the Y-axis direction of the node shape requested by a job
Function name	int pjmsx_getinfo_node_require_y(uint **node_require_y_p*, const void **extended_p*)
Description	Stores the following size in **node_require_y_p*: size in the Y-axis direction of the node shape requested by a job. For a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Size in the Z-axis direction of the node shape requested by a job
Function name	int pjmsx_getinfo_node_require_z(uint **node_require_z_p*, const void **extended_p*)
Description	Stores the following size in **node_require_z_p*: size in the Z-axis direction of the node shape requested by a job. For a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Size in the X-axis direction of the node shape allocated to a job
Function name	int pjmsx_getinfo_node_alloc_x(uint **node_alloc_x_p*, const void **extended_p*)

Description	Stores the following size in **node_alloc_x_p*: size in the X-axis direction of the node shape allocated to a job. For a non-contiguously allocated job (FX server) or a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_postrscalloc()

Acquired information	Size in the Y-axis direction of the node shape allocated to a job
Function name	int pjmsx_getinfo_node_alloc_y(uint **node_alloc_y_p*, const void **extended_p*)
Description	Stores the following size in **node_alloc_y_p*: size in the Y-axis direction of the node shape allocated to a job. For a non-contiguously allocated job (FX server) or a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_postrscalloc()

Acquired information	Size in the Z-axis direction of the node shape allocated to a job
Function name	int pjmsx_getinfo_node_alloc_z(uint **node_alloc_z_p*, const void **extended_p*)
Description	Stores the following size in **node_alloc_z_p*: size in the Z-axis direction of the node shape allocated to a job. For a non-contiguously allocated job (FX server) or a job on a PRIMERGY server, this is 0.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_postrscalloc()

Acquired information	The number of nodes that cannot be used
Function name	int pjmsx_getinfo_unavailable_nodenum(uint **unavailable_nodenum_p*, const void **extended_p*)
Description	Stores the number of allocated nodes that cannot be used, in **unavailable_nodenum_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_postrscalloc()

Acquired information	Applying backfill to a job
Function name	int pjmsx_getinfo_backfill_flg(uint8_t **backfill_flg_p*, const void **extended_p*)
Description	Stores the value indicating whether to apply backfill to scheduling of the target job, in **backfill_flg_p*. 0: Did not apply backfill. 1: Applied backfill.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_postrscalloc()

Acquired information	Planned execution start time
Function name	int pjmsx_getinfo_spec_date(time_t **spec_date_p*, const void **extended_p*)
Description	Stores the planned execution start time for a job in **spec_date_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Method of specifying the elapsed time limit of a job
Function name	int pjmsx_getinfo_elapsed_time_mode(uint8_t **elapsed_time_mode_p*, const void **extended_p*)
Description	Stores the method of specifying the elapsed time limit of a job in **elapsed_time_mode_p*. PJM_ELAPSED_TIME_MODE_ADAPTIVE: Specify a range of values for the elapsed time limit (elapse=*min*-*max*) PJM_ELAPSED_TIME_MODE_FIXED: Specify a fixed value for the elapsed time limit (elapse=*limit*)

Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Minimum value in the range of values specified for the elapsed time limit
Function name	int pjmsx_getinfo_adaptive_elapsed_time_min(uint64_t **adaptive_elapsed_time_min_p*, const void **extended_p*)
Description	Stores the minimum value (seconds) in the range of values specified for the elapsed time limit in **adaptive_elapsed_time_min_p*. PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is not specified for the elapsed time limit, 3 (An invalid parameter is specified.) is set as the return value.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Maximum value in the range of values specified for the elapsed time limit
Function name	int pjmsx_getinfo_adaptive_elapsed_time_max(uint64_t **adaptive_elapsed_time_max_p*, const void **extended_p*)
Description	Stores the maximum value (seconds) in the range of values specified for the elapsed time limit in **adaptive_elapsed_time_max_p*. PJM_RLIM_INFINITY is the setting that indicates no limit. If a range of values is not specified for the elapsed time limit, 3 (An invalid parameter is specified.) is set as the return value.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Requested quantity of a custom resource
Function name	int pjmsx_getinfo_req_cstmrsc_numeric(const char **name_p*, int64_t **req_num_p*, const void **extend_p*)
Description	Stores the requested amount of the custom resource specified by **name_p*, in **req_num_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Requested type of a custom resource
Function name	int pjmsx_getinfo_req_cstmrsc_string(const char **name_p*, char **req_str_p*, const void **extend_p*)
Description	Stores the requested type of the custom resource specified by **name_p*, in **req_str_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Whether a resource is a custom resource per node
Function name	int pjmsx_getinfo_cstmrsc_is_pernode(const char **name_p*, uint32_t **result_p*, const void **extended_p*)
Description	Stores whether the custom resource specified by **name_p* is a resource per node, in **result_p*. The meaning of each value is as follows: 0: The custom resource is not a resource per node. 1: The custom resource is a resource per node.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Number of the custom resource information

Function name	int pjmsx_getinfo_req_cstmrsc_num_all(uint32_t **num_cstmrsc_p*, const void **extended_p*)
Description	Stores number of custom resources requested by the job, in **num_cstmrsc_p*.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

Acquired information	Custom resource information
Function name	int pjmsx_getinfo_req_cstmrsc_info_all(uint32_t *num_cstmrsc*, PjmsxCstmRscInfo_t **cstmrscinfo_p*,const void **extended_p*)
Description	Stores a custom resource information list (array) in **cstmrscinfo_p*. *num_cstmrsc* stores the values that can be acquired with pjmsx_getinfo_req_cstmrsc_num_all(). Secure the storage area on the calling side, and specify the size as follows: Number of custom resources acquired with pjmsx_getinfo_req_cstmrsc_num_all() x sizeof(PjmsxCstmRscInfo_t)
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()

The structure PjmsxCstmRscInfo_t is shown below.

```
typedef struct PjmsxCstmRscInfo {
    char        cstmrsc_name[PJMSX_MAX_CSTM_NAME_LEN]; /* Custom resource name */
    uint8_t     value_type;                            /* Custom resource type */
    uint8_t     is_pernode;                            /* Custom resource range */
    union {
        int64_t num_value;                             /* Requested quantity of custom resource */
        char   string_value[PJMSX_MAX_CSTM_TYPE_LEN];  /* Requested type of custom resource */
    } req_value_rsc;
} PjmsxCstmRscInfo_t;
```

Table C.17 Members of the structure PjmsxCstmRscInfo_t

Member	Type	Meaning
cstmrsc_name[]	char	Set custom resource name

value_type	uint8_t	Set custom resource type
		The set value is either of the following:
		`typedef enum {`
		`  PJMSX_CSTMRSC_VALUE_TYPE_NUMERIC = 1,`
		`  PJMSX_CSTMRSC_VALUE_TYPE_STRING = 2`
		`} Pjmsx_cstmrsc_value_type_t;`

is_pernode	uint8_t	If the custom resource is a resource per node, 1 is set. If the custom resource is not a resource per node, 0 is set.
req_value_rsc	union	Set information on the requested custom resource If value_type is PJMSX_CSTMRSC_VALUE_TYPE_NUMERIC, a value is set in num_value. If value_type is PJMSX_CSTMRSC_VALUE_TYPE_STRING, a value is set in string_value.

C.4 Job Information Setting Function

This section describes the job information setting function APIs that can be used in exit functions.

There are three types of job information setting functions, classified as follows based on what functions they can call:

- Functions that can be called only from job manager exit functions (the names of the functions begin with "pjmx")
- Functions that can be called only from job scheduler exit functions (the names of the functions begin with "pjsx")
- Functions that can be called from both (the names of the functions begin with "pjmsx")

The job information setting function returns the following values.

Table C.18 Return values of the job information setting function

Return value	Meaning
0	The function ended normally.
Value other than 0	The function ended abnormally. For details on the value, see the description of the function.

C.4.1 pjmx functions

This section describes the job information setting functions (functions whose names begin with "pjmx") that can be called only from job manager exit functions.

When using a job information setting function, provide the member *extended_p* as an argument of the job information structure UsrJobInfo_t. The member is passed as an argument of an exit function.

The following table lists the job information setting function.

Set information	Message at job submission
Function name	int pjmx_setinfo_message(const char **message*, const void **extended_p*)
Description	Outputs the character string displayed by the argument *message* as detailed information for the following message when this function is called: - pjsub command : PJM 0079 - pjalter command : PJM 0579 - pmalter command : PJM 1079 The maximum length of the character string that can be specified in the argument *message* is 512 bytes (including the trailing NULL character). The available characters are single-byte alphanumeric characters and other displayable characters. If this function ends abnormally, the meaning of the return value is as follows: 1: The timing does not allow output. 4: Job information could not be acquired. 5: The character string specified in the argument *message* is longer than 512 bytes (including the trailing NULL character). 6: The character string specified in the argument *message* contains a character that cannot be used.
Possible caller	pjmx_quejob(), pjmx_alterjob()

Set information	Environment variable
Function name	int pjmx_setinfo_envdata(const void **data*, const void **extended_p*)
Description	Registers environment variables in the area specified by *data*. Store each environment variable in the "*name=value*" format, and separate multiple environment variables by a NULL character (\0). Set two consecutive NULL characters at the end of the area. This function copies the environment variables pointed to by *data*, and replaces all the set environment variables with the copied variables. Therefore, for environment variables that you do not want to change, reference them with the function pjmx_getinfo_envdata() in advance. Then, in this function, set them along with the environment variables to change. If this function ends abnormally, the meaning of the return value is as follows: 1: The timing does not allow the setting. 4: The value could not be set. 6: An invalid parameter is specified.
Possible caller	pjmx_quejob(), pjmx_startjob()

Set information	Supplementary information (Item SUPPLEMENTARY INFORMATION in job statistical information)

Function name	int pjmx_setinfo_supplementary_info(const char **info_p*, const void **extend_p*)
Description	Sets the character string shown by *info_p* as supplementary information in job statistical information. The set supplementary information is output to a statistical information file (.stats file), job statistical information (-s or -S option specified in the pjstat command), and a statistical information record (pmdumpjobinfo command). The character string pointed to by *info_p* must be 256 bytes or less, including the trailing NULL character. Note that if it contains a newline code or any other special character, the pjstat or pjsub command may not be able to output job statistical information properly. If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called by a exit function that cannot call the function. 4: Job information could not be set. 5: The length of the character string including the trailing character exceeds 256 bytes. 6: An invalid parameter is specified.
Possible caller	pjmx_quejob()

Set information	Requested quantity of a custom resource
Function name	int pjmx_setinfo_req_cstmrsc_numeric(const char **name_p*, int64_t *num*, const void **extend_p*)
Description	Specifies the amount of a custom resource defined in the pmpjm.conf file. *name_p* specifies the custom resource name (Name item in the pmpjm.conf file). *num* specifies the amount of the resource (a range of values defined by the Value item in the pmpjm.conf file). For details on custom resource definitions, see "Custom resource settings" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management." If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called by a exit function that cannot call the function. 4: Job information could not be set. 6: An invalid parameter is specified.
Possible caller	pjmx_quejob(), pjmx_alterjob()

Set information	Requested type of a custom resource
Function name	int pjmx_setinfo_req_cstmrsc_string(const char **name_p*, char **str_p*, const void **extend_p*)
Description	Specifies the request type of a custom resource defined in the pmpjm.conf file. *name_p* specifies the custom resource name (Name item in the pmpjm.conf file). *str_p* specifies the request type (a range of types defined by the Value item in the pmpjm.conf file). For details on custom resource definitions, see "Custom resource settings" in "Chapter 3 Job Operation Management Function Settings" in "Job Operation Software Administrator's Guide for Job Management." If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called by a exit function that cannot call the function. 4: Job information could not be set. 6: An invalid parameter is specified.
Possible caller	pjmx_quejob(), pjmx_alterjob()

Set information	Operation when a Tofu interconnect link goes down
Function name	int pjmx_setinfo_net_route(uint8_t *net_route_dynamic*, const void **extended_p*)
Description	Specifies the operation for whether to change the communication path when a link goes down for a communication path of the Tofu interconnect used by a job, in *net_route_dynamic*. 0: Do not change the communication path. The job ends abnormally. 1: Dynamically change the communication path, and job execution continues. The specification in this function has priority over the specification in the --net-route option of the pjsub command and the job ACL function settings. If this function ends abnormally, the meaning of the return value is as follows:

	1: The function was called by an exit function that cannot call this function. 4: Job information could not be acquired. 6: The value specified in *net_route_dynamic* is incorrect. 7: An internal error occurred. Confirm that the system contains no faults.
Possible caller	pjmx_quejob()

Set information	Power consumption estimate
Function name	int pjmx_setinfo_power_estimation(const char **cstmrsc_p*, double *ave_power*, double *weight*, double *max_power*, size_t *detailed_power_info_size*, const void **detailed_power_info_p*, const void **extended_p*)
Description	Sets a power consumption estimate for the job operation management function. The custom resource name *cstmrsc_p* is the name of the custom resource defined as system power consumption in the pmpjm.conf configuration file of the job operation management function. Set the following information: - Average power consumption estimate (W) shown by *ave_power* - Correction value (multiplicative factor) for the average power consumption estimate shown by *weight* The job scheduler uses "average power consumption estimate *ave_power* x correction value *weight*" as an estimate of power consumption. If you do not want correction, set 1 in *weight*. - Maximum power consumption estimate (W) shown by *max_power* It is used for future extension. - Area size of *detailed_power_info_p* shown by *detailed_power_info_size* It is used for future extension. - Initial address of the storage area for detailed power information (e.g., time series power data) shown by *detailed_power_info_p* It is used for future extension. You can set this information only in the job registration functions pjmx_quejob() and pjmx_alterjob(). If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called from an exit function that cannot call the function. 3: Failure (An invalid parameter is specified.) 4: The value could not be set. 6: An invalid parameter is specified.
Possible caller	pjmx_quejob(), pjmx_alterjob()

Set information	Value of a job statistical information item defined by the job operation administrator
Function name	int pjmx_setinfo_stats_item_*type*(const char **record_name*, uint32_t *id*, const char **item_name*, *type value*, const void **extended_p*) The function is one of multiple functions that differ in the *type* part of the function name.
Description	Sets the item *value* of the *type* type as the value of the item *item_name* of the record specified by *record_name* and *id*. There is a function for each type of value to set, and the *type* part of the function name indicates the type. For details on each function name, see the table below. *record_name*: Record name ("JI": Job statistical information, "JN": Node statistical information, "SUM": Summary information of a bulk or step job). Attempting to set node statistical information by this function calling from pjmx_quejob() or pjmx_startjob() results in an error (Return value 3). *id*: Record ID. For node statistical information, specify a node ID. For job statistical information, *id* is ignored. *item_name*: Item name. You can specify only the items defined by the job operation administrator. *value*: Value of the *type* type *extended_p*: Job information passed to an exit function The type of item to set is DataType defined in the papjmstats.conf file.

	If this function ends abnormally, the meaning of the return value is as follows: 3: Failure (An invalid parameter is specified.) 4: The value could not be set. 8: The type of the value does not match the type defined for the item. 9: A value cannot be set for the specified item.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob()

The following table lists the functions prepared for each type of item.

Type of item	Function
PJMX_DATATYPE_CHAR	int pjmx_setinfo_stats_item_char(const char **record_name*, uint32_t *id*, const char **item_name*, char *value*, const void **extended_p*)
PJMX_DATATYPE_INT8	int pjmx_setinfo_stats_item_int8(const char **record_name*, uint32_t *id*, const char **item_name*, int8 *value*, const void **extended_p*)
PJMX_DATATYPE_UINT8	int pjmx_setinfo_stats_item_uint8(const char **record_name*, uint32_t *id*, const char **item_name*, uint8 *value*, const void **extended_p*)
PJMX_DATATYPE_INT16	int pjmx_setinfo_stats_item_int16(const char **record_name*, uint32_t *id*, const char **item_name*, int16 *value*, const void **extended_p*)
PJMX_DATATYPE_UINT16	int pjmx_setinfo_stats_item_uint16(const char **record_name*, uint32_t *id*, const char **item_name*, uint16 *value*, const void **extended_p*)
PJMX_DATATYPE_INT32	int pjmx_setinfo_stats_item_int32(const char **record_name*, uint32_t *id*, const char **item_name*, int32 *value*, const void **extended_p*)
PJMX_DATATYPE_UINT32	int pjmx_setinfo_stats_item_uint32(const char *record_name, uint32_t *id*, const char **item_name*, uint32 *value*, const void **extended_p*)
PJMX_DATATYPE_INT64	int pjmx_setinfo_stats_item_int64(const char *record_name, uint32_t *id*, const char **item_name*, int64 *value*, const void **extended_p*)
PJMX_DATATYPE_UINT64	int pjmx_setinfo_stats_item_uint64(const char *record_name, uint32_t *id*, const char **item_name*, uint64 *value*, const void **extended_p*)
PJMX_DATATYPE_FLOAT	int pjmx_setinfo_stats_item_float(const char *record_name, uint32_t *id*, const char **item_name*, float *value*, const void **extended_p*)
PJMX_DATATYPE_DOUBLE	int pjmx_setinfo_stats_item_double(const char *record_name, uint32_t *id*, const char **item_name*, double *value*, const void **extended_p*)
PJMX_DATATYPE_TIMESPEC	int pjmx_setinfo_stats_item_timespec(const char *record_name, uint32_t *id*, const char **item_name*, struct timespec *value*, const void **extended_p*)
PJMX_DATATYPE_TIME	int pjmx_setinfo_stats_item_time(const char *record_name, uint32_t *id*, const char **item_name*, time_t *value*, const void **extended_p*)
PJMX_DATATYPE_SIZE	int pjmx_setinfo_stats_item_size(const char *record_name, uint32_t *id*, const char **item_name*, size_t *value*, const void **extended_p*)
PJMX_DATATYPE_STRING	int pjmx_setinfo_stats_item_string(const char *record_name, uint32_t *id*, const char **item_name*, char **value*, const void **extended_p*)

C.4.2 pjsx functions

This section describes the job information setting functions (functions whose names begin with "pjsx") that can be called only from job scheduler exit functions.

When using a job information setting function, provide the pointer *extended_p* for extended information as an argument. The pointer is passed as an argument of a job scheduler exit function.

The following table shows the job information setting function API.

Set information	Flag for judging whether the job is excluded from scheduling

Function name	int pjsx_setinfo_sched_skip(int *sched_skip*, void **extended_p*)
Description	Specifies 1 for the argument *sched_skip* for a job excluded from scheduling. If the specified value is other than 1, the job is regarded as subject to scheduling. Only the job preselection exit function pjsx_prejobselect() can call this function. If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called from an exit function that cannot call the function (called from a function other than the job preselection exit function). 3: Failure (An invalid parameter is specified.) 4: Failed setting (The argument *extended_p* is NULL or has invalid contents.)
Possible caller	pjsx_prejobselect()

C.4.3 pjmsx functions

This section describes the job information setting functions (functions whose names begin with "pjmsx") that can be called from both job manager exit functions and job scheduler exit functions.

When using a job information setting function, provide the member *extended_p* as an argument of the job information structure UsrJobInfo_t, or provide the pointer *extended_p* for extended information as an argument. The member is passed as an argument of a job manager exit function. The pointer is passed as an argument of a job scheduler exit function.

Set information	Job processing error message (job statistical information item REASON)
Function name	int pjmsx_setinfo_reason(const char **reason*, const void **extended_p*)
Description	Normally, when an exit function returns a value other than PJMX_SUCCESS, the character string "GATE CHECK" is output to the REASON item in job statistical information. When this function is called, the character string shown by the argument *reason* instead of the character string "GATE CHECK" is output to the REASON item. The maximum length of the character string that can be specified in the argument *reason* is 64 bytes (including the trailing NULL character). The available characters are single-byte alphanumeric characters and other displayable characters. If this function ends abnormally, the meaning of the return value is as follows: 1: The function was called from an exit function that cannot call the function. 3: Failure (An invalid parameter is specified.) 4: Job information could not be acquired. 5: The character string specified in the argument *reason* is longer than 64 characters (including the trailing NULL character). 6: The character string specified in the argument *reason* contains a character that cannot be used.
Possible caller	pjmx_quejob(), pjmx_startjob(), pjmx_alterjob(), pjmx_endjob(), pjmx_deljob() pjsx_prejobselect(), pjsx_postjobselect(), pjsx_postrscalloc()